

НИУ «МЭИ»

Кафедра «Релейной защиты и автоматизации энергосистем»

Алгоритмы РЗА и их программная реализация

## КУРСОВАЯ РАБОТА

«Реализация функции АПВ с контролем синхронизма в соответствии МЭК-61850»

Группа: Э-13м-24

ФИО студентов: Воложанин А. С.

ФИО преподавателя: Рыжков А. К.

Москва

2025

## **Исходные данные**

Для демонстрации работы автоматического повторного включения (АПВ) дополнительно была реализованна токовая направленная ступенчатая защита. АПВ было выполнено с пофазным исполнением в соответствии со стандартом СТО 56947007-29.240.10.248-2017 и СТО 56947007-29.240.10.299-2020. В ходе работы был реализован блок контроля синхронизма и напряжения со следующими режимами контроля напряжения:

- 0-Выведено;
- 1-ОНЛ+ННШ;
- 2-ННЛ+ОНШ;
- 3-ОНЛ+ННШ/ННЛ+ОНШ

## Теоретическая справка

АПВ предусматривается на воздушных линиях электропередач (ВЛ) и сборных шин открытых распределительных устройств.

Если линия электропередачи является кабельно-воздушной (КВЛ) 110 кВ и выше, то АПВ должна срабатывать только на участках воздушных линиях электропередач.

На линиях электропередач (ЛЭП), по которым возможна синхронизация несинхронно работающих частей электроэнергетической системы, должны устанавливаться устройства трёхфазных АПВ с улавливанием синхронизма.

На ЛЭП 330-750 кВ должны устанавливаться однофазное АПВ (ОАПВ), трёхфазное АПВ (ТАПВ) и ускоренное трёхфазное АПВ (УТАПВ).

АПВ на ВЛ и сборных шин 330-750 кВ должны быть реализованы:

- однократность действия;
- действие на включение выключателя по факту наличия готовности выключателя линии и устройства АПВ, с установленной выдержкой времени;
- запрет действия АПВ при отключении выключателя персоналом от ключа управления или с АРМ ОП ПС;
- возможность запрета ТАПВ от внешних устройств (УРОВ, защиты от неполнофазного режима и т.п.);
- возможность запрета ТАПВ при неуспешном автоматическом включении одной фазы (неуспешное ОАПВ);
- возможность реализации ТАПВ выключателя с увеличенной выдержкой времени после неуспешного ОАПВ;
- взаимный запрет ТАПВ выключателей при неуспешном ТАПВ выключателя, включаемого первым (при наличии двух выключателей на линии);
- отсутствие запрета ТАПВ в цикле ОАПВ при возникновении КЗ на другой фазе;
- оперативный ввод/вывод ОАПВ, ТАПВ, изменение алгоритма контроля ТАПВ посредством местного и дистанционного управления;

- контроль погасания дуги на отключенной фазе/фазах;
- разные выдержки времени ТАПВ для линии и шин (при использовании автоматического опробования систем шин).

Должны предусматриваться следующие виды контроля цепи пуска ТАПВ:

- с контролем отсутствия напряжения на линии (шинах) и наличия симметричного напряжения на шинах (АТ, Т);
- с контролем отсутствия напряжения на шинах и наличия симметричного напряжения на линии (АТ, Т);
- с контролем наличия синхронизма и контролем наличия симметричного напряжения на линии (АТ, Т) и на шинах;
- с улавливанием синхронизма и контролем наличия симметричного напряжения на линии (АТ, Т) и на шинах.

Применение на ЛЭП 330-750 кВ адаптивного ОАПВ должно быть обосновано при проектировании.

На ВЛ, ОВ, шинах (ошиновке) напряжением 110-220 кВ должно применяться ТАПВ с пуском по цепи «несоответствия» и/или от защит.

ОАПВ на линиях 220 кВ может выполняться только при наличии обосновывающих расчётов. Для линий с ОАПВ каждый из комплектов защиты должен иметь логику однофазного отключения выключателей и возможность перевода ее действия на отключение трёх фаз.

На ВЛ с двухсторонним питанием ТАПВ должно выполняться с однократным действием, а на ВЛ с односторонним питанием - с двукратным действием.

При выполнении ТАПВ должно быть реализовано:

- действие на включение выключателя по факту наличия готовности выключателя линии и устройства АПВ с установленной выдержкой времени;
- запрет при отключении выключателя персоналом от ключа управления или с АРМ ОП ПС;
- возможность запрета ТАПВ от внешних устройств;
- взаимный запрет ТАПВ выключателей при неуспешном ТАПВ выключателя, включаемого первым (при наличии двух выключателей на линии);
- оперативный ввод/вывод ТАПВ, изменение алгоритма контроля ТАПВ посредством местного и (при наличии АСУ ТП) дистанционного управления;
- разные выдержки времени ТАПВ для линии и шин (при использовании автоматического опробования систем шин).

На линиях с двухсторонним питанием при обосновании должны предусматриваться следующие виды контроля цепи пуска ТАПВ:

- с контролем отсутствия напряжения на линии (шинах) и наличия напряжения на шинах (АТ, Т);
- с контролем отсутствия напряжения на шинах и наличия напряжения на линии (АТ, Т);

– с контролем наличия синхронизма напряжений на линии (АТ, Т) и на шинах.

На линиях 110-220 кВ должна предусматриваться возможность реализации АПВ без контроля напряжений и синхронизма.

Стандарт МЭК 61850 представляет собой инновационный подход к преобразованию обычных подстанций в цифровые. Основное отличие заключается в использовании передового цифрового оборудования, которое осуществляет передачу информации в цифровом формате вместо аналогового. Для этого рядом с каждым компонентом оборудования устанавливается специальное цифровое устройство, известное как merging unit, которое ответственно за передачу дискретных цифровых сигналов на коммутатор.

Данные передаются в формате SV (Sampled value) или GOOSE (Generic Object-Oriented Substation Event) пакетов. Коммутатору необходимы значительные вычислительные мощности, так как, например, SV пакеты передаются с высокой частотой, требуя надежной обработки и оцифровки данных.

Через GOOSE пакеты передается информация о текущем состоянии оборудования, таких как положение выключателей или заземляющих ножей. Чтобы снизить нагрузку на сеть, пакеты отправляются периодически: если состояние оборудования не меняется, то пакеты отправляются реже, но при изменениях данные передаются чаще.

Стандарт МЭК также устанавливает требования к структуре цифровых устройств для обеспечения их унификации. Каждый элемент оборудования задается конфигурационными файлами на языке SCL. В первых разделах МЭК описывается IED (intelligent electronic device) как основной цифровой терминал, сгруппированный в LD (Logical device), содержащий LN (Logical node) и DO (Data object), включающий DA (Data attribute).

## Практическая часть

В ходе работы был реализован функции АПВ с контролем синхронизма в соответствии МЭК-61850. Для имитации входного сигнала использовались рсар файлы.

Основное назначение АПВ — повышение надежности и бесперебойности электроснабжения потребителей за счет восстановления работы объекта энергосистемы (потребитель, участок линии электропередачи, шины, трансформатор и т.п.), повторно включаемых после отключения. Значительная часть коротких замыканий на воздушных ЛЭП, вызванных перекрытием изоляции, схлестыванием проводов и др. причинами, при достаточно быстром отключении повреждения релейной защитой самоустраняется. Такие самоустраняющиеся повреждения принято называть неустойчивыми. Доля неустойчивых повреждений в зависимости от номинального напряжения установки, согласно статистическим исследованиям, составляет 50–90 %.

Ниже приведен алгоритм расчета АПВ с релейной защитой на линиях с односторонним питанием:

Выдержка времени:

1. Ожидание деионизации среды  $t_{\text{АПВ}} = t_{\text{д.с.}} + t_{\text{зап}} - t_{\text{вкл } Q}$
2. Ожидание готовности привода  $t_{\text{АПВ}} = t_{\text{г.п.}} + t_{\text{зап}}$
3. Ожидание возврата защиты  $t_{\text{АПВ}} = t_{\text{в.з.}} + t_{\text{зап}} - t_{\text{вкл } Q}$

УДЗ после АПВ: на каждой линии предусмотрена установка селективной защиты и АПВ. В случае неуспешного АПВ время срабатывания защиты этого элемента может быть сделано равным нулю, что не приведёт к неселективному отключению повреждённого элемента системы.

$$t_{\text{УДЗ после}} = t_{\text{с.з.}} + t_{\text{зап}}$$

УДЗ до АПВ: предусматривается быстрое отключение КЗ в любой точке рассматриваемой сети неселективной защитой. При возникновении КЗ на одном из участков сети без выдержки времени срабатывает НО и отключает выключатель головного участка сети. Затем НО выводится из работы, и, если КЗ оказалось устойчивым, селективная защита отключает поврежденный участок сети. В радиальной сети на головном участке: НО, СЗ, АПВ; на остальных: СЗ.

$$t_{\text{УДЗ до } i} = t_{\text{с.з. } i} + t_{\text{зап}}$$

Поочерёдное АПВ: для предотвращения нарушения устойчивости на всех участках сети, на которых при К(3)  $U_{\text{ш}} \leq 0,6U_{\text{ном}}$  необходимо установить

неселективные отсечки без выдержки времени. В случае поочередного АПВ УДЗ происходит комбинированно. До АПВ защита действует ускорено. После АПВ в защиту приходит сигнал о срабатывании АПВ, от которого запускаются две выдержки времени:  $t_{удз(п)}$  — выдержка времени ускорения защиты после АПВ, и  $t_{удз(до)}$  — выдержка времени вывода ускорения защиты после АПВ. В этом случае защита будет действовать ускорено после АПВ в течение времени  $t_{удз(п)}$ , после чего УДЗ выводится на время:  $t_{удз(до+п)}$

$$t_{удз \text{ до и после } i} = t_{АПВ \text{ max}} - t_{АПВ i} \quad t_{АПВ i+1} = t_{АПВ i} + t_{зап}$$

АПВ линии с двухсторонним питанием:

- АПВ должно выполняться после того, как ВЛ будет отключено с 2х сторон, чтобы не подпитывать КЗ и восстановить изоляционные свойства воздуха после гашения дуги
- АПВ действует на выключатели с обоих концов поверженного участка. Выдержка времени с обоих концов разное во избежание включения линии с двух сторон на устойчивое КЗ
- За время бестоковой паузы АПВ, источники питания по концам линии могут выпасть из синхронизма

Когда имеется 3 и более линий, связывающих энергосистемы, то отключение одной из них не приведёт к расхождению синхронизма, поэтому можно использовать несинхронное АПВ.

Первой включается система с меньшими токами короткого замыкания (более слабая система)

$$t_{АПВ 1} = t_{с.з.2 \text{ max}} - t_{с.з.1 \text{ min}} + t_{откл Q 2} - t_{откл Q 1} + t_{д.с.} + t_{зап} - t_{вкл Q 1}$$

$$t_{АПВ 2} = t_{с.з.1 \text{ max}} - t_{с.з.2 \text{ min}} + t_{откл Q 1} - t_{откл Q 2} + t_{АПВ 1} + t_{зап} - t_{вкл Q 2}$$

Для АПВ с контролем напряжения:

$$t_{АПВ 2} = t_{с.з.1 \text{ max}} - t_{с.з.2 \text{ min}} + t_{откл Q 1} - t_{откл Q 2} + t_{зап}$$

На рис. 2 показана UML-диаграмма реализации функции АПВ с контролем синхронизма в соответствии МЭК-61850.

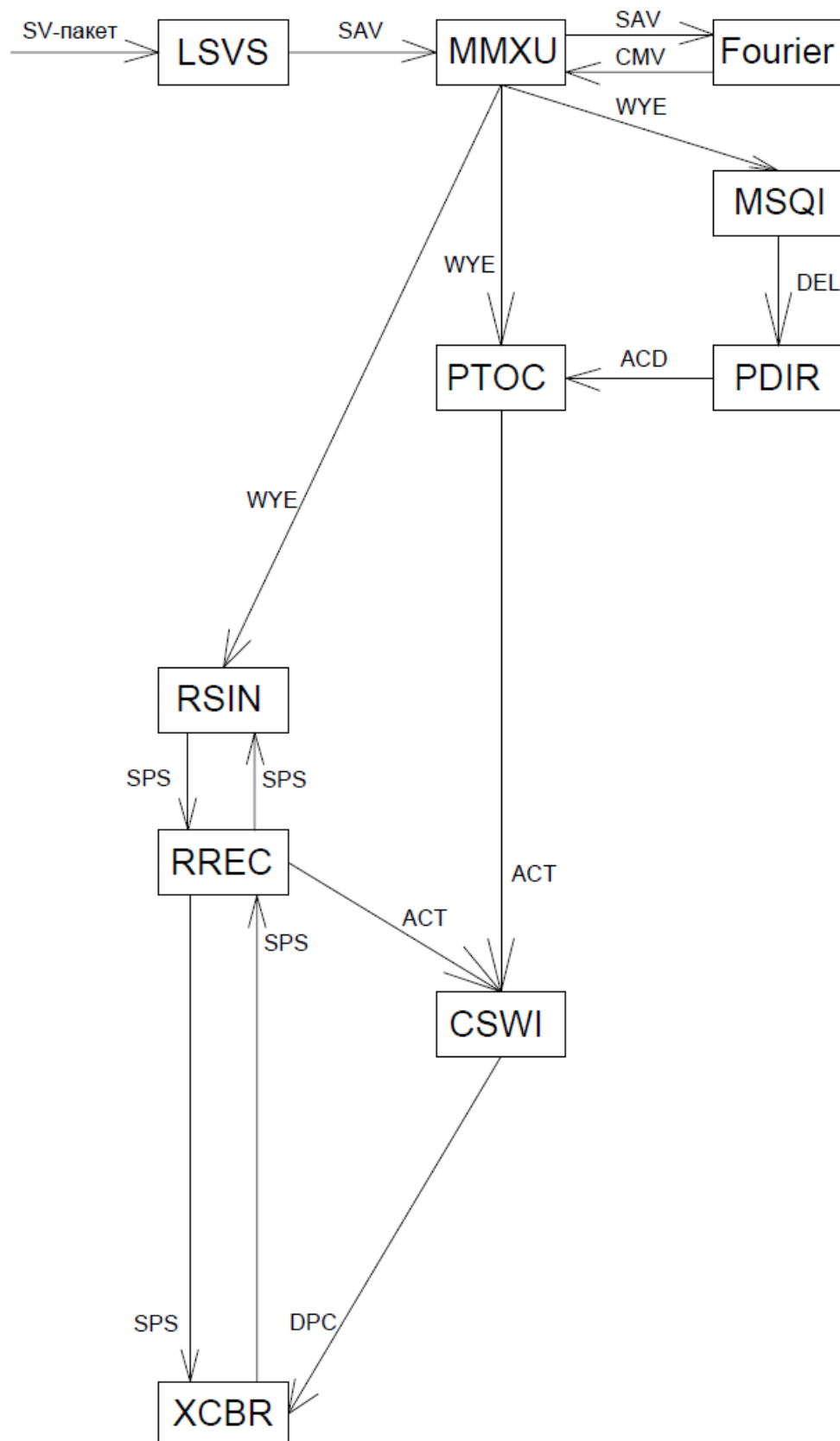


Рис. 1 UML-диаграмма

Представленные на данном рисунке логические узлы выполняют следующие функции:



- LSVS – осуществляет считывание входного сигнала;
- MMXU – совершает необходимые операции над полученными значениями и направляет их в фильтр;
- Fourier – фильтр Фурье, позволяющий получить величины в ортогональном формате;
- RDIR – реализует контроль направления мощности;
- MSQI – вычисляет симметричные составляющие, необходимые для блокировки при синхронных качаниях;
- PTOC – реализует токовую направленную защиту
- RSYN – реализует контроль синхронизма и напряжения
- RREC – реализует функцию АПВ
- CSWI – контроль сигналов с логического узла защиты на выключатель;
- XCBR – контроль состояния выключателя.

## Результаты проведения опытов

### Опыт 1:

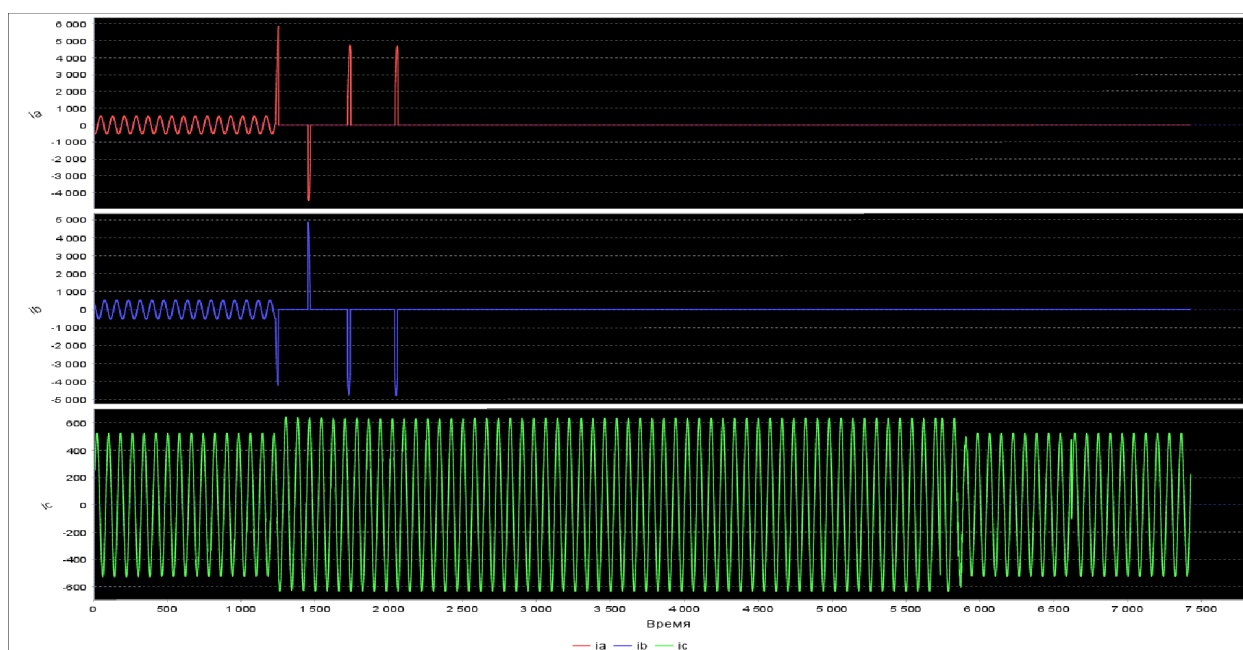


Рис. 2. Мгновенные значения опыта 1

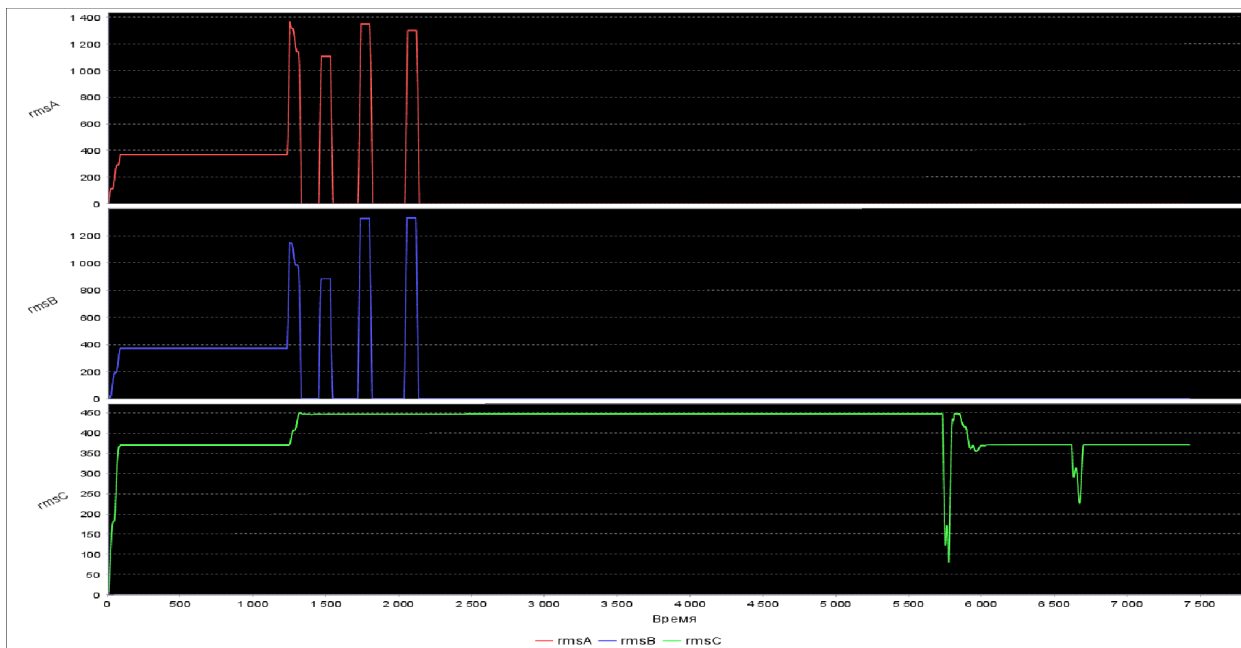


Рис. 3. Действующие значения опыта 1

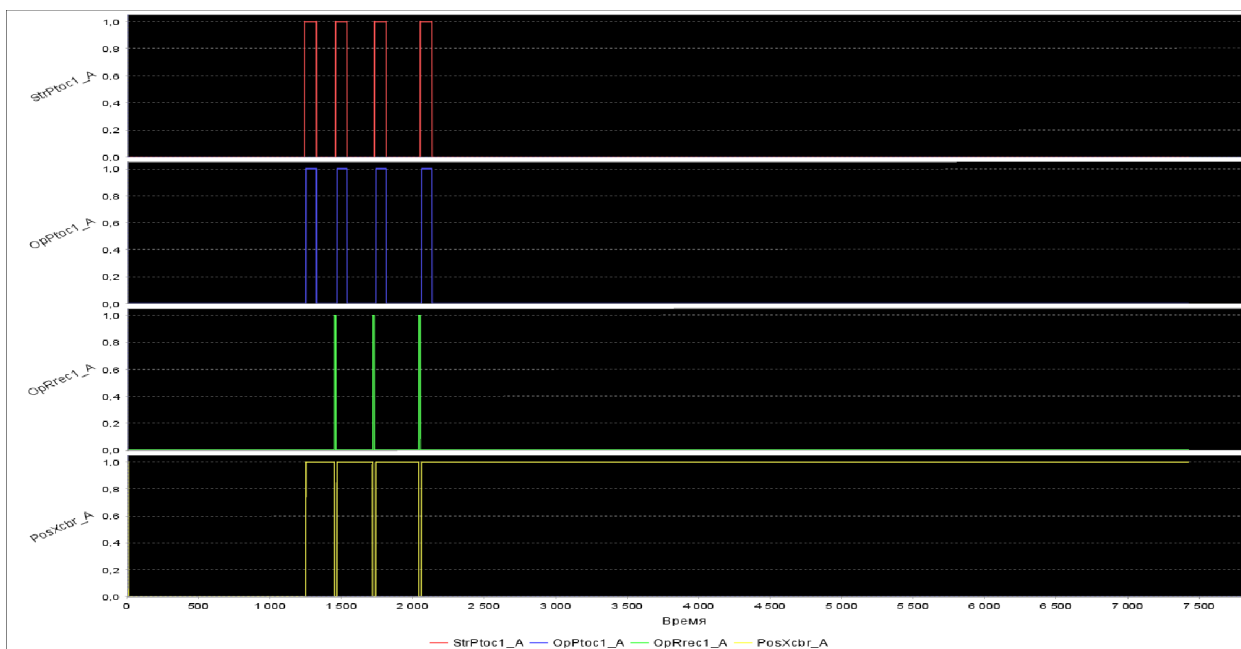


Рис. 4. Дискретные значения фазы А опыта 1

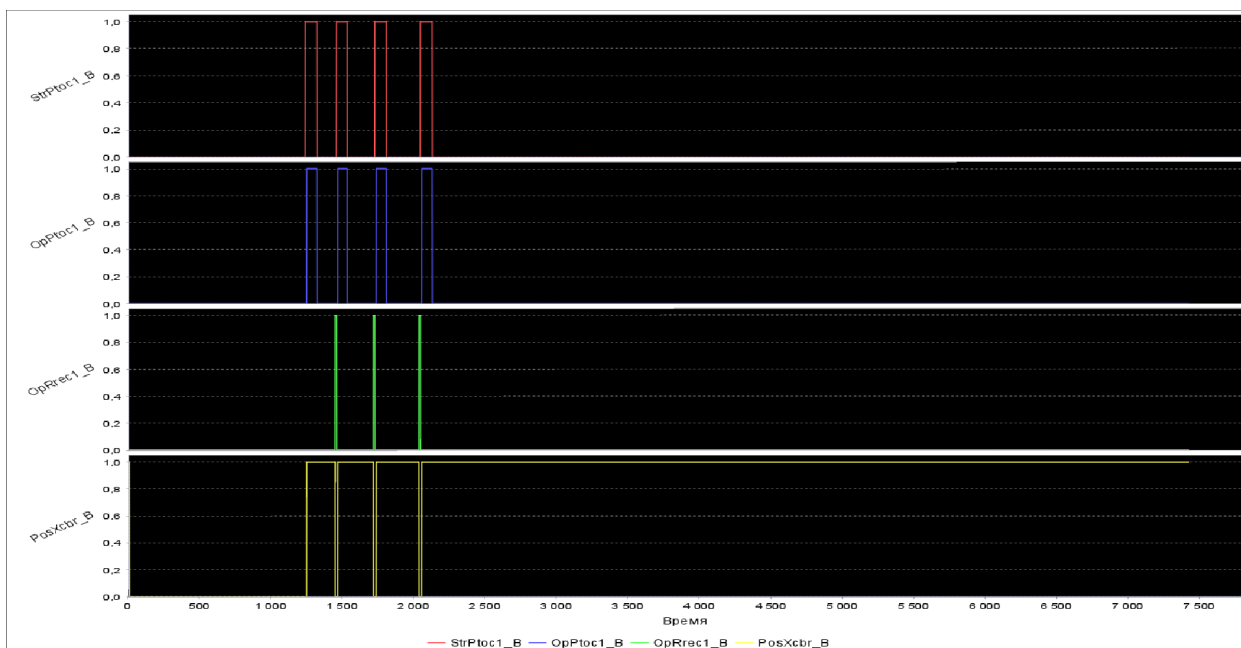


Рис. 5. Дискретные значения фазы В опыта 1

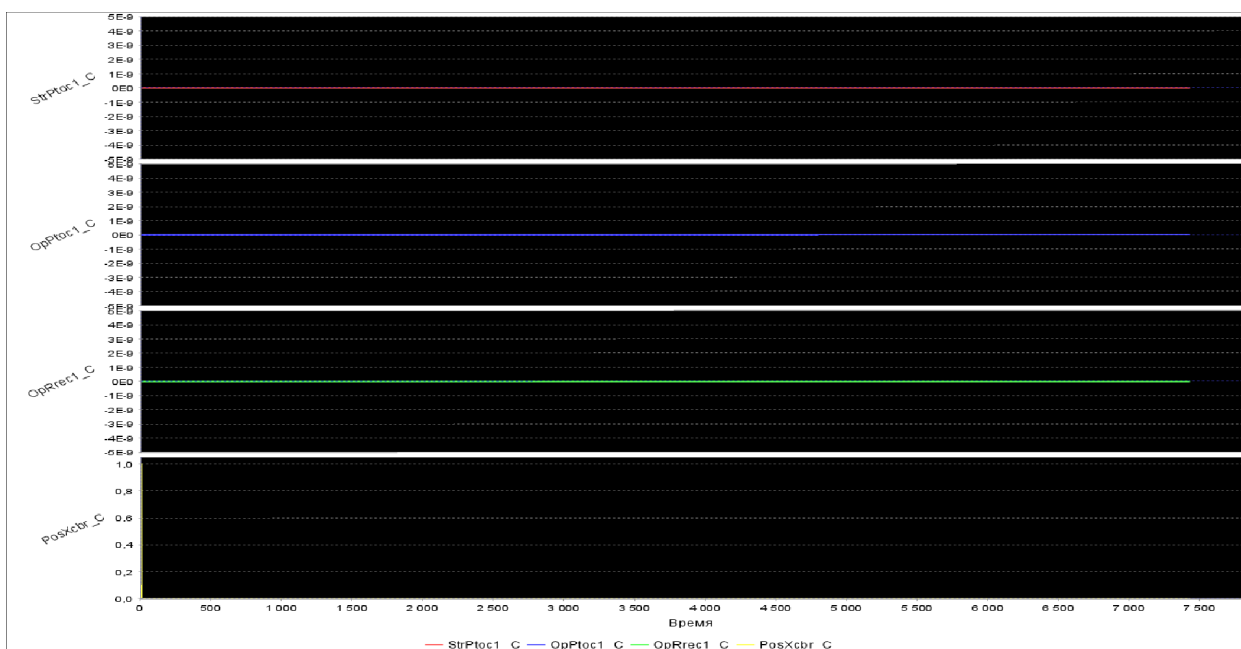


Рис. 6. Дискретные значения фазы С опыта 1

КЗ в ф.А возникло в: 21:01:54.519833800

Защита ф.А сработала в: 21:01:54.532006900

АПВ ф.А сработало в: 21:01:54.618455300

КЗ в ф.А возникло в: 21:01:54.621153100

Защита ф.А сработала в: 21:01:54.623649800

АПВ ф.А сработало в: 21:01:54.683220700

КЗ в ф.А возникло в: 21:01:54.686558100

Защита ф.А сработала в:21:01:54.689749300Z

АПВ ф.А сработало в: 21:01:54.825482900

КЗ в ф.А возникло в: 21:01:54.829681500

Защита ф.А сработала в: 21:01:54.834873600

## Опыт 2:

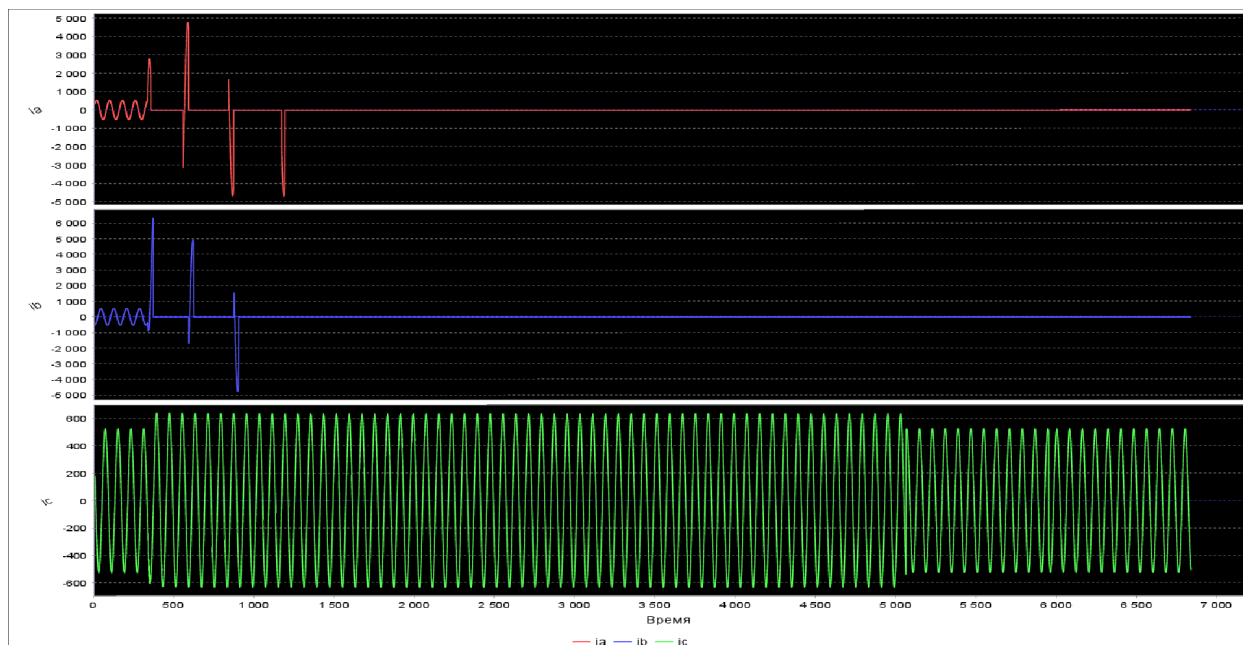


Рис. 7. Мгновенные значения опыта 2

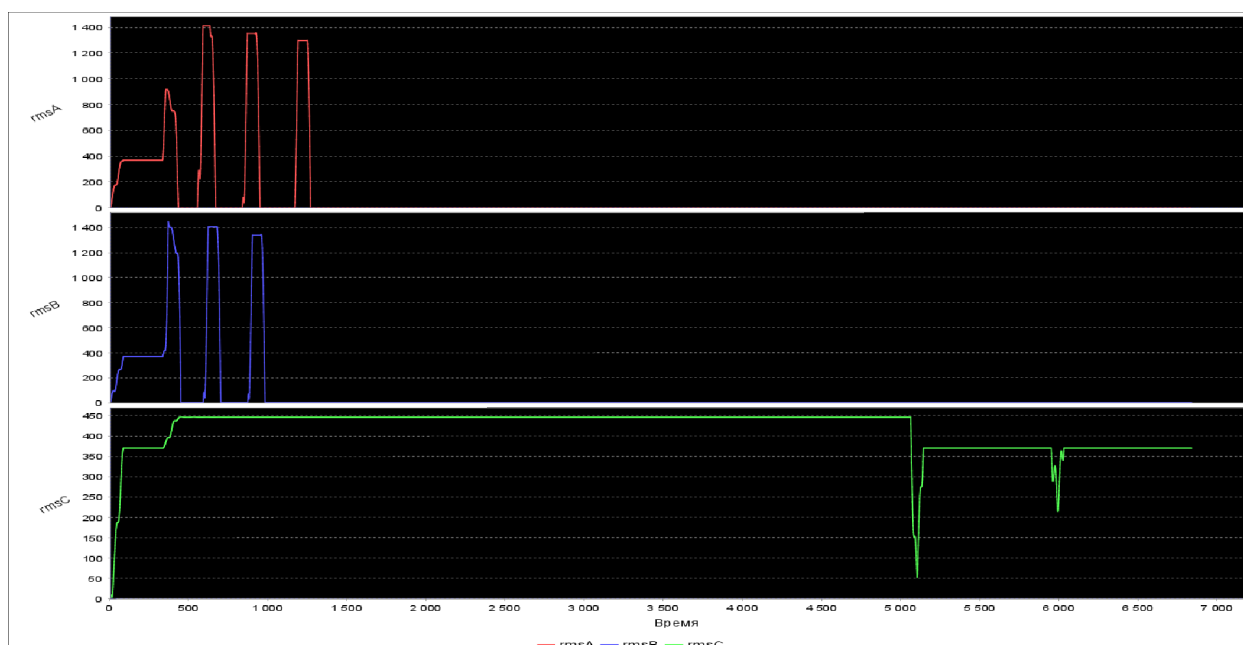


Рис. 8. Действующие значения опыта 2

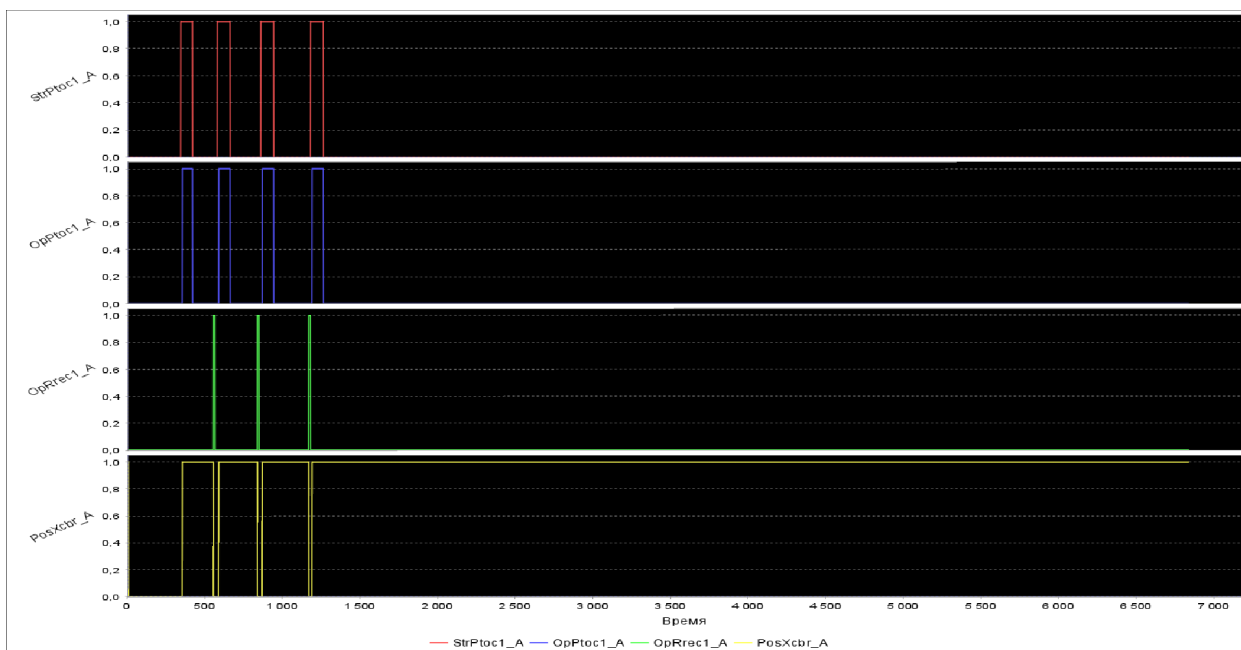


Рис. 9. Дискретные значения фазы А опыта 2

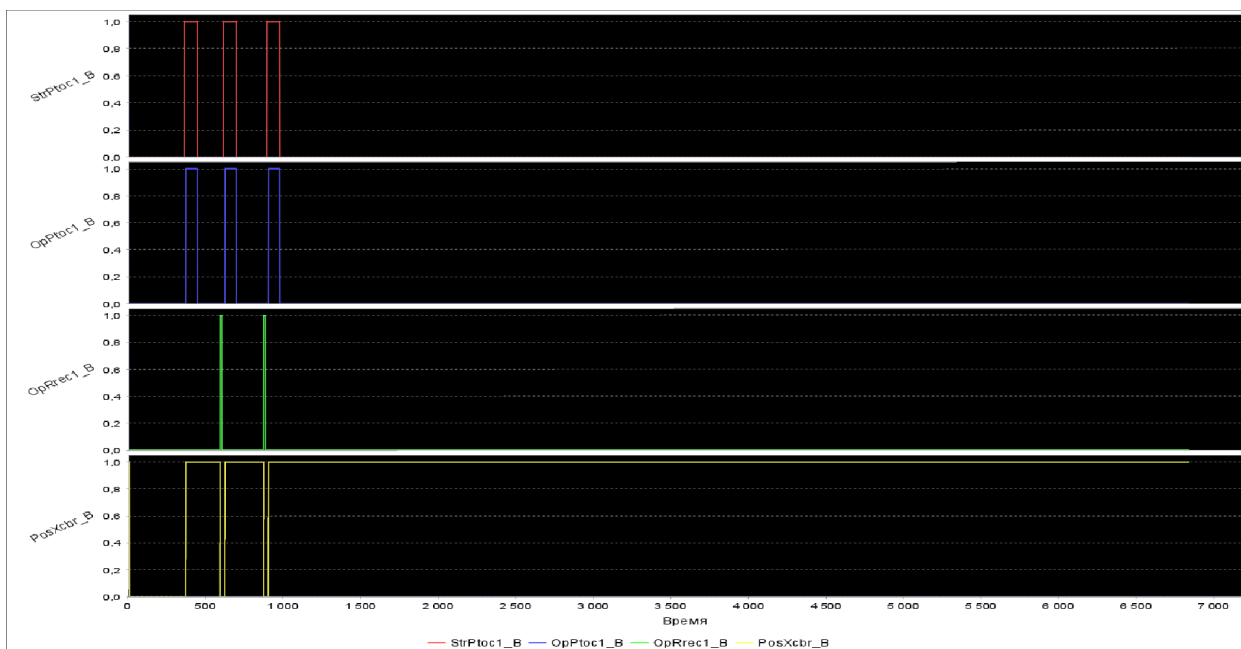


Рис. 10. Дискретные значения фазы В опыта 2

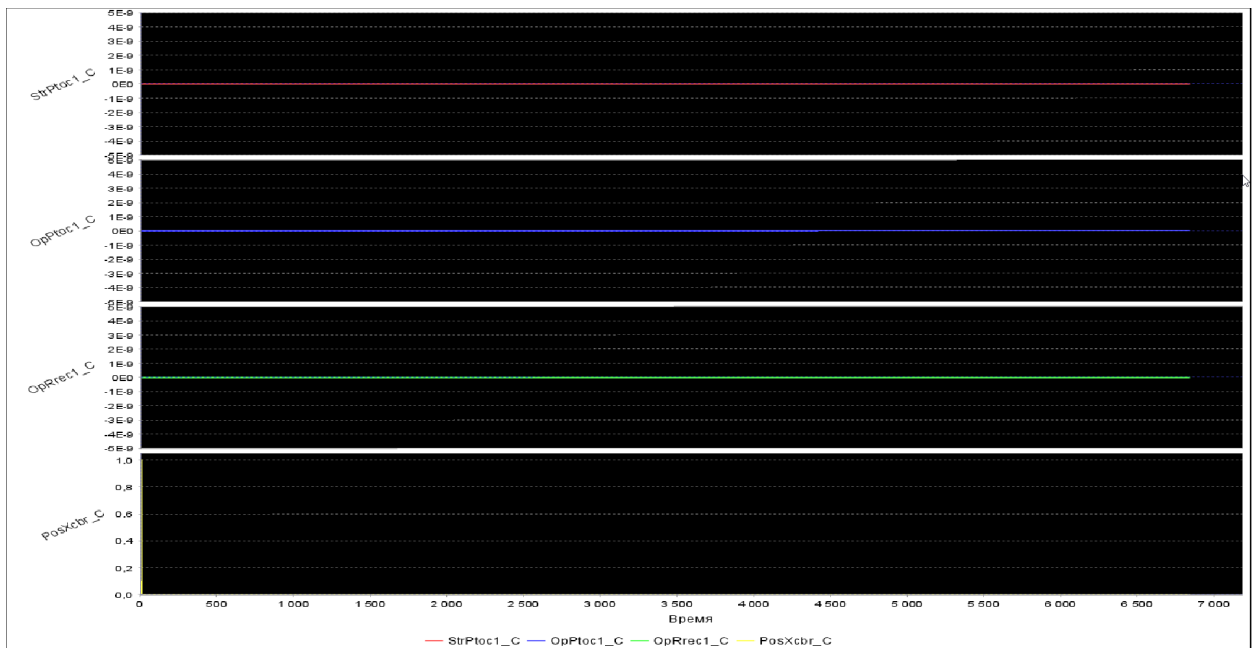


Рис. 11. Дискретные значения фазы С опыта 2

КЗ в ф.А возникло в: 21:07:32.178147200

Защита ф.А сработала в: 21:07:32.191702800

АПВ ф.А сработало в: 21:07:32.296857400

КЗ в ф.А возникло в: 21:07:32.314388300

Защита ф.А сработала в: 21:07:32.316896

АПВ ф.А сработало в: 21:07:32.460000500

КЗ в ф.А возникло в: 21:07:32.468111500

Защита ф.А сработала в: 21:07:32.472653100

АПВ ф.А сработало в: 21:07:32.558982900

КЗ в ф.А возникло в: 21:07:32.565653800

Защита ф.А сработала в: 21:07:32.567653400

### Опыт 3:

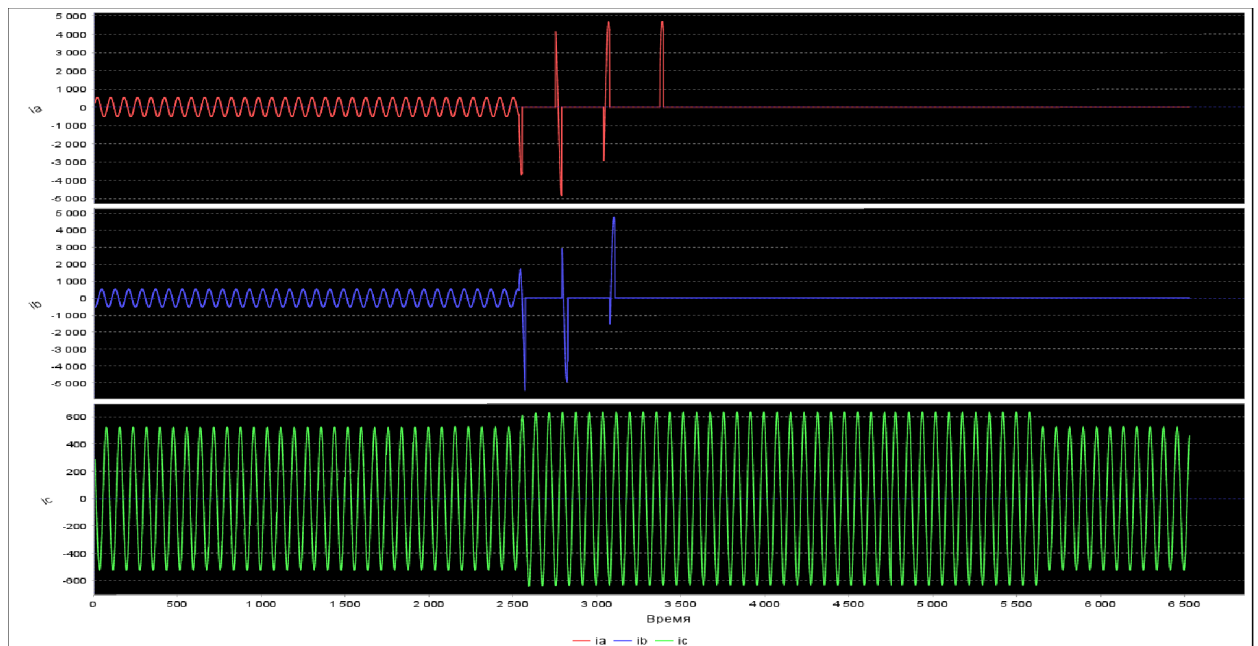


Рис. 12. Мгновенные значения опыта 3

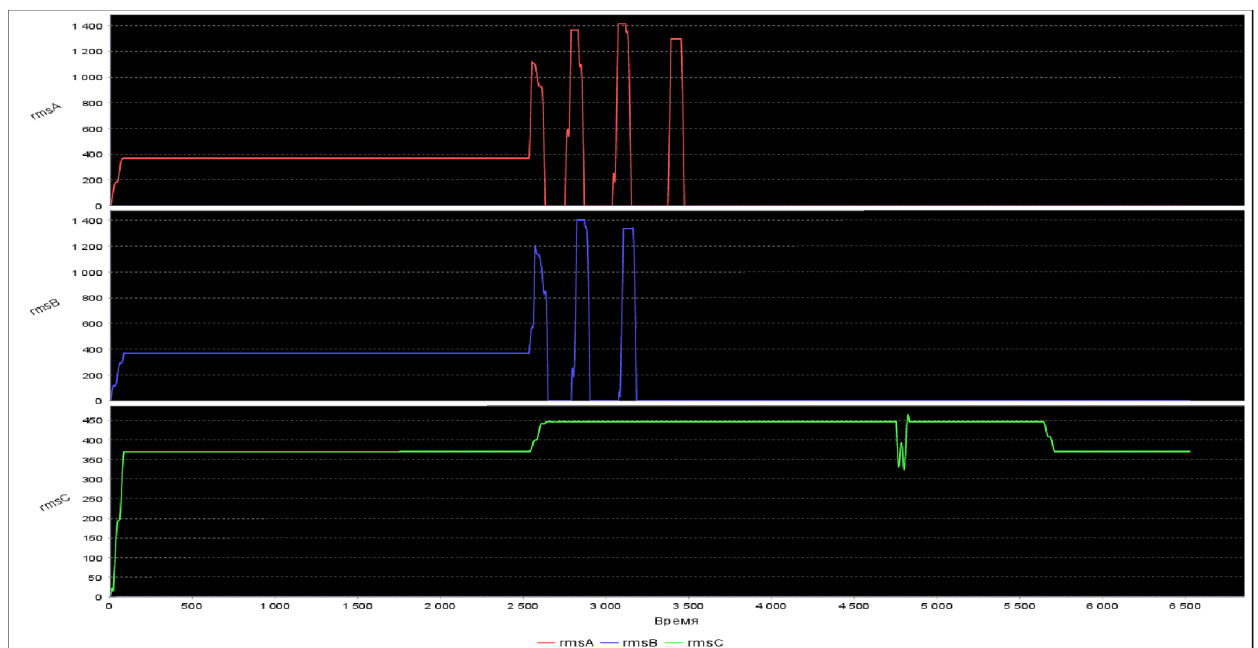


Рис. 13. Действующие значения опыта 3

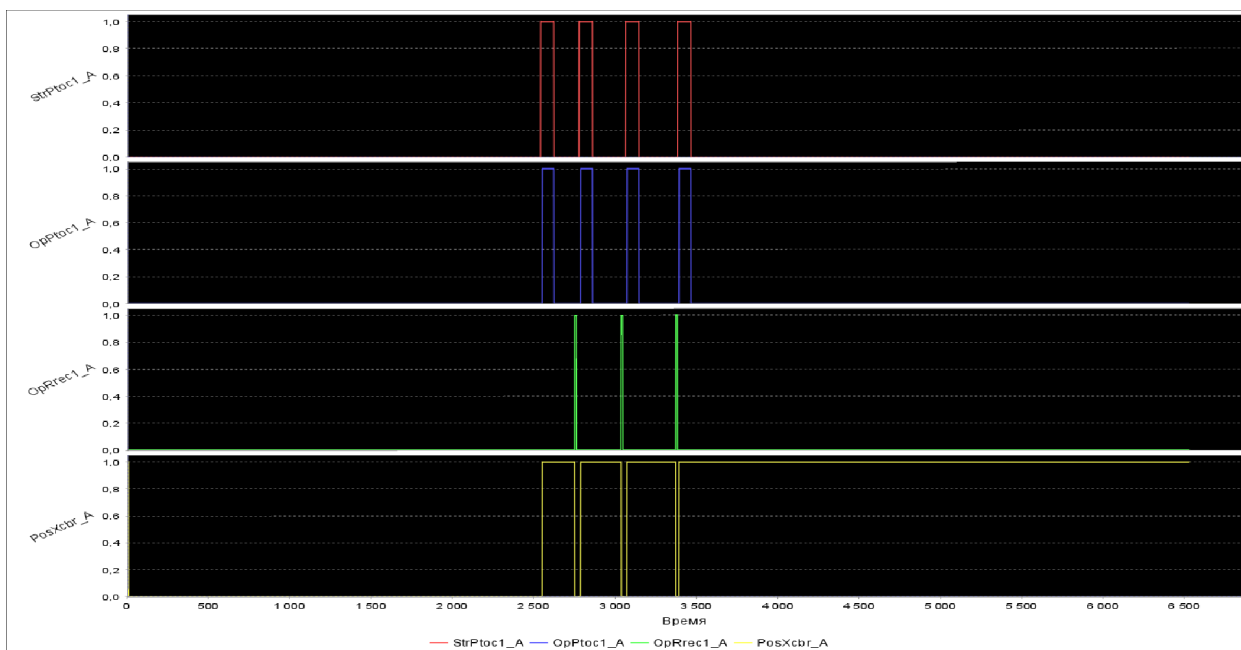


Рис. 14. Дискретные значения фазы А опыта 3

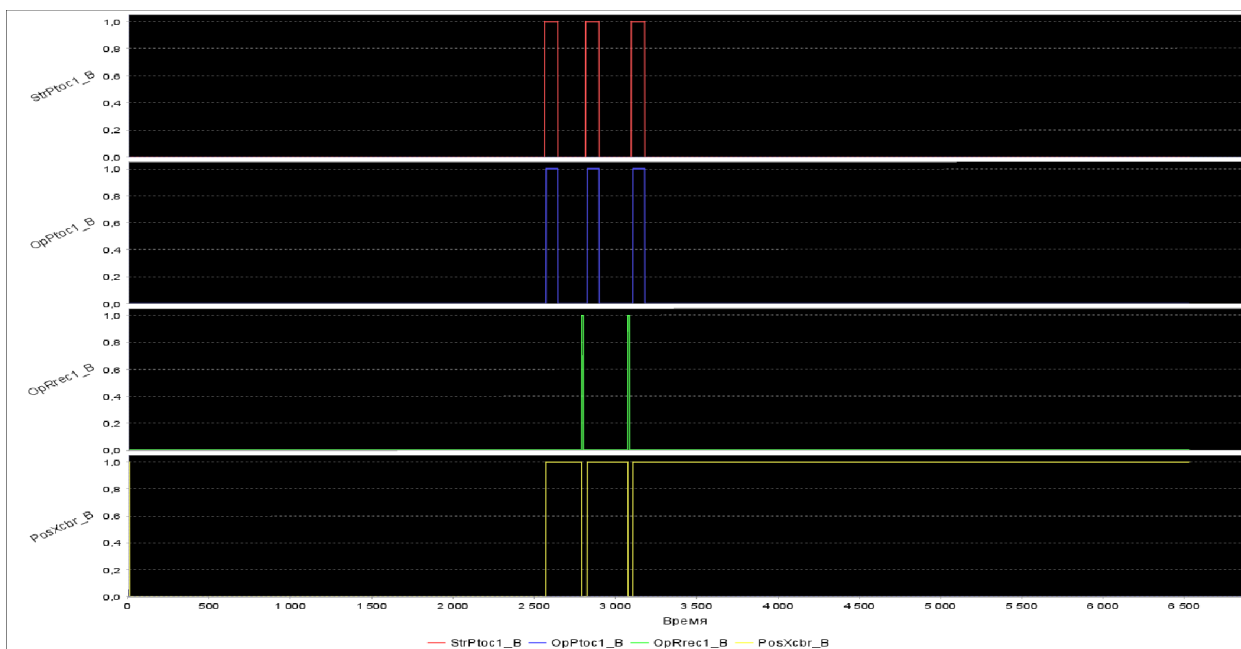


Рис. 15. Дискретные значения фазы В опыта 3



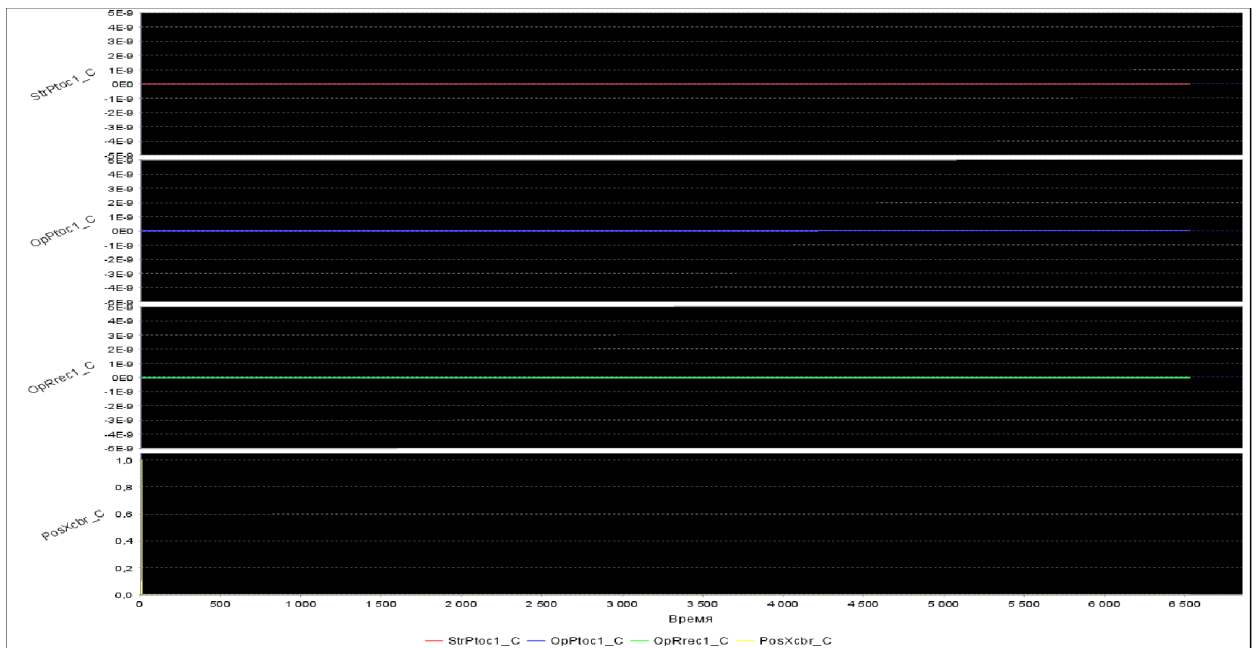


Рис. 16. Дискретные значения фазы С опыта 3

КЗ в ф.А возникло в: 21:09:33.946869900

Защита ф.А сработала в: 21:09:33.960197200

АПВ ф.А сработало в: 21:09:34.078923400

КЗ в ф.А возникло в: 21:09:34.091475600

Защита ф.А сработала в: 21:09:34.098610900

АПВ ф.А сработало в: 21:09:34.253912700

КЗ в ф.А возникло в: 21:09:34.272975

Защита ф.А сработала в: 21:09:34.274330200

АПВ ф.А сработало в: 21:09:34.448290500

КЗ в ф.А возникло в: 21:09:34.453639600

Защита ф.А сработала в: 21:09:34.457860300

## Опыт 4:

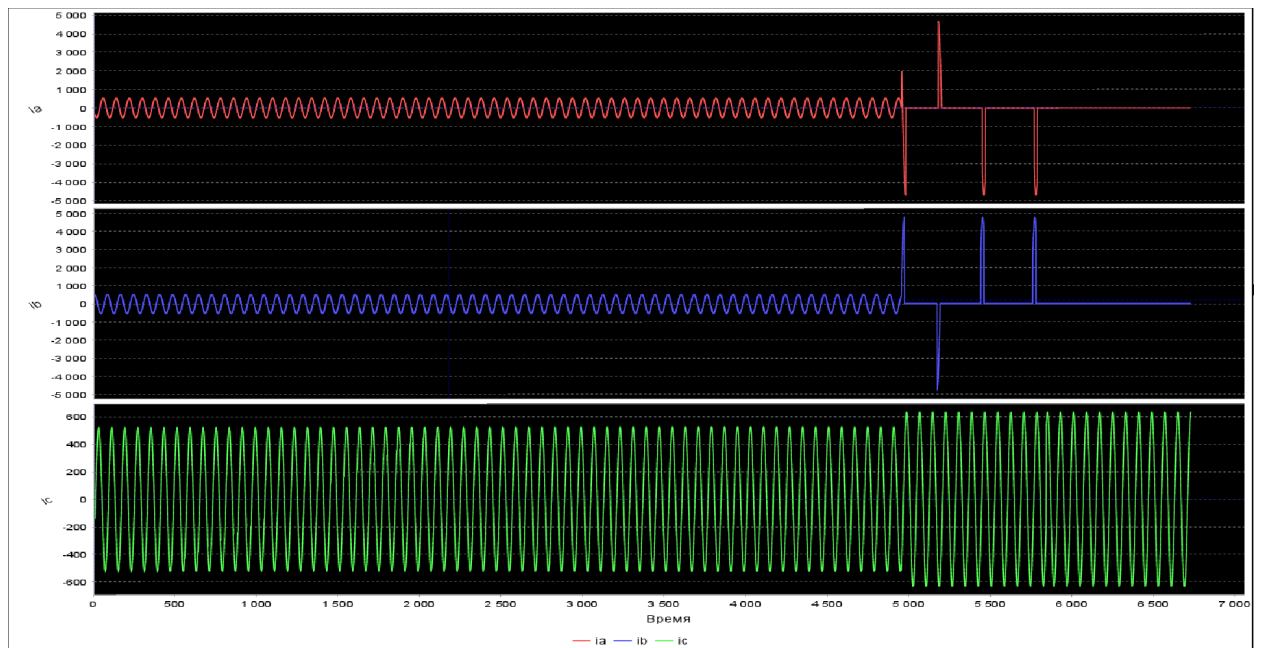


Рис. 17. Мгновенные значения опыта 4

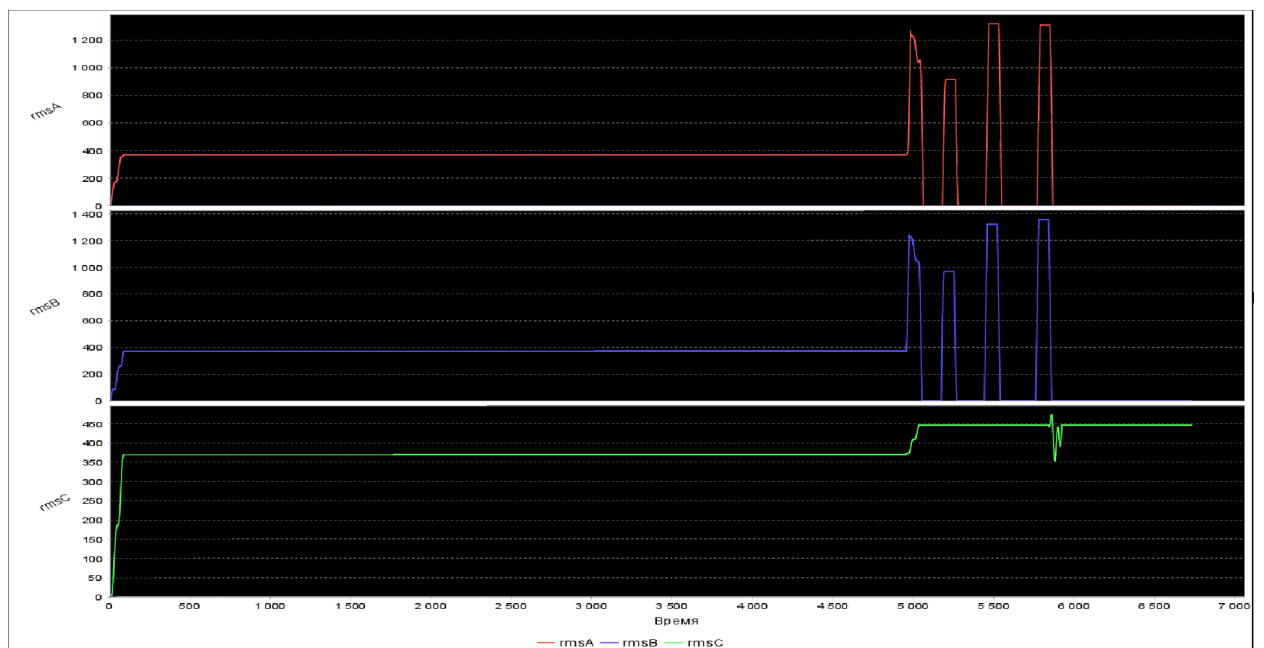


Рис. 18. Действующие значения опыта 4

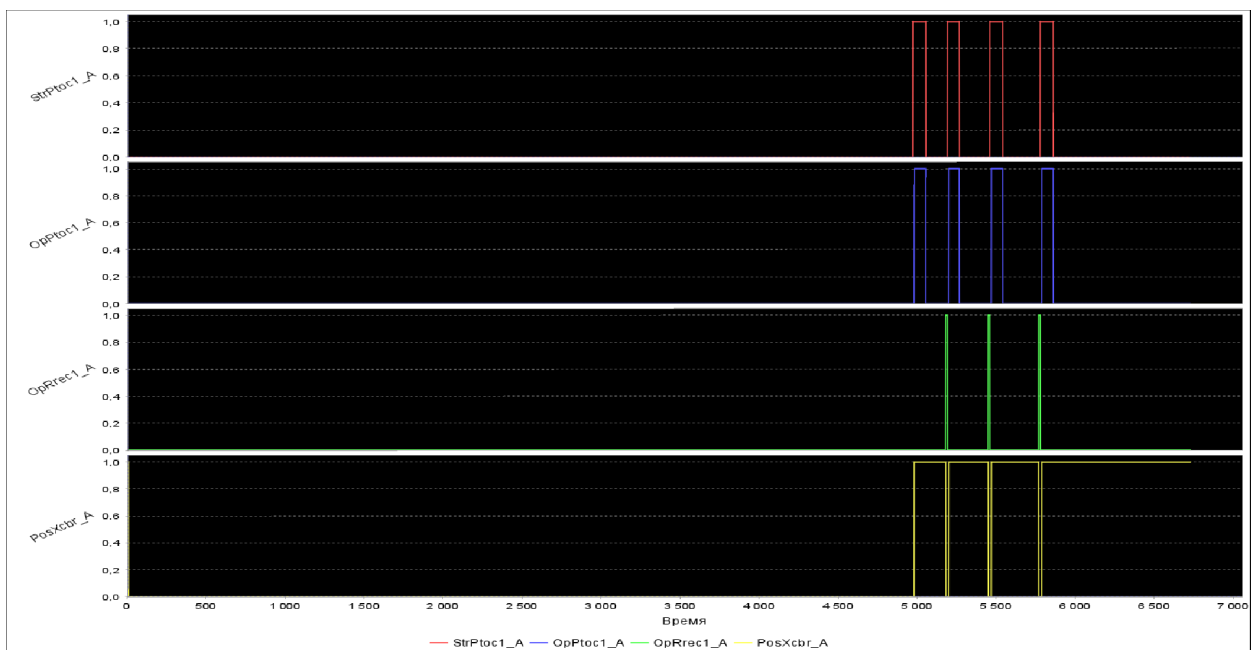


Рис. 19. Дискретные значения фазы А опыта 4

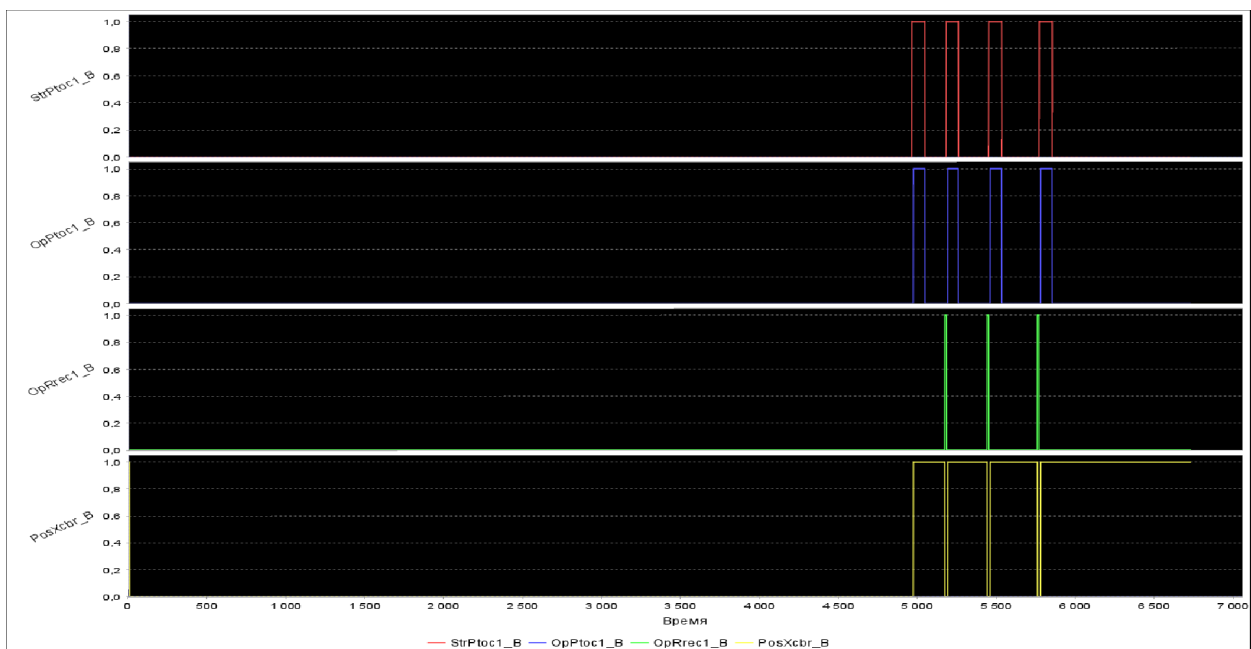


Рис. 20. Дискретные значения фазы В опыта 4

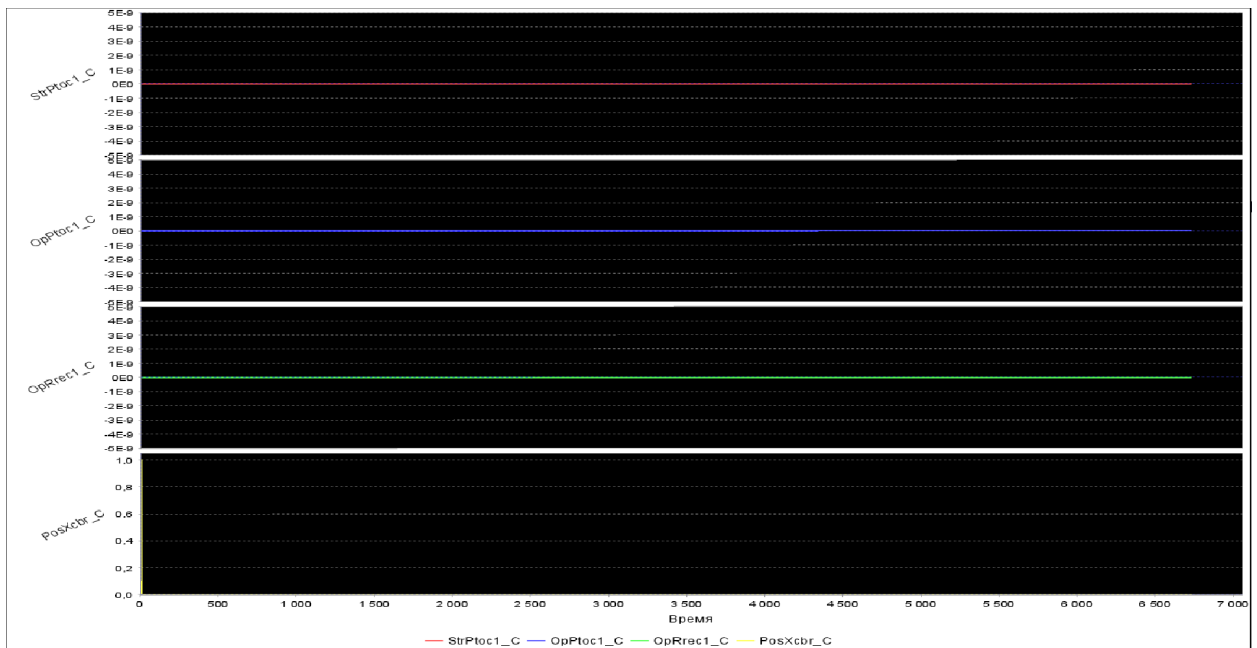


Рис. 21. Дискретные значения фазы С опыта 4

КЗ в ф.А возникло в: 21:11:20.008770900

Защита ф.А сработала в: 21:11:20.026880400

АПВ ф.А сработало в: 21:11:20.163965800

КЗ в ф.А возникло в: 21:11:20.172619600

Защита ф.А сработала в: 21:11:20.173770200

АПВ ф.А сработало в: 21:11:20.341641500

КЗ в ф.А возникло в: 21:11:20.350360600

Защита ф.А сработала в: 21:11:20.355417

АПВ ф.А сработало в: 21:11:20.535893300

КЗ в ф.А возникло в: 21:11:20.540740300

Защита ф.А сработала в: 21:11:20.542792200

## Вывод

В результате курсовой работы была реализована функция АПВ с контролем синхронизма со стандартом МЭК 61850. Для демонстрации работы АПВ необходимо было дополнительно реализовывать алгоритм релейной защиты. Для её реализации была выбрана функция токовой направленной защиты ввиду относительной простоты алгоритма. В качестве входных данных использовались пакеты по SV-протоколу, которые проходили предварительную обработку для получения действующих значений токов.

В результате выполнения программы были получены осциллограммы с аналоговыми и дискретными сигналами, на основании которых был сделан вывод о корректности работы функции АПВ.

# Приложение

## Logical nodes

### LN

```
/**
 * Базовый класс для всех логических узлов
 */

@Getter @Setter
public abstract class LN {
    private String pref; // Пишется в названии перед узлом (название для
    релейщика)
    private String clazz;
    private int inst;

    public abstract void process();
}
```

### LSVS (SV)

```
@Slf4j
@Getter
@Setter
public class LSVS extends LN {
    static {
        try {for (PcapNetworkInterface nic : Pcaps.findAllDevs())
        {log.info("found nic {}", nic);}}
        catch (PcapNativeException e) {throw new RuntimeException(e);}
    }

    public DPC Pos = new DPC();
    public DPC PosA = new DPC();
    public DPC PosB = new DPC();
    public DPC PosC = new DPC();

    public SAV phsAIInst = new SAV();
    public SAV phsBIInst = new SAV();
    public SAV phsCIInst = new SAV();
    public SAV phsAUInst = new SAV();
    public SAV phsBUInst = new SAV();
    public SAV phsCUInst = new SAV();

    @Setter
    private String nicName;
    private PcapHandle handle;

    @Setter
    private int datasetSize;
    private int selector = 0;
    private final List<MyPacketListener> packetListeners = new
    CopyOnWriteArrayList<>();

    private final PacketListener defaultPacketListener = packet -> {
        if(selector == 0) {
            decode(packet);
            packetListeners.forEach(MyPacketListener::listen);
        }
        selector++;
    }
}
```

```

        if (selector == 2) selector = 0;
    };

    @Override
    @SneakyThrows
    public void process() {
        if (handle == null){
            initializeNetworkInterface();

            if (handle != null) {
                String filter = "ether proto 0x88ba && ether dst
01:0C:CD:04:00:01";
                handle.setFilter(filter, BpfProgram.BpfCompileMode.OPTIMIZE);

                Thread captureThread = new Thread(() -> {
                    try {
                        log.info("Starting packet capture");
                        handle.loop(0, defaultPacketListener);
                    } catch (PcapNativeException | InterruptedException |
NotOpenException e) {
                        throw new RuntimeException(e);
                    }
                    log.info("Packet capture finished");
                });
                captureThread.start();
            }
        }
    }

    @SneakyThrows
    private void initializeNetworkInterface() {
        Optional<PcapNetworkInterface> nic = Pcaps.findAllDevs().stream()
            .filter(i -> nicName.equals(i.getDescription()))
            .findFirst();
        if (nic.isPresent()) {
            handle = nic.get().openLive(1500,
PcapNetworkInterface.PromiscuousMode.PROMISCUOUS, 10);
            log.info("Network Handler created: {}", nic);
        } else {
            log.error("Network interface not found");
        }
    }

    public void addListener(MyPacketListener listener) {
        packetListeners.add(listener);
    }

    public void decode(PcapPacket packet){
        try {
            byte[] data = packet.getRawData();
            int strDataByte = data.length - datasetSize;

            phsAIIInst.getInstMag().getF().setValue((double)
(byteArrayToInt(data, strDataByte) / 1000));
            phsBIIInst.getInstMag().getF().setValue((double)
(byteArrayToInt(data, strDataByte + 8) / 1000));
            phsCIIInst.getInstMag().getF().setValue((double)
(byteArrayToInt(data, strDataByte + 16) / 1000));
            phsAUIInst.getInstMag().getF().setValue((double)
(byteArrayToInt(data, strDataByte + 32) / 62));
            phsBUIInst.getInstMag().getF().setValue((double)

```

```

(byteArrayToInt(data, strDataByte + 40) / 62));
    phsCUIInst.getInstMag().getF().setValue((double)
(byteArrayToInt(data, strDataByte + 48) / 62));

    if (PosA.getStVal().getValue() != null &&
PosA.getStVal().getValue().equals(DPC.Values.OFF)) {
        phsAIIInst.getInstMag().getF().setValue(0.0);
    }
    if (PosB.getStVal().getValue() != null &&
PosB.getStVal().getValue().equals(DPC.Values.OFF)) {
        phsBIIInst.getInstMag().getF().setValue(0.0);
    }
    if (PosC.getStVal().getValue() != null &&
PosC.getStVal().getValue().equals(DPC.Values.OFF)) {
        phsCIIInst.getInstMag().getF().setValue(0.0);
    }

    } catch (Exception e) {log.error("Cannot parse sv packet");}

    }

    public static int byteArrayToInt(byte[] b, int offset){
        return b[offset + 3] & 0xFF | (b[offset + 2] & 0xFF) << 8 | (b[offset
+ 1] & 0xFF) << 16 | (b[offset] & 0xFF) << 24;
    }
}

```

## MMXU

```

/**
 * Класс для описания узла измерений и передачи данных в фильтр
 */
public class MMXU extends LN {

    public static int busSize = 80;

    private SAV TotW = new SAV();
    private SAV TotVAr = new SAV();
    private SAV TotVA = new SAV();
    private SAV TotPF = new SAV();
    private SAV Hz = new SAV();
    private DEL PPV = new DEL();

    private WYE PhV = new WYE();
    private WYE W = new WYE();
    private WYE VAr = new WYE();
    private WYE VA = new WYE();
    private WYE PF = new WYE();
    private WYE Z = new WYE();

    /* Входы */
    public SAV IaInst = new SAV();
    public SAV IbInst = new SAV();
    public SAV IcInst = new SAV();
    public SAV UaInst = new SAV();
    public SAV UbInst = new SAV();
    public SAV UcInst = new SAV();

    /* Выходы */

```



```

public WYE A = new WYE(); // Фазные токи (IL1, IL2, IL3)
public WYE PNV = new WYE();

/* Переменные */
private final Filter ia = new Fourier(busSize);
private final Filter ib = new Fourier(busSize);
private final Filter ic = new Fourier(busSize);

private final Filter ua = new Fourier(busSize);
private final Filter ub = new Fourier(busSize);
private final Filter uc = new Fourier(busSize);

@Override
public void process() {
    this.ia.process(this.IaInst, A.getPhsA().getCVal());
    this.ib.process(this.IbInst, A.getPhsB().getCVal());
    this.ic.process(this.IcInst, A.getPhsC().getCVal());

    this.ua.process(this.UaInst, PNV.getPhsA().getCVal());
    this.ub.process(this.UbInst, PNV.getPhsB().getCVal());
    this.uc.process(this.UcInst, PNV.getPhsC().getCVal());
}
}

```

## Fourier

```

/**
 * Класс для реализации фильтра Фурье
 */
public class Fourier extends Filter{
    private ING bSize = new ING();
    private final SAV[] buffer;
    public Attribute<Integer> bCount = new Attribute<>();
    public Attribute<Double> rVal = new Attribute<>();
    public Attribute<Double> imVal = new Attribute<>();
    public Attribute<Double> freq = new Attribute<>();
    public Attribute<Double> dT = new Attribute<>();

    public Fourier(int bufferSize){
        bSize.getSetVal().setValue(bufferSize);
        bCount.setValue(0);
        rVal.setValue(0D);
        imVal.setValue(0D);
        freq.setValue(50D);
        dT.setValue(0.02 / bSize.getSetVal().getValue());
        buffer = new SAV[bSize.getSetVal().getValue()];

        for (int i = 0; i < bSize.getSetVal().getValue(); i++) {
            SAV tempVal = new SAV();
            tempVal.getInstMag().getF().setValue(0D);
            buffer[i] = tempVal;
        }
    }

    @Override
    public void process(SAV measuredValue, Vector result) {

        rVal.setValue(rVal.getValue()
            + (measuredValue.getInstMag().getF().getValue()
            - buffer[bCount.getValue()].getInstMag().getF().getValue())

```

```

        * Math.sin(2 * Math.PI * freq.getValue() * bCount.getValue()
* dT.getValue())
        * 2 / bSize.getSetVal().getValue()
    );
    imVal.setValue(imVal.getValue()
        + (measuredValue.getInstMag().getF().getValue()
        - buffer[bCount.getValue()].getInstMag().getF().getValue())
        * Math.cos(2 * Math.PI * freq.getValue() * bCount.getValue()
* dT.getValue())
        * 2 / bSize.getSetVal().getValue()
    );

    result.getMag().getF().setValue(
        0.7071068 * Math.sqrt(Math.pow(rVal.getValue(), 2) +
Math.pow(imVal.getValue(), 2))
    );

    if ( rVal.getValue() <0){
        result.getAng().getF().setValue(
            Math.atan(imVal.getValue() / rVal.getValue()) * 180 /
Math.PI -180 );
    }
    else {
        result.getAng().getF().setValue(
            Math.atan(imVal.getValue() / rVal.getValue()) * 180 /
Math.PI
        );
    }

    buffer[bCount.getValue()].getInstMag().getF().setValue(measuredValue.getInstM
ag().getF().getValue());
    bCount.setValue(bCount.getValue() + 1);
    if (bCount.getValue() >= bSize.getSetVal().getValue()){
        bCount.setValue(0);
    }
}
}

```

## RDIR

```

/**
 * Класс, описывающий орган направления мощности
 */
@Getter @Setter
public class RDIR extends LN{
    private ACD Dir = new ACD();
    private ASG ChrAng = new ASG();
    private ASG MinFwdAng = new ASG();
    private ASG MinRvAng = new ASG();
    private ASG MaxFwdAng = new ASG();
    private ASG MaxRvAng = new ASG();
    private ASG BlkValA = new ASG();
    private ASG BlkValV = new ASG();
    private ASG MinPPV = new ASG();

    public DEL lineVoltages = new DEL();
    public WYE A = new WYE();
    public WYE PNV = new WYE();
    private DEL ImbPPV = new DEL();

    public RDIR (double ChrAng, double MaxFwdAng, double MinFwdAng, double
minI, double MinPPV ){

```

```

        this.ChrAng.getSetMag().getF().setValue(ChrAng);
        this.MaxFwdAng.getSetMag().getF().setValue(MaxFwdAng);
        this.MinFwdAng.getSetMag().getF().setValue(MinFwdAng);
        this.BlkValA.getSetMag().getF().setValue(minI);
        this.MinPPV.getSetMag().getF().setValue(MinPPV);
    }

    @Override
    public void process() {
        Dir.getPhsA().setValue(
            checkAngle(lineVoltages.getPhsBC(), A.getPhsA())
        );
        Dir.getPhsB().setValue(
            checkAngle(lineVoltages.getPhsCA(), A.getPhsB())
        );
        Dir.getPhsC().setValue(
            checkAngle(lineVoltages.getPhsAB(), A.getPhsC())
        );
        Dir.getGeneral().setValue(
            Dir.getPhsA().getValue() || Dir.getPhsB().getValue() ||
            Dir.getPhsC().getValue()
        );
    }

    private boolean checkAngle(CMV v, CMV i){
        if (i.getCVal().getMag().getF().getValue() <
            BlkValA.getSetMag().getF().getValue())
            return false;
        if (v.getCVal().getMag().getF().getValue() <
            MinPPV.getSetMag().getF().getValue())
            return false;
        double angle = v.getCVal().getAng().getF().getValue() -
            i.getCVal().getAng().getF().getValue();
        return angle >= MinFwdAng.getSetMag().getF().getValue() && angle
            <= MaxFwdAng.getSetMag().getF().getValue();
    }
}

```

## MSQI

```

/**
 * Класс для описания узла получения последовательностей
 */
@Getter @Setter
public class MSQI extends LN {
    private SEQ SeqA = new SEQ();
    private SEQ SeqV = new SEQ();
    private SEQ DQ0Seq = new SEQ();

    private WYE ImbA = new WYE();

    private SAV ImbNgA = new SAV();
    private SAV ImbNgV = new SAV();

    private DEL ImbPPV = new DEL();

    private WYE ImbV = new WYE();

    private SAV ImbZroA = new SAV();
    private SAV ImbZroV = new SAV();
    private SAV MaxImbA = new SAV();
    private SAV MaxImbPPV = new SAV();
}

```

```

private SAV MaxImbV = new SAV();

//      Входные данные: мгновенные значения токов и напряжений

public WYE A = new WYE();
public WYE PNV = new WYE();

@Override
public void process() {
    double MagIa = A.getPhsA().getCVal().getMag().getF().getValue();
    double MagIb = A.getPhsB().getCVal().getMag().getF().getValue();
    double MagIc = A.getPhsC().getCVal().getMag().getF().getValue();

    double AngIa =
toRadians(A.getPhsA().getCVal().getAng().getF().getValue());
    double AngIb =
toRadians(A.getPhsB().getCVal().getAng().getF().getValue());
    double AngIc =
toRadians(A.getPhsC().getCVal().getAng().getF().getValue());

    double MagUa = PNV.getPhsA().getCVal().getMag().getF().getValue();
    double MagUb = PNV.getPhsB().getCVal().getMag().getF().getValue();
    double MagUc = PNV.getPhsC().getCVal().getMag().getF().getValue();

    double AngUa =
toRadians(PNV.getPhsA().getCVal().getAng().getF().getValue());
    double AngUb =
toRadians(PNV.getPhsB().getCVal().getAng().getF().getValue());
    double AngUc =
toRadians(PNV.getPhsC().getCVal().getAng().getF().getValue());

    ImbPPV.getPhsAB().getCVal().getMag().getF().setValue(
        sqrt( pow(MagUa * cos(AngUa) + MagUb * cos(AngUb), 2) +
pow(MagUa * sin(AngUa) + MagUb * sin(AngUb) , 2))
    );
    ImbPPV.getPhsAB().getCVal().getAng().getF().setValue(
        atan((MagUa * sin(AngUa) + MagUb * sin(AngUb)) / (MagUa *
cos(AngUa) + MagUb * cos(AngUb))) * 180 / PI
    );
    ImbPPV.getPhsBC().getCVal().getMag().getF().setValue(
        sqrt( pow(MagUb * cos(AngUb) + MagUc * cos(AngUc), 2) +
pow(MagUb * sin(AngUb) + MagUc * sin(AngUc) , 2))
    );
    ImbPPV.getPhsBC().getCVal().getAng().getF().setValue(
        atan((MagUb * sin(AngUb) + MagUc * sin(AngUc)) / (MagUb *
cos(AngUb) + MagUc * cos(AngUc))) * 180 / PI
    );
    ImbPPV.getPhsCA().getCVal().getMag().getF().setValue(
        sqrt( pow(MagUc * cos(AngUc) + MagUa * cos(AngUa), 2) +
pow(MagUc * sin(AngUc) + MagUa * sin(AngUa) , 2))
    );
    ImbPPV.getPhsCA().getCVal().getAng().getF().setValue(
        atan((MagUc * sin(AngUc) + MagUa * sin(AngUa)) / (MagUc *
cos(AngUc) + MagUa * cos(AngUa))) * 180 / PI
    );

    SeqA.getC3().getCVal().getMag().getF().setValue(1.0 / 3.0 *
(sqrt((pow(MagIa * cos(AngIa) + MagIb * cos(AngIb) + MagIc * cos(AngIc), 2))
+ pow(MagIa * sin(AngIa) + MagIb * sin(AngIb) + MagIc * sin(AngIc), 2))));
    SeqA.getC3().getCVal().getAng().getF().setValue(atan(toRadians(MagIa

```

```

* sin(AngIa) + MagIb * sin(AngIb) + MagIc * sin(AngIc) / (MagIa * cos(AngIa)
+ MagIb * cos(AngIb) + MagIc * cos(AngIc))));

    SeqA.getC1().getCVal().getMag().getF().setValue(1.0 / 3.0 *
(sqrt((pow(MagIa * cos(AngIa) + rotateVector(MagIb * cos(AngIb), MagIb *
sin(AngIb), 120)[0] +
        rotateVector(MagIc * cos(AngIc), MagIc * sin(AngIc), 240)[0],
2)) +
        pow(MagIa * sin(AngIa) + rotateVector(MagIb * cos(AngIb),
MagIb * sin(AngIb), 120)[1] +
        rotateVector(MagIc * cos(AngIc), MagIc * sin(AngIc),
240)[1], 2))));

    SeqA.getC1().getCVal().getAng().getF().setValue(atan(toRadians(
        (MagIa * sin(AngIa) + rotateVector(MagIb * cos(AngIb), MagIb
* sin(AngIb), 120)[1] +
        rotateVector(MagIc * cos(AngIc), MagIc * sin(AngIc),
240)[1]) /
        (MagIa * cos(AngIa) + rotateVector(MagIb *
cos(AngIb), MagIb * sin(AngIb), 120)[0] +
        rotateVector(MagIc * cos(AngIc), MagIc *
sin(AngIc), 240)[0])
        )));

    SeqA.getC2().getCVal().getMag().getF().setValue(1.0 / 3.0 *
(sqrt((pow(MagIa * cos(AngIa) + rotateVector(MagIb * cos(AngIb), MagIb *
sin(AngIb), 240)[0] +
        rotateVector(MagIc * cos(AngIc), MagIc * sin(AngIc), 120)[0],
2)) +
        pow(MagIa * sin(AngIa) + rotateVector(MagIb * cos(AngIb),
MagIb * sin(AngIb), 240)[1] +
        rotateVector(MagIc * cos(AngIc), MagIc * sin(AngIc),
120)[1], 2))));

    SeqA.getC2().getCVal().getAng().getF().setValue(atan(toRadians(
        (MagIa * sin(AngIa) + rotateVector(MagIb * cos(AngIb), MagIb
* sin(AngIb), 240)[1] +
        rotateVector(MagIc * cos(AngIc), MagIc * sin(AngIc),
120)[1]) /
        (MagIa * cos(AngIa) + rotateVector(MagIb *
cos(AngIb), MagIb * sin(AngIb), 240)[0] +
        rotateVector(MagIc * cos(AngIc), MagIc *
sin(AngIc), 120)[0])
        )));

    SeqV.getC3().getCVal().getMag().getF().setValue(1.0 / 3.0 *
(sqrt((pow(MagUa * cos(AngUa) + MagUb * cos(AngUb) + MagUc * cos(AngUc), 2))
+ pow(MagUa * sin(AngUa) + MagUb * sin(AngUb) + MagUc * sin(AngUc), 2))));
    SeqV.getC3().getCVal().getAng().getF().setValue(atan(toRadians(MagUa
* sin(AngUa) + MagUb * sin(AngUb) + MagUc * sin(AngUc) / (MagUa * cos(AngUa)
+ MagUb * cos(AngUb) + MagUc * cos(AngUc))));

    SeqV.getC1().getCVal().getMag().getF().setValue(1.0 / 3.0 *
(sqrt((pow(MagUa * cos(AngUa) + rotateVector(MagUb * cos(AngUb), MagUb *
sin(AngUb), 120)[0] +
        rotateVector(MagUc * cos(AngUc), MagUc * sin(AngUc), 240)[0],
2)) +
        pow(MagUa * sin(AngUa) + rotateVector(MagUb * cos(AngUb),
MagUb * sin(AngUb), 120)[1] +
        rotateVector(MagUc * cos(AngUc), MagUc * sin(AngUc),
240)[1], 2))));

```

```

SeqV.getC1().getCVal().getAng().getF().setValue(atan(toRadians(
    (MagUa * sin(AngUa) + rotateVector(MagUb * cos(AngUb), MagUb
* sin(AngUb), 120)[1] +
    rotateVector(MagUc * cos(AngUc), MagUc * sin(AngUc),
240)[1])) /
    (MagUa * cos(AngUa) + rotateVector(MagUb *
cos(AngUb), MagUb * sin(AngUb), 120)[0] +
    rotateVector(MagUc * cos(AngUc), MagUc *
sin(AngUc), 240)[0])));

SeqV.getC2().getCVal().getMag().getF().setValue(1.0 / 3.0 *
(sqrt((pow(MagUa * cos(AngUa) + rotateVector(MagUb * cos(AngUb), MagUb *
sin(AngUb), 240)[0] +
    rotateVector(MagUc * cos(AngUc), MagUc * sin(AngUc), 120)[0],
2)) +
    pow(MagUa * sin(AngUa) + rotateVector(MagUb * cos(AngUb),
MagUb * sin(AngUb), 240)[1] +
    rotateVector(MagUc * cos(AngUc), MagUc * sin(AngUc),
120)[1], 2))));

SeqV.getC2().getCVal().getAng().getF().setValue(atan(toRadians(
    (MagUa * sin(AngUa) + rotateVector(MagUb * cos(AngUb), MagUb
* sin(AngUb), 240)[1] +
    rotateVector(MagUc * cos(AngUc), MagUc * sin(AngUc),
120)[1])) /
    (MagUa * cos(AngUa) + rotateVector(MagUb *
cos(AngUb), MagUb * sin(AngUb), 240)[0] +
    rotateVector(MagUc * cos(AngUc), MagUc *
sin(AngUc), 120)[0])));
    });
}

public static double[] rotateVector(double x, double y, double angle) {
    double[] rotatedVector = new double[3];

    double sin = Math.sin(toRadians(angle));
    double cos = Math.cos(toRadians(angle));

    double oldx = x;
    double oldy = y;

    x = oldx * cos - oldy * sin;
    y = oldx * sin + oldy * cos;

    double angleAfterRotation = Math.atan2(y, x) * 180 / Math.PI;

    rotatedVector[0] = x;
    rotatedVector[1] = y;
    rotatedVector[2] = angleAfterRotation;

    return rotatedVector;
}
}

```

## PTOC

```
/**
 * Класс, описывающий токовую защиту
 */
public class PTOC extends LN {

    public static double dt = 1; // миллисек

    private INC OpCntRs = new INC();
    private CURVE TmACrv = new CURVE();
    private CSG TmAChr33 = new CSG();
    private CSD TmASt = new CSD();

    public ASG TmMult = new ASG();
    private ING MinOpTmms = new ING();
    private ING MaxOpTmms = new ING();
    private ENG TypRsCrv = new ENG();
    private ING RsDiTmms = new ING();
    private ENG DirMod = new ENG();

    /* Входы */
    public ACD direction = new ACD();
    public SEQ currentSeq = new SEQ();
    public WYE A = new WYE();

    /* Выходы */
    public ACD Str = new ACD();
    public ACT Op = new ACT();

    /* Уставки */
    public ASG StrVal = new ASG();
    public ING OpDiTmms = new ING();
    public boolean isDir = false;

    /* Переменные */
    private int cntTimeA = 0;
    private int cntTimeB = 0;
    private int cntTimeC = 0;

    @Override
    public void process() {
        boolean strA = A.getPhsA().getCVal().getMag().getF().getValue() >
StrVal.getSetMag().getF().getValue();
        boolean strB = A.getPhsB().getCVal().getMag().getF().getValue() >
StrVal.getSetMag().getF().getValue();
        boolean strC = A.getPhsC().getCVal().getMag().getF().getValue() >
StrVal.getSetMag().getF().getValue();

        boolean str = strA || strB || strC;

        boolean dir = direction.getGeneral().getValue();

        if (isDir) {
            Str.getPhsA().setValue(strA && dir);
            Str.getPhsB().setValue(strB && dir);
            Str.getPhsC().setValue(strC && dir);
            Str.getGeneral().setValue(str && dir);

            cntTimeA = strA && dir ? cntTimeA + 1 : 0;
            cntTimeB = strB && dir ? cntTimeB + 1 : 0;
            cntTimeC = strC && dir ? cntTimeC + 1 : 0;
        } else {
```

```

        Str.getPhsA().setValue(strA);
        Str.getPhsB().setValue(strB);
        Str.getPhsC().setValue(strC);
        Str.getGeneral().setValue(str);

        cntTimeA = strA ? cntTimeA + 1 : 0;
        cntTimeB = strB ? cntTimeB + 1 : 0;
        cntTimeC = strC ? cntTimeC + 1 : 0;
    }

    Op.getPhsA().setValue(cntTimeA * dt >
OpDITmms.getSetVal().getValue());
    Op.getPhsB().setValue(cntTimeB * dt >
OpDITmms.getSetVal().getValue());
    Op.getPhsC().setValue(cntTimeC * dt >
OpDITmms.getSetVal().getValue());

    Op.getGeneral().setValue(Op.getPhsA().getValue() ||
Op.getPhsB().getValue() || Op.getPhsC().getValue());
}
}

```

## RSYN

```

/**
 * Класс, описывающий контроль синхронизма
 */
public class RSYN extends LN {

    private SPC RHz = new SPC();
    private SPC LHz = new SPC();
    private SPC RV = new SPC();
    private SPC LV = new SPC();
    private SPS VInd = new SPS();
    private SPS AngInd = new SPS();
    private SPS HzInd = new SPS();
    private CMV DifVClc = new CMV();
    private CMV DifHzClc = new CMV();
    private CMV DifAngClc = new CMV();

    public WYE LinePhV = new WYE();
    public WYE BusPhV = new WYE();
    public CMV LineHz = new CMV();
    public CMV BusHz = new CMV();
    public SPS SynPrg = new SPS();
    public SPS Rel = new SPS();
    public ASG DifV = new ASG();
    public ASG DifHz = new ASG();
    public ASG DifAng = new ASG();
    public ING LivDeaMod = new ING();
    public ASG DeaLinVal = new ASG();
    public ASG LivLinVal = new ASG();
    public ASG DeaBusVal = new ASG();
    public ASG LivBusVal = new ASG();
    public ING PlsTmms = new ING();
    public ING BkrTmms = new ING();

    @Override
    public void process() {
        Rel.getStVal().setValue(false);
    }
}

```



```

        if (SynPrg.getStVal().getValue() != null &&
SynPrg.getStVal().getValue()) {
            if (LivDeaMod.getSetVal().getValue() != null &&
LivDeaMod.getSetVal().getValue() == 0) {
                Rel.getStVal().setValue(true);
                return;
            }
            if (isValidData()) {
                boolean ONL =
LinePhV.getPhsA().getCVal().getMag().getF().getValue() <
DeaLinVal.getSetMag().getF().getValue();
                boolean>NNL =
LinePhV.getPhsA().getCVal().getMag().getF().getValue() >
LivLinVal.getSetMag().getF().getValue();
                boolean>ONSH =
BusPhV.getPhsA().getCVal().getMag().getF().getValue() <
DeaBusVal.getSetMag().getF().getValue();
                boolean>NNSH =
BusPhV.getPhsA().getCVal().getMag().getF().getValue() >
LivBusVal.getSetMag().getF().getValue();
                boolean>voltageConditionsMet = false;
                switch (LivDeaMod.getSetVal().getValue()) {
                    case 1:
                        voltageConditionsMet = ONL &&>NNSH;
                        break;
                    case 2:
                        voltageConditionsMet =>NNL &&>ONSH;
                        break;
                    case 3:
                        voltageConditionsMet = (ONL &&>NNSH) || (>>NNL &&
ONSH);
                        break;
                    default:
                        voltageConditionsMet = false;
                        break;
                }

                if (voltageConditionsMet) {
                    DifVClc = calcDifMag(LinePhV, BusPhV);
                    DifAngClc = calcDifAng(LinePhV, BusPhV);
                    DifHzClc = calcDifHz(LineHz, BusHz);

                    VInd.getStVal().setValue(isLargerThenSetting(DifVClc,
DifV));
                    AngInd.getStVal().setValue(isLargerThenSetting(DifAngClc,
DifAng));
                    HzInd.getStVal().setValue(isLargerThenSetting(DifHzClc,
DifHz));

                    if (!VInd.getStVal().getValue() &&
!AngInd.getStVal().getValue() &&>!HzInd.getStVal().getValue()) {
                        Rel.getStVal().setValue(true);
                    }
                }
            }
        }

        private boolean>isValidData() {
            return>LinePhV.getPhsA().getCVal().getMag().getF().getValue() != null
&&
                BusPhV.getPhsA().getCVal().getMag().getF().getValue() != null
&&

```

```

        LineHz.getCVal().getMag().getF().getValue() != null &&
        BusHz.getCVal().getMag().getF().getValue() != null &&
        DeaLinVal.getSetMag().getF().getValue() != null &&
        LivLinVal.getSetMag().getF().getValue() != null &&
        DeaBusVal.getSetMag().getF().getValue() != null &&
        LivBusVal.getSetMag().getF().getValue() != null &&
        LivDeaMod.getSetVal().getValue() != null;
    }

    private CMV calcDifMag(WYE lineV, WYE busV) {
        CMV difMag = new CMV();
        double dif =
Math.abs(lineV.getPhsA().getCVal().getMag().getF().getValue() -
busV.getPhsA().getCVal().getMag().getF().getValue());
        difMag.getCVal().getMag().getF().setValue(dif);
        return difMag;
    }

    private CMV calcDifAng(WYE lineV, WYE busV) {
        CMV difAng = new CMV();
        double dif =
Math.abs(lineV.getPhsA().getCVal().getAng().getF().getValue() -
busV.getPhsA().getCVal().getAng().getF().getValue());
        difAng.getCVal().getMag().getF().setValue(dif);
        return difAng;
    }

    private CMV calcDifHz(CMV lineF, CMV busF) {
        CMV difHz = new CMV();
        double dif = Math.abs(lineF.getCVal().getMag().getF().getValue() -
busF.getCVal().getMag().getF().getValue());
        difHz.getCVal().getMag().getF().setValue(dif);
        return difHz;
    }

    private boolean isLargerThenSetting(CMV value, ASG settingValue) {
        return value.getCVal().getMag().getF().getValue() >
settingValue.getSetMag().getF().getValue();
    }
}

```

## RREC

```

/**
 * Класс, описывающий АПВ
 */
public class RREC extends LN {

    public static int dt = 1;

    private INC OpCntRs = new INC();
    private SPC ChkRec = new SPC();
    private SPS Auto = new SPS();
    private INS AutoRecSt = new INS();

    public ACT OpOpn = new ACT();
    public DPC PosA = new DPC();
    public DPC PosB = new DPC();
    public DPC PosC = new DPC();
    public SPS Rel = new SPS();
    public SPS Blk = new SPS();
}

```

```

public SPC BlkRec = new SPC();

public ACT Op = new ACT();
public SPS SynPrg = new SPS();

public ING Rec1Tmms = new ING();
public ING Rec2Tmms = new ING();
public ING Rec3Tmms = new ING();
public ING PlsTmms = new ING();
public ING RclTmms = new ING();

private int recCntTimeA = 0;
private int currentCycleNumA = 0;
private boolean prevValueOfOpOpnA = false;
private int impulseStartTimeA = 0;
private boolean isImpulseActiveA = false;
private int readyTimeA = 0;
private boolean inReadyStateA = false;

private int recCntTimeB = 0;
private int currentCycleNumB = 0;
private boolean prevValueOfOpOpnB = false;
private int impulseStartTimeB = 0;
private boolean isImpulseActiveB = false;
private int readyTimeB = 0;
private boolean inReadyStateB = false;

private int recCntTimeC = 0;
private int currentCycleNumC = 0;
private boolean prevValueOfOpOpnC = false;
private int impulseStartTimeC = 0;
private boolean isImpulseActiveC = false;
private int readyTimeC = 0;
private boolean inReadyStateC = false;

@Override
public void process() {
    if (SynPrg.getStVal().getValue() == null)
SynPrg.getStVal().setValue(false);
    if (Rel.getStVal().getValue() == null)
Rel.getStVal().setValue(false);
    if (Blk.getStVal().getValue() == null)
Blk.getStVal().setValue(false);
    if (BlkRec.getStVal().getValue() == null)
BlkRec.getStVal().setValue(false);
    if (OpOpn.getPhsA().getValue() == null)
OpOpn.getPhsA().setValue(false);
    if (OpOpn.getPhsB().getValue() == null)
OpOpn.getPhsB().setValue(false);
    if (OpOpn.getPhsC().getValue() == null)
OpOpn.getPhsC().setValue(false);
    if (PosA.getStVal().getValue() == null)
PosA.getStVal().setValue(DPC.Values.ON);
    if (PosB.getStVal().getValue() == null)
PosB.getStVal().setValue(DPC.Values.ON);
    if (PosC.getStVal().getValue() == null)
PosC.getStVal().setValue(DPC.Values.ON);

    boolean relAllowed = Rel.getStVal().getValue();
    boolean blk = Blk.getStVal().getValue() ||
BlkRec.getStVal().getValue();

    if (inReadyStateA) {

```

```

        readyTimeA += dt;
        if (readyTimeA >= RclTmms.getSetVal().getValue()) {
            inReadyStateA = false;
            readyTimeA = 0;
            currentCycleNumA = 0;
            isImpulseActiveA = false;
            recCntTimeA = 0;
        }
    } else if (currentCycleNumA > 3) {
        inReadyStateA = true;
        SynPrg.getStVal().setValue(false);
    }

    if (inReadyStateB) {
        readyTimeB += dt;
        if (readyTimeB >= RclTmms.getSetVal().getValue()) {
            inReadyStateB = false;
            readyTimeB = 0;
            currentCycleNumB = 0;
            isImpulseActiveB = false;
            recCntTimeB = 0;
        }
    } else if (currentCycleNumB > 3) {
        inReadyStateB = true;
        SynPrg.getStVal().setValue(false);
    }

    if (inReadyStateC) {
        readyTimeC += dt;
        if (readyTimeC >= RclTmms.getSetVal().getValue()) {
            inReadyStateC = false;
            readyTimeC = 0;
            currentCycleNumC = 0;
            isImpulseActiveC = false;
            recCntTimeC = 0;
        }
    } else if (currentCycleNumC > 3) {
        inReadyStateC = true;
        SynPrg.getStVal().setValue(false);
    }

    if (!inReadyStateA && currentCycleNumA <= 3) {
        boolean opOpnA = OpOpn.getPhsA().getValue() &&
!prevValueOfOpOpnA;
        boolean breakerOffA =
PosA.getStVal().getValue().equals(DPC.Values.OFF);
        prevValueOfOpOpnA = OpOpn.getPhsA().getValue();
        if (opOpnA && breakerOffA && !blk) {
            currentCycleNumA++;
            SynPrg.getStVal().setValue(true);
            isImpulseActiveA = false;
            recCntTimeA = 0;
        }
        boolean strA = false;
        if (currentCycleNumA >= 1 && currentCycleNumA <= 3) {
            recCntTimeA += dt;
            int requiredTime = switch (currentCycleNumA) {
                case 1 -> Rec1Tmms.getSetVal().getValue();
                case 2 -> Rec2Tmms.getSetVal().getValue();
                case 3 -> Rec3Tmms.getSetVal().getValue();
                default -> 0;
            };
        }
    }
};

```

```

        if (breakerOffA && relAllowed && recCntTimeA >= requiredTime
&& !isImpulseActiveA) {
            isImpulseActiveA = true;
            impulseStartTimeA = recCntTimeA;
        }

        if (isImpulseActiveA && (recCntTimeA - impulseStartTimeA) <=
PlsTmms.getSetVal().getValue()) {
            strA = true;
        } else if (isImpulseActiveA && (recCntTimeA -
impulseStartTimeA) > PlsTmms.getSetVal().getValue()) {
            isImpulseActiveA = false;
            SynPrg.getStVal().setValue(false);
            recCntTimeA = 0;
        }
    }
    Op.getPhsA().setValue(strA);
} else {
    Op.getPhsA().setValue(false);
}

if (!inReadyStateB && currentCycleNumB <= 3) {
    boolean opOpnB = OpOpn.getPhsB().getValue() &&
!prevValueOfOpOpnB;
    boolean breakerOffB =
PosB.getStVal().getValue().equals(DPC.Values.OFF);
    prevValueOfOpOpnB = OpOpn.getPhsB().getValue();
    if (opOpnB && breakerOffB && !blk) {
        currentCycleNumB++;
        SynPrg.getStVal().setValue(true);
        isImpulseActiveB = false;
        recCntTimeB = 0;
    }

    boolean strB = false;
    if (currentCycleNumB >= 1 && currentCycleNumB <= 3) {
        recCntTimeB += dt;
        int requiredTime = switch (currentCycleNumB) {
            case 1 -> Rec1Tmms.getSetVal().getValue();
            case 2 -> Rec2Tmms.getSetVal().getValue();
            case 3 -> Rec3Tmms.getSetVal().getValue();
            default -> 0;
        };
        if (breakerOffB && relAllowed && recCntTimeB >= requiredTime
&& !isImpulseActiveB) {
            isImpulseActiveB = true;
            impulseStartTimeB = recCntTimeB;
        }
        if (isImpulseActiveB && (recCntTimeB - impulseStartTimeB) <=
PlsTmms.getSetVal().getValue()) {
            strB = true;
        } else if (isImpulseActiveB && (recCntTimeB -
impulseStartTimeB) > PlsTmms.getSetVal().getValue()) {
            isImpulseActiveB = false;
            SynPrg.getStVal().setValue(false);
            recCntTimeB = 0;
        }
    }
    Op.getPhsB().setValue(strB);
} else {
    Op.getPhsB().setValue(false);
}

```

```

        if (!inReadyStateC && currentCycleNumC <= 3) {
            boolean opOpnC = OpOpn.getPhsC().getValue() &&
!prevValueOfOpOpnC;
            boolean breakerOffC =
PosC.getStVal().getValue().equals(DPC.Values.OFF);
            prevValueOfOpOpnC = OpOpn.getPhsC().getValue();
            if (opOpnC && breakerOffC && !blk) {
                currentCycleNumC++;
                SynPrg.getStVal().setValue(true);
                isImpulseActiveC = false;
                recCntTimeC = 0;
            }
            boolean strC = false;
            if (currentCycleNumC >= 1 && currentCycleNumC <= 3) {
                recCntTimeC += dt;
                int requiredTime = switch (currentCycleNumC) {
                    case 1 -> Rec1Tmms.getSetVal().getValue();
                    case 2 -> Rec2Tmms.getSetVal().getValue();
                    case 3 -> Rec3Tmms.getSetVal().getValue();
                    default -> 0;
                };
                if (breakerOffC && relAllowed && recCntTimeC >= requiredTime
&& !isImpulseActiveC) {
                    isImpulseActiveC = true;
                    impulseStartTimeC = recCntTimeC;
                }
                if (isImpulseActiveC && (recCntTimeC - impulseStartTimeC) <=
PlsTmms.getSetVal().getValue()) {
                    strC = true;
                } else if (isImpulseActiveC && (recCntTimeC -
impulseStartTimeC) > PlsTmms.getSetVal().getValue()) {
                    isImpulseActiveC = false;
                    SynPrg.getStVal().setValue(false);
                    recCntTimeC = 0;
                }
            }
            Op.getPhsC().setValue(strC);
        } else {
            Op.getPhsC().setValue(false);
        }

        Op.getGeneral().setValue(Op.getPhsA().getValue() ||
Op.getPhsB().getValue() || Op.getPhsC().getValue());

        if (inReadyStateA || inReadyStateB || inReadyStateC) {
            AutoRecSt.getStVal().setValue(2);
        } else if (currentCycleNumA > 0 || currentCycleNumB > 0 ||
currentCycleNumC > 0) {
            AutoRecSt.getStVal().setValue(1);
        } else {
            AutoRecSt.getStVal().setValue(0);
        }
    }
}

```

## CSWI

```
/**
 * Класс для управления всеми состояниями переключений
 */
public class CSWI extends LN {
    private SPS LocKey = new SPS();
    private SPS Loc = new SPS();
    public ACT OpOpn = new ACT();
    public ACT OpCls = new ACT();
    private INC OpCntRs = new INC();
    private SPC LocSta = new SPC();
    public DPC Pos = new DPC();
    public DPC PosA = new DPC();
    public DPC PosB = new DPC();
    public DPC PosC = new DPC();

    /* Входы */
    private SPS SelOpnA = new SPS();
    private SPS SelOpnB = new SPS();
    private SPS SelOpnC = new SPS();

    private SPS SelClsA = new SPS();
    private SPS SelClsB = new SPS();
    private SPS SelClsC = new SPS();

    /* Блокировки (общие для всех фаз) */
    private SPS BlkOpn = new SPS(); // Блокировка отключения
    private SPS BlkCls = new SPS(); // Блокировка включения

    public List<ACT> protSignalsList = new ArrayList<>(); // Сигналы защиты
    public List<ACT> automaticSignalsList = new ArrayList<>(); // Сигналы
автоматики

    @Override
    public void process() {

        // Защита от null в начале
        if (SelOpnA.getStVal().getValue() == null)
SelOpnA.getStVal().setValue(false);
        if (SelOpnB.getStVal().getValue() == null)
SelOpnB.getStVal().setValue(false);
        if (SelOpnC.getStVal().getValue() == null)
SelOpnC.getStVal().setValue(false);
        if (SelClsA.getStVal().getValue() == null)
SelClsA.getStVal().setValue(false);
        if (SelClsB.getStVal().getValue() == null)
SelClsB.getStVal().setValue(false);
        if (SelClsC.getStVal().getValue() == null)
SelClsC.getStVal().setValue(false);
        if (BlkOpn.getStVal().getValue() == null)
BlkOpn.getStVal().setValue(false);
        if (BlkCls.getStVal().getValue() == null)
BlkCls.getStVal().setValue(false);

        // Сбрасываем все селекторы
        SelOpnA.getStVal().setValue(false);
        SelOpnB.getStVal().setValue(false);
        SelOpnC.getStVal().setValue(false);
        SelClsA.getStVal().setValue(false);
        SelClsB.getStVal().setValue(false);
        SelClsC.getStVal().setValue(false);
    }
}
```

```

// Обрабатываем сигналы защиты (отключение)
for (ACT act : protSignalsList) {
    if (act.getPhsA().getValue() != null && act.getPhsA().getValue())
    {
        SelOpnA.getStVal().setValue(true);
    }
    if (act.getPhsB().getValue() != null && act.getPhsB().getValue())
    {
        SelOpnB.getStVal().setValue(true);
    }
    if (act.getPhsC().getValue() != null && act.getPhsC().getValue())
    {
        SelOpnC.getStVal().setValue(true);
    }
}

// Обрабатываем сигналы автоматики (включение)
for (ACT act : automaticSignalsList) {
    if (act.getPhsA().getValue() != null && act.getPhsA().getValue())
    {
        SelClsA.getStVal().setValue(true);
    }
    if (act.getPhsB().getValue() != null && act.getPhsB().getValue())
    {
        SelClsB.getStVal().setValue(true);
    }
    if (act.getPhsC().getValue() != null && act.getPhsC().getValue())
    {
        SelClsC.getStVal().setValue(true);
    }
}

// Управление положением фаз с учетом блокировок
// Фаза А
if (SelOpnA.getStVal().getValue() && !BlkOpn.getStVal().getValue()) {
    PosA.getStVal().setValue(DPC.Values.OFF);
} else if (SelClsA.getStVal().getValue() &&
!BlkCls.getStVal().getValue()) {
    PosA.getStVal().setValue(DPC.Values.ON);
}

// Фаза В
if (SelOpnB.getStVal().getValue() && !BlkOpn.getStVal().getValue()) {
    PosB.getStVal().setValue(DPC.Values.OFF);
} else if (SelClsB.getStVal().getValue() &&
!BlkCls.getStVal().getValue()) {
    PosB.getStVal().setValue(DPC.Values.ON);
}

// Фаза С
if (SelOpnC.getStVal().getValue() && !BlkOpn.getStVal().getValue()) {
    PosC.getStVal().setValue(DPC.Values.OFF);
} else if (SelClsC.getStVal().getValue() &&
!BlkCls.getStVal().getValue()) {
    PosC.getStVal().setValue(DPC.Values.ON);
}

// Общее положение выключателя
updateGeneralPosition();
}

private void updateGeneralPosition() {
    // Защита от null

```



```

        if (PosA.getStVal().getValue() == null)
PosA.getStVal().setValue(DPC.Values.ON);
        if (PosB.getStVal().getValue() == null)
PosB.getStVal().setValue(DPC.Values.ON);
        if (PosC.getStVal().getValue() == null)
PosC.getStVal().setValue(DPC.Values.ON);

        // Общее положение = ON только если все фазы включены
        boolean allPhasesOn =
PosA.getStVal().getValue().equals(DPC.Values.ON) &&
        PosB.getStVal().getValue().equals(DPC.Values.ON) &&
        PosC.getStVal().getValue().equals(DPC.Values.ON);

        // Общее положение = OFF если хотя бы одна фаза отключена
        boolean anyPhaseOff =
PosA.getStVal().getValue().equals(DPC.Values.OFF) ||
        PosB.getStVal().getValue().equals(DPC.Values.OFF) ||
        PosC.getStVal().getValue().equals(DPC.Values.OFF);

        if (anyPhaseOff) {
            Pos.getStVal().setValue(DPC.Values.OFF);
        } else if (allPhasesOn) {
            Pos.getStVal().setValue(DPC.Values.ON);
        }
    }

    // Методы для установки блокировок
    public void setBlkOpn(boolean value) {
        BlkOpn.getStVal().setValue(value);
    }

    public void setBlkCls(boolean value) {
        BlkCls.getStVal().setValue(value);
    }
}

```

## XCBR

```

/**
 * Класс для реализации состояний силового выключателя
 */
public class XCBR extends LN {
    private DPL EEName = new DPL();
    private ENS EEHealth = new ENS();
    private SPS LocKey = new SPS();
    private SPS Loc = new SPS();
    private INS OpCnt = new INS();
    private ENS CBOpCap = new ENS();
    private ENS POWCap = new ENS();
    private INS MaxOpCap = new INS();
    private SPS Dsc = new SPS();
    private BCR SumSwARs = new BCR();
    private SPC LocSta = new SPC();

    // Пофазные положения
    public DPC Pos = new DPC(); // Общее положение
    public DPC PosA = new DPC(); // Положение фазы А
    public DPC PosB = new DPC(); // Положение фазы В
    public DPC PosC = new DPC(); // Положение фазы С

    // Блокировки (общие для всех фаз)

```

```

public SPC BlkOpn = new SPC(); // Блокировка отключения
public SPS BlkCls = new SPS(); // Блокировка включения

private SPC ChaMotEna = new SPC();
private ING CBTmms = new ING();

// Пофазные статусы
public Attribute<Boolean> isOpenA = new Attribute<>();
public Attribute<Boolean> isOpenB = new Attribute<>();
public Attribute<Boolean> isOpenC = new Attribute<>();
public Attribute<Boolean> isOpen = new Attribute<>(); // Общий статус

@Override
public void process() {
    BlkCls.getStVal().setValue(false);

    // Обновляем статусы фаз
    updatePhaseStatus('A', PosA, isOpenA);
    updatePhaseStatus('B', PosB, isOpenB);
    updatePhaseStatus('C', PosC, isOpenC);

//      System.out.println(PosA.getStVal().getValue());

    // Обновляем общий статус
//      updateGeneralStatus();
}

private void updatePhaseStatus(char phase, DPC pos, Attribute<Boolean>
isOpenAttr) {
    if (pos.getStVal().getValue() != null &&
pos.getStVal().getValue().equals(DPC.Values.ON)) {
        isOpenAttr.setValue(false); // Выключатель включен
    } else {
        isOpenAttr.setValue(true); // Выключатель отключен
    }
}

private void updateGeneralStatus() {
    // Общий статус = отключен, если хотя бы одна фаза отключена
    boolean anyPhaseOpen = isOpenA.getValue() || isOpenB.getValue() ||
isOpenC.getValue();

    // Общий статус = включен, если все фазы включены
    boolean allPhasesClosed = !isOpenA.getValue() && !isOpenB.getValue()
&& !isOpenC.getValue();

    if (anyPhaseOpen) {
        isOpen.setValue(true);
        Pos.getStVal().setValue(DPC.Values.OFF);
    } else if (allPhasesClosed) {
        isOpen.setValue(false);
        Pos.getStVal().setValue(DPC.Values.ON);
    }
}

// Методы для управления положением фаз
public void setPhaseA(boolean closed) {
    PosA.getStVal().setValue(closed ? DPC.Values.ON : DPC.Values.OFF);
    isOpenA.setValue(!closed);
}

public void setPhaseB(boolean closed) {
    PosB.getStVal().setValue(closed ? DPC.Values.ON : DPC.Values.OFF);
}

```

```

        isOpenB.setValue(!closed);
    }

    public void setPhaseC(boolean closed) {
        PosC.getStVal().setValue(closed ? DPC.Values.ON : DPC.Values.OFF);
        isOpenC.setValue(!closed);
    }

    // Методы для проверки состояний
    public boolean isPhaseAClosed() {
        return !isOpenA.getValue();
    }

    public boolean isPhaseBClosed() {
        return !isOpenB.getValue();
    }

    public boolean isPhaseCClosed() {
        return !isOpenC.getValue();
    }

    public boolean isAllPhasesClosed() {
        return isPhaseAClosed() && isPhaseBClosed() && isPhaseCClosed();
    }

    public boolean isAnyPhaseOpen() {
        return isOpenA.getValue() || isOpenB.getValue() ||
        isOpenC.getValue();
    }
}

```

## Common

### DATA

```

/**
 * Родительский класс для всех остальных
 */

@Getter @Setter
public class Data {
    private String name;
    private String ref;
}

```

### Attribute

```

/**
 * Класс содержит значение переменной любого типа
 */

@Getter @Setter
public class Attribute<T> extends Data {
    private T value;

    public Attribute() {

```

```

    }

    public Attribute(T value) {
    }
}

```

## Originator

```

/**
 * Должен содержать сведения об инициаторе последнего изменения атрибута
 * данных
 */

@Getter @Setter
public class Originator extends Data{
    private Attribute<OrCat> orCat = new Attribute<>(); // Категория
инициатора
    private Attribute<String> orIdent = new Attribute<>(); // Адрес
инициатора

    public enum OrCat{
        NOT_SUPPORTED, BAY_CONTROL, STATION_CONTROL,
        REMOTE_CONTROL, AUTOMATIC_BAY, AUTOMATIC_STATION,
        AUTOMATIC_REMOTE, MAINTENANCE, PROCESS
    }
}

```

## TimeStamp

```

@Getter @Setter
public class Timestamp extends Data {
    private long value;
}

```

## Unit

```

@Getter @Setter
public class Unit extends Data{
    private Attribute<Units> SIUnit = new Attribute<>(); // Физические
величины
    private Attribute<Multiplier> multiplier = new Attribute<>(); //
Множители

    public enum Units{
        AMPERE, VOLT, SECOND, DEGREES, RADIAN, OHM, HERTZ,
        VOLT_AMPERE, WATTS, VOLT_AMPERE_REACTIVE
    }

    public enum Multiplier{
        YOCTO, ZEPTO, ATTO, FEMTO, PICO, NANO, MICRO, MILLI, CENTI, DECI,
        DECA, HECTO, KILO, MEGA, GIGA, TERA, PETA, EXA, ZETTA, YOTTA
    }
}

```

## Quality

```
@Getter @Setter
public class Quality extends Data {
    private Attribute<VALIDITY> validity = new Attribute<>(); //
Идентификатор качества
    private Attribute<Boolean> overflow = new Attribute<>(false); //
Переполнение
    private Attribute<Boolean> outOfRange = new Attribute<>(false); // За
пределами предопределенного диапазона
    private Attribute<Boolean> badReference = new Attribute<>(false); //
Источник информации не откалиброван
    private Attribute<Boolean> oscillatory = new Attribute<>(false); //
Колесательный идентификатор качества
    private Attribute<Boolean> failure = new Attribute<>(false); //
Повреждение
    private Attribute<Boolean> oldData = new Attribute<>(false); //
Обновление давно не производилось
    private Attribute<Boolean> inconsistent = new Attribute<>(false); //
Несогласованный
    private Attribute<Boolean> inaccurate = new Attribute<>(false); //
Неудовлетворительная точность
    private SOURCE source = SOURCE.PROCESS; // Идентификатор происхождения
значения (получен из процесса)
    private Attribute<Boolean> test = new Attribute<>(false); // Для указания
тестового значения
    private Attribute<Boolean> operatorBlocked = new Attribute<>(false); //
Оператор блокирует дальнейшие обновления

    public enum VALIDITY{
        GOOD, INVALID, RESERVED, QUESTIONABLE
    }
    public enum SOURCE{
        PROCESS, SUBSTITUTED
    }
}
```

## Objects extended Data

### ACD

```
/**
 * Класс для представления сведений об активации направленной защиты
 */

@Getter
@Setter
public class ACD extends ACT {
    private Attribute<Boolean> general = new Attribute<>();
    private Attribute<Direction> dirGeneral = new Attribute<>();
    private Attribute<Boolean> phsA = new Attribute<>();
    private Attribute<Direction> dirPhsA = new Attribute<>();
    private Attribute<Boolean> phsB = new Attribute<>();
    private Attribute<Direction> dirPhsB = new Attribute<>();
    private Attribute<Boolean> phsC = new Attribute<>();
    private Attribute<Direction> dirPhsC = new Attribute<>();
    private Attribute<Boolean> neut = new Attribute<>();
    private Attribute<Direction> dirNeut = new Attribute<>();
    private Quality q = new Quality();
    private Timestamp t = new Timestamp();
}
```

```

public enum Direction {
    UNKNOWN, FORWARD, BACKWARD, BOTH
}

```

## ACT

```

/**
 * Класс для представления сведений об активации защиты
 */

@Getter @Setter
public class ACT extends Data {
    private Attribute<Boolean> general = new Attribute<>();
    private Attribute<Boolean> phsA = new Attribute<>();
    private Attribute<Boolean> phsB = new Attribute<>();
    private Attribute<Boolean> phsC = new Attribute<>();
    private Attribute<Boolean> neut = new Attribute<>();
    private Quality q = new Quality();
    private Timestamp t = new Timestamp();
    private Timestamp operTmPhsA = new Timestamp();
    private Timestamp operTmPhsB = new Timestamp();
    private Timestamp operTmPhsC = new Timestamp();
}

```

## AnalogueValue

```

/**
 * Класс для хранения значений аналоговых сигналов
 */

@Getter @Setter
public class AnalogueValue extends Data {
    private Attribute<Double> f = new Attribute<>();
    private Attribute<Integer> i = new Attribute<>();
}

```

## ASG

```

/**
 * Класс для задания значения аналогового сигнала
 */

@Getter @Setter
public class ASG extends Data {
    private AnalogueValue setMag = new AnalogueValue();
    private Unit units = new Unit();
    private AnalogueValue minVal = new AnalogueValue();
    private AnalogueValue maxVal = new AnalogueValue();
    private AnalogueValue stepSize = new AnalogueValue();
}

```

## BCR

```

/**
 * Класс для считывания значений двоичного счетчика
 */

```

```
@Getter @Setter
public class BCR extends Data {
    private Attribute<Integer> actVal = new Attribute<>();
    private Attribute<Integer> frVal = new Attribute<>();
    private Timestamp frTm = new Timestamp();
    private Quality q = new Quality();
    private Timestamp t = new Timestamp();
}
```

## CMV

```
/**
 * Класс для описания комплексных измеренных значений
 */
@Getter @Setter
public class CMV extends Data {
    private Vector instCVal = new Vector();
    private Vector cVal = new Vector();
    private Attribute<Range> range = new Attribute<>();
    private Attribute<Range> rangeAng = new Attribute<>();
    private Quality q = new Quality();
    private Timestamp t = new Timestamp();

    public enum Range{
        NORMAL, HIGH, LOW, HIGH_HIGH, LOW_LOW
    }
}
```

## CSD

```
/**
 * Класс для описания формы кривой
 */
@Getter @Setter
public class CSD extends Data {
    private Unit xUnits = new Unit();
    private Attribute<String> xD = new Attribute<>();
    private Attribute<Character.UnicodeBlock> xDU = new Attribute<>();
    private Unit yUnits = new Unit();
    private Attribute<String> yD = new Attribute<>();
    private Attribute<Character.UnicodeBlock> yDU = new Attribute<>();
    private Unit zUnits = new Unit();
    private Attribute<String> zD = new Attribute<>();
    private Attribute<Character.UnicodeBlock> zDU = new Attribute<>();
    private Attribute<Integer> numPts = new Attribute<>();
    private List<Double> crvPts = new ArrayList<>();
    private Attribute<String> d = new Attribute<>();
    private Attribute<Character.UnicodeBlock> dU = new Attribute<>();
    private Attribute<String> cdcNs = new Attribute<>();
    private Attribute<String> cdcName = new Attribute<>();
    private Attribute<String> dataNs = new Attribute<>();
}
```

## CSG

```

/**
 * Класс для настройки формы кривой
 */

@Getter @Setter
public class CSG extends Data {
    private Attribute<Float> pointZ = new Attribute<>();
    private Attribute<Integer> numPts = new Attribute<>();
    private List<Double> crvPts = new ArrayList<>();
}

```

## CURVE

```

/**
 * Класс для настройки кривой
 */

@Getter @Setter
public class CURVE extends Data {
    private Attribute<Float> setParA = new Attribute<>();
    private Attribute<Float> setParB = new Attribute<>();
    private Attribute<Float> setParC = new Attribute<>();
    private Attribute<Float> setParD = new Attribute<>();
    private Attribute<Float> setParE = new Attribute<>();
    private Attribute<Float> setParF = new Attribute<>();
}

```

## DEL

```

/**
 * Класс для представления набора междуфазных измеренных значений
 */

@Getter @Setter
public class DEL extends Data {
    private CMV phsAB = new CMV();
    private CMV phsBC = new CMV();
    private CMV phsCA = new CMV();
    private Attribute<AngRef> angRef = new Attribute<>();
    private Attribute<String> d = new Attribute<>();
    private Attribute<Character.UnicodeBlock> dU = new Attribute<>();
    private Attribute<String> cdcNs = new Attribute<>();
    private Attribute<String> cdcName = new Attribute<>();
    private Attribute<String> dataNs = new Attribute<>();

    public enum AngRef {
        Va, Vb, Vc, Aa, Ab, Ac, Vab, Vbc,
        Vca, Vother, Aother, Synchrophasor
    }
}

```

## DPC

```

/**
 * Класс дублированного управления и состояния
 */

```



```
@Getter @Setter
public class DPC extends Data {
    private Attribute<Values> stVal = new Attribute<>();
    private Quality q = new Quality();
    private Timestamp t = new Timestamp();

    public enum Values{
        INTERMEDIATE_STATE, OFF, ON, BAD_STATE
    }
}
```

## DPL

```
/**
 * Класс - паспортная табличка устройства
 */

@Getter @Setter
public class DPL extends Data {
    private Attribute<String> vendor = new Attribute<>();
}
```

## ENG

```
/**
 * Класс - Enumerated Status Setting (строка состояния в виде перечисления)
 */

@Getter @Setter
public class ENG extends Data {
    private Attribute<Enumerated> setVal = new Attribute<>();
    private enum Enumerated{}
}
```

## ENS

```
/**
 * Класс - Enumerated status
 */

@Getter @Setter
public class ENS extends Data {
    private Attribute<Values> stVal = new Attribute<>();
    private Quality q = new Quality();
    private Timestamp t = new Timestamp();

    private enum Values{}
}
```

## INC

```
/**
 * Класс - Controllable integer status - целочисленное управление и состояние
 */
```

```
@Getter @Setter
public class INC extends Data {
    private Originator origin = new Originator();
    private Attribute<Integer> ctlNum = new Attribute<>();
    private Attribute<Integer> stVal = new Attribute<>();
    private Quality q = new Quality();
    private Timestamp t = new Timestamp();
    private Attribute<Boolean> stSeld = new Attribute<>();
    private Attribute<Boolean> opRcvd = new Attribute<>();
    private Attribute<Boolean> opOk = new Attribute<>();
    private Timestamp tOpOk = new Timestamp();
}
```

## ING

```
/**
 * Класс - integer status setting - установка состояния целочисленная
 */

@Getter @Setter
public class ING extends Data {
    private Attribute<Integer> setVal = new Attribute<>();
}
```

## INS

```
/**
 * Класс - Integer status - целочисленное состояние
 */

@Getter @Setter
public class INS {
    private Attribute<Integer> stVal = new Attribute<>();
    private Quality q = new Quality();
    private Timestamp t = new Timestamp();
}
```

## SAV

```
/**
 * Класс для представления выборок мгновенных значений аналоговых сигналов
 */

@Getter @Setter
public class SAV extends Data {
    private AnalogueValue instMag = new AnalogueValue();
    private Quality q = new Quality();
    private Timestamp t = new Timestamp();
}
```

## SPC

```
/**
 * Класс - Controllable single point - недублированное управление и состояние
 */
```

```
@Getter @Setter
public class SPC extends Data {
    private Originator origin = new Originator();
    private Attribute<Integer> ctlNum = new Attribute<>();
    private Attribute<Boolean> stVal = new Attribute<>();
    private Quality q = new Quality();
    private Timestamp t = new Timestamp();
    private Attribute<Boolean> stSeld = new Attribute<>();
    private Attribute<Boolean> opRcvd = new Attribute<>();
    private Attribute<Boolean> opOk = new Attribute<>();
    private Timestamp tOpOk = new Timestamp();
}
```

## SPS

```
/**
 * Класс - Single point status - недублированное состояние
 */

@Getter @Setter
public class SPS extends Data {
    private Attribute<Boolean> stVal = new Attribute<>();
    private Quality q = new Quality();
    private Timestamp t = new Timestamp();
}
```

## Vector

```
/**
 * Класс для описания векторных значений (амплитуда + фаза)
 */

@Getter @Setter
public class Vector extends Data {

    private AnalogueValue mag = new AnalogueValue();
    private AnalogueValue ang = new AnalogueValue();
}
```

## WYE

```
/**
 * Класс для описания трехфазного сигнала в комплексном виде
 */

@Getter @Setter
public class WYE extends Data {
    private CMV phsA = new CMV();
    private CMV phsB = new CMV();
    private CMV phsC = new CMV();
    private CMV neut = new CMV();
    private CMV net = new CMV();
    private CMV res = new CMV();
    private Attribute<DEL.AngRef> angRef = new Attribute<>();
    private Attribute<Boolean> phsToNeut = new Attribute<>(false);
    private Attribute<String> d = new Attribute<>();
    private Attribute<Character.UnicodeBlock> dU = new Attribute<>();
    private Attribute<String> cdcNs = new Attribute<>();
}
```

```
private Attribute<String> cdcName = new Attribute<>();  
private Attribute<String> dataNs = new Attribute<>();  
}
```

## Filters

### Filter

```
/**
 * Базовый класс для всех видов фильтров
 */

public abstract class Filter {
    public abstract void process(SAV measuredValue, CMV result);
}
```

### Fourier

```
/**
 * Класс для реализации фильтра Фурье
 */

public class Fourier extends Filter{
    private ING bSize = new ING();
    private final SAV[] buffer;
    public Attribute<Integer> bCount = new Attribute<>();
    public Attribute<Double> rVal = new Attribute<>();
    public Attribute<Double> imVal = new Attribute<>();
    public Attribute<Double> freq = new Attribute<>();
    public Attribute<Double> dT = new Attribute<>();

    public Fourier(int bufferSize) {
        bSize.getSetVal().setValue(bufferSize);
        bCount.setValue(0);
        rVal.setValue(0D);
        imVal.setValue(0D);
        freq.setValue(50D);
        dT.setValue(0.02 / bSize.getSetVal().getValue());
        buffer = new SAV[bSize.getSetVal().getValue()];

        for (int i = 0; i < bSize.getSetVal().getValue(); i++) {
            SAV tempVal = new SAV();
            tempVal.getInstMag().getF().setValue(0D);
            buffer[i] = tempVal;
        }
    }

    @Override
    public void process(SAV measuredValue, Vector result) {

        rVal.setValue(rVal.getValue()
            + (measuredValue.getInstMag().getF().getValue()
            - buffer[bCount.getValue()].getInstMag().getF().getValue())
            * Math.sin(2 * Math.PI * freq.getValue() * bCount.getValue()
            * dT.getValue())
            * 2 / bSize.getSetVal().getValue()
        );
        imVal.setValue(imVal.getValue()
            + (measuredValue.getInstMag().getF().getValue()
            - buffer[bCount.getValue()].getInstMag().getF().getValue())
            * Math.cos(2 * Math.PI * freq.getValue() * bCount.getValue()
            * dT.getValue())
            * 2 / bSize.getSetVal().getValue()
        );
    }
}
```

```

        result.getMag().getF().setValue(
            0.7071068 * Math.sqrt(Math.pow(rVal.getValue(), 2) +
Math.pow(imVal.getValue(), 2))
        );

        if ( rVal.getValue() <0){
            result.getAng().getF().setValue(
                Math.atan(imVal.getValue() / rVal.getValue()) * 180 /
Math.PI -180 );
        }
        else {
            result.getAng().getF().setValue(
                Math.atan(imVal.getValue() / rVal.getValue()) * 180 /
Math.PI
            );
        }
    }

    buffer[bCount.getValue()].getInstMag().getF().setValue(measuredValue.getInstM
ag().getF().getValue());
    bCount.setValue(bCount.getValue() + 1);
    if (bCount.getValue() >= bSize.getSetVal().getValue()){
        bCount.setValue(0);
    }
}
}

```

## MainX

```

public class MainX {
    private static final List<LN> logicalNodes = new ArrayList<>();

    public static void main(String[] args) throws IOException {

        LSVS lsvs = new LSVS();
        lsvs.setDatasetSize(64);
        lsvs.setNicName("Qualcomm Atheros QCA9377 Wireless Network Adapter");
        MMXU mmxu = new MMXU();
        logicalNodes.add(mmxu);
        mmxu.IaInst = lsvs.phsAIIInst;
        mmxu.IbInst = lsvs.phsBIIInst;
        mmxu.IcInst = lsvs.phsCIIInst;
        mmxu.UaInst = lsvs.phsAUIInst;
        mmxu.UbInst = lsvs.phsBUIInst;
        mmxu.UcInst = lsvs.phsCUIInst;

        MSQI msqi = new MSQI();
        msqi.A = mmxu.A;
        msqi.PNV = mmxu.PNV;
        logicalNodes.add(msqi);

        RDIR rdir = new RDIR(0, 80, -80, 35, 5000);
        logicalNodes.add(rdir);
        rdir.A = mmxu.A;
        rdir.PNV = mmxu.PNV;
        rdir.lineVoltages = msqi.getImbPPV();

        PTOC ptocl = new PTOC();
        logicalNodes.add(ptocl);
        ptocl.A = mmxu.A;
        ptocl.currentSeq = msqi.getSeqA();
        ptocl.direction = rdir.getDir();
    }
}

```

```

ptocl.StrVal.getSetMag().getF().setValue(600.0);
ptocl.OpDITmms.getSetVal().setValue(10);
ptocl.isDir = false;

RREC rrec1 = new RREC();
rrec1.Rec1Tmms.getSetVal().setValue(200);
rrec1.Rec2Tmms.getSetVal().setValue(250);
rrec1.Rec3Tmms.getSetVal().setValue(300);
rrec1.PlsTmms.getSetVal().setValue(10);
rrec1.RclTmms.getSetVal().setValue(10_000);

rrec1.OpOpn.setPhsA(ptocl.Op.getPhsA());
rrec1.OpOpn.setPhsB(ptocl.Op.getPhsB());
rrec1.OpOpn.setPhsC(ptocl.Op.getPhsC());

RSYN rsyn1 = new RSYN();
logicalNodes.add(rsyn1);
rsyn1.DifV.getSetMag().getF().setValue(1000.0);
rsyn1.DifHz.getSetMag().getF().setValue(0.5);
rsyn1.DifAng.getSetMag().getF().setValue(10.0);
rsyn1.LivDeaMod.getSetVal().setValue(0);
rsyn1.DeaLinVal.getSetMag().getF().setValue(1000.0);
rsyn1.LivLinVal.getSetMag().getF().setValue(10_000.0);
rsyn1.DeaBusVal.getSetMag().getF().setValue(1000.0);
rsyn1.LivBusVal.getSetMag().getF().setValue(10_000.0);
rsyn1.PlsTmms.getSetVal().setValue(10);
rsyn1.BkrTmms.getSetVal().setValue(50);
rsyn1.LinePhV = mmxu.PNV;
rsyn1.BusPhV = mmxu.PNV;
rsyn1.LineHz.getCVal().getMag().getF().setValue(50.0);
rsyn1.BusHz.getCVal().getMag().getF().setValue(50.0);
rsyn1.SynPrg = rrec1.SynPrg;
rrec1.Rel = rsyn1.Rel;

CSWI cswi1 = new CSWI();
logicalNodes.add(cswi1);
cswi1.protSignalsList.add(ptocl.Op);
cswi1.automaticSignalsList.add(rrec1.Op);
cswi1.Pos.getStVal().setValue(DPC.Values.ON);

XCBR xcbr1 = new XCBR();
logicalNodes.add(xcbr1);

xcbr1.PosA = cswi1.PosA;
xcbr1.PosB = cswi1.PosB;
xcbr1.PosC = cswi1.PosC;

lsvs.PosA = xcbr1.PosA;
lsvs.PosB = xcbr1.PosB;
lsvs.PosC = xcbr1.PosC;

rrec1.Blk = xcbr1.BlkCls;

rrec1.PosA = xcbr1.PosA;
rrec1.PosB = xcbr1.PosB;
rrec1.PosC = xcbr1.PosC;

LGOS lgos1 = new LGOS();
rrec1.BlkRec = lgos1.BlkRec;
logicalNodes.add(lgos1);
logicalNodes.add(rrec1);

NHMI nhmiCurrents = new NHMI();

```

```

        logicalNodes.add(nhmiCurrents);
        nhmiCurrents.addSignals(new NHMISignal("ia",
lsvs.phsAIInst.getInstMag().getF()));
        nhmiCurrents.addSignals(new NHMISignal("ib",
lsvs.phsBIInst.getInstMag().getF()));
        nhmiCurrents.addSignals(new NHMISignal("ic",
lsvs.phsCIInst.getInstMag().getF()));

        NHMI nhmiAnalogCurrent = new NHMI();
        logicalNodes.add(nhmiAnalogCurrent);
        nhmiAnalogCurrent.addSignals(new NHMISignal("rmsA",
mmxu.A.getPhsA().getCVal().getMag().getF()));
        nhmiAnalogCurrent.addSignals(new NHMISignal("rmsB",
mmxu.A.getPhsB().getCVal().getMag().getF()));
        nhmiAnalogCurrent.addSignals(new NHMISignal("rmsC",
mmxu.A.getPhsC().getCVal().getMag().getF()));

        NHMI nhmiProtectionWorkingA = new NHMI();
        logicalNodes.add(nhmiProtectionWorkingA);
        nhmiProtectionWorkingA.addSignals(new NHMISignal("StrPtocl_A",
ptocl1.Str.getPhsA()));
        nhmiProtectionWorkingA.addSignals(new NHMISignal("OpPtocl_A",
ptocl1.Op.getPhsA()));
        nhmiProtectionWorkingA.addSignals(new NHMISignal("OpRrec1_A",
rrec1.Op.getPhsA()));
        nhmiProtectionWorkingA.addSignals(new NHMISignal("PosXcbr_A",
xcbr1.isOpenA));

        NHMI nhmiProtectionWorkingB = new NHMI();
        logicalNodes.add(nhmiProtectionWorkingB);
        nhmiProtectionWorkingB.addSignals(new NHMISignal("StrPtocl_B",
ptocl1.Str.getPhsB()));
        nhmiProtectionWorkingB.addSignals(new NHMISignal("OpPtocl_B",
ptocl1.Op.getPhsB()));
        nhmiProtectionWorkingB.addSignals(new NHMISignal("OpRrec1_B",
rrec1.Op.getPhsB()));
        nhmiProtectionWorkingB.addSignals(new NHMISignal("PosXcbr_B",
xcbr1.isOpenB));

        NHMI nhmiProtectionWorkingC = new NHMI();
        logicalNodes.add(nhmiProtectionWorkingC);
        nhmiProtectionWorkingC.addSignals(new NHMISignal("StrPtocl_C",
ptocl1.Str.getPhsC()));
        nhmiProtectionWorkingC.addSignals(new NHMISignal("OpPtocl_C",
ptocl1.Op.getPhsC()));
        nhmiProtectionWorkingC.addSignals(new NHMISignal("OpRrec1_C",
rrec1.Op.getPhsC()));
        nhmiProtectionWorkingC.addSignals(new NHMISignal("PosXcbr_C",
xcbr1.isOpenC));

        lsvs.addListener(new MyPacketListener() {
            @Override
            public void listen() {
                logicalNodes.forEach(LN::process);
            }
        });
        lsvs.process();
    }
}

```



```
@Getter @Setter
public class SEQ {
    private CMV c1 = new CMV();
    private CMV c2 = new CMV();
    private CMV c3 = new CMV();

    public enum seqT {
        POS, NEG, ZERO
    }
}
```