

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Вычислительные машины, комплексы, системы и сети»

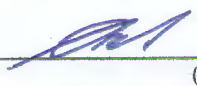
БАКАЛАВРСКАЯ РАБОТА

на тему

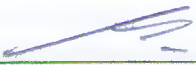
ПРОГРАММНАЯ ИМИТАЦИОННАЯ
МОДЕЛЬ ПРОИЗВОДСТВЕННОЙ СИСТЕМЫ
FESTO И ЕЕ ИССЛЕДОВАНИЕ

Студент  31.05.2022 Дубинин Алексей Викторович
(подпись, дата) (ФИО полностью)

Руководитель  31.05.2022 Дубинин В. Н.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр  31.05.2022 Бычков А. С.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 02.06.2022 г. № 13)

Заведующий кафедрой  Митрохин М.А.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от

07.06.2022 г. № 6)

Секретарь ГЭК

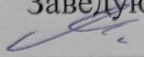

(подпись)

Гурин Е.И.
(фамилия, инициалы)

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
 М.А. Митрохин
«26» апреля 2022г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Дубинин Алексей Викторович гр. 18BB1 факультета вычислительной техники
2. Тема работы «Программная имитационная модель производственной системы *FESTO* и ее исследование»

Тема утверждена приказом ПГУ № 357/о от "26" апреля 2022г.

3. Руководитель бакалаврской работы д.т.н. профессор кафедры ВТ ПГУ Дубинин В.Н.

4. Задание на работу

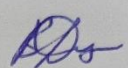
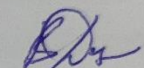
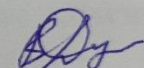
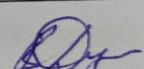
Разработать две программные имитационные модели тестирующей станции производственной системы *FESTO*, одна из которых на основе языка программирования высокого уровня, позволяющая формировать и записывать журнал событий, а другая – на основе функциональных блоков стандарта *IEC 61499*, которая может использоваться при реализации системы управления данной станцией.

По журналам событий, полученных в ходе имитационных экспериментов, с использованием технологии *Process mining* сгенерировать формальные модели процессов, протекающих в тестирующей станции производственной системы *FESTO*, а также произвести их анализ.

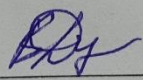
5. Перечень подлежащих разработке вопросов

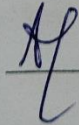
- 1) Обзор и анализ методов моделирования производственной системы *FESTO*
- 2) Разработка программной имитационной модели тестирующей станции производственной системы *FESTO*
- 3) Генерация формальных моделей процессов тестирующей станции на основе метода *Process mining* и их исследование
- 4) Разработка распределенной функционально-блочной модели тестирующей станции производственной системы *FESTO*

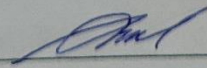
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Обзор и анализ методов моделирования производственной системы <i>FESTO</i>	15%	26.04.22-30.04.22	
2	Разработка программной имитационной модели тестирующей станции производственной системы <i>FESTO</i>	25%	01.05.22-07.05.22	
3	Генерация формальных моделей процессов тестирующей станции на основе метода <i>Process mining</i> и их исследование	25%	08.05.22-15.05.22	
4	Разработка распределенной функционально-блочной модели тестирующей станции производственной системы <i>FESTO</i>	25%	16.05.22-23.05.22	
5	Оформление выпускной работы.	10%	23.05.22-31.05.22	

Дата выдачи задания "26" 04 2022 г.

Руководитель бакалаврской работы  26.04.22 Дубинин В.Н.
(подпись, дата)

Задание к исполнению принял студент  26.04.22 Дубинин А.В.
(подпись, дата)

Работу к защите допустить
Декан факультета  31.05.2022 Фионова Л.Р.
(подпись, дата)

Реферат

Пояснительная записка содержит 104 листов, 68 рисунков, 24 источника, 4 приложения.

ПРОИЗВОДСТВЕННАЯ СИСТЕМА, ИМИТАЦИОННОЕ
МОДЕЛИРОВАНИЕ, ПРОЦЕССНАЯ АНАЛИТИКА, ФУНКЦИОНАЛЬНЫЕ
БЛОКИ, IEC 61499, FESTO, PROCESS MINING, NXTSTUDIO

Цель работы – разработка и исследование имитационных моделей тестирующей станции производственной системы *FESTO*.

Основные результаты работы:

- разработана программная имитационная модель тестирующей станции производственной системы *FESTO*, позволяющая записывать журнал событий;
- по журналам событий, полученных в ходе имитационных экспериментов, с использованием технологии *Process mining* сгенерированы формальные модели процессов, протекающих в тестирующей станции производственной системы *FESTO*, а также произведено их исследование;
- с использованием паттерна проектирования *MVC*, а также сгенерированных формальных моделей разработана (имитационная) функционально-блочная модель тестирующей станции производственной системы *FESTO* на основе международного стандарта *IEC 61499*, причем управляющая компонента может быть перенесена на реальную систему практически без изменений.

Использованный в работе подход может быть адаптирован и использован в организациях, работающих в сфере промышленной автоматизации при разработке управляющих функционально-блочных приложений для промышленных киберфизических систем.

ПГУ 1.090301.11.001 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата
Разраб.		Дубинин А.В.		
Пров.		Дубинин В.Н.		
Н. Контр.		Бычков А.С.		
Утверд.		Митрохин М.А.		

Программная имитационная модель производственной системы <i>FESTO</i> и ее исследование Пояснительная записка			Лит.	Лист	Листов
			2	104	
ФВТ, каф. ВТ гр. 18ВВ1					

Содержание

Введение	5
1 Программная имитационная модель производственной системы <i>FESTO</i> и ее исследование	8
1.1 Обзор и анализ методов моделирования производственной системы <i>FESTO</i>	8
1.1.1 Неформальное описание установки <i>FESTO</i>	8
1.1.2 <i>UML</i> -модель установки <i>FESTO</i>	10
1.1.3 Графическое моделирование установки <i>FESTO</i>	12
1.1.4 Моделирование установки <i>FESTO</i> на основе <i>NCES</i> -сетей	15
1.1.5 Моделирование установки <i>FESTO</i> с использованием раскрашенных сетей Петри	17
1.1.6 Постановка задачи исследований	19
1.2 Разработка программной имитационной модели тестирующей станции производственной системы <i>FESTO</i>	20
1.2.1 Сигналы устройств тестирующей станции	21
1.2.2 Основной алгоритм событийного имитационного моделирования	22
1.2.3 Алгоритмы обработки событий в имитационной модели тестирующей станции	25
1.2.4 Описание программы	29
1.2.5 Тестирование программы	32
1.3 Генерация формальных моделей процессов тестирующей станции <i>FESTO</i> на основе метода <i>process mining</i>	34
1.3.1 Использование метода <i>Process Mining</i> в проектировании	34
1.3.2 Генерация формальных моделей процессов на основе журналов событий	37
1.3.3 Анализ сетевых моделей процессов в системе <i>TINA</i>	42

1.4 Разработка распределенной функционально-блочной модели	
тестирующей станции системы <i>FESTO</i>	45
1.4.1 MVC-паттерн проектирования	45
1.4.2 Модели переходов состояний сценариев обработки деталей	46
1.4.3 Системная конфигурация	47
1.4.4 Управляющее приложение	48
1.4.5 Компонента <i>M</i> (Модель)	50
1.4.6 Компонента <i>V</i> (Представление)	52
1.4.7 Компонента <i>C</i> (Контроллер)	60
1.4.8 Отладка и тестирование	62
2 Требования к освещению рабочего места, оборудованного	
ПЭВМ	65
2.1 Общие положения	65
2.2 Требования к освещению на рабочих местах, оборудованных ПЭВМ . . .	67
2.3 Выводы	71
Заключение	72
Список использованных источников	74
Приложение А – Листинг программы	77
Приложение Б – Схемы программ	89
Приложение В – UML-диаграммы.	92
Приложение Г – Презентация	95

Введение

В связи с принятием концепции производства «Индустрия 4.0» как наиболее перспективной актуальным становится разработка методов проектирования промышленных киберфизических систем (КФС), которые являются одним из главных атрибутов «Индустрии 4.0». Данное направление является новым и исследования в данной области далеки от завершения. К особенностям КФС можно отнести их структурную и поведенческую сложность, сильную взаимосвязь физических процессов, протекающих в системе с управляющими элементами, распределенность в пространстве и времени.

Важным подклассом КФС являются промышленные КФС, непосредственно связанные с промышленным производством. Примерами таких КФС являются производственные системы, сборочные производства, разумные сети электроснабжения (сети *Smart Grid*), системы транспортировки и логистические системы и т.д. Каждая из таких систем имеет свои особенности. Производственные системы, как правило, связаны с производством изделий на основе обработки исходных материалов. Одной из главных их особенностей является то, что они в основном относятся к классу дискретных событийных систем.

В проектировании сложных технических систем выделяются два основных вида деятельности – анализ и синтез. Имитационное моделирование является одним из эффективных методов изучения и анализа сложных систем. Компьютерные модели проще и удобнее исследовать в силу их возможности проводить т. н. вычислительные эксперименты, в тех случаях, когда реальные эксперименты затруднены из-за финансовых или физических препятствий, или могут дать непредсказуемый результат. Недостатком имитационного моделирования является неполная формальность модели, а также неполнота анализа, особенно в области валидации и верификации системы. Неполная формализация препятствует изучению и восприятию модели, ее

документированию, коммуникации между разработчиками в процессе проектирования, а также применению формальных методов анализа и синтеза.

Перспективным подходом к синтезу формальных моделей является метод *Process mining*, известный в русскоязычной литературе как «процессная аналитика». Данный метод позволяет на основе журналов событий автоматически генерировать формальные модели. В качестве основных формализмов используются конечно-автоматные модели, сети Петри и их расширения, а также другие модели переходов состояний. На последующих стадиях метод *Process mining* позволяет проводить проверку соответствия реального поведения системы сгенерированной формальной системе (*conformance checking*), а также производить «дообучение» формальной модели на основе новых журналов событий.

С точки зрения информатики большой интерес представляет проектирование кибернетической части КФС (иначе, проектирование систем управления). В настоящее время наиболее привлекательным подходом для построения распределенных систем управления промышленными процессами является использование международного стандарта *IEC 61499*. По сути дела, стандарт *IEC 61499* вводит класс систем управления нового поколения.

На основе вышеизложенного тема бакалаврской работы, посвященной разработке, анализу и реализации имитационных программных и функционально-блочных моделей производственной системы *FESTO*, является актуальной. В работе будет исследоваться только тестирующая станция производственной системы *FESTO*, однако полученные результаты можно будет распространить и на другие станции.

Задачами работы являются:

- разработка программной имитационной модели тестирующей станции производственной системы *FESTO*, позволяющей записывать журнал событий;

- синтез формальных моделей процессов, протекающих в тестирующей станции *FESTO* по журналам событий, полученных в ходе имитационных экспериментов, а также их анализ;

- разработка имитационной функционально-блочной модели тестирующей станции *FESTO* на основе стандарта *IEC 61499*, включая модель системы управления (блок контроллера).

					<i>ПГУ 1.090301.11.001 ПЗ</i>	Лист
						7
Изм.	Лист	№ докум	Подпись	Дата		

1 Программная имитационная модель производственной системы *FESTO* и ее исследование

1.1 Обзор и анализ методов моделирования производственной системы *FESTO*

1.1.1 Неформальное описание установки *FESTO*

Производственная система (ПС) *FESTO* предназначена для изготовления некоторых законченных деталей из поступающих на ее вход заготовок. Она состоит из трех станций: распределительной (*Distribution Station*), тестирующей (*Testing Station*) и обрабатывающей (*Processing Station*) (рисунок 1).



Рисунок 1 – Производственная система *FESTO*

Заготовка представляет собой шайбу из пластмассы или металла. В процессе изготовления детали заготовка последовательно проходит через все три станции. Распределительная станция служит для буферирования поступающих от внешнего производителя заготовок, тестирующая станция (ТС) – для контроля и отбраковки нестандартных заготовок, а обрабатывающая станция – для сверления в заготовке отверстия. Обработанная деталь может быть передана внешнему потребителю готовых деталей. В дальнейшем для краткости заготовку будет называть деталью (*workpiece*).

Техпроцесс обработки деталей является одним из определяющих факторов при разработке системы управления. Ниже приводится краткое описание техпроцесса, положенного в основу создания системы управления *FESTO*.

Внешний производитель помещает детали в магазин распределительной станции, где они могут накапливаться. Буферирование деталей в магазине способствует сглаживанию темпов работы внешнего производителя и ПС *FESTO*. При помощи специального выталкивателя (поршня) деталь выталкивается из дна магазина и сдвигается в выходное место магазина, откуда шарнирный рычаг перемещает ее на тестирующую станцию. В данном случае под выходным местом магазина понимается площадка, на которую из магазина выталкивается деталь. Выходное место магазина не является исполнительным магазином и с ним не связаны никакие датчики. Удерживание детали на шарнирном рычаге осуществляет расположенная на его конце присоска. Для присасывания детали включается вакуум, для отсоединения детали вакуум отключается.

Деталь поступает на ТС с распределительной станции при помощи транспортного рычага. При поступлении детали на тестирующую станцию (а именно, на тестирующее место) датчики-детекторы определяют цвет и материал, их которого изготовлена деталь. При ненормальном цвете или материале деталь выталкивается (отбраковывается) из тестирующей станции. Если цвет и материал детали находятся в норме, то деталь поднимается лифтом к устройству измерения высоты детали (измерителю высоты). Если в результате измерения оказывается, что высота детали нормальная, то деталь выталкивается в скользящий желоб, по которому она поступает на обрабатывающую станцию. В противном случае (ненормальная высота) деталь опускается на лифте вниз, где она выталкивается из тестирующей станции и таким образом исключается из дальнейшей обработки. Скользящий желоб имеет заслонку, которая позволяет заблокировать поступление детали на обрабатывающую станцию.

На обрабатывающую станцию деталь поступает в лунку рабочего стола, имеющего форму круга. Рабочий стол поворачивается вокруг своей оси на одну позицию по часовой стрелке, перенося тем самым деталь в позицию для сверления. Сверление осуществляется с помощью дрели, способной двигаться по направляющим вверх и вниз. Перед сверлением деталь прижимается специальным прижимным стержнем, входящим в боковую прорезь лунки. После окончания сверления рабочий стол поворачивается еще на одну позицию по часовой стрелке и деталь оказывается в позиции для проверки отверстия. Проверка осуществляется с помощью специального проверочного стержня, опускаемого вниз по центру детали. Если отверстие нормального качества, то стержень проходит сквозь него и касается специального датчика, что свидетельствует о норме. После окончания проверки отверстия рабочий стол поворачивается на одну позицию по часовой стрелке, перенося деталь в позицию для выгрузки детали из обрабатывающей станции. Выгружаемая деталь передается внешнему потребителю деталей.

1.1.2 UML-модель установки *FESTO*

Перспективным направлением в разработке систем промышленной автоматизации является использование современных технологий инженерии программных систем [1], в число которых входит объектно-ориентированный анализ и проектирование. Исследования в области проектирования систем управления промышленными системами на основе объектно-ориентированного подхода начались в 90-е годы. К числу этих работ относятся также исследования, связанные с использованием языка *UML* [2].

В работе [3] рассмотрен объектно-ориентированный подход к проектированию систем управления промышленной автоматизации на основе языка *UML* и нового международного стандарта *IEC 61499*. Предложено расширение *UML-FB* для реализации систем управления на основе функциональных блоков *IEC 61499*, а также соответствующие правила трансформации. В качестве примера рассматривается производственная система *FESTO*.

Особенностью языка *UML-FB* является его близость стандарту *IEC 61499*. Все конструкции, представимые с использованием языка спецификации ФБ (функциональных блоков) [4], являются представимыми в *UML-FB*. Язык *UML-FB* можно рассматривать, с одной стороны, как полноценный язык моделирования систем управления, с другой стороны – как некоторое расширение языка спецификации ФБ стандарта *IEC 61499* средствами *UML*. При использовании *UML-FB*, например, появляется возможность явного описания иерархии системы в виде диаграммы классов. Для описания функционирования системы могут использоваться диаграммы последовательностей и кооперации, отсутствующие в стандарте *IEC 61499*. Кроме того, появляется возможность использования диаграмм *UML* в качестве языка запросов (для валидации и тестирования модели) и языка структурных и функциональных ограничений (для использования в процессе разработки корректных моделей). Язык *UML-FB* является расширением языка *UML*, поскольку использует аппарат стереотипов. В то же время жесткие правила использования диаграмм и особая семантика определяют его ограниченность в плане применимости в других предметных областях.

В языке *UML-FB* используются следующие виды диаграмм: диаграмма классов, диаграмма последовательностей, диаграмма кооперации и диаграмма состояний. Диаграмма классов используется для представления:

- полной системной иерархии компонентов (по отношениям агрегации и наследования);
- типов ФБ, субприложений, адаптерных интерфейсов, ресурсов и устройств;
- конфигураций систем и устройств;
- интерфейсов ФБ, включающих событийные и информационные входы и выходы, а также *WITH*-связи между событийными и информационными линиями;
- связей между ФБ, характерных для всех экземпляров заданных типов;

- констант, представляющих параметры ФБ, субприложений, ресурсов и устройств.

На рисунке 2 приведен пример диаграмм классов и последовательности выталкивателя, входящего в состав производственной системы *FESTO* [3].

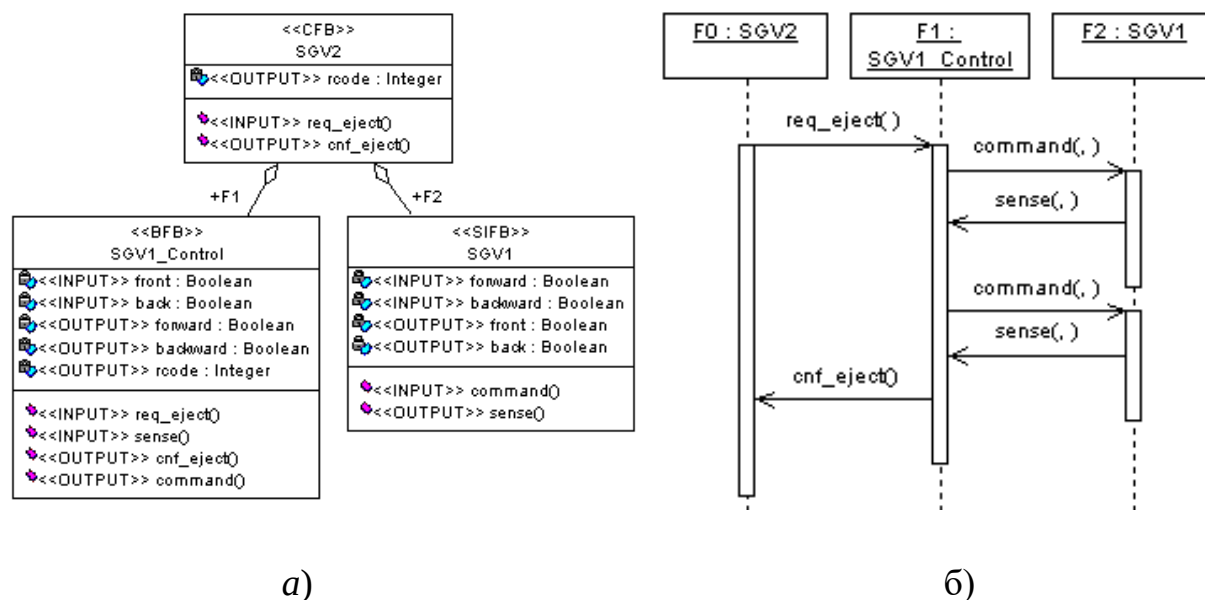


Рисунок 2 – Выталкиватель уровня 2: а) диаграмма классов; б) диаграмма последовательности

Недостатком языка *UML-FB* является его ориентация на описание системы. В то же время не существует программных инструментов для проведения имитационного моделирования на его основе.

1.1.3 Графическое моделирование установки *FESTO*

В данном подразделе рассматривается ряд визуальных (графических) моделей производственной установки *FESTO*, начиная лот простых *Flash*-моделей и заканчивая сложными моделями на основе технологий *OpenGL* и экспертных систем, представленных на сайте [5].

Для графического моделирования установки *FESTO* были разработаны две различные модели на основе технологии *Flash*. Первая модель только визуализирует поведение системы *FESTO*, в то время как другая предоставляет пользователю удобное меню с возможностью выбора различных сценариев

обработки детали, а также интерактивный режим, позволяющий управлять с экрана операциями, связанными с оборудованием.

Во второй модели имеется возможность визуализировать следующие сценарии технологического процесса обработки детали:

- “обработка нормальной детали”;
- “обработка детали, имеющей недопустимый цвет”;
- “обработка детали, изготовленной из несоответствующего материала”;
- “обработка детали, имеющей неподходящую высоту”. Пункт меню

“*Interactive model*” предоставляет пользователю интерактивный режим работы с моделью.

На сайте [5] представлены две программные модели установки *FESTO*, реализованные с использованием объектно-ориентированного подхода на языках *C++* и *Java*. Характерной особенностью обеих моделей является использование в них специально разработанных языков графического моделирования техпроцессов обработки деталей, что позволяет гибко строить и перестраивать различные визуальные модели функционирования системы *FESTO*.

Особенностью первой программной модели, реализованной на языке *C++*, является возможность описания и имитации параллельно протекающих процессов в установке *FESTO*. Все команды языка состоят из пятисимвольного обозначения исполнительного механизма и двухсимвольной команды. Например, строка *table.mf* вращает стол на 90 градусов.

Особенностью второй программной модели (по сравнению с моделью 1) является более выразительный интерфейс и более богатые возможности в моделировании техпроцессов, протекающих в системе *FESTO*. Модель *FESTO* представлена набором объектов (некоторые из них связаны между собой), имеющих определенные свойства и методы. Объекты имеют графические атрибуты (например, текущие и начальные координаты *X*, *Y*, цвет, геометрические размеры – ширину, высоту и т.д.), а также атрибуты,

определяющие логическое состояние объекта (например, наличие детали в выходном месте магазина). Методы объектов представляют действия, производимые на экране (пространственные действия - сдвинуть, передвинуть, опустить, поднять и т.д., действия по изменению цвета объектов, действия по изменению логического состояния объектов).

Для работы с объектами разработан специальный язык сценариев, в котором имеются два типа команд: инициализации и выполнения. В скриптовой программе также выделяются блок инициализации и блок выполнения. Команда скрипта имеет простой формат. Она содержит имя объекта и имя функции или свойства, разделяемые точкой. После имени функции в круглых скобках следует список параметров. После имени свойства следует значение свойства.

На рисунке 3 представлен скриншот программы графического моделирования установки *FESTO* (модель 2).

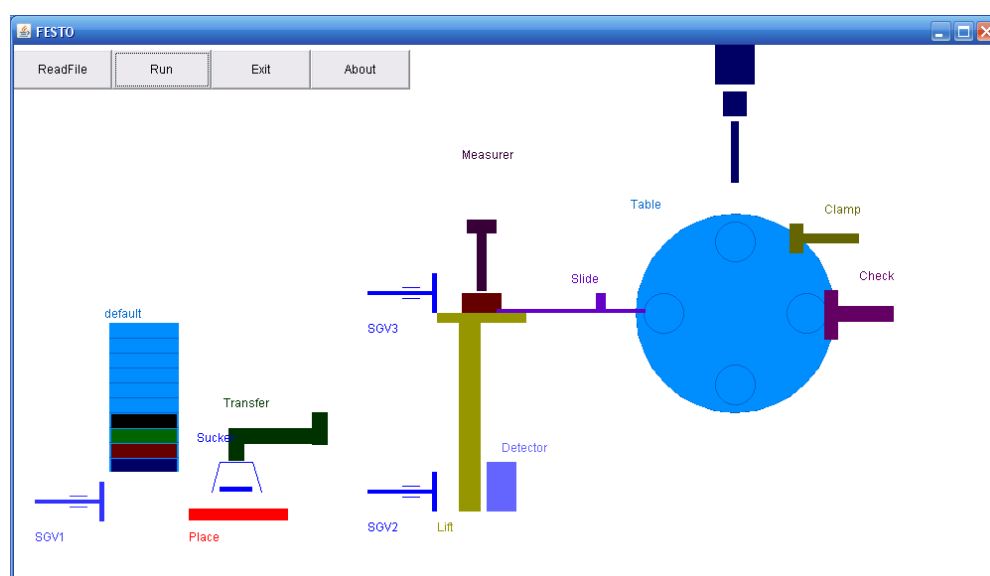


Рисунок 3 – Графическое представление *FESTO* в программной модели 2
(поднятие детали на лифте)

На сайте [5] также представлена система графического моделирования *FESTO* на основе использования базы знаний продукционного типа и пакета графического моделирования *OpenGL* [6]. Создание базы знаний осуществляется на языке *CLIPS* и языке *COOL*, предоставляющем объектно-ориентированные

возможности. Система *CLIPS* представляет собой современный инструмент, предназначенный для создания экспертных систем [7].

1.1.4. Моделирование установки *FESTO* на основе *NCES*-сетей

Для моделирования и верификации инфомехатронных систем со сложными видами взаимодействий часто используются сети Петри и их расширения. Адекватным средством моделирования дискретных событийных систем, к каким относятся многие промышленные КФС, являются *NCES* (*net condition/event systems*) [8]. *NCES* представляют собой сети Петри, расширенные событийными сигналами (дугами) и особым порядком срабатывания переходов. В сетях данного класса принято синхронное срабатывание группы переходов, называемой шагом. В то же время срабатывание шагов в рамках всей сети является асинхронным. Инструментальное средство *ViVe* поддерживает построение и верификацию модульных *NCES* [9]. Моделирование КФС предполагает построения замкнутой модели системы, включающей взаимодействующие модели оборудования и управления. В работе [10] предложена комплексная методология проектирования и валидации систем промышленной автоматизации, основанная на использовании *UML*, *Simulink* и *NCES*.

В данной работе [11] с использованием подхода, предложенного в [10], разрабатывается *NCES*-модель *Pick-and-Place* (*PnP*)-манипулятора, а также производится ее исследование в системе *ViVe*.

В работе [12] рассматривается моделирование и верификация установки *FESTO* с применением *NCES*-сетей. *NCES*-сети программно поддерживаются инструментальной системой *ViVe*, которая имеет в своем составе графический редактор сетей *ViEd* и верификатор *ViVe*.

В качестве примера на рисунке 4 приведена *NCES*-модель контроллера для тестирующей станции, сформированная в системе *ViVe*.

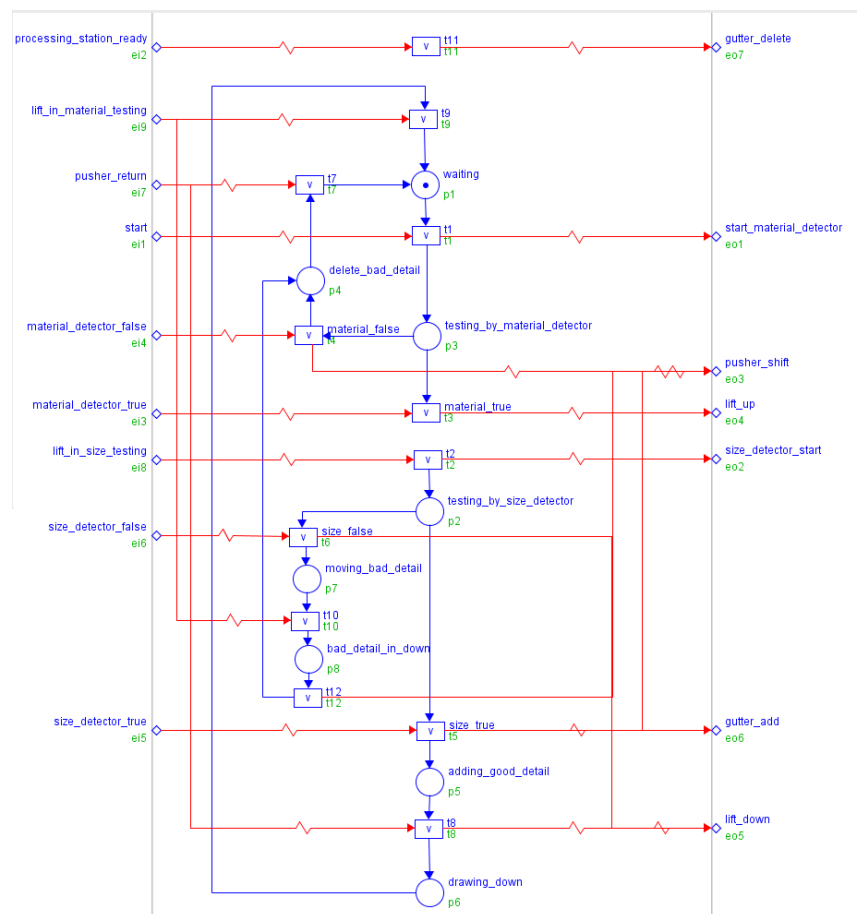


Рисунок 4 – NCES-модель контроллера тестирующей станции системы FESTO

Модель контроллера, по сути, отображает путь движения детали по всей станции. Изначально станция находится в «режиме ожидания» (*waiting*), т.е. ждет, когда шарнирный рычаг принесет деталь и освободит рабочее пространство.

При поступлении сигнала *start*, запускается детектор материала. В случае если деталь оказалось негодной, срабатывает выталкиватель, и при поступлении сигнала о том, что выталкиватель вернулся, станция снова возвращается в «режим ожидания». Если же деталь оказалась подходящей, посылается сигнал поднятия лифта.

Когда лифт достигает верхней точки, запускается детектор размера. Если деталь оказывается неподходящей, на лифте она опускается вниз, там ее удаляет выталкиватель и станция возвращается в исходное состояние. Если же размер детали удовлетворяет нужным качествам, выталкиватель добавляет ее в

скользящий желоб. После возвращения выталкивателя на место, опускается лифт, и станция снова в «режиме ожидания».

Недостатком использования *NCES*-сетей для моделирования системы *FESTO* является сложность представления арифметических выражений для условий переходов из одного состояния в другое, а также сложность реализации системы управления станциями на основе языков программирования логических контроллеров.

1.1.5 Моделирование установки *FESTO* с использованием раскрашенных сетей Петри

В работе [13] были рассмотрены вопросы моделирования и верификации производственной системы *FESTO* на основе высокоуровневых сетей Петри в системе *CPNTools*.

С использованием высокоуровневых сетей Петри (например, цветных сетей Петри [14]) может быть получено более компактное формальное описание системы. Они позволяют проведение как верификации, так и оценки производительности [15]. Существует ряд систем моделирования на основе высокоуровневых сетей Петри, наиболее известной из которых является система *CPN Tools* [16].

В работе [13] сетевая модель системы *FESTO* иерархически разбита на три уровня:

- верхний уровень системы (*TOP*);
- средний уровень системы (*FESTO*);
- нижний уровень системы (*Distribution Station, Testing Station, Processing Station*).

В качестве примера на рисунке 5 представлена сетевая модель тестирующей станции.

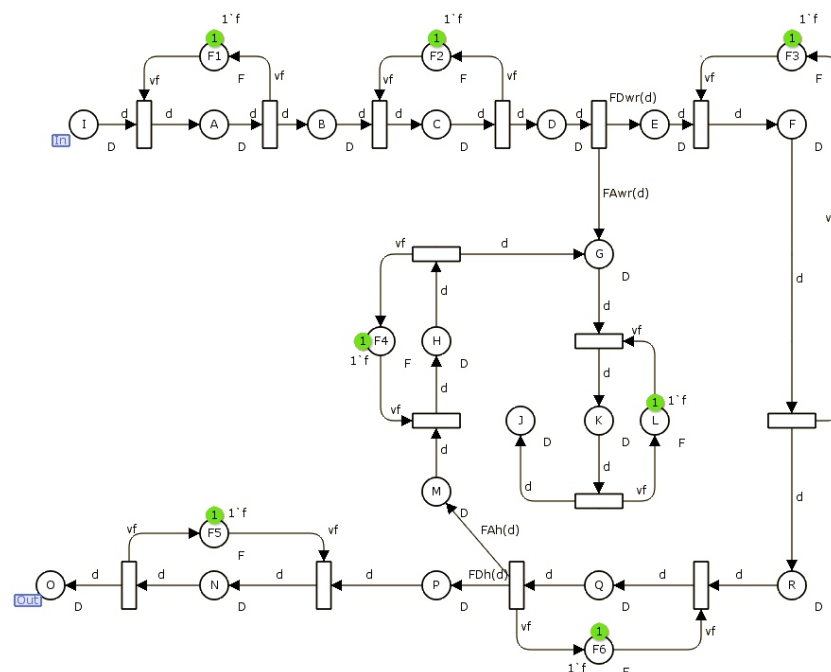


Рисунок 5 – Сетевая модель тестирующей станции

В данной сетевой модели используются два класса фишек – для представления деталей и для представления ресурсов. Фишки первого класса имеют три атрибута:

- высота детали (значения: m – средняя, h – высокая);
- цвет заготовки (значения r – красная, g – зеленая, b – синяя);
- материал заготовки (значения: w – дерево, s – сталь).

Формальная верификация систем в среде *CPN Tools* в принципе основана на методе *Model Checking*. С использованием инструмента Текст (*Text*) из набора Вспомогательные (*Auxiliary*) необходимо разместить на новой странице запросы на специальном языке временной логики *CTL*, принятым в *CPNTools*. Далее вычисляется пространство состояний и граф достижимости модели. После этого можно приступить к выполнению сформированных *CTL*-запросов. Это делается с использованием инструмента *MLEvaluate* из набора *Simulation*. Следует отметить, что из-за большой размерности полученного графа достижимости и недостаточной ориентацией системы *CPNTools* на *Modelchecking*, формальная верификация системы *FESTO* в *CPNTools* представляет большие трудности. Поэтому единственным приемлемым способом (неполной) верификации

является имитационное моделирование. Имитационные эксперименты с сетевой моделью показали правильность логического поведения системы *FESTO*.

Недостатком использования раскрашенных сетей Петри для моделирования системы *FESTO* является сложность реализации системы управления станциями на основе языков программирования логических контроллеров.

Недостатком использования системы *CPN Tools* является низкая скорость имитации модели, поскольку данная система является имитационной системой моделирования в полном смысле этого слова.

1.1.6 Постановка задачи исследований

На основе проведенного анализа методов моделирования производственной системы *FESTO* было установлено, что рассмотренные методы моделирования обладают рядом недостатков, основным из которых является сложность использования моделей для реализации системы управления станциями на основе языков программирования логических контроллеров, в частности, на основе функциональных блоков стандарта *IEC 61499*. Кроме того, процесс построения формальных моделей производственной системы *FESTO* в форме конечных автоматов или сетей Петри является неформализованным и трудно поддается формализации, что требует разработки методов автоматической генерации формальных моделей. На основе вышеизложенного целями работы являются:

- Разработка программной имитационной модели тестирующей станции производственной системы *FESTO*, позволяющей произвести проверку правильности логического поведения системы, а также сгенерировать журналы событий в процессе проведения имитационных экспериментов;

- Построение формальных моделей тестирующей станции производственной системы *FESTO* на основе сгенерированных журналов событий с использованием метода *Process Mining* и инструментальной системы *ProM*, а также их исследование и анализ. Используемые модельные формы – конечные автоматы и сети Петри;

- Разработка функционально-блочной модели тестирующей станции производственной системы *FESTO*, облегчающей реализацию системы управления тестирующей станцией на основе стандарта *IEC 61499*.

1.2 Разработка программной имитационной модели тестирующей станции производственной системы *FESTO*

Тестирующая станция производственной системы *Festo*, которая будет моделироваться представлена на рисунке 6.

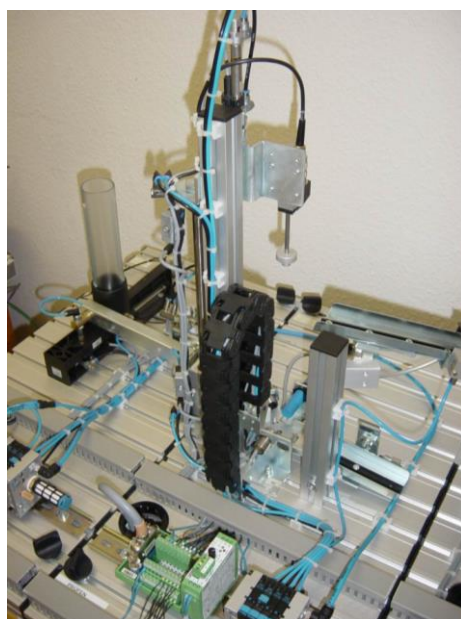


Рисунок 6 – Тестирующая станция производственной системы *Festo*

Тестирующая станция производственной системы *FESTO* включает следующие электромеханические устройства (ЭМУ):

- блок детекторов;
- выталкиватель;
- блок измерений;
- лифт.

1.2.1 Сигналы устройств тестирующей станции

В общем виде множество сигнальных линий L в замкнутой системе «Управление-Оборудование» определяется следующим образом:

$$L = S \cup C; \quad S \cap C = \emptyset,$$

где $S = \{s_1, s_2, \dots, s_m\}$ - множество информационных линий, идущих от датчиков (сенсоров) оборудования к контроллеру;

$C = \{c_1, c_2, \dots, c_n\}$ - множество управляющих линий, идущих от контроллера к исполнительным механизмам оборудования.

На рисунке 7 представлена замкнутая система «Управление-Оборудование» с обозначенными сигнальными линиями. Линии из этих двух множеств имеют одинаковую структуру.

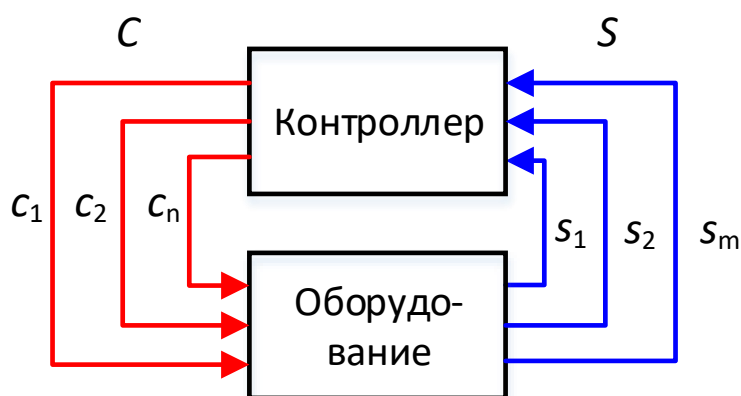


Рисунок 7 – Замкнутая система «Управление-Оборудование»

Множество L будет включать все $(n+m)$ линий:

$$L = \{l_1, l_2, \dots, l_{n+m}\}.$$

Далее будем использовать следующие обозначения сигналов. Сигналы с датчиков (сенсоров) будут начинаться с буквы «S», а управляющие сигналы - с буквы «C». Сигналы с датчиков идут в систему управления, а управляющие сигналы из системы управления на исполнительные механизмы ЭМУ.

Ниже представлены сигналы для каждого из ЭМУ.

С блоком детекторов (*Detectors*) связаны следующие сигналы:

- S_COLOR_TRUE (деталь годна по цвету);

- *S_COLOR_FALSE* (деталь не подходит по цвету).

С лифтом (*Lift*) связаны следующие сигналы:

- *C_LIFT_DOWN_TRUE* (подать сигнал опускания лифта);
- *C_LIFT_DOWN_FALSE* (снять сигнал опускания лифта);
- *C_LIFT_UP_TRUE* (подать сигнал поднятия лифта);
- *C_LIFT_UP_FALSE* (снять сигнал поднятия лифта);
- *S_LIFT_DOWN* (лифт находится в нижнем положении);
- *S_LIFT_UP* (лифт находится в верхнем положении).

С выталкивателем (*Slidegate valve*) связаны следующие сигналы:

- *C_EXTEND_TRUE* (подать сигнал на выдвижение поршня вперед);
- *C_EXTEND_FALSE* (снять сигнал на выдвижение поршня вперед);
- *C_RETRACT_TRUE* (подать сигнал на втягивание поршня назад);
- *C_RETRACT_FALSE* (снять сигнал на втягивание поршня назад);
- *S_END* (поршень достиг крайнего положения).

С блоком измерений (*Measurer*) связаны следующие сигналы:

- *C_MEASURER_DOWN_TRUE* (подать сигнал опускания измерителя);
- *C_MEASURER_DOWN_FALSE* (снять сигнал опускания измерителя);
- *C_MEASURER_UP_TRUE* (подать сигнал поднятия измерителя);
- *C_MEASURER_UP_FALSE* (снять сигнал поднятия измерителя);
- *S_MEASURER_DOWN* (измеритель находится в нижнем положении);
- *S_MEASURER_UP* (измеритель находится в верхнем положении);
- *S_HEIGHT_TRUE* (деталь годна по высоте);
- *S_HEIGHT_FALSE* (деталь не подходит по высоте).

1.2.2 Основной алгоритм событийного имитационного моделирования

Существует два программных метода имитационного моделирования сложных систем [17]:

- дискретно-событийное;
- моделирования по принципу дельта t .

В данной бакалаврской работе был выбран метод дискретно-событийного моделирования т.к. он менее затратен по времени.

Дискретно-событийное моделирование (*discrete event modeling*) - подход к моделированию, предлагающий абстрагироваться от непрерывной природы событий и рассматривать процессы, протекающие в моделируемой системе, как последовательность отдельных событий.

В дискретно-событийном моделировании выделяются состояния системы и описываются действия, которые переводят ее из одного состояния в другое. Состояние системы может меняться только в моменты свершения событий. Так как состояние системы не изменяется между этими моментами, полный динамический портрет состояний системы может быть получен путем продвижения имитационного времени от одного события к другому. При дискретной имитации значения зависимых переменных изменяются дискретно в моменты совершения событий и, как правило, не изменяются в промежутках между моментами совершения событий.

Дискретно-событийное моделирование наиболее развито и имеет огромную сферу приложений – от логистики и систем массового обслуживания до транспортных и производственных систем. Этот вид моделирования наиболее подходит для моделирования производственных процессов. Неотъемлемым атрибутом дискретно-событийного моделирования является использование списка будущих событий (СБС).

В приведенных ниже алгоритмах имитационного моделирования использовались следующие параметры, переменные и структуры данных.

Деталь имеет следующие параметры:

- цвет детали (p_1);
- высота детали (p_2).

Каждый из этих параметров может иметь значения:

- годен;
- не годен.

В модели используются переменные состояния *Lift* и *Ready*. Переменная *Lift* имеет два значения:

- «лифт находится в нижнем положении» (0);
- «лифт находится в верхнем положении» (1).

Переменная *Ready* имеет два значения:

- «станция не готова принять новую деталь» (0);
- «станция готова принять новую деталь» (1).

В модели используются системные переменные *sysstime* (системное модельное время) и *limtime* (общее время моделирования).

Структура данных СОБЫТИЕ включает поля *eventID* (идентификатор события) и *time* (планируемое время совершения события).

Для упрощения моделирования был разработан граф событий и действий тестирующей станции, который представлен ниже на рисунке 8.

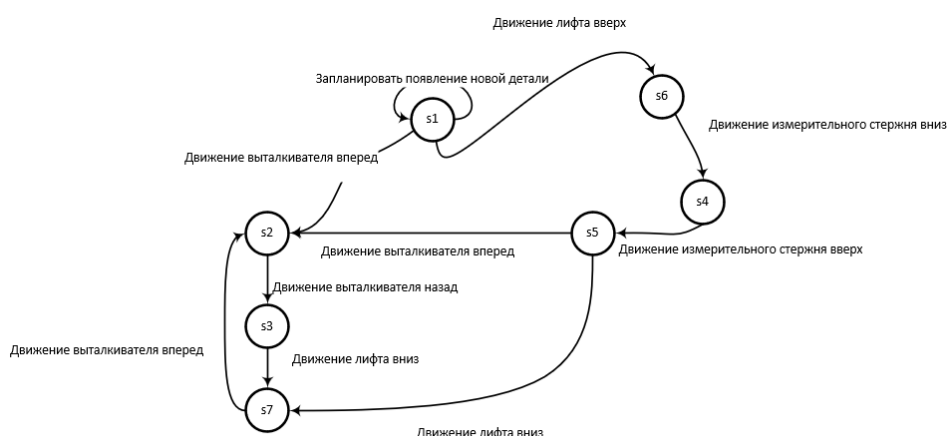


Рисунок 8 – Граф событий и действий тестирующей станции

На данном рисунке использовались следующие события:

- сгенерирована новая деталь (s1);
- выталкиватель достиг крайнего (выдвинутого) положения(s2);
- выталкиватель достиг крайнего (втянутого) положения(s3);
- измерительный стержень достиг крайнего (выдвинутого) положения(s4);
- измерительный стержень достиг крайнего (втянутого) положения(s5);

- лифт достиг крайнего верхнего положения(s_6);
- лифт достиг крайнего нижнего положения(s_7).

Ниже на рисунке 9 представлен общий алгоритм событийного моделирования тестирующей станции *FESTO*.

```

ПОКА (СБС не пуст AND  $simtime < limtime$ )
    ДЕЛАТЬ
        Выбрать первый элемент СОБЫТИЕ из СБС
         $systime = time$  – Скачок времени
        ЕСЛИ  $EventID = s_1$ , ТО Обработка события  $s_1$ 
        ИНАЧЕ ЕСЛИ  $EventID = s_2$ , ТО Обработка события  $s_2$ 
        ИНАЧЕ ЕСЛИ  $EventID = s_3$ , ТО Обработка события  $s_3$ 
        ИНАЧЕ ЕСЛИ  $EventID = s_4$ , ТО Обработка события  $s_4$ 
        ИНАЧЕ ЕСЛИ  $EventID = s_5$ , ТО Обработка события  $s_5$ 
        ИНАЧЕ ЕСЛИ  $EventID = s_6$ , ТО Обработка события  $s_6$ 
        ИНАЧЕ ЕСЛИ  $EventID = s_7$ , ТО Обработка события  $s_7$ 
    КОНЕЦ
КОНЕЦ_ПОКА

```

Рисунок 9 – Алгоритм событийного моделирования тестирующей станции *FESTO*

1.2.3 Алгоритмы обработки событий в имитационной модели тестирующей станции

Ниже на рисунках 10–16 представлены алгоритмы обработки событий, которые используются в общем алгоритме имитационного моделирования.

```

Сгенерировать структуру данных Деталь для новой детали
 $p_1 = random(0,1)$  – Цвет детали
 $p_2 = random(0,1)$  – Высота детали
ЕСЛИ станция не готова принять новую деталь ( $Ready=0$ ), ТО
    Поставить деталь во входную очередь деталей
ИНАЧЕ
    ДЕЛАТЬ
        Запланировать появление новой детали
        Начало обработки детали, которая связана с событием
    КОНЕЦ

```

Рисунок 10 – Алгоритм обработки события s_1 “Сгенерирована новая деталь”

Выдать сигнал S_END – Поршень достиг крайнего положения
 Выдать сигнал C_EXTEND_FALSE – Снять сигнал на выдвижение поршня вперед
 Выдать сигнал $C_RETRACT_TRUE$ – Подать сигнал на втягивание поршня назад
 Запустить движение Выталкивателя назад

Рисунок 11 – Алгоритм обработки события $s2$ “Выталкиватель достиг крайнего (выдвинутого) положения”

Выдача сигнала S_END – Поршень достиг крайнего положения
 Выдача сигнала $C_RETRACT_FALSE$ – Снять сигнал на втягивание поршня назад
 ЕСЛИ Лифт поднят ($Lift=1$)), ТО
 Выдача сигнала $C_LIFT_DOWN_TRUE$ – Подать сигнал опускания лифта
 Запустить опускание Лифта
 ИНАЧЕ – Лифт опущен
 ДЕЛАТЬ
 $Ready=1$
 ЕСЛИ Очередь деталей ПУСТА, то Пустой оператор
 ИНАЧЕ
 ДЕЛАТЬ
 Выбрать очередную деталь из очереди деталей
 Начало обработки новой детали
 КОНЕЦ
 КОНЕЦ

Рисунок 12 – Алгоритм обработки события $s3$ “Выталкиватель достиг крайнего (втянутого) положения”.

Выдача сигнала $S_MEASURER_DOWN$ – Измеритель находится в нижнем положении
 Выдача сигнала $C_MEASURER_DOWN_FALSE$ – Снять сигнал опускания измерителя
 Выдача сигнала $C_MEASURER_UP_TRUE$ – Подать сигнал поднятия измерителя
 Запустить движение измерительного стержня вверх

Рисунок 13 – Алгоритм обработки события $s4$ “Измерительный стержень достиг крайнего (выдвинутого) положения”

Выдача сигнала $S_MEASURER_UP$ – Измеритель находится в верхнем положении
 Выдача сигнала $C_MEASURER_UP_FALSE$ – Снять сигнал поднятия измерителя
 ЕСЛИ $p2=0$, ТО – Деталь не годна по высоте
 Выдача сигнала S_HEIGHT_FALSE – Деталь не подходит по высоте
 Выдача сигнала $C_LIFT_DOWN_TRUE$ – Подать сигнал опускания лифта
 Запустить опускание Лифта
 ИНАЧЕ – Деталь годна по высоте
 Выдача сигнала S_HEIGHT_TRUE – Деталь годна по высоте
 Выдача сигнала C_EXTEND_TRUE – Подать сигнал на выдвижение поршня вперед
 Запустить движение Выталкивателя вперед

Рисунок 14 – Алгоритм обработки события $s5$ “Измерительный стержень достиг крайнего (втянутого) положения”

Выдача сигнала S_LIFT_UP – Лифт находится в верхнем положении
 Выдача сигнала $C_LIFT_UP_FALSE$ – Снять сигнал поднятия лифта
 Выдача сигнала $C_MEASURER_DOWN_TRUE$ – Подать сигнал опускания лифта
 $Lift=1$
 Запустить движение измерительного стержня вниз

Рисунок 15 – Алгоритм обработки события $s6$ “Лифт достиг крайнего верхнего положения”

Выдача сигнала S_LIFT_DOWN – Лифт находится в нижнем положении
 Выдача сигнала $C_LIFT_DOWN_FALSE$ – Снять сигнал опускания лифта
 $Lift=0$
 ЕСЛИ $p2=0$, ТО – Деталь не годна по высоте
 Выдача сигнала S_HEIGHT_FALSE – Деталь не подходит по высоте
 Выдача сигнала C_EXTEND_TRUE – Подать сигнал на выдвижение поршня вперед
 Запустить движение Выталкивателя вперед
 ИНАЧЕ – Деталь была годна по высоте
 ДЕЛАТЬ – Цикл обработки годной детали закончен. Подать новую деталь
 $Ready=1$
 Попытка выбрать очередную деталь из очереди деталей
 ЕСЛИ Деталь выбрана успешно, ТО Начало обработки новой детали
 ИНАЧЕ – Ничего не делать (очередь деталей пуста)
 КОНЕЦ

Рисунок 16 – Алгоритм обработки события $s7$ “Лифт достиг крайнего нижнего положения”

1.2.3.1 Алгоритмы вспомогательных процедур

Ниже на рисунках 17–26 представлены алгоритмы процедур планирования новых событий различного вида, а также другие процедуры, необходимые для функционирования имитационной модели.

Вызов процедуры обработки события $s1$

Рисунок 17 – Алгоритм процедуры инициализации

$Ready=0$
ЕСЛИ $p1=0$, ТО – Деталь не годна по цвету
Запустить движение Выталкивателя вперед
ИНАЧЕ – Деталь годна по цвету
Запустить подъем Лифта)

Рисунок 18 – Алгоритм процедуры «Начало обработки новой детали»

$ta=random(a, b)$ – интервал времени между появлением деталей
 $t= systime+ta$ – Планируемое время совершения события
 $schedule(s1, t)$ – Запланировать событие $s1$ на время t (создать элемент СБС, заполнить поля элемента $Node$, включить элемент $Node$ в список будущих событий в хронологическом порядке).

Рисунок 19 – Алгоритм процедуры «Запланировать появление новой детали»

Выдать управляющий сигнал движения выталкивателя вперед C_EXTEND_TRUE
 $ta=random(a, b)$ – Время движения поршня выталкивателя
 $t= systime+ta$ – Планируемое время совершения события
 $schedule(s2, t)$ – Запланировать событие $s2$ на время t .

Рисунок 20 – Алгоритм процедуры «Запустить движение Выталкивателя вперед»

$ta=random(a, b)$ – Время движения поршня выталкивателя
 $t= systime+ta$ – Планируемое время совершения события
 $schedule(s3, t)$ – Запланировать событие $s3$ на время t .

Рисунок 21 – Алгоритм процедуры «Запустить движение Выталкивателя назад»

$ta=random(a, b)$ – Время движения измерительного стержня
 $t= systime+ta$ – Планируемое время совершения события
 $schedule(s4, t)$ – Запланировать событие $s4$ на время t .

Рисунок 22 – Алгоритм процедуры «Запустить движение измерительного стержня вниз»

$ta = \text{random}(a, b)$ – Время движения измерительного стержня
 $t = \text{sys\textit{time}} + ta$ – Планируемое время совершения события
 $\text{schedule}(s5, t)$ – Запланировать событие $s5$ на время t .

Рисунок 23 – Алгоритм процедуры «Запустить движение измерительного стержня вверх»

$ta = \text{random}(a, b)$ – Время движения лифта
 Выдать управляющий сигнал поднятия лифта $C_LIFT_UP_TRUE$
 $t = \text{sys\textit{time}} + ta$ – Планируемое время совершения события
 $\text{schedule}(s6, t)$ – Запланировать событие $s6$ на время t .

Рисунок 24 – Алгоритм процедуры «Запустить движение лифта вверх»

$ta = \text{random}(a, b)$ – Время движения лифта
 $t = \text{sys\textit{time}} + ta$ – Планируемое время совершения события
 $\text{schedule}(s7, t)$ – Запланировать событие $s7$ на время t .

Рисунок 25 – Алгоритм процедуры «Запустить движение лифта вниз»

Создать элемент списка будущих событий (СБС) $Node$
 Заполнить поля элемента $Node$
 $Node.eventID = s$
 $Node.time = t$
 Включить элемент $Node$ в список будущих событий (СБС) в хронологическом порядке

Рисунок 26 – Алгоритм процедуры планирования нового события с интерфейсом $\text{schedule}(s, t)$

1.2.4 Описание программы

Ниже представлено краткое описание программы имитационного моделирования тестирующей станции *FESTO* включая описание переменных, структур данных, а также процедур функций.

В программе используются следующие переменные:

- *bool lift* – состояние лифта (по умолчанию лифт опущен);
- *bool SGV* – состояние выталкивателя (по умолчанию выталкиватель находится в задвинутом положении);
- *bool TS* – состояние тестирующей станции (по умолчанию тестирующая станция свободна);

- *bool msr* – состояние измерительного стержня (по умолчанию измерительный стержень находится в верхнем положении);
- *int systime* – системное время (по умолчанию равно 0);
- *int lim* – лимит системного времени (по умолчанию равно 1000).

Переменная *lift* может иметь следующие значения:

- лифт опущен (0);
- лифт поднят (1).

Переменная *SGV* может иметь следующие значения:

- выталкиватель находится в задвинутом положении (0);
- выталкиватель находится в выдвинутом положении (1).

Переменная *TS* может иметь следующие значения:

- тестирующая станция свободна (1);
- тестирующая станция занята (0).

Переменная *msr* может иметь следующие значения:

- измерительный стержень находится в верхнем положении (1);
- измерительный стержень находится в нижнем положении (0).

1.2.4.1 Структуры данных программы представлены ниже.

Структура данных *Node*, представляющая элемент списка будущих событий, имеет следующие поля:

- *int time* (запланированное время совершения события);
- *string event_type* (тип события);
- *string event_id* (наименование события);
- *string device_id* (наименование устройства тестирующей станции);
- *struct Node *next* (указатель на следующий элемент списка).

Структура данных *Detal*, представляющая деталь, имеет следующие поля:

- *bool height* (высота детали);
- *bool color* (цвет детали).

Структура данных *Queue* для представления элемента очереди деталей, имеет следующие поля:

- *struct Detal* detale* (указатель на структуру, описывающую деталь);
- *struct Queue *next* (указатель на следующий элемент очереди).

1.2.4.2 Процедуры программы представлены ниже.

Процедура *void generate (Detal** detal)* предназначена для генерации новой детали с параметрами *height* и *color*. Параметры генерируются случайным образом (0 или 1). Процедура имеет один входной параметр *Detal** detal* – указатель на структуру, представляющую деталь.

Процедура *void insert_queue (Queue** head, Queue** final, Detal* k)* предназначена для занесения детали в очередь, имеет следующие поля:

- *Queue** head* (указатель на начало очереди);
- *Queue** final* (указатель на конец очереди);
- *Detal* k* (указатель на структуру, представляющую деталь, которая заносится в очередь).

Процедура *Detal* take_queue (Queue** head, Queue** final)* предназначена для выборки детали из очереди, имеет следующие поля:

- *Queue** head* (указатель на начало очереди);
- *Queue** final* (указатель на конец очереди).

Процедура *void insert (Node** head, Node* t)* предназначена для занесения структуры данных, представляющей событие, в список будущих событий, имеет следующие поля:

- *Node** head* (указатель на начало списка будущих событий);
- *Node* t* (указатель на структуру данных, представляющей событие).

Процедура *void action (string s)* предназначена для вывода данных о событии в выходной файл (журнал событий). Процедура имеет один входной параметр *string s* – имя сигнала.

Список других процедур программы представлен ниже:

- *SGV_go_forward* (процедура «Запуск выталкивателя вперед»);
- *SGV_go_back* (процедура «Запуск выталкивателя назад»);
- *lift_go_up* (процедура «Запуск движения лифта в верх»);

- *lift_go_down* (процедура «Запуск движения лифта в вниз»);
- *msr_go_down* (процедура «Запуск движения проверочного стржня вниз»);
- *msr_go_up* (процедура «Запуск движения проверочного стржня вверх»);
- *shedule_new_detal* (процедура «Запланировать появление новой детали»);
- *new_detal_proc_begin* (процедура «Начало обработки детали»);
- процедура *s1()* – процедура обработки события *s1* “Сгенерирована новая деталь”;
- процедура *s2()* – процедура обработки события *s2* “Выталкиватель достиг крайнего (выдвинутого) положения”;
- процедура *s3()* – процедура обработки события *s3* “Выталкиватель достиг крайнего (втянутого) положения”;
- процедура *s4()* – процедура обработки события *s4* “Измерительный стержень достиг крайнего (выдвинутого) положения”;
- процедура *s5()* – процедура обработки события *s5* “Измерительный стержень достиг крайнего (втянутого) положения”;
- процедура *s6()* – процедура обработки события *s6* “Лифт достиг крайнего верхнего положения”;
- процедура *s7()* – процедура обработки события *s7* “Лифт достиг крайнего нижнего положения”.

1.2.5 Тестирование программы

Тестирование программы производилось в разных режимах работы, для этого использовались следующие параметры: *height* – высота детали, *color* – цвет детали.

Для тестирования сценария «Плохой цвет» использовались следующие значения параметров: *height* = 1, *color* = 1.

Для тестирования сценария «Хороший цвет, хорошая высота» использовались следующие значения параметров: *height* = 1, *color* = 1.

Для тестирования сценария «Хороший цвет, плохая высота» использовались следующие значения параметров: $height = 0$, $color = 1$.

Для комплексного тестирования использовались случайные значения параметров из диапазона $[0, 1]$: $height = rand() \% 2$, $color = rand() \% 2$.

Скриншот комплексного тестирования программы со случайными значениями представлен на рисунке 27. На данном рисунке приведена последовательность диагностических сообщений, описывающая ход выполнения программы.

```

2 Запуск движения лифта в верх s6 Measurer
6 Запуск движения проверочного стрелки вниз s4 Measurer
7 Запуск движения проверочного стрелки вверх s5 Measurer
12 Запуск выталкивателя вперед s2 Slidegate valve
17 Запуск выталкивателя назад s3 Slidegate valve
24 Запуск движения лифта в вниз s7 Lift
107 Сгенерирована новая деталь s1 Application Generator
115 Запуск движения лифта в верх s6 Measurer
120 Запуск движения проверочного стрелки вниз s4 Measurer
126 Запуск движения проверочного стрелки вверх s5 Measurer
135 Запуск движения лифта в вниз s7 Lift
138 Запуск выталкивателя вперед s2 Slidegate valve
146 Запуск выталкивателя назад s3 Slidegate valve
216 Сгенерирована новая деталь s1 Application Generator
221 Запуск движения лифта в верх s6 Measurer
230 Запуск движения проверочного стрелки вниз s4 Measurer
240 Запуск движения проверочного стрелки вверх s5 Measurer
248 Запуск выталкивателя вперед s2 Slidegate valve
255 Запуск выталкивателя назад s3 Slidegate valve
259 Запуск движения лифта в вниз s7 Lift
322 Сгенерирована новая деталь s1 Application Generator
330 Запуск движения лифта в верх s6 Measurer
333 Запуск движения проверочного стрелки вниз s4 Measurer
335 Запуск движения проверочного стрелки вверх s5 Measurer
342 Запуск движения лифта в вниз s7 Lift
345 Запуск выталкивателя вперед s2 Slidegate valve
349 Запуск выталкивателя назад s3 Slidegate valve
427 Сгенерирована новая деталь s1 Application Generator
433 Запуск выталкивателя вперед s2 Slidegate valve
442 Запуск выталкивателя назад s3 Slidegate valve
529 Сгенерирована новая деталь s1 Application Generator
536 Запуск выталкивателя вперед s2 Slidegate valve
544 Запуск выталкивателя назад s3 Slidegate valve
639 Сгенерирована новая деталь s1 Application Generator
647 Запуск движения лифта в верх s6 Measurer
654 Запуск движения проверочного стрелки вниз s4 Measurer
658 Запуск движения проверочного стрелки вверх s5 Measurer
661 Запуск выталкивателя вперед s2 Slidegate valve
664 Запуск выталкивателя назад s3 Slidegate valve
674 Запуск движения лифта в вниз s7 Lift
740 Сгенерирована новая деталь s1 Application Generator
746 Запуск движения лифта в верх s6 Measurer
749 Запуск движения проверочного стрелки вниз s4 Measurer
754 Запуск движения проверочного стрелки вверх s5 Measurer
760 Запуск выталкивателя вперед s2 Slidegate valve
763 Запуск выталкивателя назад s3 Slidegate valve
772 Запуск движения лифта в вниз s7 Lift
844 Сгенерирована новая деталь s1 Application Generator
851 Запуск выталкивателя вперед s2 Slidegate valve
855 Запуск выталкивателя назад s3 Slidegate valve
954 Сгенерирована новая деталь s1 Application Generator
962 Запуск выталкивателя вперед s2 Slidegate valve
972 Запуск выталкивателя назад s3 Slidegate valve
1062 Сгенерирована новая деталь s1 Application Generator
1066 Запуск движения лифта в верх s6 Lift
1168 Сгенерирована новая деталь s1 Application Generator

```

Рисунок 27 – Скриншот выполнения программы для комплексного тестирования

1.3 Генерация формальных моделей процессов тестирующей станции *Festo* на основе метода *process mining*

1.3.1 Использование метода *Process Mining* в проектировании

Process Mining – это совокупность методов, методик и инструментов для обнаружения, наблюдения и улучшения реальных процессов путем извлечения знаний из журналов событий, обычно доступных в информационных и управляющих системах [18]. В русскоязычной литературе *Process Mining* часто называют процессной аналитикой [19]. Это довольно молодое научное направление, возникшее на стыке нескольких направлений в информатике.

Методы *Process Mining* подразделяют на три блока:

- обнаружение (*discovery*): модель процесса строится на основании журнала событий, содержащего запись функционирования информационной системы за некоторый промежуток времени;
- проверка соответствия (*conformance checking*): существующая модель сравнивается с журналом событий, анализируются выявленные несоответствия в поведении реальной системы и моделируемом поведении;
- усовершенствование (*enhancement*): существующая модель усовершенствуется путём расширения моделируемого поведения или повышения эффективности моделирования с использованием информации, полученной в ходе проверки соответствия модели и журнала событий.

Основным форматом для представления журналов событий является XML-базированный формат XES. Однако допустимыми являются и другие форматы, в частности, тестовый формат CSV.

В настоящее время существует достаточно большое число инструментальных средств (как бесплатных, так и коммерческих), поддерживающих технологию *Process mining*. В данной бакалаврской работе используется бесплатное программное средство ProM 6.11 [20], разработанное в Эндховенском университете (Нидерланды) под руководством основоположника технологии *Process Mining* профессора Ван дер Аальста.

На рисунке 28 представлен технологический маршрут использования процессной аналитики в данной выпускной квалификационной работе бакалавра.

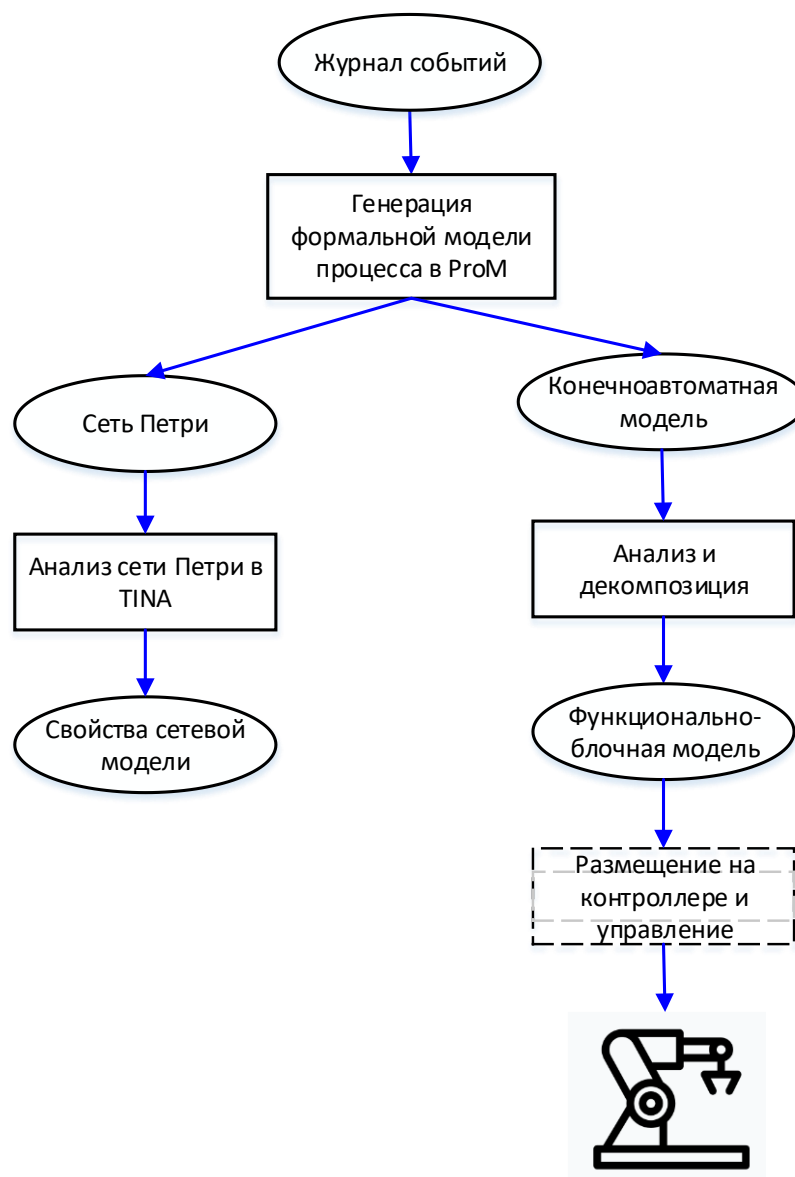


Рисунок 28 – Последовательность работ *Process mining* для проектирования тестирующей станции *FESTO*

Как видно из данного рисунка, стартовой точкой проектирования является журнал событий, описывающий функционирование тестирующей станции системы *FESTO* и полученный в ходе проведения экспериментов с

имитационной моделью тестирующей станции. Данный журнал генерируется в текстовом формате CSV и имеет следующую структуру:

- *processId* – идентификатор варианта (экземпляра) процесса. В нашем случае используется один единственный вариант, поскольку описываемый процесс носит непрерывный циклический характер;
- *timestamp* – временная метка, определяющая время возникновения события. В данном случае используется модельное время (а не реальное);
- *component* – устройство тестирующей станции, с которым связано возникшее событие;
- *action* – идентификатор (имя) сигнала, который ассоциируется с данным событием.

Фрагмент журнала событий в формате CSV приведен на рисунке 29.

```
ProcessId, Timestamp, Component, Action
...
1, 954, Slidegate valve, C_EXTEND_FALSE
1, 954, Slidegate valve, C_RETRACT_TRUE
1, 963, Slidegate valve, S_END
1, 963, Slidegate valve, C_RETRACT_FALSE
2, 0, Lift, C_LIFT_UP_TRUE
2, 1, Lift, S_LIFT_UP
2, 1, Lift, C_LIFT_UP_FALSE
2, 1, Measurer, C_MEASURER_DOWN_TRUE
2, 6, Measurer, S_MEASURER_DOWN
2, 6, Measurer, C_MEASURER_DOWN_FALSE
....
```

Рисунок 29 – Фрагмент журнала событий для тестирующей станции в формате CSV

Генерация формальных моделей производится в системе ProM 6.11. Для этого используются могут использоваться следующие майнеры: *Alpha miner*, *ILP miner* и *Inductive miner* (для сетей Петри) *Transition system miner* (для конечноавтоматной модели) и другие майнеры.

1.3.2 Генерация формальных моделей процессов на основе журналов событий

Для построения формальных моделей в системе *ProM* с помощью программной имитационной модели тестирующей станции *FESTO* был сгенерирован журнал событий, содержащий три реализации процесса (три прогона) с идентификаторами процессов (*ProcessId*), равными 1,2,3. Каждая реализация процесса содержала по 1000 записей (событий).

Первой формальной моделью, которая была сгенерирована в системе *ProM*, был граф причинно-следственных зависимостей событий (иначе, сигналов) (рисунок 30). Для генерации данной модели использовался майнер *Directly follows model*.

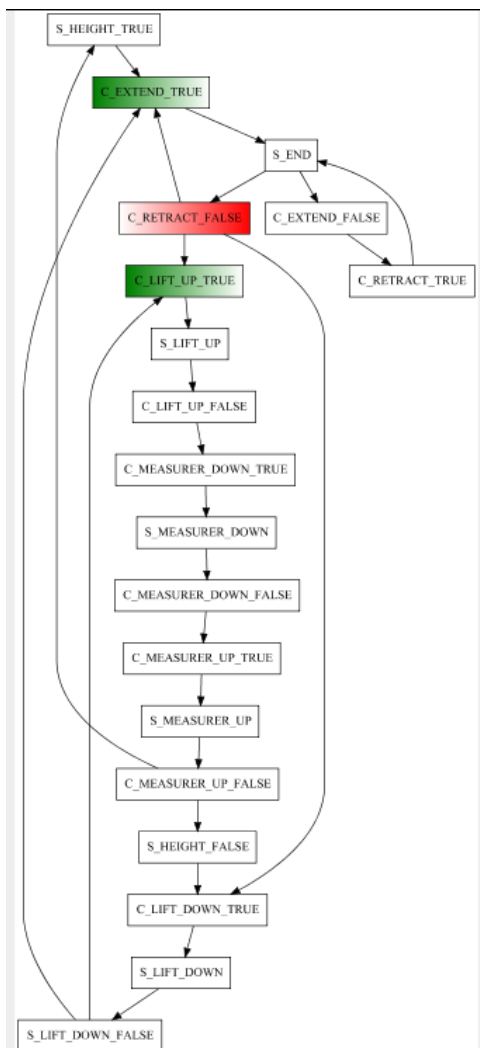


Рисунок 30 – Граф причинно-следственных зависимостей событий (сигналов)

На данном графе зеленым цветом отмечены события, с которых начинаются реализации процессов, а красным цветом отмечены финальные события данных цепочек. Приведенный граф позволяет лучше понять техпроцесс обработки деталей, а также визуально проверить его на корректность (вручную).

В тестирующей станции *FESTO* существует три сценария обработки деталей (см. рисунки 31–33), и каждый из этих сценариев был проверен вручную на формальных моделях.

C_EXTEND_TRUE
S_END
C_EXTEND_FALSE
C_RETRACT_TRUE
S_END
C_RETRACT_FALSE

Рисунок 31 – Последовательность сигналов для сценария «Плохой цвет»

C_LIFT_UP_TRUE
S_LIFT_UP
C_LIFT_UP_FALSE
C_MEASURER_DOWN_TRUE
S_MEASURER_DOWN
C_MEASURER_DOWN_FALSE
C_MEASURER_UP_TRUE
S_MEASURER_UP
C_MEASURER_UP_FALSE
S_HEIGHT_TRUE
C_EXTEND_TRUE
S_END
C_EXTEND_FALSE
C_RETRACT_TRUE
S_END
C_RETRACT_FALSE
C_LIFT_DOWN_TRUE
S_LIFT_DOWN
S_LIFT_DOWN_FALSE

Рисунок 32 – Последовательность сигналов для сценария «Хороший цвет, хорошая высота»

```

C_LIFT_UP_TRUE
S_LIFT_UP
C_LIFT_UP_FALSE
C_MEASURER_DOWN_TRUE
S_MEASURER_DOWN
C_MEASURER_DOWN_FALSE
C_MEASURER_UP_TRUE
S_MEASURER_UP
C_MEASURER_UP_FALSE
S_HEIGHT_FALSE
C_LIFT_DOWN_TRUE
S_LIFT_DOWN
S_LIFT_DOWN_FALSE
C_EXTEND_TRUE
S_END
C_EXTEND_FALSE
S_END
C_RETRACT_FALSE

```

Рисунок 33 – Последовательность сигналов для сценария «Хороший цвет, плохая высота»

Второй формальной моделью, которая была сгенерирована в системе *ProM*, была конечно-автоматная модель (типа *Transition System*), построенная с помощью соответствующего майнера. Данная формальная модель описывает динамику поведения всей системы (на уровне входных и выходных сигналов), включая оборудование и управление.

Данная модель может быть сохранена в *XML*-базируемом формате *TSML*, а затем с использованием специального конвертера [21] преобразована в *GraphML*-формат, который может быть открыт в редакторах графов (например, *yEd*) для визуализации и редактирования. В дальнейшем *GraphML*-описание может быть сконвертировано в функционально-блочные модели оборудования [22] и мониторы последовательностей событий [23]. В нашем случае конечно-автоматная модель тестирующей станции использовалась для построения конечно-автоматной модели системы управления, в именно: автомата Мура (в ручном режиме). Для преобразования использовались специальные правила. При этом сигналы с датчиков (начинающиеся на *S_*) принимались в качестве входных

сигналов, а управляющие сигналы (начинающиеся на С_) принимались как Выходные сигналы.

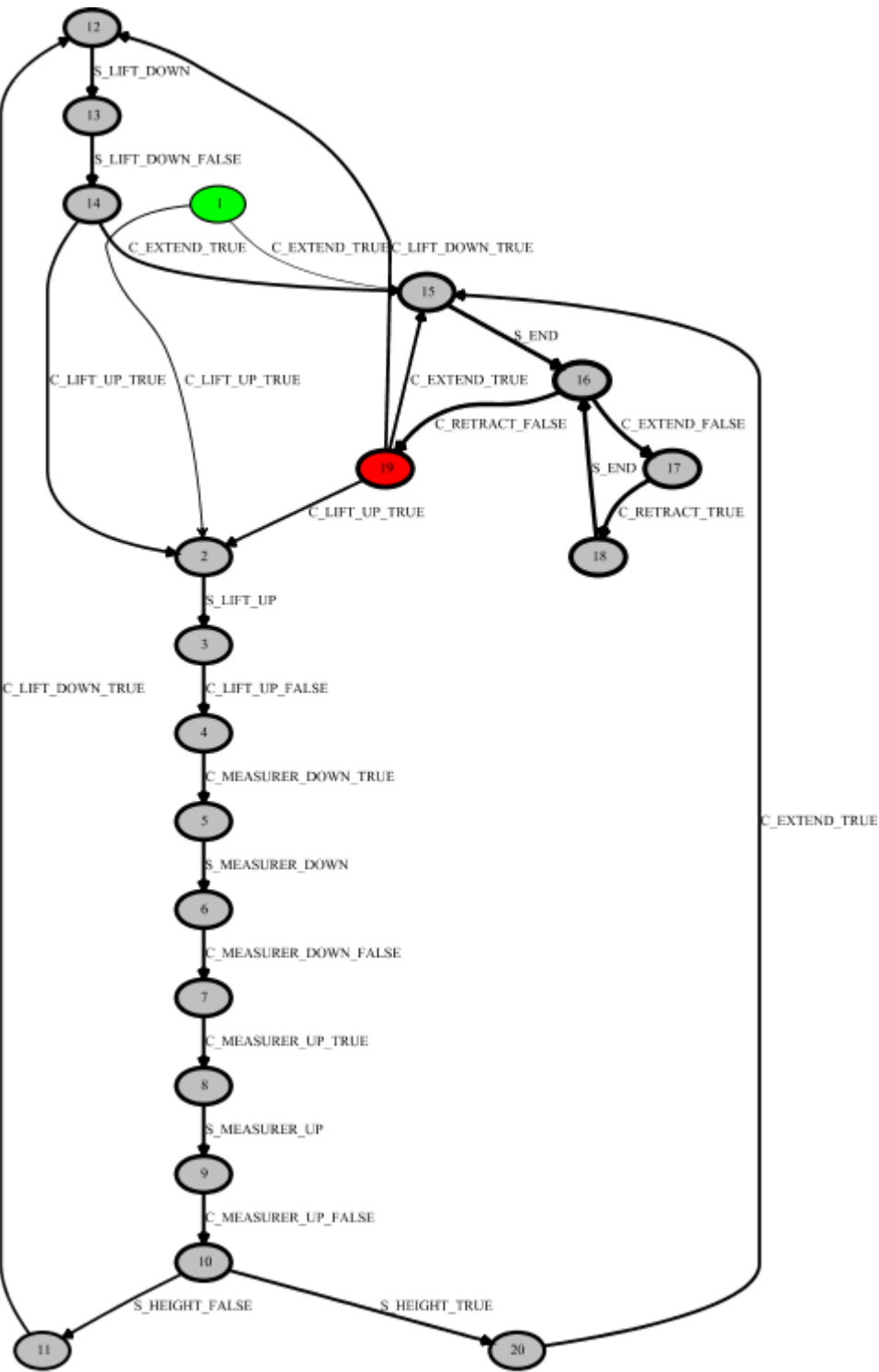


Рисунок 34 – Конечно-автоматная модель тестирующей станции

Третьей формальной моделью, которая была сгенерирована в системе *ProM*, была сетевая модель (сеть Петри). Для ее построения использовался индуктивный майнер (*Inductive miner*). Внешний вид сгенерированной сети Петри представлен на рисунке 35. Данная сетевая модель использовалась для анализа и исследования.

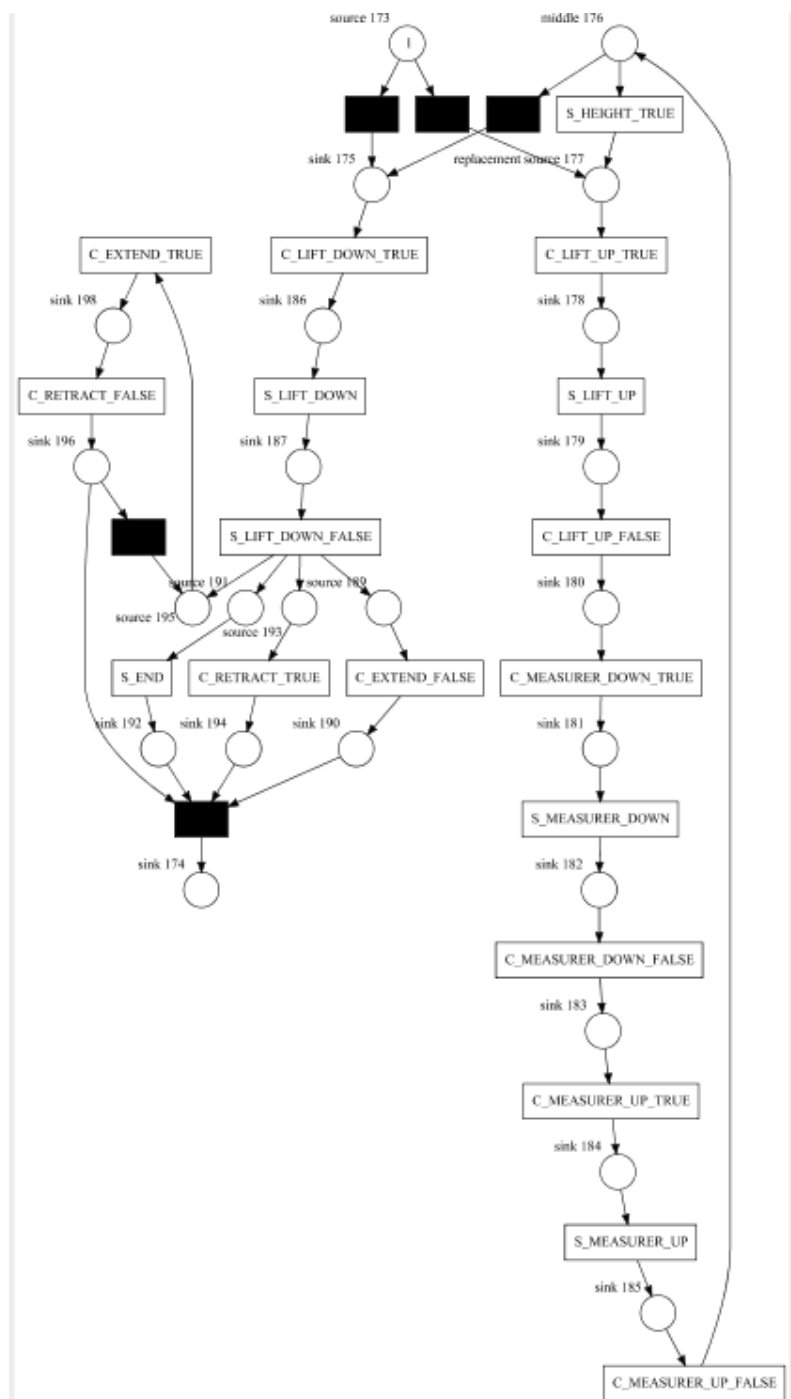


Рисунок 35 – Сгенерированная сетевая модель тестирующей станции в системе *ProM*

1.3.3 Анализ сетевых моделей процессов в системе *TINA*

Для анализа сетевой модели в данной работе была выбрана система *TINA* [24]. Выбор этой системы был обусловлен следующими причинами:

- возможность работать с форматом *PNML*, именно в этом формате система *ProM* генерирует сетевую модель;
- наличие графического редактора;
- развитые средства анализа сетевых моделей.

Сетевые модели, сгенерированные майнерами системы *ProM*, в их исходном виде малоприспособлены для анализа в системах моделирования сетей Петри (поскольку цель генерации была иной, скорее *Conformance checking*) и для целей анализа требуют определенной доводки. В частности, требуется корректировка структуры сетевой модели, чтобы:

- обеспечить начальные условия (установка начальной маркировки);
- избежать бесконечного накопления меток в определенных позициях;
- обеспечить условия для срабатывания некоторых переходов.

На рисунке 36 представлена скорректированная сетевая модель в системе *TINA*.

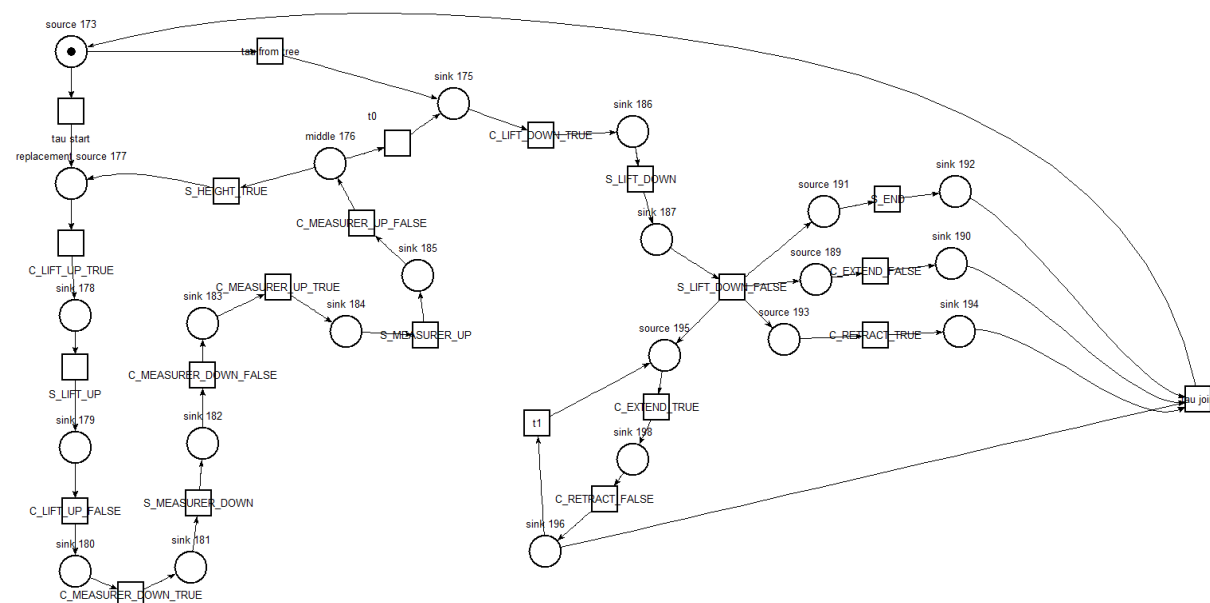


Рисунок 36 – Скорректированная сетевая модель тестирующей станции в системе *TINA*

Из корректировок были выполнены следующие действия:

- слита начальная позиция сети Петри с конечной позицией;
- скорректированы конфигурации (входные и выходные дуги) спонтанных переходов (эти переходы представлены в системе *ProM* в виде прямоугольников со сплошной черной заливкой). Дело в том, что при открытии *PNML*-файла, сгенерированного в *ProM*, система *TINA* слила два спонтанных перехода в один.

Первым использованным методом анализа сетевой модели было ее имитационное моделирование с использованием встроенного симулятора сетевой модели. Использовался пошаговый режим, когда пользователь может интерактивно активировать любой разрешенный переход. Данные переходы окрашены в красный цвет.

Вторым использованным методом анализа сетевой модели был анализ пространства состояний. Его результат в системе *TINA* приведен на рисунке 37.

digest

places

23

transitions

23

net

bounded

Y

live

?

reversible

?

abstraction

count

props

psets

dead

live

states

38

23

38

?

?

transitions

77

23

23

?

?

help

```

props {sink 192} {sink 194} {sink 196} {source 189}
trans t1/20 C_EXTEND_FALSE/35

state 34
props {sink 184}
trans S_MEASURER_UP/36

state 35
props {sink 190} {sink 192} {sink 194} {sink 196}
trans {tau join}/0 t1/25

state 36
props {sink 185}
trans C_MEASURER_UP_FALSE/37

state 37
props {middle 176}
trans t0/1 S_HEIGHT_TRUE/2

```

Рисунок 37 – Результаты анализа сетевой модели, связанный с пространством состояний

Из приведенного скриншота можно сделать следующие выводы. Граф достижимости является конечным и содержит 38 состояний и 77 переходов. Сетевая модель является ограниченной (*bounded*), то есть, в любом состоянии маркировка позиций сетевой модели не превосходит 1.

Сетевая модель не является живой и реверсивной. Это связано с тем, что в сети имеется цикл, содержащий позиции *source_195*, *sink_196*, *sink_198*, по которому фишка может ходить бесконечно долго (но выход из этого цикла есть). Таким образом, переходы, входящие в сеть, можно отнести лишь к потенциально-живым (но не безусловно живым). Из-за этого же цикла сеть не является реверсивной, то есть, не всегда сеть может вернуться в начальное состояние. Наличие этого цикла объясняется особенностями майнинга.

Третьим использованным методом анализа сетевой модели был структурный анализ. В рамках этого анализа производилось вычисление полупотоков (*semiflows*) (иначе, инвариантов). Результаты этого структурного анализа представлены на рисунок 38.

P-SEMI-FLOWS GENERATING SET ----- <i>invariant</i> {middle 176} {replacement source 177} {sink 175} {sink 178} {sink 179} {sink 180} {sink 181} {sink 182} {sink 183} {sink 184} {sink 185} {sink 186} {sink 187} {sink 192} {source 173} {source 191} (1) {middle 176} {replacement source 177} {sink 175} {sink 178} {sink 179} {sink 180} {sink 181} {sink 182} {sink 183} {sink 184} {sink 185} {sink 186} {sink 187} {sink 194} {source 173} {source 193} (1) {middle 176} {replacement source 177} {sink 175} {sink 178} {sink 179} {sink 180} {sink 181} {sink 182} {sink 183} {sink 184} {sink 185} {sink 186} {sink 187} {sink 196} {sink 198} {source 173} {source 195} (1) {middle 176} {replacement source 177} {sink 175} {sink 178} {sink 179} {sink 180} {sink 181} {sink 182} {sink 183} {sink 184} {sink 185} {sink 186} {sink 187} {sink 190} {source 173} {source 189} (1)
T-SEMI-FLOWS GENERATING SET ----- <i>consistent</i> C_LIFT_UP_FALSE C_LIFT_UP_TRUE C_MEASURER_DOWN_FALSE C_MEASURER_DOWN_TRUE C_MEASURER_UP_FALSE C_MEASURER_UP_TRUE S_HEIGHT_TRUE S_LIFT_UP S_MEASURER_DOWN S_MEASURER_UP C_EXTEND_TRUE C_RETRACT_FALSE t1 C_EXTEND_FALSE C_EXTEND_TRUE C_LIFT_DOWN_TRUE C_LIFT_UP_FALSE C_LIFT_UP_TRUE C_MEASURER_DOWN_FALSE C_MEASURER_DOWN_TRUE C_MEASURER_UP_FALSE C_MEASURER_UP_TRUE C_RETRACT_FALSE C_RETRACT_TRUE S_END S_LIFT_DOWN S_LIFT_DOWN_FALSE S_LIFT_UP S_MEASURER_DOWN S_MEASURER_UP t0 {tau join} {tau start} C_EXTEND_FALSE C_EXTEND_TRUE C_LIFT_DOWN_TRUE C_RETRACT_FALSE C_RETRACT_TRUE S_END S_LIFT_DOWN S_LIFT_DOWN_FALSE {tau from tree} {tau join}

Рисунок 38 – Результаты структурного анализа сетевой модели

Суть *P*-инварианта заключается в том, что взвешенная сумма значений маркировок некоторых позиций является величиной постоянной для всех состояний системы. Суть *T*-инварианта заключается в том, срабатывание переходов, входящих в *T*-инвариант, приводит сеть к той же самой маркировке. Беступиковая сеть обязательно имеет хотя бы один *T*-инвариант. Следовательно, наша сетевая модель свободна от тупиков.

1.4 Разработка распределенной функционально-блочной модели тестирующей станции системы *FESTO*

1.4.1 MVC-паттерн проектирования

Model-View-Controller (MVC, «Модель-Представление-Контроллер», «Модель-Вид-Контроллер») (рисунок 39) — схема разделения данных приложения и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо:

- модель (*Model*) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние;
- представление (*View*) отвечает за отображение данных модели пользователю, реагируя на изменения модели;
- контроллер (*Controller*) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

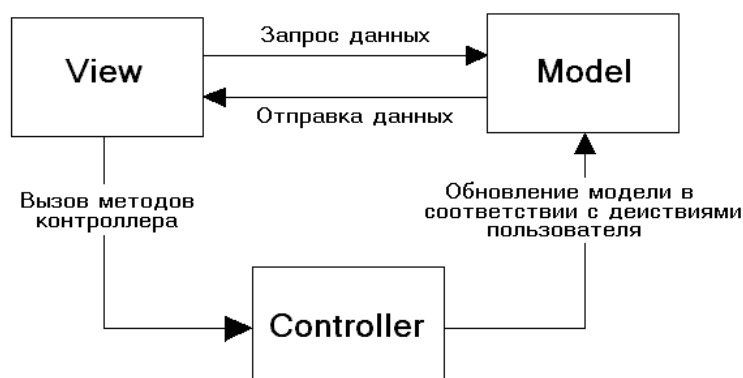


Рисунок 39 – Схема шаблона MVC

1.4.2 Модели переходов состояний сценариев обработки деталей

Состояния тестирующей станции перечислены ниже:

- состояние s0. Лифт опущен, Поршень втянут, Стержень поднят (0, 0, 0);
- состояние s1. Лифт опущен, Поршень выдвинут, Стержень поднят (0, 1, 0);
- состояние s2. Лифт поднят, Поршень втянут, Стержень поднят (1, 0, 0);
- состояние s3. Лифт поднят, Поршень втянут, Стержень опущен (1, 0, 1);
- состояние s4. Лифт поднят, Поршень выдвинут, Стержень поднят (1, 1, 0).

Диаграммы состояний сценариев представлены на рисунках 40–42.

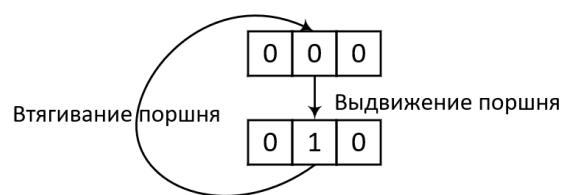


Рисунок 40 – Диаграмма состояний, описывающая сценарий поведения «плохой цвет детали»

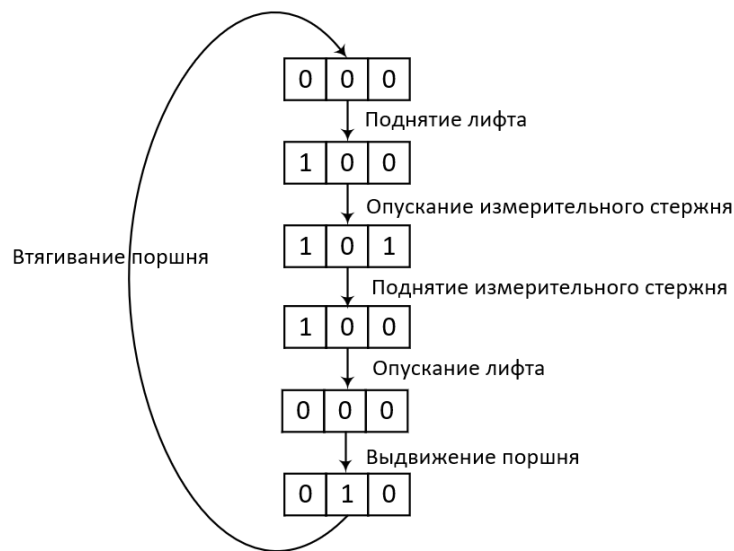


Рисунок 41 – Диаграмма состояний, описывающая сценарий поведения «хороший цвет, плохая высота»

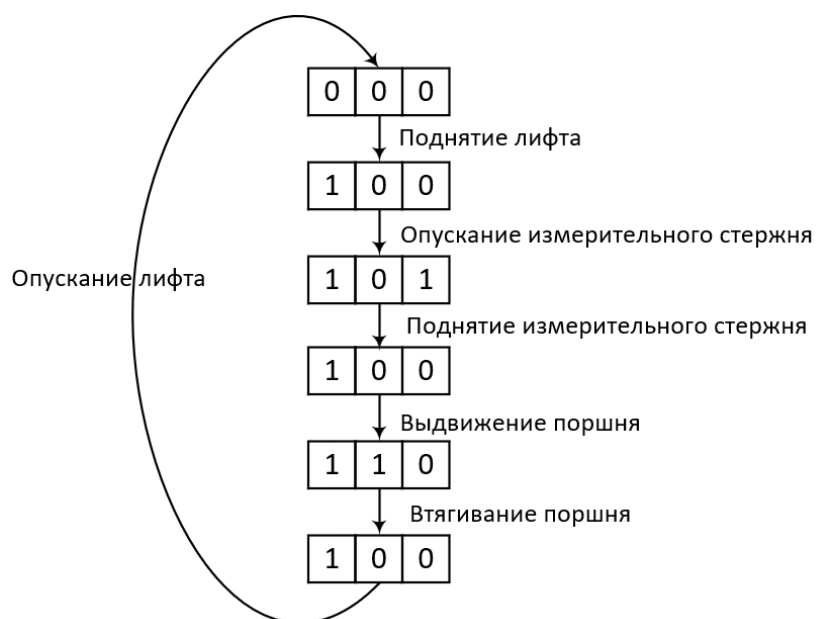


Рисунок 42 – Диаграмма состояний, описывающая сценарий поведения
«хороший цвет, хорошая высота»

1.4.3 Системная конфигурация

Система, представляющая имитационную модель ТС, включает в себя три виртуальных (логических) устройства, определенных по стандарту *IEC 61499* (рисунок 43). Все три устройства относятся к типу *RMTDEV_V1*, определенному в *nxtSTUDIO*.

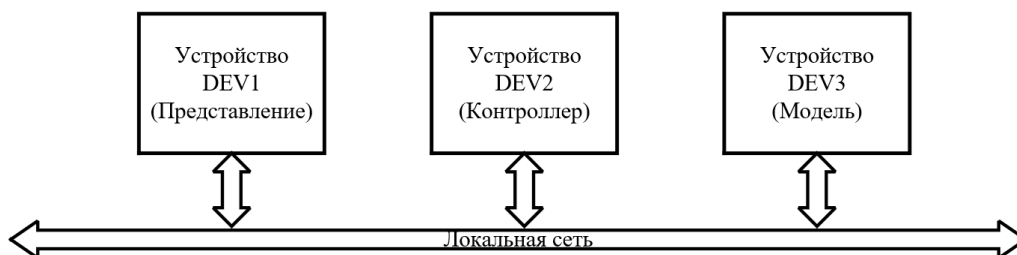


Рисунок 43 – Распределенная система для моделирования ТС

Каждое устройство включает в себя два ресурса:

- ресурс *MGR* типа *RMT_RES*;
- ресурс *RES0* типа *EMB_RES_ENH*.

Ресурс *MGR* является системным и предназначен для диспетчирования выполнения функциональных блоков, расположенных на данном ресурсе. Ресурс *RES0* является пользовательским. Его содержимое определяется исходя из логики функционирования системы.

Устройство типа *RMTDEV_V1* содержит два ресурса: *MGR* типа *RMT_RES* и *RES0* типа *EMB_RES_ENH*. Ресурс *MGR* является системным и используется для управления выполнением функциональных блоков на ресурсе. Ресурс *RES0* является пользовательским и на нем располагаются функциональные блоки, разработанные пользователем.

В качестве примера содержимое ресурса *RES0* в устройства *DEV3* представлено на рисунке 44.

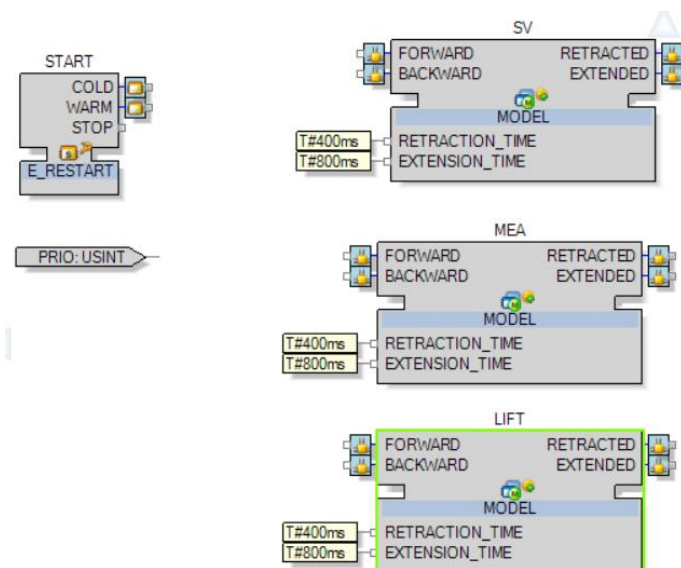


Рисунок 44 – Содержимое ресурса *RES0* устройства *DEV3*

Стандартный (библиотечный) ФБ *E_RESTART* предназначен для «холодного» (первичного) и «теплого» (вторичного) запуска системы.

1.4.4 Управляющее приложение

Управляющее функционально-блочное приложение приведено на рисунке 45. Оно включает в себя три компонента: модель (ФБ *SV*, *LIFT*, *MEA* типа

MODEL), контроллер (ФБ FB типа CONTROLLER_N), представление (ФБ VIEW типа VIEW).

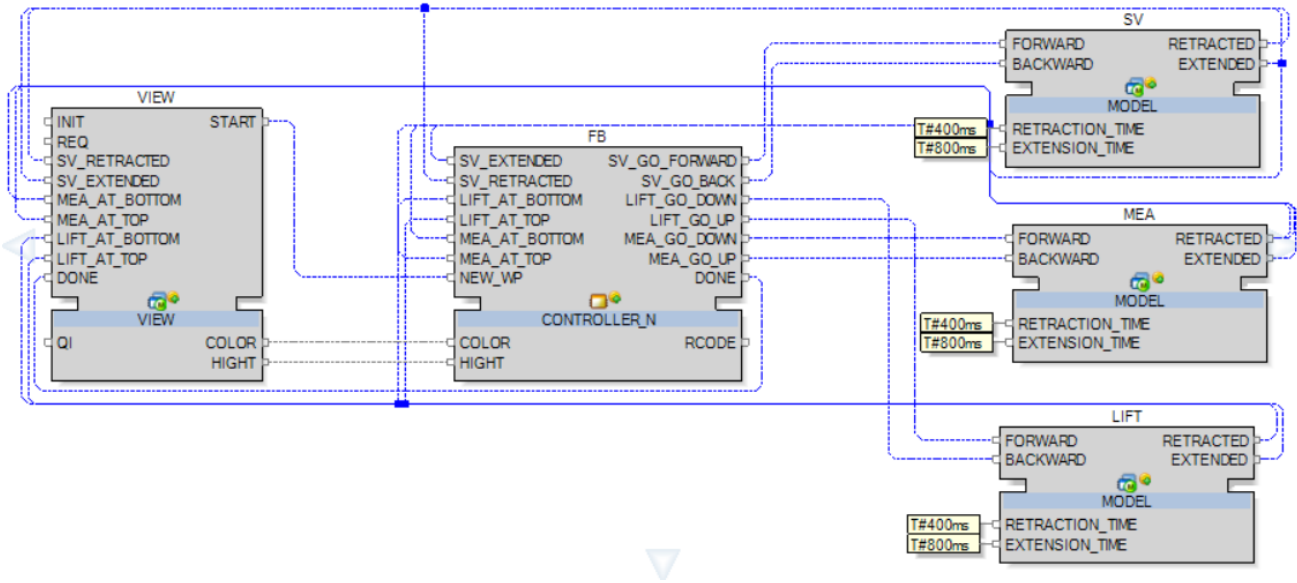


Рисунок 45 – Управляющее приложение

Диаграмма классов *UML* для управляющего приложения представлена на рисунке 46 ниже.

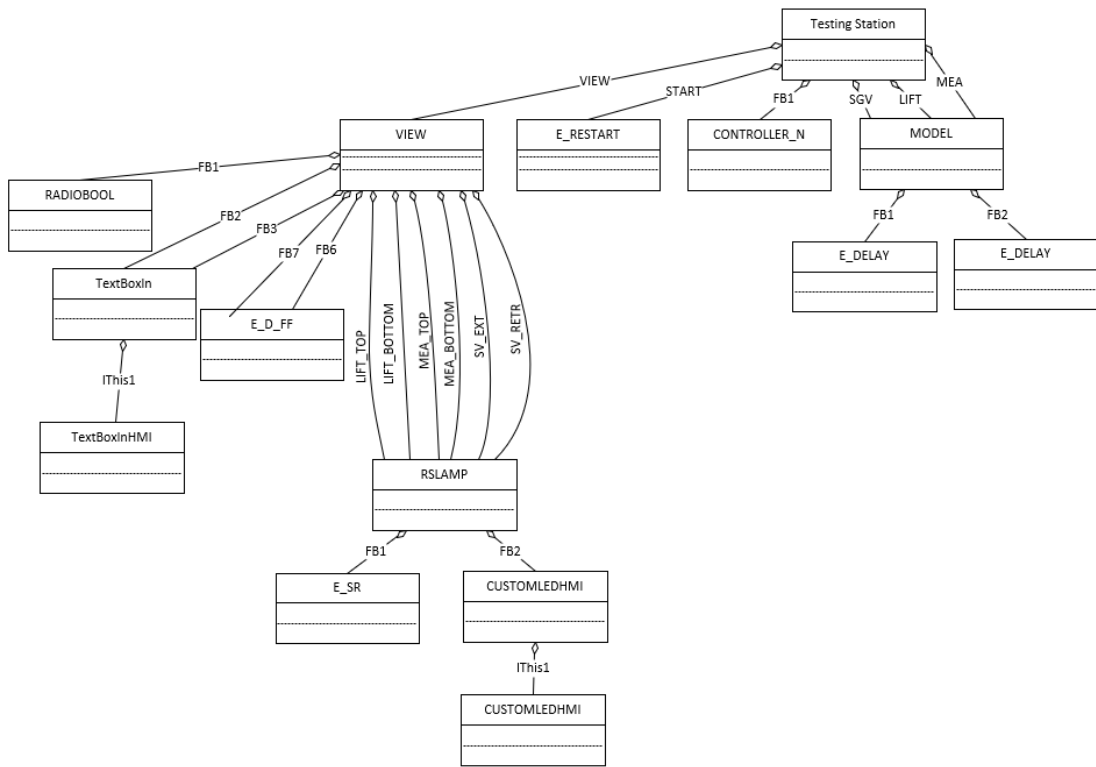


Рисунок 46 – Диаграмма классов *UML* для управляющего приложения

1.4.5 Компонента М (Модель)

Функциональный блок *MODEL* предназначен для моделирования каждого из мехатронных устройств осуществляется в виде двух временных задержек, причем первая временная задержка связана с движением подвижной части устройства вперед, а вторая – назад. Для моделирования был разработан ФБ *MODEL* (рисунок 47).

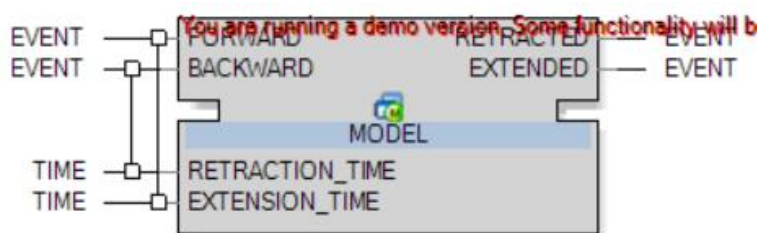


Рисунок 47 – Интерфейс ФБ *MODEL*

В зависимости от моделируемого устройства будут отличаться управляющие сигналы.

Для экземпляра ФБ выталкивателя (*SV*) определены следующие входы-выходы:

- событийные входы:

- 1) *FORWARD* – запустить выталкивание поршня;
- 2) *BACKWORD* – запустить втягивание поршня;

- событийные выходы:

- 1) *RETRACTED* – поршень втянут;
- 2) *EXTENDED* – поршень выдвинут;

- информационные входы:

- 1) *RETRACTION_TIME* – время втягивания поршня;
- 2) *EXTENSION_TIME* – время выдвигания поршня.

Для экземпляра ФБ лифта (*LIFT*) определены следующие входы-выходы:

-событийные входы:

- 1) *FORWARD* – запустить поднятие лифта;
- 2) *BACKWORD* – запустить опускание лифта;

- событийные выходы:

- 1) *RETRACTED* – лифт опущен;
- 2) *EXTENDED* – лифт поднят;

- информационные входы:

- 1) *RETRACTION_TIME* – время опускания лифта;
- 2) *EXTENSION_TIME* – время поднятия лифта.

Для экземпляра ФБ измерительного стержня (*MEA*) определены следующие входы-выходы:

- событийные входы:

- 1) *FORWARD* – запустить опускание проверочного стержня;
- 2) *BACKWORD* – запустить поднятие проверочного стержня;

- событийные выходы:

- 1) *RETRACTED* – проверочный стержень втянут;
- 2) *EXTENDED* – проверочный стержень опущен;

- информационные входы:

- 1) *RETRACTION_TIME* – время поднятия проверочного стержня;
- 2) *EXTENSION_TIME* – время опускания проверочного стержня.

Содержимое ФБ *MODEL* представлено на рисунке 48.

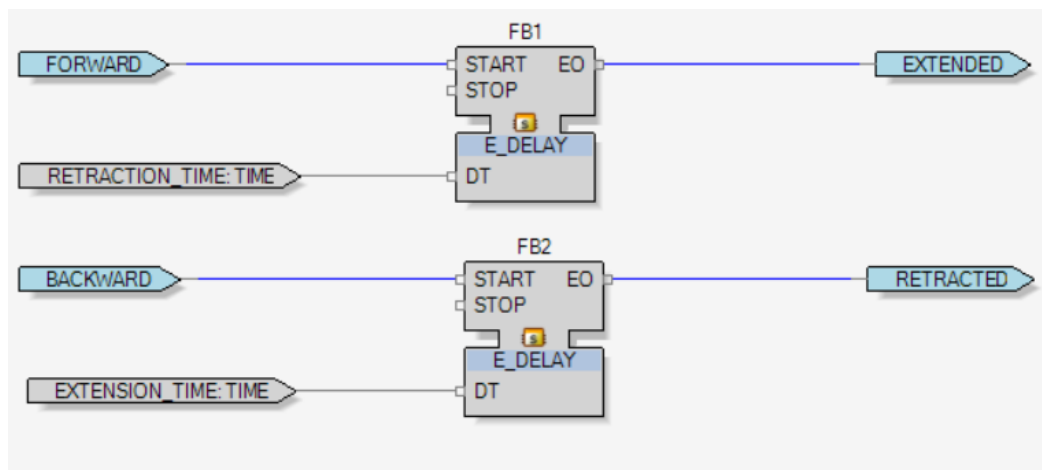


Рисунок 48 – Сеть функциональных блоков, входящих в состав блока *MODEL*

1.4.6 Компонента V (Представление)

Функциональный блок *VIEW* предназначен для организации человеко-машинного интерфейса, включая визуальное отображение состояний моделируемых устройств. Для моделирования был разработан ФБ *VIEW* (рисунок 49).

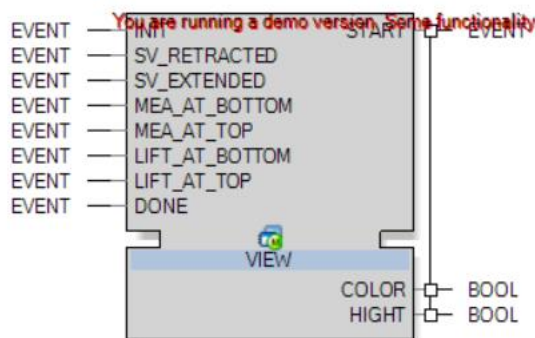


Рисунок 49 – Интерфейс ФБ *VIEW*

Для ФБ *VIEW* определены следующие входы-выходы:

- событийные входы:

- 1) *INIT* – инициализация;
- 2) *SV_RETRACTED* – поршень втянут;
- 3) *SV_EXTENDED* – поршень выдвинут;
- 4) *MEA_AT_BOTTOM* – измерительный стержень опущен;
- 5) *MEA_AT_TOP* – измерительный стержень втянут;
- 6) *LIFT_AT_BOTTOM* – лифт опущен;
- 7) *LIFT_AT_TOP* – лифт поднят;
- 8) *DONE* – обработка детали завершена.

- событийные выходы:

- 1) *START* – запуск моделирования обработки новой детали;

- информационные выходы:

- 1) *COLOR* – цвет детали (годна/не годна);
- 2) *HIGHT* – высота детали (годна/не годна).

Визуальное представление человеко-машинного интерфейса на экране монитора в системе *nxtSTUDIO* представлено на рисунке 50.



Рисунок 50 – Визуальное представление человеко-машинного интерфейса (блок *VIEW*)

Содержимое ФБ *VIEW* представлено на рисунке 51.

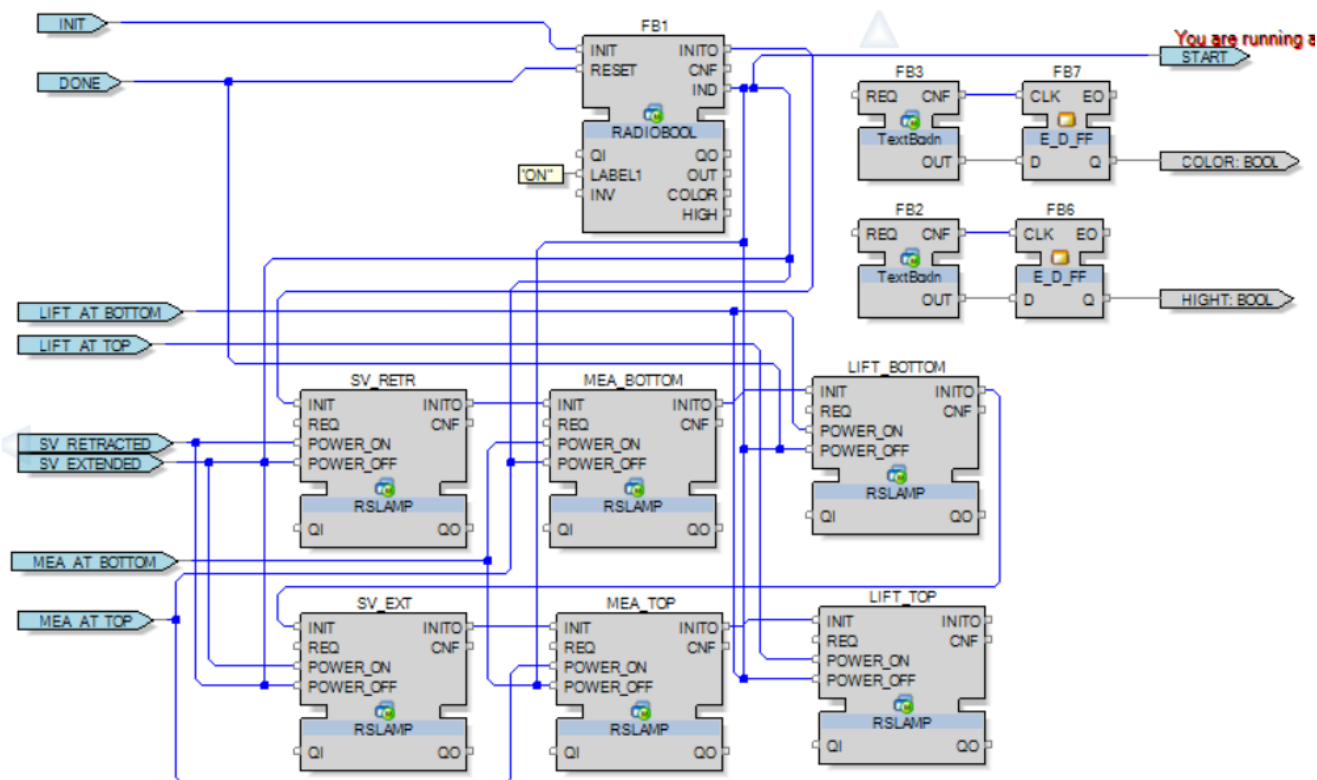


Рисунок 51 – Сеть функциональных блоков, входящих в состав блока *VIEW*

Функциональный блок *RSLAMP* предназначен для фиксации и отображения состояния светодиода. Для моделирования был разработан ФБ *RSLAMP* (рисунок 52).

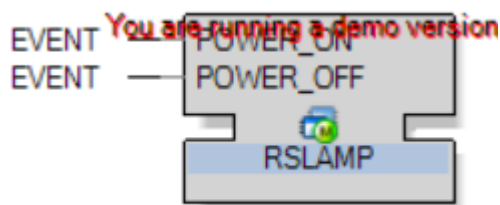


Рисунок 52 – Интерфейс ФБ *RSLAMP*

Для ФБ *RSLAMP* определены следующие событийные входы:

- *POWER_ON* – включить светодиод;
- *POWER_OFF* - выключить светодиод.

Содержимое ФБ *RSLAMP* представлено на рисунке 53.

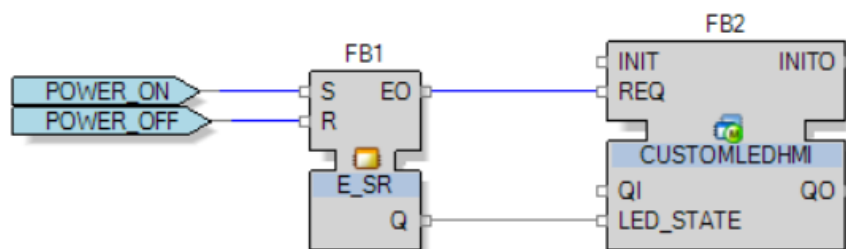


Рисунок 53 – Сеть функциональных блоков, входящих в состав блока *RSLAMP*

Функциональный блок *E_SR* используется как триггер для фиксации состояния светодиода. Данный блок является библиотечным.

Для ФБ *E_SR* определены следующие входы-выходы:

- событийные входы:
 - 1) *S* – установить в единицу;
 - 2) *R* – сбросить в ноль;
- событийные выходы:
 - 1) *EO* – изменение состояния триггера;

- информационные выходы:

1) *Q* – состояние триггера.

Функциональный блок *CUSTOMLEDHMI* предназначен для отображения светодиода на экране. Для ФБ *CUSTOMLEDHMI* определены следующие входы-выходы:

- событийные входы:

1) *INIT* – старт инициализации;

2) *REQ* – запрос на рисование;

- событийные выходы:

1) *INITO* – инициализация завершена;

- информационные входы:

1) *LED_STATE* – цвет отображения светодиода (если «истина» - зеленый цвет, «ложь» - серый);

- выходы не используются.

Функциональный блок *RADIOBOOL* предназначен для организации «кнопки» в человеко-машинном интерфейсе (рисунок 54).

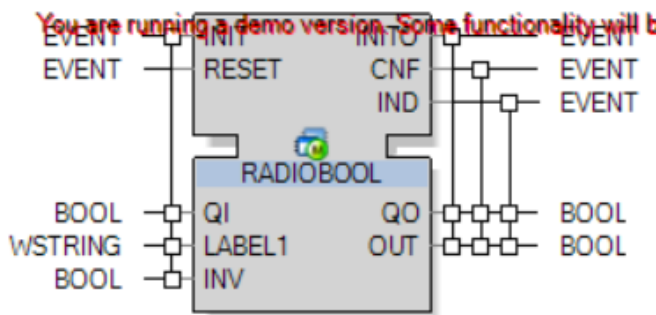


Рисунок 54 – Интерфейс ФБ *RADIOBOOL*

Для ФБ *RADIOBOOL* определены следующие входы-выходы:

- событийные входы:

1) *INIT* – старт инициализации;

2) *RESET* – сброс;

- событийные выходы:

- 1) *INITO* – инициализация завершена;
- 2) *CNF* – не используется;
- 3) *IND* – индикация нажатия кнопки;

- информационные входы:

- 1) *QI* – не используется;
- 2) *LABEL1* – надпись на кнопке;
- 3) *INV* - не используется;

- информационные выходы не используются.

Содержимое ФБ *RADIOBOOL* представлено на рисунке 55.

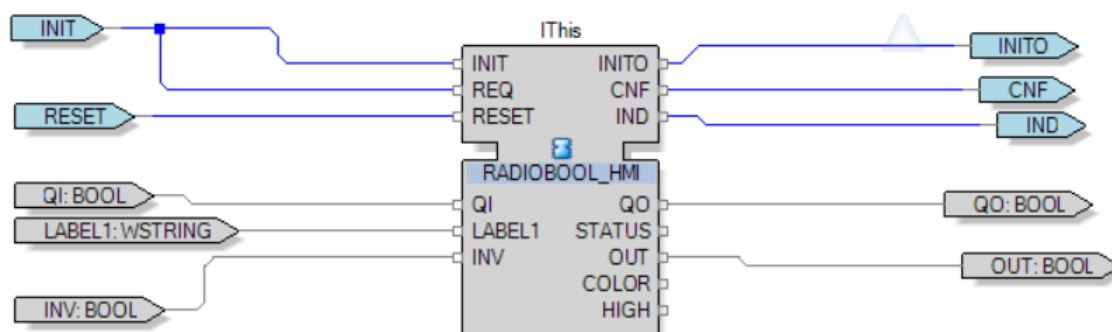


Рисунок 55 – Сеть функциональных блоков, входящих в состав блока *RADIOBOOL*

Функциональный блок *RADIOBOOL_HMI* предназначен для визуального отображение кнопки на экране монитора (рисунок 56).

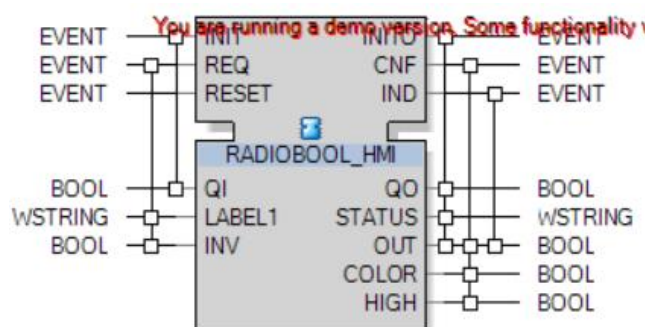


Рисунок 56 – Интерфейс ФБ *RADIOBOOL_HMI*

Для ФБ *RADIOBOOL_HMI* определены следующие входы-выходы:

- событийные входы:

- 1) *INIT* – старт инициализации;
- 2) *REQ* – запрос на «рисование» кнопки на экране монитора;
- 3) *RESET* – сброс;

- событийные выходы:

- 1) *INITO* – инициализация завершена;
- 2) *CNF* – подтверждение выполнения действия по сигналу *REQ*;
- 3) *IND* – индикация нажатия кнопки;

- информационные входы:

- 1) *QI* – модификатор;
- 2) *LABEL1* – имя кнопки;
- 3) *INV* – не используется;

- информационные выходы не используются.

Визуальное представление *RADIOBOOL* на экране монитора в системе *nxtSTUDIO* представлено на рисунке 57.

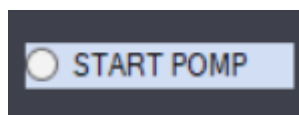


Рисунок 57 – Визуальное представление *RADIOBOOL*

Функциональный блок *TextBoxIn* предназначен для организации текстового поля в человеко-машинном интерфейсе (рисунок 58).

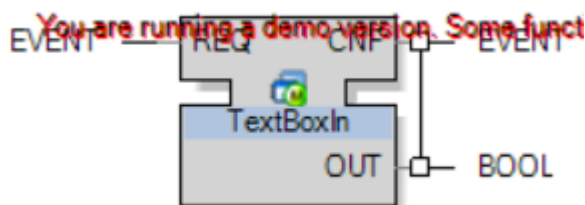


Рисунок 58 – Интерфейс ФБ *TextBoxIn*

Для ФБ *TextBoxIn* определены следующие входы-выходы:

- событийные входы:

1) *REQ* – не используется;

- событийные выходы:

1) *CNF* – подтверждение нажатия ввода в текстовом поле;

- информационные выходы:

1) *OUT* – значение текстового поля (истина/ложь).

Содержимое ФБ *TextBoxIn* представлено на рисунке 59.

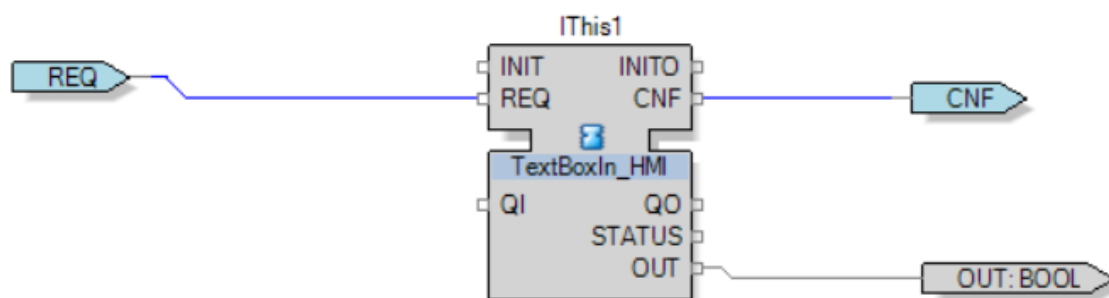


Рисунок 59 – Сеть функциональных блоков, входящих в состав блока *TextBoxIn*

Функциональный блок *TextBoxIn_HMI* предназначен для отрисовки текстового поля на холсте (рисунок 60).

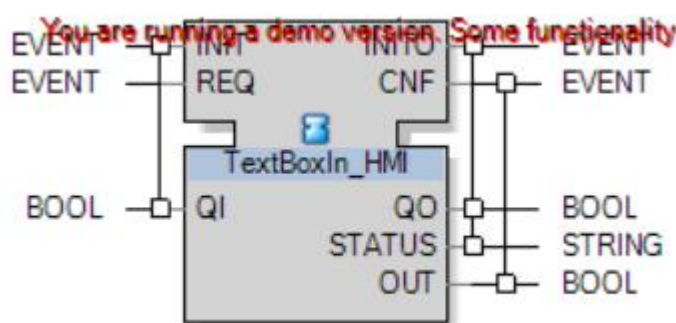


Рисунок 60 – Интерфейс ФБ *TextBoxIn_HMI*

Для ФБ *TextBoxIn_HMI* определены следующие входы-выходы:

- событийные входы:

- 1) *INIT* – старт инициализации;
- 2) *REQ* – не используется;

- событийные выходы:

- 1) *INITO* – инициализация завершена;
- 2) *CNF* – подтверждение нажатия ввода в текстовом поле;

- информационные выходы:

- 1) *Q0* – не используется;
- 2) *STATUS* – не используется;
- 3) *OUT* – значение текстового поля (истина/ложь).

Визуальное представление *TextBoxIn* на экране монитора в системе *nxtSTUDIO* представлено на рисунке 61.



Рисунок 61 – Визуальное представление *TextBoxIn*

Функциональный блок *E_D_FF* используется как *D* – триггер для фиксации введенного цвета и высоты детали (рисунок 62).

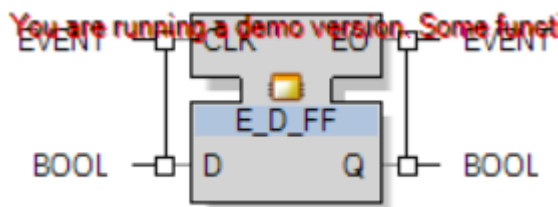


Рисунок 62 – Интерфейс ФБ *E_D_FF*

Для ФБ *E_D_FF* определены следующие входы-выходы:

- событийные входы:

1) *CLK* – сигнал синхронизации;

- событийные выходы:

1) *EO* – подтверждение фиксации значения в триггере;

- информационные входы:

1) *D* – входное значение;

- информационные выходы:

1) *Q* – выходное значение;

1.4.7 Компонента С (Контроллер)

Функциональный блок *CONTROLLER_N* предназначен для управления процессом обработки детали (рисунок 63).

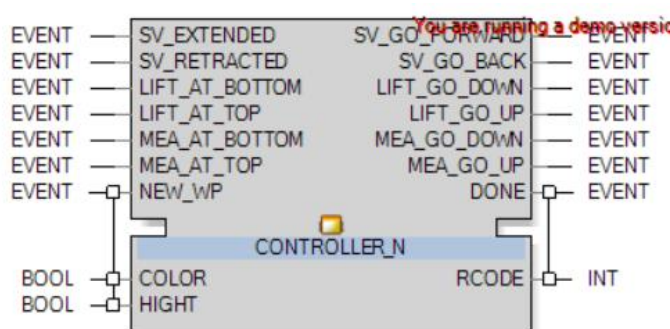


Рисунок 63 – Интерфейс ФБ *CONTROLLER_N*

Для ФБ *CONTROLLER_N* определены следующие входы-выходы:

- событийные входы:

1) *SV_RETRACTED* –поршень втянут;

2) *SV_ETENDED* –поршень выдвинут;

3) *MEA_AT_BOTTOM* – измерительный стержень опущен;

4) *MEA_AT_TOP* - измерительный стержень втянут;

5) *LIFT_AT_BOTTOM* – лифт опущен;

6) *LIFT_AT_TOP* – лифт поднят;

7) *NEW_WP* – приход новой детали;

- событийные выходы:

- 1) *SV_GO_FORWARD* – запустить движение поршня вперед;
- 2) *SV_GO_BACK* – запустить движение поршня назад;
- 3) *LIFT_GO_DOWN* – запустить движение лифта вниз;
- 4) *LIFT_GO_UP* – запустить движение лифта вверх;
- 5) *MEA_GO_DOWN* – запустить движение измерительного стержня вниз;
- 6) *MEA_GO_UP* – запустить движение измерительного стержня вверх;
- 7) *DONE* – обработка детали завершена;

- информационные входы:

- 1) *COLOR* – цвет детали;
- 2) *HIGHT* – высота детали;

- информационные выходы:

- 1) *RCODE* – код завершения.

Диаграмма управления выполнением представлена на рисунках 64–65.

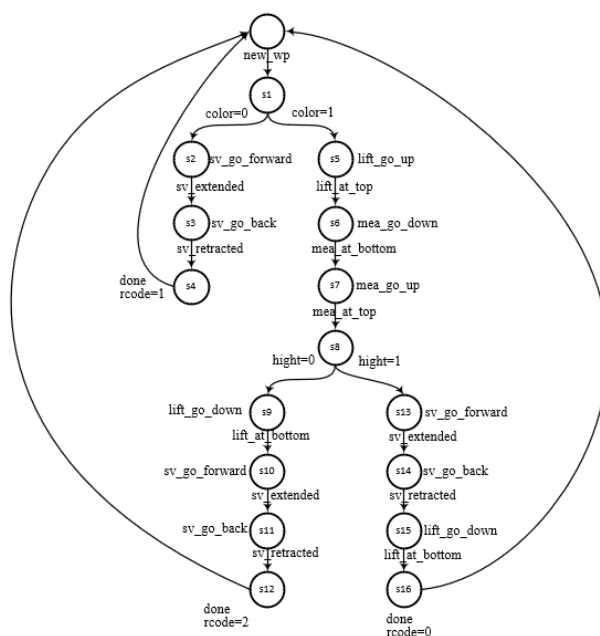


Рисунок 64 – Диаграмма управления выполнением (диаграмма *ECC*),
разработанная вручную

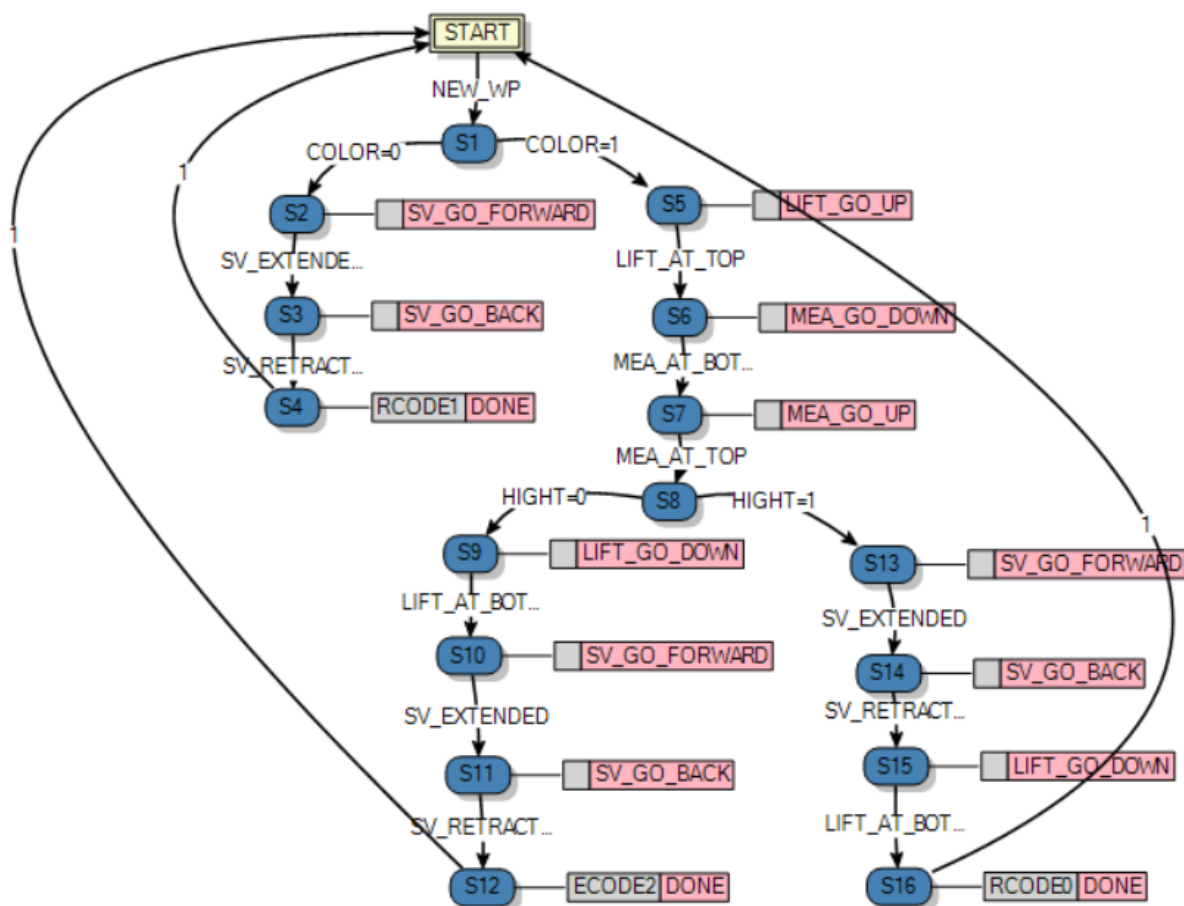


Рисунок 65 – Диаграмма управления выполнением (диаграмма ECC), сформированным в *nxtSTUDIO*

1.4.8 Отладка и тестирование

Работоспособность программы была проверена при следующих сценариях обработки делали: «плохой цвет», «плохая высота», «нормальный цвет, нормальная высота».

На рисунке 66 представлена работа программы при сценарии «хороший цвет, хорошая высота» (последовательность скриншотов идет слева направо, сверху вниз).

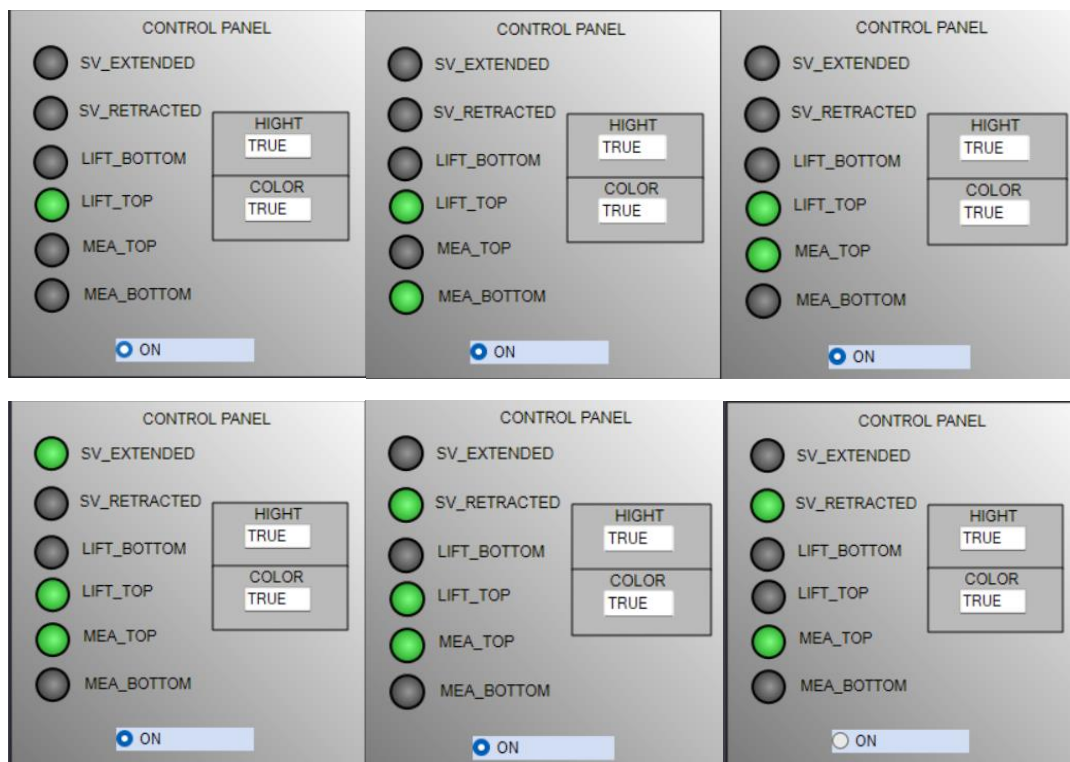


Рисунок 66 – Работа программы при сценарии «хороший цвет, хорошая
высота»

На рисунке 67 представлена работа программы при сценарии «плохой
цвет» (последовательность скриншотов идет слева направо).

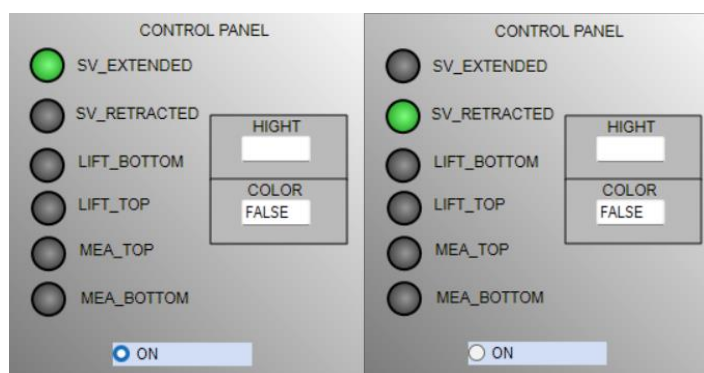


Рисунок 67 – Работа программы при сценарии «плохой цвет»

На рисунке 68 представлена работа программы при сценарии «хороший
цвет, плохая высота» (последовательность скриншотов идет слева направо,
сверху вниз).

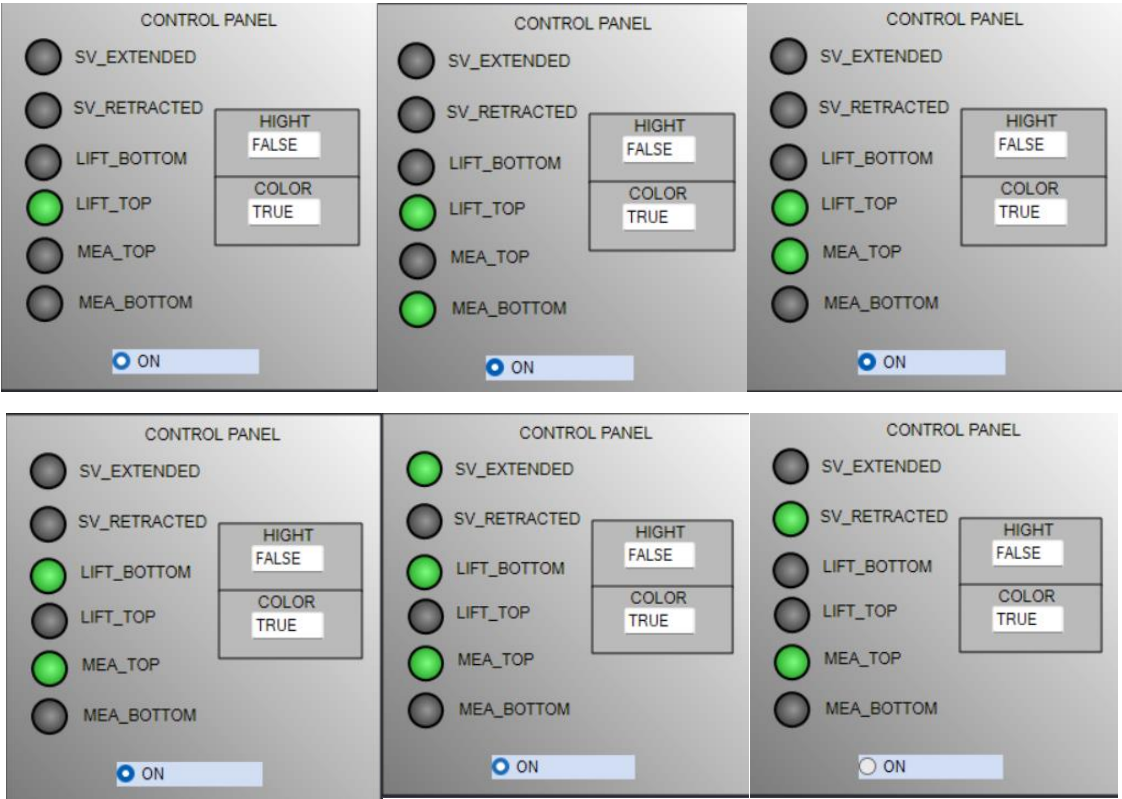


Рисунок 68 – Работа программы при сценарии «хороший цвет, плохая высота»

2 Требования к освещению рабочего места, оборудованного ПЭВМ

2.1 Общие положения

Эффективность труда специалистов всех категорий независимо от характера условий их деятельности зависит от того, как устроено и оснащено рабочее место. Организация рабочего места включает в себя оснащение всем необходимым в соответствии с характером работы, рациональное расположение этого необходимого оснащения, создание удобств, комфортных условий работы, предотвращение вредного воздействия на человека неблагоприятных факторов внешней среды. Работа оператора ЭВМ утомительна, так как связана с постоянным статическим напряжением мышц и многими другими вредными для организма явлениями, возникающими из-за относительной неподвижности рабочей позы. К этому добавляется также постоянное нервное напряжение. Такой труд нуждается в облегчении и, прежде всего в рациональном устройстве рабочего места.

Важнейшие положения в области охраны труда закреплены в 10 разделе Трудового Кодекса РФ от 30 декабря 2001 г. N 197-ФЗ.

Государственными нормативными требованиями охраны труда, содержащимися в федеральных законах и иных нормативных правовых актах Российской Федерации и законах и иных нормативных правовых актах субъектов Российской Федерации об охране труда, устанавливаются правила, процедуры и критерии, направленные на сохранение жизни и здоровья работников в процессе трудовой деятельности. Требования охраны труда обязательны для исполнения юридическими и физическими лицами при осуществлении ими любых видов деятельности, в том числе при проектировании, строительстве (реконструкции) и эксплуатации объектов, конструировании машин, механизмов и другого оборудования, разработке технологических процессов, организации производства и труда.

Порядок разработки и утверждения подзаконных нормативных правовых актов об охране труда, а также сроки их пересмотра устанавливаются Правительством Российской Федерации.

Нормы охраны труда содержатся в правилах внутреннего трудового распорядка предприятий (учреждений).

Обеспечение здоровых и безопасных условий труда возлагается на администрацию предприятий (учреждений). Администрация обязана внедрить современные средства техники безопасности, предупреждающие производственный травматизм, и обеспечивать санитарно-гигиенические условия, предотвращающие возникновение профессиональных заболеваний рабочих и служащих.

Производственные здания, сооружения, оборудование, технологические процессы должны отвечать требованиям, обеспечивающим здоровые и безопасные условия труда. Эти требования включают: рациональное использование территории и производственных помещений, правильную эксплуатацию оборудования и организацию технологических процессов, защиту работающих от воздействия вредных условий труда, содержание производственных помещений и рабочих мест в соответствии с санитарно-гигиеническими нормами и правилами, устройство санитарно-бытовых помещений. При проектировании, строительстве и эксплуатации производственных зданий и сооружений должны соблюдаться правила и нормы по охране труда.

Администрация предприятий (учреждений) обязана осуществлять организационную работу по обеспечению безопасных и здоровых условий труда (планирование и финансирование различных мероприятий по охране труда, проведение инструктажа рабочих и служащих по технике безопасности и производственной санитарии и т.п.).

2.2 Требования к освещению на рабочих местах, оборудованных ПЭВМ

Основная задача освещения внутри помещения — создание наилучших условий для зрения. Требования к освещению на рабочих местах, оборудованных ПЭВМ подробно освещены в шестом разделе СанПиН 2.2.2./2.4.1340-03. Согласно данному документу, освещение должно соответствовать следующим требованиям:

2.2.1 Освещенность на рабочем месте должна соответствовать характеру зрительной работы, который определяется следующими тремя параметрами:

объект различения — наименьший размер рассматриваемого предмета, отдельная его часть или дефект, который необходимо различить в процессе работы (например, при работе с приборами — толщина линии градуировки шкалы; при чертежных работах — толщина самой тонкой линии на чертеже);

фон — поверхность, прилегающая непосредственно к объекту различения, на которой он рассматривается; характеризуется коэффициентом отражения, зависящим от цвета и фактуры поверхности, значения которого находятся в пределах 0.02—0.95; при коэффициенте отражения поверхности более 0,4 фон считается светлым; 0,2 — 0,4 - средним и менее 0,2 — темным;

контраст объекта с фоном K характеризуется соотношением яркостей рассматриваемого объекта (точка, линия, знак, пятно, трещина, риска, раковина или другие элементы, которые требуется различить в процессе работы) и фона. Контраст определяют по формуле:

$$K = \frac{|L_0 - L_{\phi}|}{L_{\phi}},$$

где L_{ϕ} — яркость фона;

L_0 — яркость объекта.

Контраст объекта с фоном считается большим при значениях K более 0,5 (объект и фон резко отличаются по яркости), средним при значениях $K = 0.2 -$

0.5 (объект и фон заметно отличаются по яркости) и малым при значениях K менее 0.2 (объект и фон мало отличаются по яркости).

Увеличение освещенности рабочей поверхности улучшает видимость объектов за счет повышения их яркости, увеличивает скорость различения деталей, что сказывается на росте производительности труда.

2.2.2 Необходимо обеспечить достаточно равномерное распределение яркости на рабочей поверхности, а также в пределах окружающего пространства. Если в поле зрения находятся поверхности, значительно отличающиеся между собой по яркости, то при переводе взгляда с ярко освещенной на слабо освещенную поверхность глаз вынужден переадаптироваться, что ведет к повышенному утомлению зрения.

2.2.3 На рабочей поверхности должны отсутствовать резкие тени. Наличие резких теней создает неравномерное распределение поверхностей с различной яркостью в поле зрения, искажает размеры и формы объектов различения, в результате повышается утомляемость, снижается производительность труда. Особенно вредны движущиеся тени, которые могут привести к травмам. Тени необходимо смягчать, применяя, например, светильники со светорассеивающими молочными стеклами.

В отделах, оборудованных рабочими местами с ПЭВМ, необходимо предусматривать на окнах солнцезащитные устройства (жалюзи, козырьки, светорассеивающие стеклопластики), предотвращающие проникновение прямых солнечных лучей, которые создают на рабочих местах резкие тени.

2.2.4 В поле зрения должна отсутствовать прямая и отраженная блескость. Блескость — повышенная яркость светящихся поверхностей, вызывающая нарушение зрительных функций (ослепленность), т. е. ухудшение видимости объектов.

Видимость V характеризует способность глаза воспринимать объект; зависит от освещенности, размера объекта, его яркости, контраста объекта с

фоном, длительности экспозиции. Видимость определяется числом пороговых контрастов в контрасте объекта с фоном:

$$V = \frac{K}{K_{\text{пор}}},$$

где ($K_{\text{пор}}$ — пороговый контраст, т. е. наименьший различимый глазом контраст, при небольшом уменьшении которого объект становится неразличимым на фоне.

Прямая блескость связана с источниками света, отраженная возникает на поверхности с большим коэффициентом отражения или отражением по направлению к глазу. Ослепленность приводит к быстрому утомлению и снижению работоспособности. Критерием оценки слепящего действия, создаваемого осветительной установкой, является показатель ослепленности P_o значение которого определяют по формуле:

$$P_o = (V_1/V_2 - 1) * 1000,$$

где V_1 — видимость объекта различения при экранировании;

V_2 — видимость объекта различения при наличии ярких источников света в поле зрения.

Экранирование источников света осуществляют с помощью щитков, козырьков и т. п.

Прямую блескость ограничивают уменьшением яркости источников света, правильным выбором защитного угла светильника, увеличением высоты подвеса светильников. Отраженную блескость ослабляют правильным выбором направления светового потока на рабочую поверхность, а также изменением угла наклона рабочей поверхности. Там, где это возможно, следует заменять блестящие поверхности матовыми.

2.2.5 Величина освещенности должна быть постоянной во времени. Колебания освещенности, вызванные резким изменением напряжения в сети, имеют большую амплитуду, каждый раз вызывая переадаптацию глаза, приводят к значительному утомлению. Пульсация освещенности связана также с особенностью работы газоразрядных ламп.

Коэффициент пульсации освещенности K_p —критерий оценки относительной глубины колебаний освещенности в результате изменения во времени светового потока газоразрядных ламп при питании их переменным током.

Коэффициент пульсации освещенности $K_p(\%)$ следует определять по формуле:

$$K_p = \frac{100(E_{max} - E_{min})}{2 * E_{cp}},$$

где E_{max} — максимальное значения освещенности за период ее колебания;

E_{min} — минимальное значения освещенности за период ее колебания;

E_{cp} — среднее соответственно значения освещенности за период ее колебания.

Постоянство освещенности во времени достигается стабилизацией питающего напряжения, жестким креплением светильников, применением специальных схем включения газоразрядных ламп. Например, снижение коэффициента пульсации освещенности люминесцентных ламп с 55 до 5% (при трехфазном включении) приводит к уменьшению утомления и повышению производительности труда на 15% для работ высокой точности.

2.2.6 Следует выбирать оптимальную направленность светового потока, что позволяет в одних случаях рассмотреть внутренние поверхности деталей, в других — различить рельефность элементов рабочей поверхности.

2.2.7 Следует выбирать необходимый спектральный состав света. Это требование особенно существенно для обеспечения правильной цветопередачи, а в отдельных случаях для усиления цветовых контрастов.

Правильную цветопередачу обеспечивают естественное освещение и искусственные источники света со спектральной характеристикой, близкой к солнечной. Для создания цветовых контрастов применяют монохроматический свет, усиливающий одни цвета и ослабляющий другие.

2.2.8 Все элементы осветительных установок — светильники, групповые щитки, понижающие трансформаторы, осветительные сети — должны быть достаточно долговечными, электробезопасными, а также не должны быть причиной возникновения пожара или взрыва. Обеспечение указанных условий достигается применением зануления или заземления, ограничением напряжения для питания местных и переносных светильников до 42 В и ниже (36, 24, 12 В), выбором оборудования, соответствующего условиям среды в помещениях, и защитой элементов осветительных сетей от механических повреждений при эксплуатации. Кроме того, необходимо уменьшить до минимума теплоту, выделяемую осветительной установкой, и шум.

2.3 Выводы

В ходе выполнения раздела дипломного проекта по безопасности жизнедеятельности, был рассмотрен ряд законодательных документов по охране труда и проанализирована другая литература, связанная с тематикой раздела. Согласно заданию, было рассмотрено освещение и в соответствии с проведенным анализом рассчитано необходимое количество светильников, которое обеспечивает необходимое освещение помещения. Таким образом, выяснено, что для наиболее благоприятной работы оператора необходимо установить 9 светильников типа ОД.

Заключение

В бакалаврской работе были получены следующие результаты:

- разработана программная имитационная модель тестирующей станции производственной системы *FESTO*, позволяющая записывать журнал событий;
- по журналам событий, полученных в ходе имитационных экспериментов, с использованием технологии *Process mining* сгенерированы формальные модели процессов, протекающих в тестирующей станции производственной системы *FESTO*, а также произведено их исследование;
- с использованием паттерна проектирования *MVC*, а также сгенерированных формальных моделей разработана (имитационная) функционально-блочная модель тестирующей станции производственной системы *FESTO* на основе международного стандарта *IEC 61499*, причем управляющая компонента может быть перенесена на реальную систему практически без изменений.

Интересным вопросом, требующим дальнейшего исследования, является использование метода *Process mining* в отношении журналов событий, полученных в ходе имитационных экспериментов. С одной стороны, строится формальная модель не реальной системы (в данном случае, тестирующей станции *FESTO*), а формальная модель имитационной модели тестирующей станции *FESTO* (т.е. «модель модели»), что несколько уменьшает значимость этого метода для исследования реальных систем. С другой стороны, метод *Process mining* может служить определенным расширением возможностей имитационного моделирования. Построение формальной модели для имитационной модели важно в двух аспектах:

- в качестве языка спецификации и общения формальная модель может служить для улучшения коммуникации между разработчиками, а также для документирования;

- автоматизированный анализ формальной модели может выявить ошибки как в самой имитационной модели, так и в проектируемой или реальной системе (при условии, что имитационная модель точно и без ошибок описывает эту систему). Как правило, в ходе имитационного моделирования собирается некая обобщенная статистика, но совершенно не собираются и не анализируются данные о протекающих процессах.

					ПГУ 1.090301.11.001 ПЗ	Лист
Изм.	Лист	№ докум	Подпись	Дата		73

Список использованных источников

1. Tomilla T., Venta O., Koskinen K. Next generation industrial automation – needs and opportunities, *Automation Technology Review*, 2001, pp.34-41.
2. Гома Х. *UML. Проектирование систем реального времени, параллельных и распределенных приложений.* – М.: ДМК. – 2002. – 704 с.
3. Дубинин В.Н., Вяткин В. Проектирование распределенных систем управления промышленными процессами с использованием *UML-FB* // Известия высших учебных заведений. Поволжский регион. Сер. Технические науки. – Пенза: Изд-во Пенз. гос. ун-та, 2004. – N 2 (11). – С.136-146.
4. Дубинин В.Н., Вяткин В.В. Модели функциональных блоков IEC 61499, их проверка и трансформации в проектировании распределенных систем управления: монография. – Пенза: Изд-во ПГУ, 2012. – 348 с. – URL: https://yadi.sk/i/l_imSbayZsr4uw (дата обращения: 19.03.2022).
5. Дубинин В.Н. Графическое моделирование системы *FESTO* – URL: http://alice.pnzgu.ru/~dvn/fb61499/festo/graphic_modeling/index.htm (дата обращения: 23.03.2022).
6. Хилл Ф. *OpenGL. Программирование компьютерной графики.* - СПб: Питер, 2006. - 1088 с.
7. Частиков А. П., Гаврилова Т. А., Белов Д. Л. Разработка экспертных систем. Среда *CLIPS*. – СПб.: БХВ-Петербург, 2003. -608 с.
8. Rausch, M. Net condition/event systems with multiple condition outputs / M. Rausch, H.-M. Hanisch // *IEEE Int. Conf. on Emerging Technologies and Factory Automation.* – Paris, 1995. – Vol. 1. – P. 592–600.
9. ViVe – VisualVerifier Tool Framework. – URL: <http://www.fb61499.com/license.html> ().
10. Vyatkin, V. Closed-Loop Modeling in Future Automation System Engineering and Validation / V. Vyatkin, H.-M. Hanisch, P. Cheng, Y. Chia-Han // *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews.* – 2009. – Vol. 39. – P. 17–28.

11. Дубинин А.В., Дубинин В.Н., Ручкин М.А. *NCES-модель PnP-манипулятора* // Сб. статей VII Ежегодной всероссийской межвузовской науч.-практ. конф. «Информационные технологии в науке и образовании. Проблемы и перспективы» (ВМНПК – 2020), Пенза, 2020. – С. 391-393.

12. Отчет по проекту № 2.1.2/4257 «Разработка комплекса формальных моделей и их трансформаций для проектирования распределенных информационно-управляющих систем промышленной автоматизации».

13. *FESTODidactic*. – URL: <https://www.festo-didactic.com> (дата обращения: 18.04.2022).

14. *Jensen, K. Coloured Petri Nets. Modelling and Validation of Concurrent Systems* / K. Jensen, L. M. Kristensen // Springer Publishing Company. – 2009. – 384 p.

15. Зайцев, Д. А. Моделирование телекоммуникационных систем в *CPNTools*: учеб. пособие / Д. А. Зайцев, Т. Р. Шмелева. – Одесса: ОНАС, 2008. – 60 с.

16. *CPNTools*. – URL: <http://cpntools.org/> (дата обращения: 15.01.2022).

17. Советов Б. Я., Яковлев С. А. Моделирование систем: Учебник для вузов по спец. «Автоматизированные системы управления». — М.: Высш. шк., 1985. — 271 с., ил.

18. *Van der Aalst, W. et al. Process Mining Manifesto* // *Lecture Notes in Business Information Processing*, vol 99. Springer, Berlin, Heidelberg, 2012. - pp 169–194.

19. Процессная аналитика. Википедия. URL: <https://ru.wikipedia.org/wiki/> (дата обращения: 24.02.2022).

20. *ProM Tools*. URL: <https://www.promtools.org/doku.php> (дата обращения: 04.03.2022).

21. Муромский Д.А., Дубинин А.В., Ручкин М.А. Программа детерминизации и конвертации автоматных моделей в формат *GraphML* // Сб. ст.

IX Ежегодн. всеросс. межвуз. науч.-практ. конф. «Информационные технологии в науке и образовании. Проблемы и перспективы (ВМНПК – 2022)», Пенза, 2022.

22. Ручкин М.А., Дубинин А.В., Дубинин В.Н. Конвертер автоматных моделей процессов в функционально-блочные модели оборудования // Сб. ст. IX Ежегодн. всеросс. межвуз. науч.-практ. конф. «Информационные технологии в науке и образовании. Проблемы и перспективы (ВМНПК – 2022)», Пенза, 2022.

23. Кобзев М.И. Дубинин В.Н. Конвертер автоматных моделей процессов в функционально-блочные модели мониторов последовательностей // Сб. ст. IX Ежегодн. всеросс. межвуз. науч.-практ. конф. «Информационные технологии в науке и образовании. Проблемы и перспективы (ВМНПК – 2022)», Пенза, 2022.

24. *The TINA toolbox Home Page - Time Petri Net Analyzer.* – URL: <http://projects.laas.fr/tina/> (дата обращения: 27.01.2022).

Приложение А

(обязательное)

Программная имитационная модель производственной
системы *FESTO* и ее исследование

Листинг программы

					ПГУ 1.090301.11.001 ПЗ	Лист
Изм.	Лист	№ докум	Подпись	Дата		77

```

#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <conio.h>
#include <locale.h>
#include <string.h>
#include <windows.h>
#include <iostream>
#include <string>
#include <time.h>
#include <ctime>
#include <fstream>
using namespace std;

void s1();
void s2();
void s3();
void s4();
void s5();
void s6();
void s7();

typedef struct Node { // элемент списка будущих событий
    int time;
    string event_type;
    string event_id;
    string device_id;
    struct Node *next;
} Node;

typedef struct Detal {
    bool height;
    bool color;
} Detal;

typedef struct Queue {
    struct Detal* detalle;
    struct Queue *next;
} Queue;

Node* node = NULL;
Node* head = NULL;
Node* head1 = NULL;
Detal* detal = NULL;
Queue* first = NULL;
Queue* last = NULL;
ofstream out1; // поток для записи

bool lift = 0; // lift = 0 - лифт опущен, lift = 1 - лифт поднят

```

```

bool SGV = 0; // SGV = 0 - выталкиватель находится в задвинутом
положении, SGV = 1 - выталкиватель находится в выдвинутом
положении
bool TS = 1; // TS = 1 - тестирующая станция свободна, TS = 0 -
тестирующая станция занята
bool msr = 0; // msr = 0 - измерительный стержень находится в
верхнем положении, msr = 1 - измерительный стержень находится в
нижнем положении
int systime = 0;
int lim = 1000;
int ProcessId = 1000;
int N = 10; // число прогонов
int t;

```

```

void generate(Detal** detal) //сгенерировать новую деталь
{

```

```

    *detal = new Detal;

```

```

    (*detal)->height = rand() % 2;
    (*detal)->color = rand() % 2;
    cout <<"Высота: " <<(*detal)->height << " ";
    cout << "Цвет: " << (*detal)->color << endl;

```

```

}

```

```

void insert_queue(Queue** head, Queue** final, Detal* k) //
занести деталь в очередь
{

```

```

    if ((*head) == NULL)
    {
        (*head) = new Queue;
        (*head)->detale = k;
        *final = *head;
    }
    else
    {
        (*final)->next = new Queue;

        (*final) = (*final)->next;
        (*final)->detale = k;
    }

```

```

}

```

```

Detal* take_queue(Queue** head, Queue** final) // достать деталь
из очереди
{

```

```

    if ((*head) != NULL)
    {
        if ((*head) == (*final))

```

```

        {
            Detal* k = (*head)->detale;
            (*head) = NULL;
            (*final) = NULL;
            return k;
        }
        else
        {
            Detal* k = (*head)->detale;
            (*head) = (*head)->next;
            return k;
        }
    }
    return NULL;
}

```

void insert(Node** head, Node* t) // занесение в список будущих
СОБЫТИЙ

```

{
    Node* tmp = (*head);
    bool cheker = true;
    if ((*head) == NULL)
    {
        (*head) = t;
    }
    else if ((*head)->time > t->time)
    {
        t->next = (*head);
        (*head) = t;
    }
    else
    {
        while (tmp->next)
        {
            if (tmp->next->time > t->time)
            {
                cheker = false;
                break;
            }
            tmp = tmp->next;
        }
        if (cheker)
        {
            tmp->next = t;
        }
        else
        {
            t->next = tmp->next;
            tmp->next = t;
        }
    }
}

```

```

    }

}

void action(string s)
{
    out1 << ProcessId << ", ";
    out1 << head->time << ", ";
    out1 << head->device_id << ", ";
    out1 << s;
    out1 << endl;
}

void SGV_go_forward() //Запуск выталкивателя вперед
{
    node = new Node;
    node->time = systime + rand() % 10 + 1;
    node->event_type = "Запуск выталкивателя вперед";
    node->event_id = "s2";
    node->device_id = "Slidegate valve";

    cout << node->time << " ";
    cout << node->event_type << " ";
    cout << node->event_id << " ";
    cout << node->device_id << endl;

    node->next = NULL;
    insert(&head, node);
    SGV = 1;
}

void SGV_go_back() //Запуск выталкивателя назад
{
    node = new Node;
    node->time = systime + rand() % 10 + 1;
    node->event_type = "Запуск выталкивателя назад";
    node->event_id = "s3";
    node->device_id = "Slidegate valve";

    cout << node->time << " ";
    cout << node->event_type << " ";
    cout << node->event_id << " ";
    cout << node->device_id << endl;

    node->next = NULL;
    insert(&head, node);
    SGV = 0;
}

```

```

void lift_go_up() //Запуск движения лифта в верх
{
    node = new Node;
    node->time = systime + rand() % 10+1;
    node->event_type = "Запуск движения лифта в верх";
    node->event_id = "s6";
    node->device_id = "Lift";

    cout << node->time << " ";
    cout << node->event_type << " ";
    cout << node->event_id << " ";
    cout << node->device_id << endl;

    node->next = NULL;
    insert(&head, node);
}

void lift_go_down() //Запуск движения лифта в вниз
{
    node = new Node;
    node->time = systime + rand() % 10+1;
    node->event_type = "Запуск движения лифта в вниз";
    node->event_id = "s7";
    node->device_id = "Lift";

    cout << node->time << " ";
    cout << node->event_type << " ";
    cout << node->event_id << " ";
    cout << node->device_id << endl;

    node->next = NULL;
    insert(&head, node);
}

void msr_go_down() //Запуск движения проверочного стржня вниз
{
    node = new Node;
    node->time = systime + rand() % 10+1;
    node->event_type = "Запуск движения проверочного стржня
вниз";
    node->event_id = "s4";
    node->device_id = "Measurer";

    cout << node->time << " ";
    cout << node->event_type << " ";
    cout << node->event_id << " ";
    cout << node->device_id << endl;

    node->next = NULL;

```

```

        insert(&head, node);
        msr = 1;
    }

void msr_go_up() //Запуск движения проверочного стржня вверх"
{

    node = new Node;
    node->time = systime + rand() % 10+1;
    node->event_type = "Запуск движения проверочного стржня
вверх";
    node->event_id = "s5";
    node->device_id = "Measurer";

    cout << node->time << " ";
    cout << node->event_type << " ";
    cout << node->event_id << " ";
    cout << node->device_id << endl;

    node->next = NULL;
    insert(&head, node);
    msr = 0;
}

void shedule_new_detal() //Запланировать появление новой детали
{

    node = new Node;
    node->time = systime + 1 + rand() % 10+100;
    node->event_type = "Сгенерирована новая деталь";
    node->event_id = "s1";
    node->device_id = "Application Generator";

    cout << node->time << " ";
    cout << node->event_type << " ";
    cout << node->event_id << " ";
    cout << node->device_id << endl;

    node->next = NULL;
    insert(&head, node);
}

void new_detal_proc_begin() //Начало обработки детали
{
    cout << "Процедура: начало обработки новой детали" << endl;
    TS = 0;
    if (detal->color == 0)
    {

        //node->device_id = "Slidegate valve";
        action("C_EXTEND_TRUE");
        SGV_go_forward();
    }
}

```

```

    }

    else
    {

        //node->device_id = "Lift";
        action("C_LIFT_UP_TRUE");
        lift_go_up();
    }

}

void s1() //Алгоритм обработки события s1 "Сгенерирована новая
деталь"
{
    generate(&detal);

    if (TS == 0)
    {
        insert_queue(&first, &last, detal);
        cout << "занесение в очередь-----"
<< endl;
    }
    else //станция готова
    {
        shedule_new_detal();
        new_detal_proc_begin();
    }
}

void s2() //Алгоритм обработки события s2 "Выталкиватель достиг
крайнего (выдвинутого) положения"
{
    action("S_END");
    action("C_EXTEND_FALSE");
    action("C_RETRACT_TRUE");

    SGV_go_back();
}

void s3() //Алгоритм обработки события s3 "Выталкиватель достиг
крайнего (втянутого) положения"
{
    action("S_END");
    action("C_RETRACT_FALSE");

    if (lift == 1)
    {
        action("C_LIFT_DOWN_TRUE");
        lift_go_down();
    }
    else

```

```

{
    TS = 1;

    if ((detal = take_queue(&first, &last)) != NULL)
    {

        cout << "-----Деталь вынута из очереди:
";

        cout << detal->height ;
        cout << detal->color << endl;
        new_detal_proc_begin();
        //shedule_new_detal();
    }

}

void s4() //Алгоритм обработки события s4 "Измерительный стержень
достиг крайнего (выдвинутого) положения"
{
    action("S_MEASURER_DOWN");
    action("C_MEASURER_DOWN_FALSE");
    action("C_MEASURER_UP_TRUE");
    msr_go_up();
}

void s5() //Алгоритм обработки события s5 " Измерительный стержень
достиг крайнего (втянутого) положения"
{
    action("S_MEASURER_UP");
    action("C_MEASURER_UP_FALSE");

    if (detal->height == 0)
    {
        action("S_HEIGHT_FALSE");
        action("C_LIFT_DOWN_TRUE");
        lift_go_down();
    }
    else
    {
        action("S_HEIGHT_TRUE");
        action("C_EXTEND_TRUE");
        SGV_go_forward();
    }
}

void s6() //Алгоритм обработки события s6 " Лифт достиг крайнего
верхнего положения"
{

```

```

        action("S_LIFT_UP");
        action("C_LIFT_UP_FALSE");
        node->device_id = "Measurer";
        action("C_MEASURER_DOWN_TRUE ");
        lift = 1;
        msr_go_down();
    }

void s7() //Алгоритм обработки события s7 " Лифт достиг крайнего
нижнего положения"
{
    action("S_LIFT_DOWN");
    action("S_LIFT_DOWN_FALSE ");

    lift = 0;
    if (detal->height == 0)
    {
        //action("S_HEIGHT_FALSE");
        action("C_EXTEND_TRUE");
        SGV_go_forward();
    }
    else // Цикл обработки закончен (деталь годна)
    {

        TS = 1;

        if ((detal = take_queue(&first, &last)) != NULL)
        {

            cout << "-----Деталь вынута из очереди: ";
            cout << detal->height;
            cout << detal->color << endl;
            new_detal_proc_begin();
            //shedule_new_detal();
        }

    };
}

int main()
{
    setlocale(LC_ALL, "Russian");
    srand(time(NULL));

    out1.open("out.txt"); // открываем файл для записи
    if (!out1.is_open()) { printf("\n Ошибка"); getchar();
    exit(0); }
    out1 << "ProcessId, Timestamp, Component, Action" << endl;

```

```

s1();
head1 = head;
while (1)
{
    if (head == NULL){ cout << "Моделирование завершено, т.к.
СБС пуст" << endl; break; };
    if (systime >= lim){ cout << "Моделирование завершено,
т.к. исчерпан лимит времени" << endl; break; }

    systime = head->time; //скачок времени

    if (head->event_id == "s1")
    {
        cout << "Обработка события s1" << endl; s1();
    }
    else {
        if (head->event_id == "s2")
        {
            cout << "Обработка события s2" << endl; s2();
        }
        else {
            if (head->event_id == "s3")
            {
                cout << "Обработка события s3" << endl;
s3();
            }
            else {
                if (head->event_id == "s4")
                {
                    cout << "Обработка события s4" <<
endl;    s4();
                }
                else{
                    if (head->event_id == "s5")
                    {
                        cout << "Обработка события s5"
<< endl;    s5();
                    }
                    else{
                        if (head->event_id == "s6")
                        {
                            cout << "Обработка события
s6" << endl;    s6();
                        }
                        else{
                            if (head->event_id == "s7")
                            {
                                cout << "Обработка
события s7" << endl;    s7();
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

        }
    }
}

    }

    head = head->next;

}

cout << "-----" << endl;

while (1) // ВЫВОД ПОЛЕЙ
{
    if (head1 == NULL) break;
    cout << head1->time << " ";
    cout << head1->event_type << " ";
    cout << head1->event_id << " ";
    cout << head1->device_id << endl;
    head1 = head1->next;

}

out1.close();
getchar();
}

```

Приложение Б

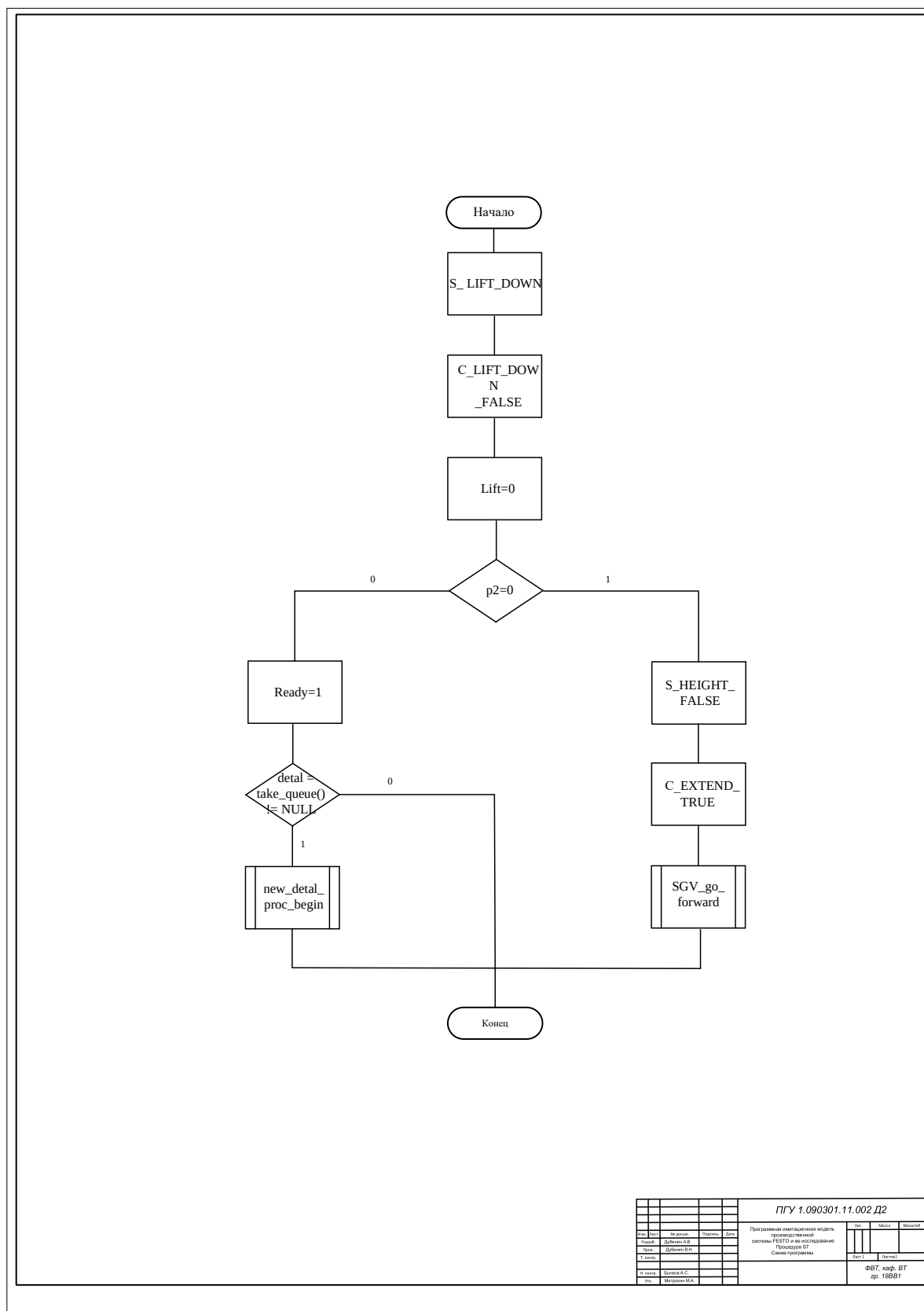
(обязательное)

Программная имитационная модель производственной
системы *FESTO* и ее исследование

Схемы программ

					ПГУ 1.090301.11.001 ПЗ	Лист
Изм.	Лист	№ докум	Подпись	Дата		89

Приложение Б.1 – Процедура S7. Схема программы



ПГУ 1.090301.11.002 Д2									
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Должность	Должность	Должность	Должность	Должность	Должность	Должность	Должность	Должность	Должность
Подпись	Подпись	Подпись	Подпись	Подпись	Подпись	Подпись	Подпись	Подпись	Подпись
И.О.	И.О.	И.О.	И.О.	И.О.	И.О.	И.О.	И.О.	И.О.	И.О.
И.О.	И.О.	И.О.	И.О.	И.О.	И.О.	И.О.	И.О.	И.О.	И.О.

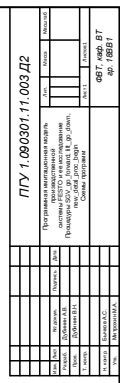
Изм.	Лист	№ докум	Подпись	Дата

ПГУ 1.090301.11.001 ПЗ

Лист

90

new_detal_proc_begin. Схемы программ



Приложение В

(обязательное)

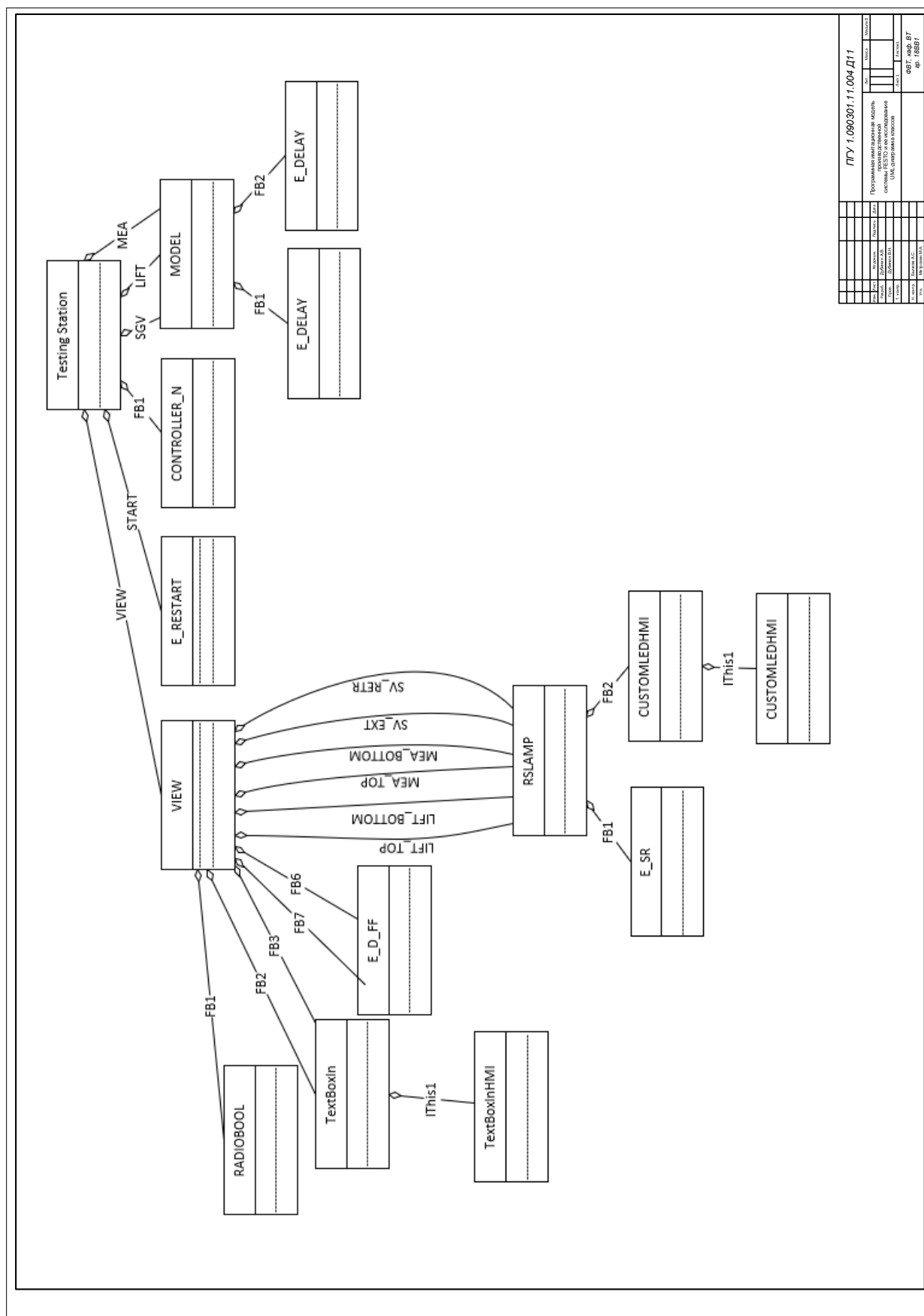
Программная имитационная модель производственной

системы *FESTO* и ее исследование

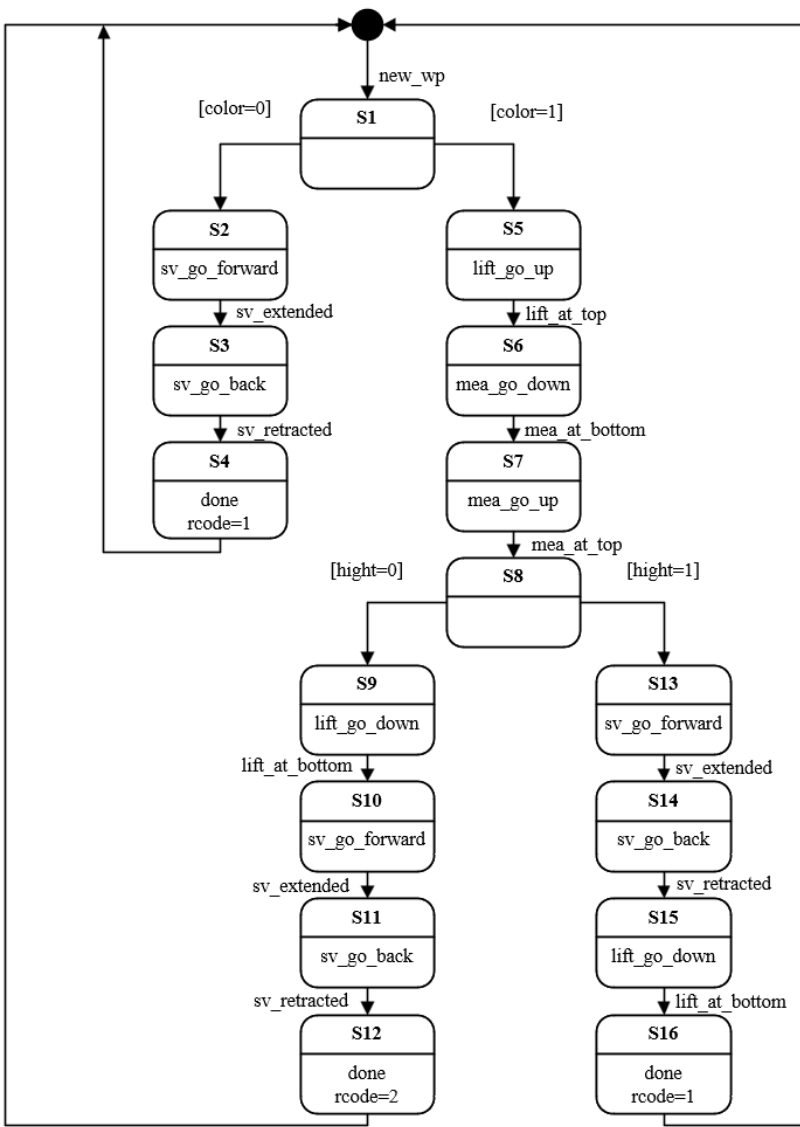
UML-диаграммы

					ПГУ 1.090301.11.001 ПЗ	Лист
Изм.	Лист	№ докум	Подпись	Дата		92

Приложение В.1–UML-диаграмма классов



Приложение В.2– UML-диаграмма состояний



ПГУ 1.090301.11.005 Д16									
Имя	Иван	Иванов	Иванов	Иванов	Иванов	Иванов	Иванов	Иванов	Иванов
Фамилия	Иванов	Иванов	Иванов	Иванов	Иванов	Иванов	Иванов	Иванов	Иванов
Имя	Иван	Иванов	Иванов	Иванов	Иванов	Иванов	Иванов	Иванов	Иванов
Фамилия	Иванов	Иванов	Иванов	Иванов	Иванов	Иванов	Иванов	Иванов	Иванов
Имя	Иван	Иванов	Иванов	Иванов	Иванов	Иванов	Иванов	Иванов	Иванов
Фамилия	Иванов	Иванов	Иванов	Иванов	Иванов	Иванов	Иванов	Иванов	Иванов
Имя	Иван	Иванов	Иванов	Иванов	Иванов	Иванов	Иванов	Иванов	Иванов
Фамилия	Иванов	Иванов	Иванов	Иванов	Иванов	Иванов	Иванов	Иванов	Иванов
Имя	Иван	Иванов	Иванов	Иванов	Иванов	Иванов	Иванов	Иванов	Иванов

Приложение Г

(обязательное)

Программная имитационная модель производственной
системы *FESTO* и ее исследование

Презентация

					ПГУ 1.090301.11.001 ПЗ	Лист
Изм.	Лист	№ докум	Подпись	Дата		95

ПРОГРАММНАЯ ИМИТАЦИОННАЯ МОДЕЛЬ ПРОИЗВОДСТВЕННОЙ СИСТЕМЫ FESTO И ЕЕ ИССЛЕДОВАНИЕ



Выполнил:
Студент группы 18ВВ1
Дубинин А.В.
Руководитель:
д.т.н., профессор каф. ВТ
Дубинин В.Н.

Актуальность



- В настоящее время ни одна сфера жизнедеятельности человека немыслима без *автоматизации*.
- Ведущее место в инфраструктуре экономик развитых стран принадлежит *промышленному производству*.
- Принятие концепции производства «*Индустрия 4.0*» как наиболее перспективной
- Производственные системы являются важным классом киберфизических систем
- Имитационное моделирование является одним из эффективных методов изучения и анализа сложных систем
- Перспективным подходом к синтезу формальных моделей является метод Process mining («процессная аналитика»)
- Привлекательным подходом для построения распределенных систем управления промышленными процессами является использование стандарта IEC 61499.
- Отсутствуют методики проектирования производственных систем, комбинирующих имитационное моделирование, процессную аналитику и стандарт МЭК 61499.



Цель работы

3



Цель работы - разработка программной и функционально-блочной имитационных моделей тестирующей станции производственной системы FESTO, а также их исследование с использованием процессной аналитики.



Производственная система FESTO

4



Производственная система (ПС) *FESTO* предназначена для изготовления некоторых законченных деталей из поступающих на ее вход заготовок. Она состоит из трех станций: распределительной (*Distribution Station*), тестирующей (*Testing Station*) и обрабатывающей (*Processing Station*)



Изм.	Лист	№ докум	Подпись	Дата

ПГУ 1.090301.11.001 ПЗ

Лист

97

Тестирующая станция FESTO

5

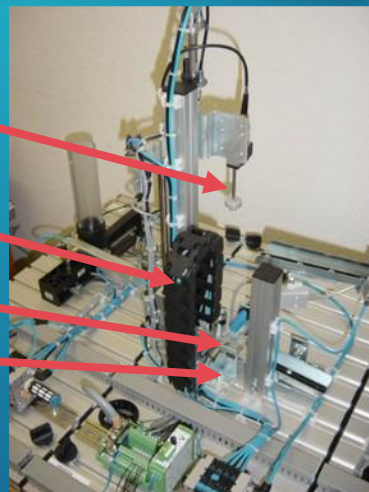


Измеритель высоты детали

Лифт

Выталкиватель

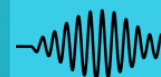
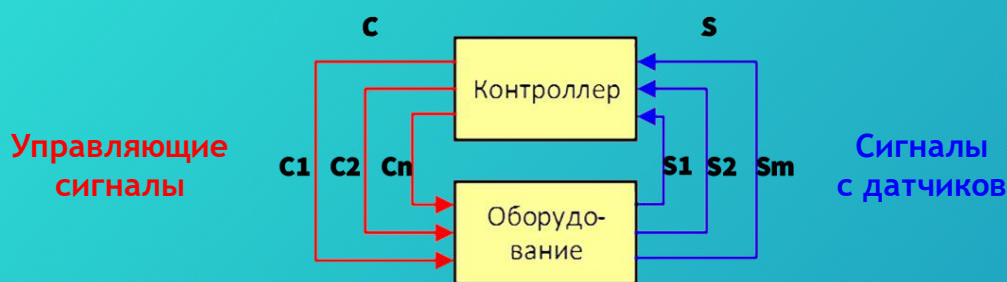
Блок детекторов



Основные результаты работы и их взаимосвязь

6

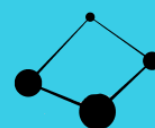




ПРИМЕР: Сигналы для лифта

C_LIFT_DOWN_TRUE - Подать сигнал опускания лифта;
C_LIFT_DOWN_FALSE - Снять сигнал опускания лифта;
C_LIFT_UP_TRUE - Подать сигнал поднятия лифта;
C_LIFT_UP_FALSE - Снять сигнал поднятия лифта;
S_LIFT_DOWN - Лифт находится в нижнем положении;
S_LIFT_UP - Лифт находится в верхнем положении

Граф событий и действий тестирующей станции



s1 - Сгенерирована новая деталь;
s2 - Выталкиватель достиг крайнего (выдвинутого) положения;
s3 - Выталкиватель достиг крайнего (втянутого) положения;
s4 - Измерительный стержень достиг крайнего (выдвинутого) положения;
s5 - Измерительный стержень достиг крайнего (втянутого) положения;
s6 - Лифт достиг крайнего верхнего положения;
s7 - Лифт достиг крайнего нижнего положения.



Алгоритм обработки события s7 “ Лифт достиг крайнего нижнего положения”

9



Выдача сигнала S_LIFT_DOWN - Лифт находится в нижнем положении
 Выдача сигнала C_LIFT_DOWN_FALSE - Снять сигнал опускания лифта
 Lift=0
 ЕСЛИ p2=0, ТО - Деталь не годна по высоте
 ДЕЛАТЬ
 Выдача сигнала S_HEIGHT_FALSE - Деталь не подходит по высоте
 Выдача сигнала C_EXTEND_TRUE - Выдвижение поршня вперед
 Запустить движение Выталкивателя вперед
 КОНЕЦ
 ИНАЧЕ - Закончить цикл обработки годной детали
 ДЕЛАТЬ
 Ready=1 -- Установить флаг «Станция свободна»
 Попытка выбрать очередную деталь из очереди деталей
 ЕСЛИ Деталь выбрана успешно, ТО Начало обработки новой детали
 КОНЕЦ

Фрагмент журнала событий для тестирующей станции в формате CSV

10



ProcessId, Timestamp, Component, Action

...

1, 954, Slidegate valve, C_EXTEND_FALSE
 1, 954, Slidegate valve, C_RETRACT_TRUE
 1, 963, Slidegate valve, S_END
 1, 963, Slidegate valve, C_RETRACT_FALSE
 2, 0, Lift, C_LIFT_UP_TRUE
 2, 1, Lift, S_LIFT_UP
 2, 1, Lift, C_LIFT_UP_FALSE
 2, 1, Measurer, C_MEASURER_DOWN_TRUE
 2, 6, Measurer, S_MEASURER_DOWN
 2, 6, Measurer, C_MEASURER_DOWN_FALSE

ЕСЛИ БЫ
КОТИК
 УМЕЛ КОДИТЬ,
 ЭТО БЫЛ БЫ
 САМЫЙ МИЛЫЙ

КОД



Изм.	Лист	№ докум	Подпись	Дата

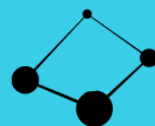
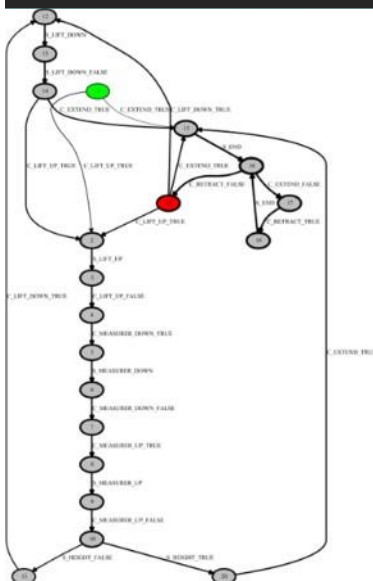
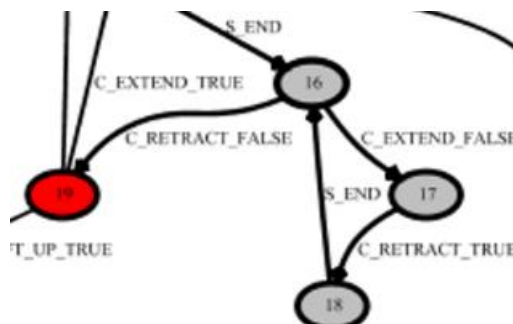
ПГУ 1.090301.11.001 ПЗ

Лист

100

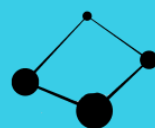
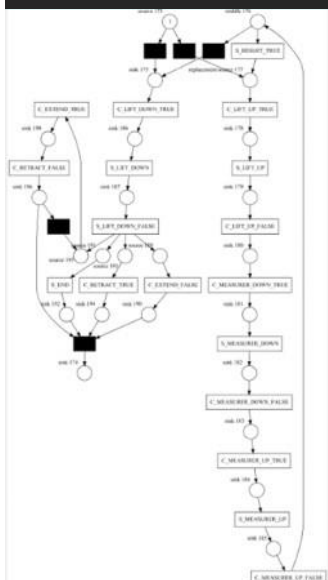
Конечно-автоматная модель тестирующей станции

11



Сетевая модель тестирующей станции

12

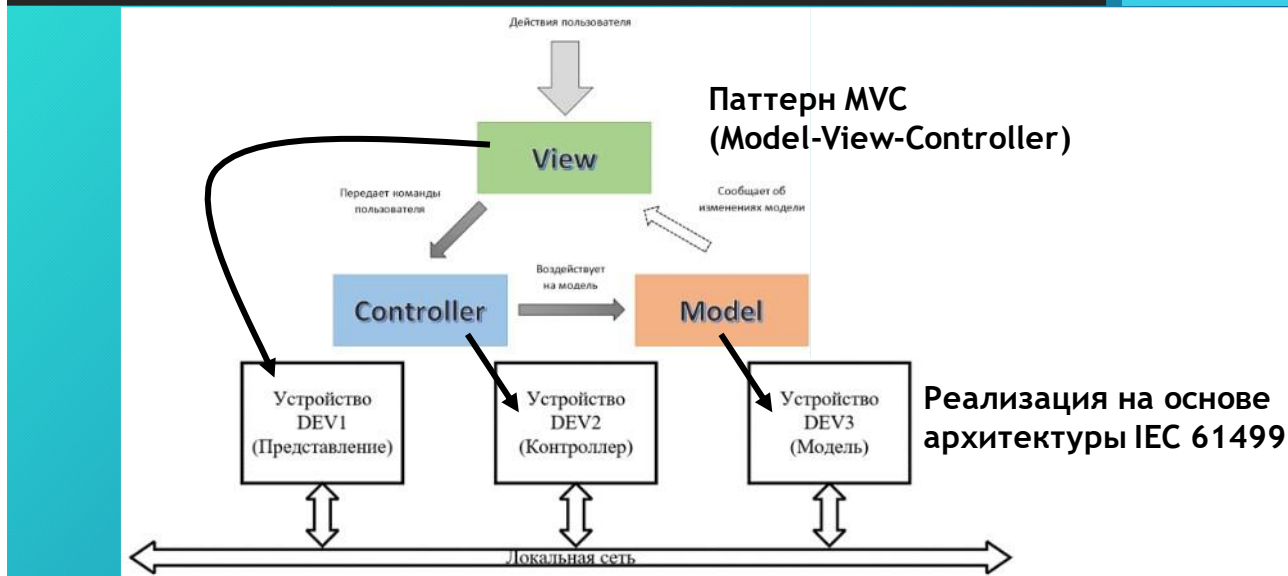
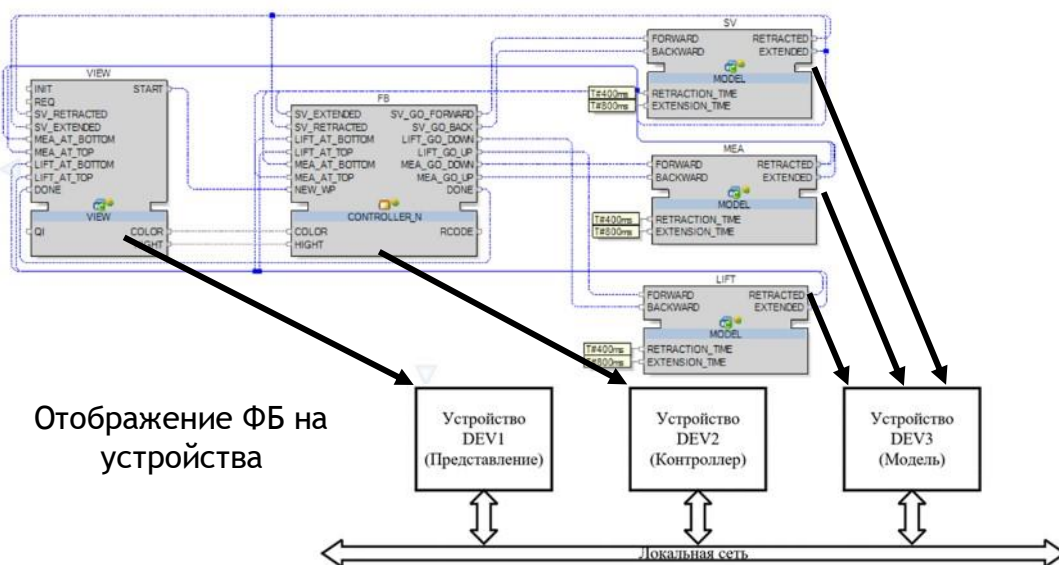


Генерация сети Петри в системе ProM (Process Mining).
- Анализ сети Петри в системе TINA (Time petri Net Analyzer)

Свойства сетевой модели

- граф достижимости содержит 28 вершин (маркировок) и 77 дуг
- сеть ограниченная
- сеть неживая и нереверсивная (т.к. имеются потенциально живые переходы)
- имеет 4 Р-инварианта и 4 Тинварианта

Вывод: алгоритмы функционирования тестирующей станции, заложенные в имитационную модель, корректны



Функциональный блок для управления тестирующей станцией

15



Интерфейс

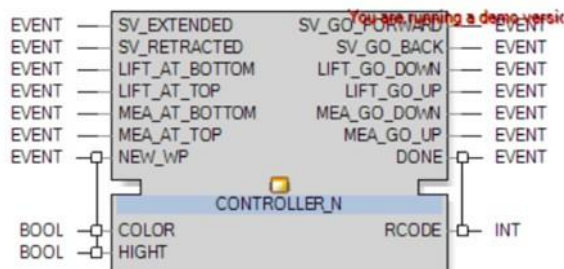
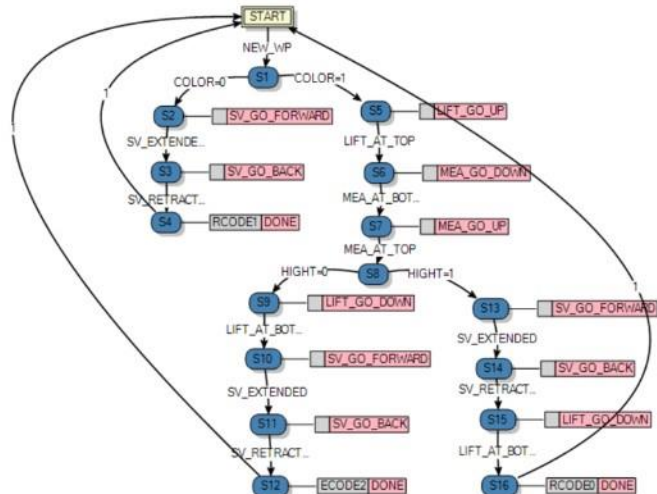
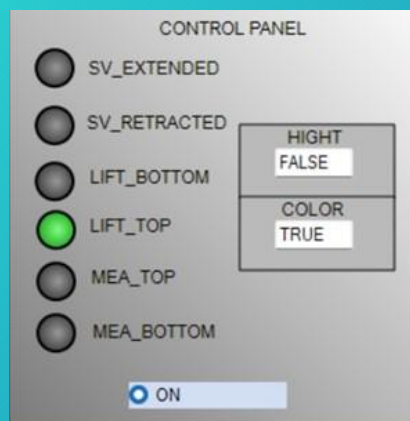
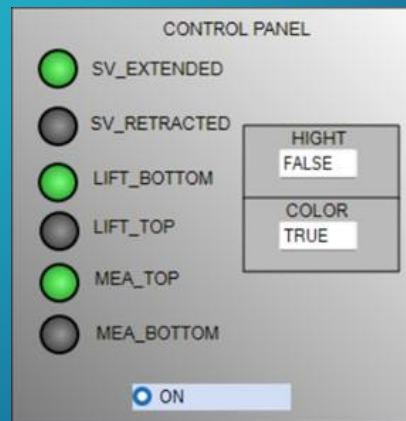


Диаграмма управления выполнением (диаграмма ECC)



Работа программы при сценарии «хороший цвет, плохая высота»

16





- разработана программная имитационная модель тестирующей станции производственной системы *FESTO*, позволяющая записывать журнал событий;
- по журналам событий, полученных в ходе имитационных экспериментов, с использованием технологии *Process mining* сгенерированы формальные модели процессов, протекающих в тестирующей станции производственной системы *FESTO*, а также произведено их исследование;
- с использованием паттерна проектирования *MVC* разработана (имитационная) функционально-блочная модель тестирующей станции производственной системы *FESTO* на основе международного стандарта *IEC 61499*, причем управляющая компонента может быть перенесена на реальную систему практически без изменений.

