ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ РФ

Московский институт электроники и математики
Национального исследовательского университета
Высшая Школа Экономики

Кафедра Вычислительных систем и сетей

Курсовая работа

на тему:

Разработка программного обеспечения для моделирования цифровых комбинационных схемотехнических устройств

по курсу
“Схемотехника”

Студент гр. СВ-71 _____ Собко С.С.

Преподаватель _____ Черноусова Т.Г.

Москва 2014

Содержание

Введение

В настоящее время существует некоторый спрос на релизацию решений по автоматизации синтеза различного рода сущностей, ранее формировавшихся в ручном или полуавтоматическом режимах. К таковым, например, можно отнести логические схемы комбинационных схемотехнических устройств [1].

Промышленные стандарты и решения для синтеза таких устройств (например, VHDL и Verilog) не позволяют в полной мере достичь гибкости в разработке и описании схемотехнических устройств с помощью принципов объектно-ориентированного программирования и использования таких инструментов как графы и символьная алгебра логики. Невозможность разложить синтезированную схему на отдельные логические вентили, влечет за собой невозможность автоматизированной смены базиса и разрядности используемых логических вентилей схем.

Эти недостатки не являются критичными, но увеличивают время синтеза нетиповых комбинационных схем по сравнению с потенциально возможной автоматической генерацией по параметрически заданным количествам входных и выходных портов с помощью объектно-ориентированного программирования.

1 Цель работы

Цель работы – изучение логических методов синтеза комбинационных схемотехнических устройств и разработка программного обеспечения для автоматического синтеза по заданным параметрам входных и выходных сигналов для представления устройств в различных форматах.

Помимо разработки непосредственно библиотеки, пригодной для использования программистами, необходимо разработать интерфейс пользователя, с помощью которого можно задавать параметры разрядности входов и выходов устройств, получая на выходе результаты преобразования в текстовом (таблицы истинности, код MATLAB, код LaTeX) и графическом (условно-графические обозначения, графы) виде.

2 Обзор существующих решений

Существуют аналогичные реализуемому в рамках курсовой работы решения, в том числе на языке программирования Python. Из наиболее схожих по функционалу можно выделить три решения – MyHDL (`http://www.myhdl.org/`), Chips (`http://dawsonjon.github.io/chips/`) и BinPy (`http://www.binpy.org/`). Первые два делают упор на симуляции схемотехнических устройств и позволяют генерировать VHDL и Verilog описания, но не позволяют представить общую схему устройства в виде графов [2, 3]. Третье наиболее схоже с реализуемым проектом, но опять же не позволяет производить декомпозицию синтезируемых устройств на отдельные логические вентили, из-за чего возникает невозможность использования готовой кодовой базы для реализации новых устройств с помощью наследования [4].

3 Исходные данные

Классы устройств, которые необходимо реализовать в рамках проекта:

- Логические вентили И, ИЛИ, НЕ.
- Мультиплексоры и демультиплексоры.
- Сумматоры.
- Цифровые компараторы.

Типы обработчиков (адаптеров) для вывода объектов устройств:

- Графы.
- Модели MATLAB / Simulink.
- Условно-графические обозначения.
- Таблицы истинности.
- Таблицы истинности и логические формулы в формате LaTeX.

4 Описание программного решения

4.1 Разработка программной библиотеки

Исходя из ограничений существующих решений, было решено в рамках проекта реализовать свою библиотеку для синтеза цифровых комбинационных схем на языке программирования Python с использованием библиотеки символьной алгебры логики SymPy и библиотеки для работы с графами NetworkX.

Декомпозиция проекта при проектировании проведена таким образом, чтобы разделить сущности устройств синтезируемых схем и адаптеров для отображения схем в различных форматах. Независимость сущностей дает возможность применять однотипные способы описания устройств к совершенно различным синтезированным устройствам, построенным на единой парадигме символьной алгебры.

Для наглядной демострации функциональных возможностей разрабатываемой библиотеки синтеза схемотехнических устройств, также был разработан интерфейс пользователя на базе языка программирования Python с использованием фреймворка Flask и языка программирования JavaScript с использованием фреймворка ExtJS 4. Программное обеспечение представляет собой веб-приложение и может быть запущено с использованием сети Интернет. Единственной особенностью связки с основной библиотекой схемотехнического моделирования, является механизм интроспекции классов устройств и классов адаптеров, автоматический импорт используемых сущностей и унификация вывода информации в приложение.

4.2 Графический интерфейс пользователя

Основной интерфейс пользователя представляет собой таблицу с перечнем типов устройств, синтез которых поддерживается библиотекой проекта, кнопку настройки сигналов устройств, выпадающий список выбора адаптера и кнопку обработки. На рисунке 1 представлена заглавная страница интерфейса.

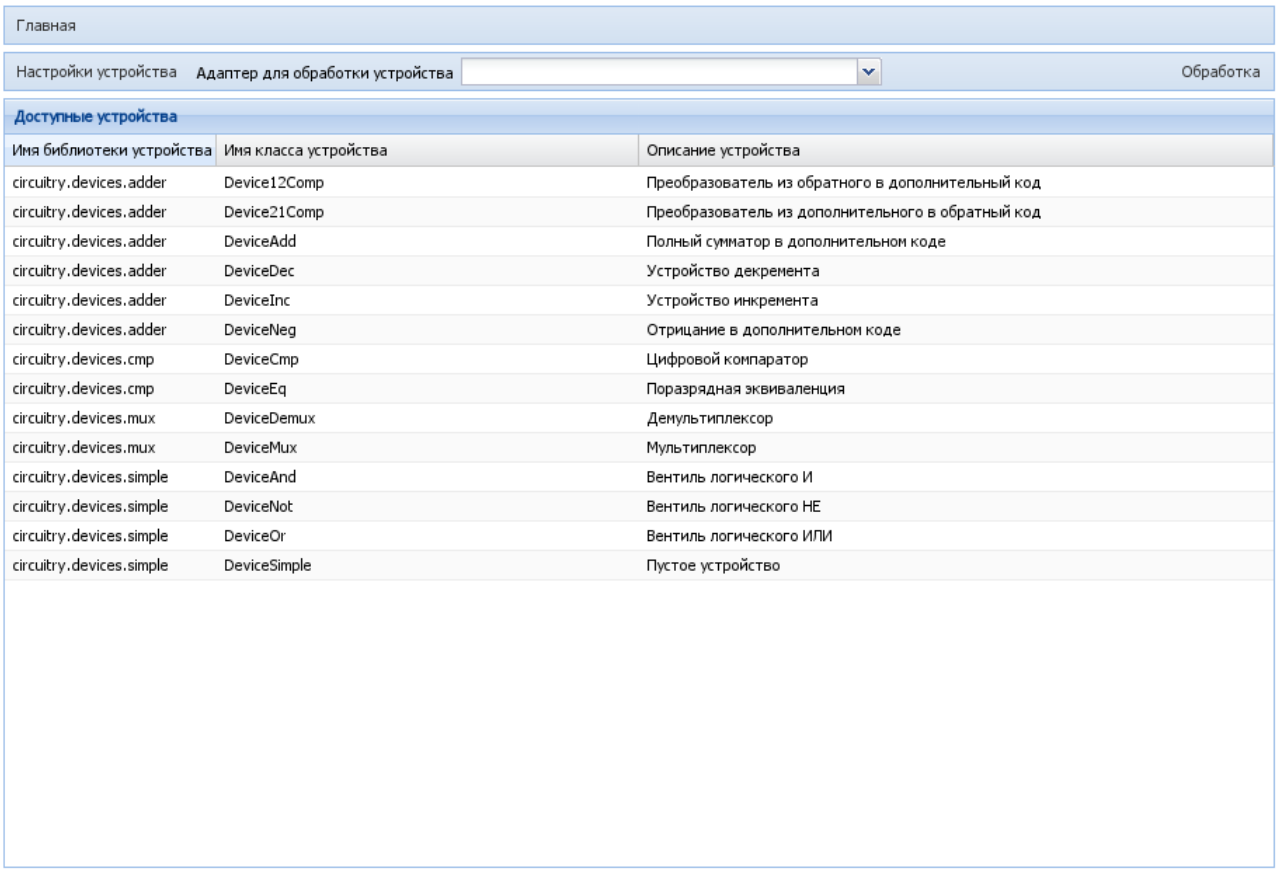

Рисунок 1 — Заглавная страница интерфейса пользователя

При выборе устройства из списка и нажатии на кнопку настройки сигналов устройства, открывается окно настроек, где с использованием полей ввода можно задать сигналы устройства. На рисунке 2 представлено окно настройки сигналов синтезируемого устройства.

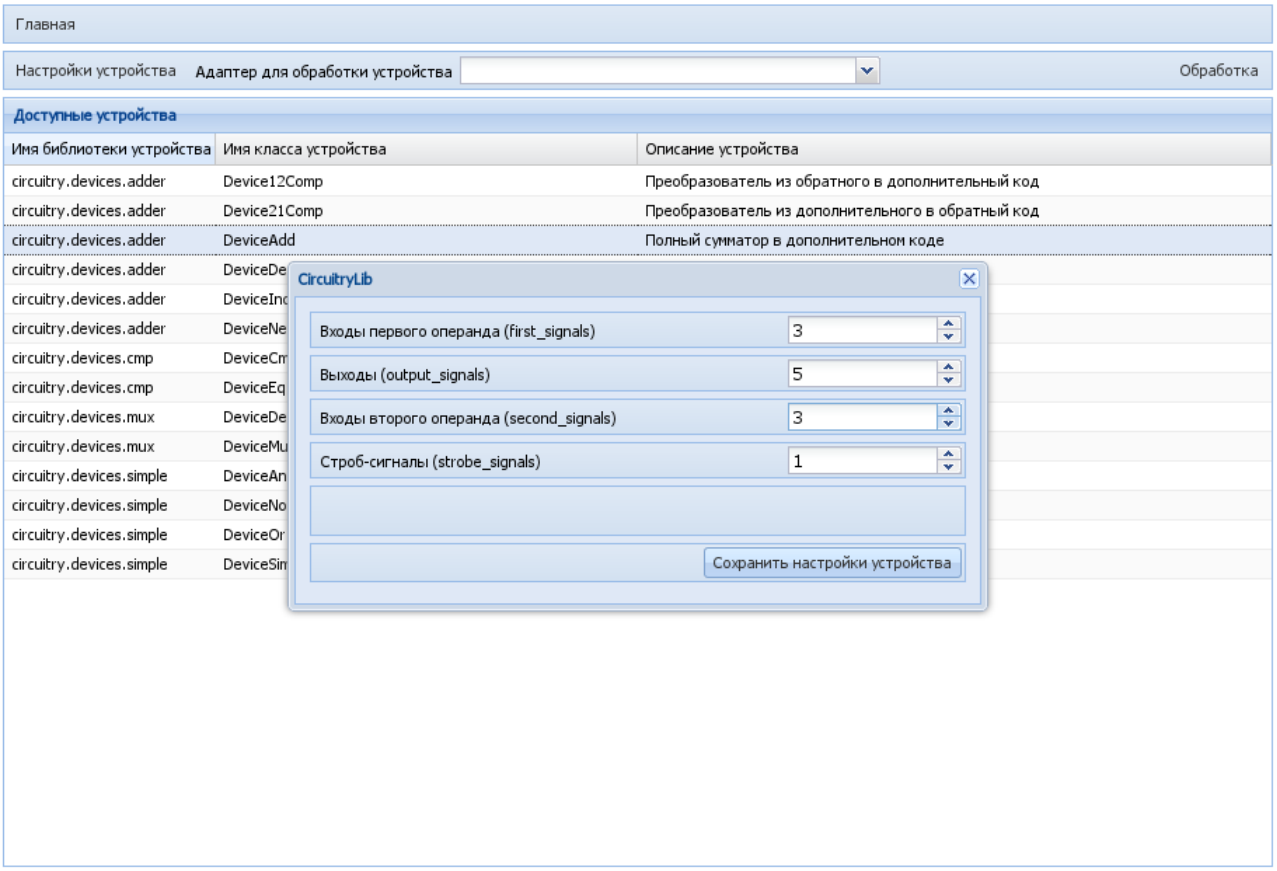

Рисунок 2 — Окно настройки сигналов синтезируемого устройства

После выбора адаптера, возвращающего текстовый вывод и нажатия на кнопку обработки, открывается окно с текстовым полем, в котором находится результат применения адаптера к синтезируемому устройству. Пример представлен на рисунке 3.

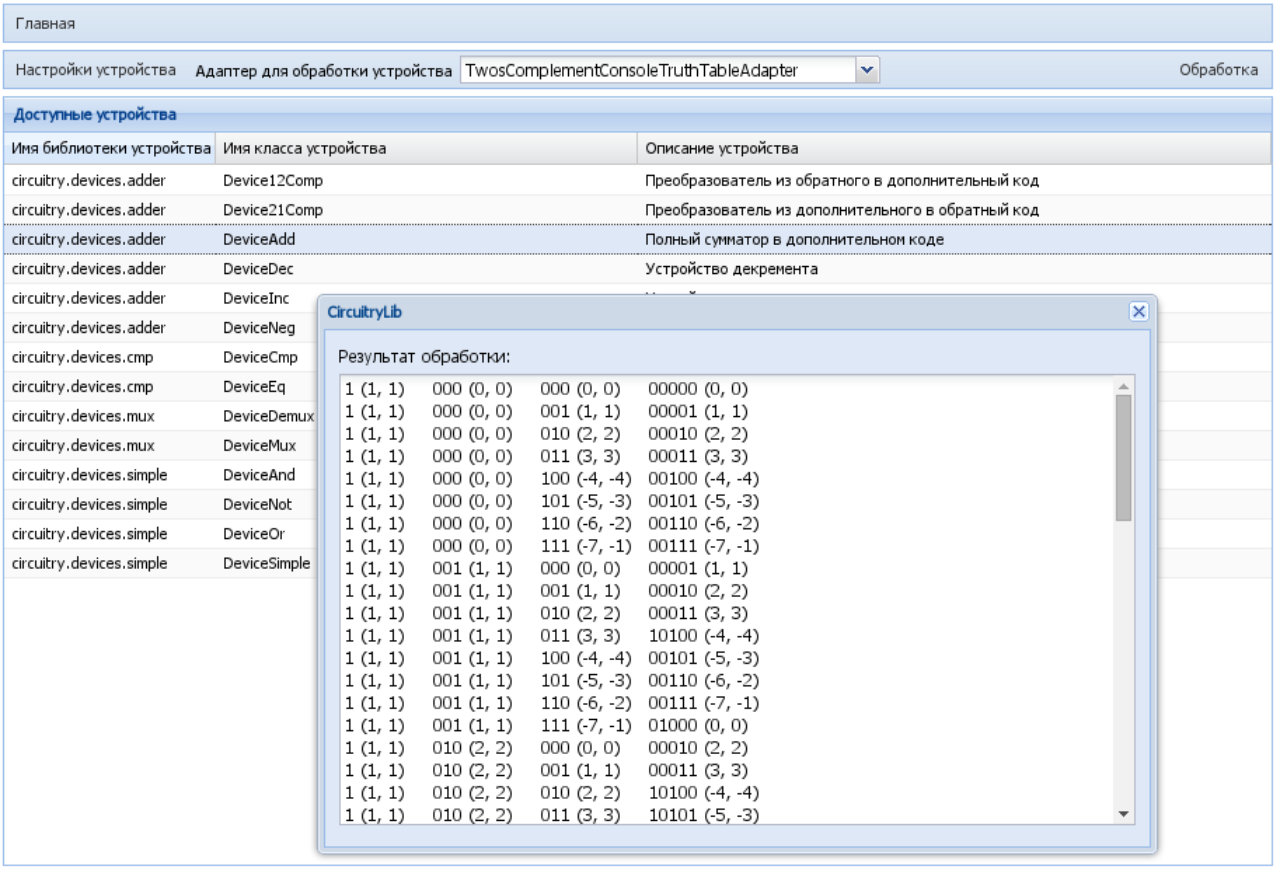

Рисунок 3 — Текстовый результат применения адаптера к синтезируемому устройству

В случае, если адаптер возвращает графический результат, окно результата будет содержать изображение. Пример представлен на рисунке 4.

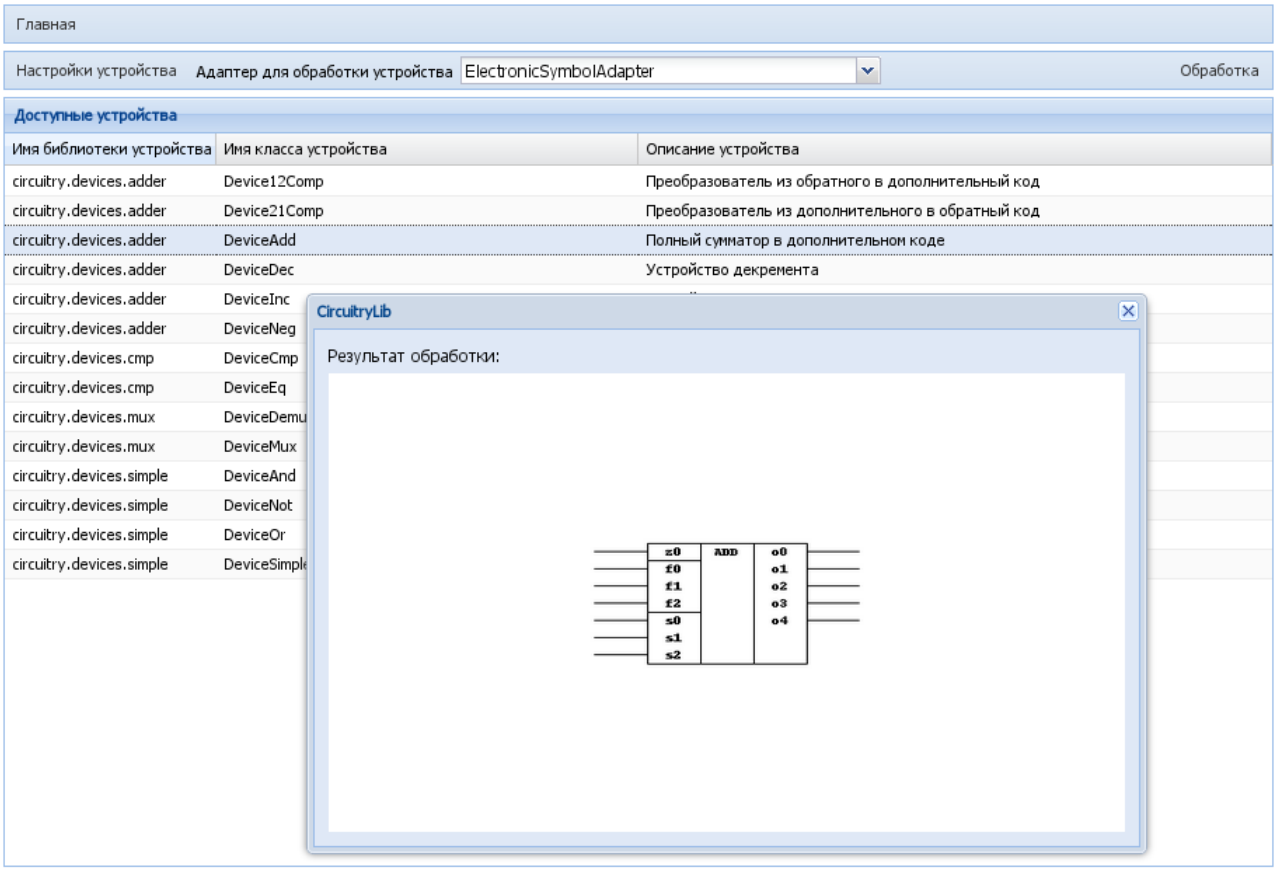

Рисунок 4 — Графический результат применения адаптера к синтезируемому устройству

4.3 Реализованные типы синтезируемых устройств

Следующие типы устройств, синтез которых производится созданием экземпляров приведенных классов, были реализованы в рамках курсовой работы (в названиях опущен префикс библиотеки – circuitry):

- devices.simple.DeviceNot – логический вентиль НЕ (инвертор) с возможностью поразрядной инверсии;
- devices.simple.DeviceAnd – логический вентиль И (логическое умножение, конъюнкция);
- devices.simple.DeviceOr – логический вентиль ИЛИ (логическое сложение, дизъюнкция);
- devices.mux.DeviceMux – мультиплексор;
- devices.mux.DeviceDemux – демультиплексор;
- devices.adder.DeviceAdd – полный сумматор;
- devices.adder.DeviceInc – инкремент;
- devices.adder.DeviceDec – декремент;
- devices.adder.Device12Comp – устройство преобразования из обратного кода в дополнительный;
- devices.adder.Device21Comp – устройство преобразования из дополнительного кода в обратный;
- devices.adder.DeviceNeg – отрицание в дополнительном коде;
- devices.cmp.DeviceEq – эквиваленция (поразрядное сравнение);
- devices.cmp.DeviceCmp – цифровой компаратор.

4.4 Реализованные типы адаптеров

Следующие типы адаптеров для обработки и преобразования к необходимым форматам синтезированных устройств были реализованы в рамках проекта (в названиях опущен префикс библиотеки – circuitry):

- adapters.matlab.MatlabAdapter – адаптер для формирования кода модели Simulink;
- adapters.matlab.extended.ExtendedMatlabAdapter – расширенный адаптер для формирования кода модели Simulink, включая константы для входных сигналов и отображение значений выходных сигналов;
- adapters.visual.ElectronicSymbolAdapter – адаптер для генерации условно-графического обозначения синтезируемого устройства;
- adapters.latex.LatexTruthTableAdapter – адаптер для генерации таблиц истинности синтезируемых устройств в формате LaTeX;
- adapters.latex.mux.DeviceMuxLatexTruthTableAdapter – адаптер для генерации таблиц истинности мультиплексоров и демультиплексоров в формате LaTeX;
- adapters.graph.GraphAdapter – адаптер для представления внутренних структур синтезируемых устройств в виде графов библиотеки NetworkX;
- adapters.console.ConsoleTruthTableAdapter – адаптер для генерации таблиц истинности в текстовом виде;
- adapters.console.TwosComplementaryConsoleTruthTableAdapter – адаптер для генерации таблиц истинности в текстовом виде с отображением десятичных значений в обратном и дополнительном кодах.

5 Использование библиотеки для синтеза устройств

В разделе представлены способы использования библиотеки для синтеза типов устройств.

Общий принцип синтеза устройств с использованием программной библиотеки – создание экземпляра класса соответствующего устройства с заполнением количества входных и выходных типов сигналов, перечисленных в классе устройства обязательными к заполнению.

5.1 Логические вентили

Логический вентиль – базовый элемент цифровой схемы, выполняющий элементарную логическую операцию, преобразуя таким образом множество входных логических сигналов в выходной логический сигнал. Логика работы вентиля основана на битовых операциях с входными цифровыми сигналами в качестве операндов. При создании цифровой схемы вентили соединяют между собой, при этом выход используемого вентиля должен быть подключен к одному или к нескольким входам других вентилей [5].

Вся кодовая база логических вентилей находится в пакете devices.simple и наследуется от абстрактного класса DeviceSimple.

В разрабатываемой библиотеке на настоящий момент присутствует техническое ограничение, не позволяющее задавать базис и ограничение на количество разрядов логических вентилей в составных устройствах. Это ограничение связано с использованием библиотеки символьной алгебры, не предоставляющей функционал ленивых логических вычислений. Все логические выражения символьной алгебры в составных устройствах раскрываются и упрощаются автоматически [6].

5.1.1 Логический вентиль НЕ (инвертор)

Логический вентиль НЕ (инвертор) с возможностью поразрядной инверсии можно описать следующим образом:

$$\forall x : O_x = \overline{I_x} \tag{1}$$

В выражении 1:

- x – индекс входного разряда;
- I – набор входных разрядов;
- O – набор выходных разрядов.

Описание логического вентиля НЕ в разрабатываемой библиотеке представлено в листинге 1.

```
class DeviceNot(DeviceSimple):
    """Logic NOT gate"""
    logic_function = Not
    constraints = {
        'data_signals': {
            'min': 1,
            'max': 10
        },
        'output_signals': {
            'min': 1,
            'max': 10
        }
    }

    @property
    def functions(self):
        return [self.logic_function(signal) for signal in self.data_signals]
```

Листинг 1 — Программное описание класса логического вентиля НЕ

В листинге 2 представлен код для программного синтеза устройства инвертора с 3 входными разрядами данных (data_signals='d:3') и 3 выходными разрядами (output_signals='o:3').

```
>>> from circuitry.devices.simple import DeviceNot
>>> pprint(
...     DeviceNot(data_signals='d:3', output_signals='o:3')
... )
{'data_signals': (d0, d1, d2),
 'output_signals': (o0, o1, o2),
 'truth_table': [([0, 0, 0], [1, 1, 1]),
                 ([1, 0, 0], [0, 1, 1]),
                 ([0, 1, 0], [1, 0, 1]),
                 ([1, 1, 0], [0, 0, 1]),
                 ([0, 0, 1], [1, 1, 0]),
                 ([1, 0, 1], [0, 1, 0]),
                 ([0, 1, 1], [1, 0, 0]),
                 ([1, 1, 1], [0, 0, 0])]}
```

Листинг 2 — Программный синтез логического вентиля НЕ

На рисунке 5 представлено условно-графическое обозначение синтезированного вентиля НЕ.

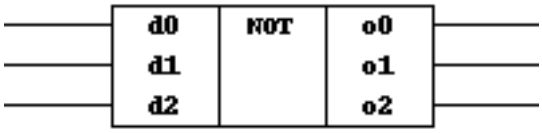

Рисунок 5 — Условно-графическое обозначение вентиля НЕ

5.1.2 Логический вентиль И (конъюнкция)

Логический вентиль И (логическое умножение, конъюнкция) описывается следующим образом:

$$O_0 = I_0 \wedge I_2 \wedge \ldots \wedge I_{n-1} \tag{2}$$

В выражении 2:

- n – количество входных разрядов;
- I – набор входных разрядов;
- O_0 – выходной разряд.

Описание логического вентиля И в разрабатываемой библиотеке представлено в листинге 3.

```
class DeviceAnd(DeviceSimple):
    """Logic AND gate"""
    logic_function = And
    constraints = {
        'data_signals': {
            'min': 2,
            'max': 10
        },
        'output_signals': {
            'min': 1,
            'max': 1
        }
    }
```

Листинг 3 — Программное описание класса логического вентиля И

В листинге 4 представлен код для программного синтеза устройства логического умножения с 2 входными разрядами данных (data_signals='d:2') и 1 прямым выходным разрядом (output_signals='o:1').

```
>>> from circuitry.devices.simple import DeviceAnd
>>> pprint(
...     DeviceAnd(data_signals='d:2', output_signals='o:1',
...               output_signals_subs=dict(o0=1))
... )
{'data_signals': (d0, d1),
 'output_signals': (o0,),
 'output_signals_function': o0,
 'output_signals_subs': {'o0': 1},
 'output_signals_truth_table': [1],
 'truth_table': [([0, 0], [0]), ([1, 0], [0]), ([0, 1], [0]), ([1, 1], [1])]}
```

Листинг 4 — Программный синтез логического вентиля И

На рисунке 6 представлено условно-графическое обозначение синтезированного вентиля И.

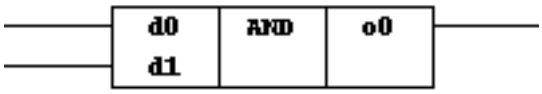

Рисунок 6 — Условно-графическое обозначение вентиля И

5.1.3 Логический вентиль ИЛИ (дизъюнкция)

Логический вентиль ИЛИ (логическое сложение, дизъюнкция) описывается следующим образом:

$$O_0 = I_0 \vee I_2 \vee \ldots \vee I_{n-1} \tag{3}$$

В выражении 3:

- n – количество входных разрядов;
- I – набор входных разрядов;
- O_0 – выходной разряд.

Описание логического вентиля ИЛИ в разрабатываемой библиотеке представлено в листинге 5.

```
class DeviceOr(DeviceSimple):
    """Logic OR gate"""
    logic_function = Or
    constraints = {
        'data_signals': {
            'min': 2,
            'max': 10
        },
        'output_signals': {
            'min': 1,
            'max': 1
        }
    }
```

Листинг 5 — Программное описание класса логического вентиля ИЛИ

В листинге 6 представлен код для программного синтеза устройства логического сложения с 3 входными разрядами данных (data_signals='d:3') и 1 прямым выходным разрядом (output_signals='o:1').

```
>>> from circuitry.devices.simple import DeviceOr
>>> pprint(
...     DeviceOr(data_signals='d:3', output_signals='o:1',
...              output_signals_subs=dict(o0=1))
... )
{'data_signals': (d0, d1, d2),
 'output_signals': (o0,),
 'output_signals_function': o0,
 'output_signals_subs': {'o0': 1},
 'output_signals_truth_table': [1],
 'truth_table': [([0, 0, 0], [0]),
                 ([1, 0, 0], [1]),
                 ([0, 1, 0], [1]),
                 ([1, 1, 0], [1]),
                 ([0, 0, 1], [1]),
                 ([1, 0, 1], [1]),
                 ([0, 1, 1], [1]),
                 ([1, 1, 1], [1])]}
```

Листинг 6 — Программный синтез логического вентиля ИЛИ

На рисунке 7 представлено условно-графическое обозначение синтезированного вентиля ИЛИ.

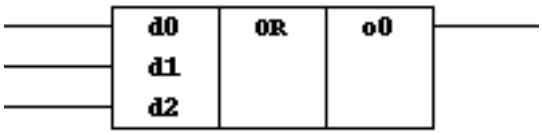

Рисунок 7 — Условно-графическое обозначение вентиля ИЛИ

5.2 Мультиплексоры и демультиплексоры

Мультиплексор – устройство, имеющее несколько сигнальных входов, один или более управляющих входов и один выход. Мультиплексор позволяет передавать сигнал с одного из входов на выход; при этом выбор желаемого входа осуществляется подачей соответствующей комбинации управляющих сигналов. Устройство, противоположное мультиплексору по своей функции, называется демультиплексором [7].

Вся кодовая база мультиплексоров и демультиплексоров находится в пакете devices.mux и наследуется от абстрактного класса Device.

5.2.1 Мультиплексор

Мультиплексор можно описать следующим образом:

$$\forall x : (D_x = 1) \wedge (A = x) \Leftrightarrow (O_0 = 1) \tag{4}$$

В выражении 4:

- x – индекс входного разряда данных;
- D – набор входных разрядов данных;
- A – набор входных разрядов адреса;
- O_0 – выходной разряд.

Описание мультиплексора в разрабатываемой библиотеке представлено в листинге 7.

```
class DeviceMux(Device):
    """Multiplexer device"""
    mandatory_signals = ('strobe_signals', 'address_signals',
                         'data_signals', 'output_signals',)
    mandatory_signals_using_subs = ('strobe_signals', 'output_signals',)
    truth_table_signals = ('strobe_signals', 'address_signals',
                           'data_signals', 'output_signals',)
    constraints = {
        'strobe_signals': {
            'min': 1,
            'max': 10
        },
        'address_signals': {
            'min': 1,
            'max': 5
        },
        'data_signals': {
            'min': 1,
            'max': 32
        },
        'output_signals': {
            'min': 1,
            'max': 1
        }
    }
```

Листинг 7 — Программное описание класса мультиплексора

В листинге 8 представлен код для программного синтеза мультиплексора с 4 входными разрядами данных (data_signals='d:4'), 2 входными разрядами адреса (address_signals='a:2'), 1 прямым входным разрядом строб-сигнала (strobe_signals='z:1') и 1 прямым выходным разрядом (output_signals='o:1').

```
>>> from circuitry.devices.mux import DeviceMux
>>> pprint(
...     DeviceMux(strobe_signals='z:1', address_signals='a:2',
...               data_signals='d:4', output_signals='o:1',
...               strobe_signals_subs=dict(z0=1),
...               output_signals_subs=dict(o0=1))
... )
{'address_and_data_function': Or(And(Not(a0), Not(a1), Not(d1),
                                 Not(d2), Not(d3), d0), And(Not(a0),
                                 Not(d0), Not(d1), Not(d3), a1, d2),
                                 And(Not(a1), Not(d0), Not(d2), Not(d3), a0, d1),
                                 And(Not(d0), Not(d1), Not(d2), a0, a1, d3)),
 'address_signals': (a0, a1),
 'data_signals': (d0, d1, d2, d3),
 'functions': [And(Or(
                      And(Not(a0), Not(a1), Not(d1), Not(d2), Not(d3), d0),
                      And(Not(a0), Not(d0), Not(d1), Not(d3), a1, d2),
                      And(Not(a1), Not(d0), Not(d2), Not(d3), a0, d1),
                      And(Not(d0), Not(d1), Not(d2), a0, a1, d3)), z0)
              ],
 'output_signals': (o0,),
 'output_signals_function': o0,
 'output_signals_subs': {'o0': 1},
 'output_signals_truth_table': [1],
 'strobe_signals': (z0,), 'strobe_signals_function': z0,
 'strobe_signals_subs': {'z0': 1},
 'strobe_signals_truth_table': [1],
 'truth_table': [([1], [0, 0], [1, 0, 0, 0], [1]),
                 ([1], [1, 0], [0, 1, 0, 0], [1]),
                 ([1], [0, 1], [0, 0, 1, 0], [1]),
                 ([1], [1, 1], [0, 0, 0, 1], [1])]}
```

Листинг 8 — Программный синтез мультиплексора

На рисунке 8 представлено условно-графическое обозначение синтезированного мультиплексора.

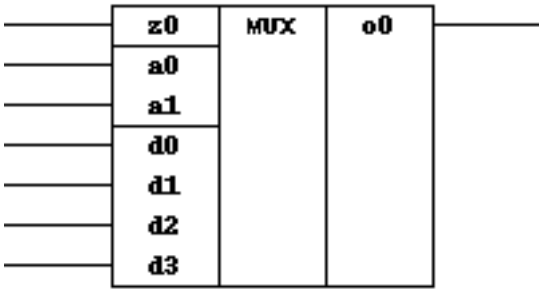

Рисунок 8 — Условно-графическое обозначение мультиплексора

На рисунке 9 представлено устройство синтезированного мультиплексора в виде модели Simulink.

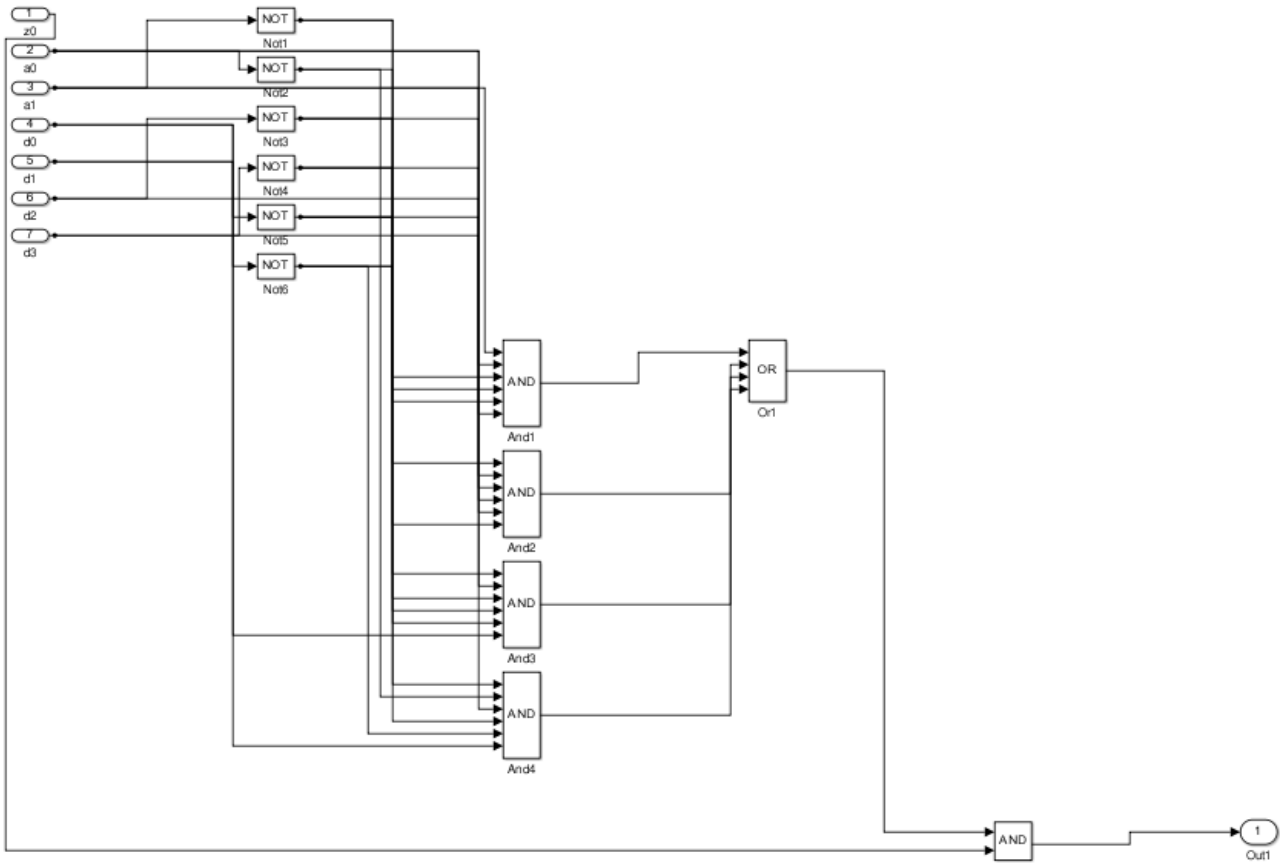

Рисунок 9 — Представление мультиплексора в виде модели Simulink

5.2.2 Демультиплексор

Деультиплексор можно описать следующим образом:

$$\forall x : (D_0 = 1) \wedge (A = x) \Leftrightarrow (O_x = 1) \tag{5}$$

В выражении 5:

- x – индекс выходного разряда;
- D_0 – входной разряд данных;
- A – набор входных разрядов адреса;
- O – набор выходных разрядов.

Описание демультиплексора в разрабатываемой библиотеке представлено в листинге 9.

```
class DeviceDemux(Device):
    """Demultiplexer device"""
    mandatory_signals = ('strobe_signals', 'address_signals',
                         'data_signals', 'output_signals',)
    mandatory_signals_using_subs = ('strobe_signals', 'data_signals',
                                    'output_signals')
    truth_table_signals = ('strobe_signals', 'address_signals',
                           'data_signals', 'output_signals',)
    constraints = {
        'strobe_signals': {
            'min': 1,
            'max': 10
        },
        'address_signals': {
            'min': 1,
            'max': 5
        },
        'data_signals': {
            'min': 1,
            'max': 1
        },
        'output_signals': {
            'min': 1,
            'max': 32
        }
    }
```

Листинг 9 — Программное описание класса демультиплексора

В листинге 10 представлен код для программного синтеза демультиплексора с 1 входным разрядом данных (data_signals='d:1'), 2 входными разрядами адреса (address_signals='a:2'), 1 прямым входным разрядом строб-сигнала (strobe_signals='z:1') и 4 прямыми выходными разрядами (output_signals='o:4').

```
>>> from circuitry.devices.mux import DeviceDemux
>>> pprint(
...     DeviceDemux(strobe_signals='z:1', address_signals='a:2',
...                 data_signals='d:1', output_signals='o:4',
...                 strobe_signals_subs=dict(z0=1),
...                 data_signals_subs=dict(d0=1),
...                 output_signals_subs=dict(o0=1, o1=1, o2=1, o3=1))
... )
{'address_signals': (a0, a1),
 'data_signals': (d0,),
 'data_signals_function': d0,
 'data_signals_subs': {'d0': 1},
 'data_signals_truth_table': [1],
 'functions': [And(Not(a0), Not(a1), d0, z0),
               And(Not(a1), a0, d0, z0),
               And(Not(a0), a1, d0, z0),
               And(a0, a1, d0, z0)],
 'output_signals': (o0, o1, o2, o3),
 'output_signals_function': And(o0, o1, o2, o3),
 'output_signals_subs': {'o0': 1, 'o1': 1, 'o2': 1, 'o3': 1},
 'output_signals_truth_table': [1, 1, 1, 1],
 'strobe_signals': (z0,),
 'strobe_signals_function': z0,
 'strobe_signals_subs': {'z0': 1},
 'strobe_signals_truth_table': [1],
 'truth_table': [([1], [0, 0], [1], [1, 0, 0, 0]),
                 ([1], [1, 0], [1], [0, 1, 0, 0]),
                 ([1], [0, 1], [1], [0, 0, 1, 0]),
                 ([1], [1, 1], [1], [0, 0, 0, 1])]}
```

Листинг 10 — Программный синтез демультиплексора

На рисунке 10 представлено условно-графическое обозначение синтезированного демультиплексора.

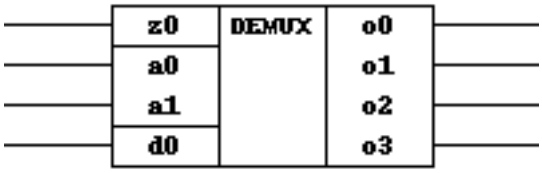

Рисунок 10 — Условно-графическое обозначение демультиплексора

На рисунке 11 представлено устройство синтезированного демультиплексора в виде модели Simulink.

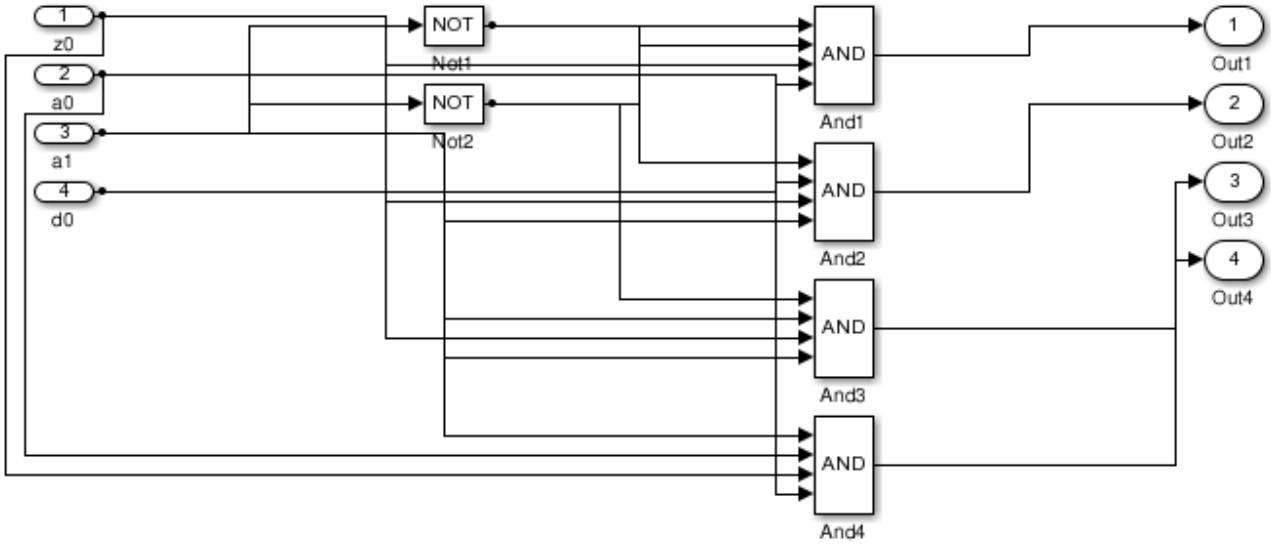

Рисунок 11 — Представление демультиплексора в виде модели Simulink

5.3 Сумматоры

Сумматор – устройство, преобразующее информационные сигналы (аналоговые или цифровые) в сигнал, эквивалентный сумме этих сигналов. Полные сумматоры – тринарные (трёхоперандные) сумматоры по модулю с разрядом переноса, характеризующиеся наличием трёх входов, на которые подаются одноимённые разряды двух складываемых чисел и перенос из предыдущего (более младшего) разряда, и двумя выходами: на одном реализуется арифметическая сумма по модулю в данном разряде, а на другом — перенос в следующий (более старший разряд) [8].

Вся кодовая база сумматоров находится в пакете devices.adder и наследуется от абстрактного класса Device.

5.3.1 Параллельный сумматор

Параллельный (многоразрядный) сумматор можно описать следующим образом:

$$\forall x : A_x = F_x \oplus S_x \oplus C_{x-1} \tag{6}$$

$$\forall x : C_x = (F_x \wedge S_x) \vee (F_x \wedge C_{x-1}) \vee (S_x \wedge C_{x-1}) \tag{7}$$

$$\forall y : O_y = \begin{cases} A_y & \text{при } y < n; \\ C_{y-1} & \text{при } y \equiv n; \\ C_{y-2} \oplus C_{y-1} & \text{при } y \equiv n+1. \end{cases} \tag{8}$$

В выражениях 6, 7 и 8:

- x – индекс входного разряда первого и второго операнда;
- y – индекс выходного разряда;
- n – количество входных разрядов первого и второго операнда;
- F – набор входных разрядов первого операнда;
- S – набор входных разрядов второго операнда;
- C – разряды переноса;
- A – разряды суммы;
- O – выходные разряды сумматора.

Основным классом, реализующим функционал сумматора (характерной особенностью которого является возможность генерации двух дополнительных выходных сигналов – переноса и переполнения), является DeviceAdd. Описание сумматора в разрабатываемой библиотеке представлено в листинге 11.

```
class DeviceAdd(Device):
    """Adder device"""
    mandatory_signals = ('strobe_signals', 'first_signals',
                         'second_signals', 'output_signals',)
    mandatory_signals_using_subs = ('strobe_signals',)
    truth_table_signals = ('strobe_signals', 'first_signals',
                           'second_signals', 'output_signals',)
    constraints = {
        'strobe_signals': {
            'min': 1,
            'max': 10
        },
        'first_signals': {
            'min': 1,
            'max': 5
        },
        'second_signals': {
            'min': 1,
            'max': 5
        },
        'output_signals': {
            'min': 1,
            'max': 7
        }
    }
```

Листинг 11 — Программное описание класса сумматора

В листинге 12 представлен код для программного синтеза параллельного сумматора с 2 входными разрядами первого операнда (first_signals='f:2'), 2 входными разрядами второго операнда (second_signals='s:2'), 1 прямым входным разрядом строб-сигнала (strobe_signals='z:1') и 4 выходными разрядами (output_signals='o:4').

```
>>> from circuitry.devices.adder import DeviceAdd
>>> pprint(
...     DeviceAdd(strobe_signals='z:1', first_signals='a:2',
...               second_signals='s:2', output_signals='o:4',
...               strobe_signals_subs=dict(z0=1))
... )
{'first_signals': (a0, a1),
 'functions': [And(Or(And(Not(a0), s0),
                      And(Not(s0), a0)), z0),
               And(Or(And(Not(a1), Or(And(Not(s1), a0, s0),
                                      And(Or(Not(a0), Not(s0)), s1))),
                      And(Or(And(a0, s0), Not(s1)),
                          Or(Not(a0), Not(s0), s1), a1)), z0),
               And(Or(And(a0, a1, s0), And(a0, s0, s1),
                      And(a1, s1)), z0),
               And(Or(And(Or(And(a0, a1, s0),
                             And(a0, s0, s1), And(a1, s1)),
                          Or(Not(a0), Not(s0))),
                      And(Or(Not(a0), Not(a1),
                             Not(s0)),
                          Or(Not(a0), Not(s0), Not(s1)),
                          Or(Not(a1), Not(s1)), a0, s0)), z0)],
 'output_signals': (o0, o1, o2, o3),
 'second_signals': (s0, s1),
 'strobe_signals': (z0,),
 'strobe_signals_function': z0,
 'strobe_signals_subs': {'z0': 1},
 'strobe_signals_truth_table': [1],
 'truth_table': [([1], [0, 0], [0, 0], [0, 0, 0, 0]),
                 ([1], [0, 0], [1, 0], [1, 0, 0, 0]),
                 ([1], [0, 0], [0, 1], [0, 1, 0, 0]),
                 ([1], [0, 0], [1, 1], [1, 1, 0, 0]),
                 ([1], [1, 0], [0, 0], [1, 0, 0, 0]),
                 ([1], [1, 0], [1, 0], [0, 1, 0, 1]),
                 ([1], [1, 0], [0, 1], [1, 1, 0, 0]),
                 ([1], [1, 0], [1, 1], [0, 0, 1, 0]),
                 ([1], [0, 1], [0, 0], [0, 1, 0, 0]),
                 ([1], [0, 1], [1, 0], [1, 1, 0, 0]),
                 ([1], [0, 1], [0, 1], [0, 0, 1, 1]),
                 ([1], [0, 1], [1, 1], [1, 0, 1, 1]),
                 ([1], [1, 1], [0, 0], [1, 1, 0, 0]),
                 ([1], [1, 1], [1, 0], [0, 0, 1, 0]),
                 ([1], [1, 1], [0, 1], [1, 0, 1, 1]),
                 ([1], [1, 1], [1, 1], [0, 1, 1, 0])]}
```

Листинг 12 — Программный синтез сумматора

На рисунке 12 представлено условно-графическое обозначение синтезированного сумматора.

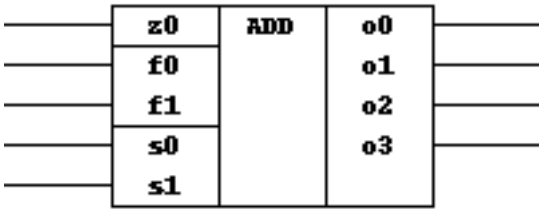

Рисунок 12 — Условно-графическое обозначение сумматора

На рисунке 13 представлено устройство синтезированного сумматора в виде модели Simulink.

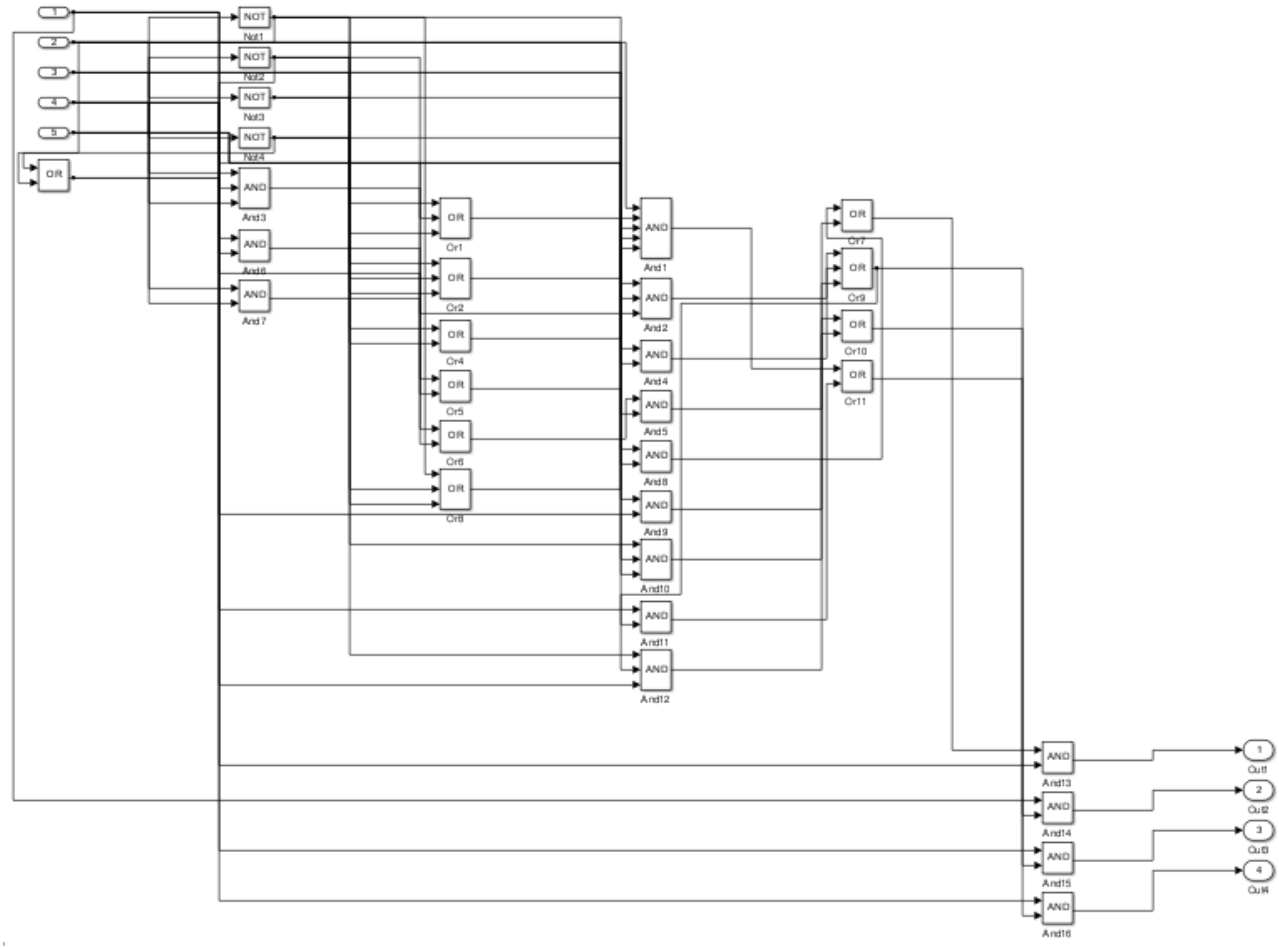

Рисунок 13 — Представление сумматора в виде модели Simulink

5.3.2 Устройство инкремента

Устройство инкремента описывается следующим образом на базе параллельного сумматора с использованием формул из выражений 6, 7 и 8:

$$\forall x : F_x = D_x \tag{9}$$

$$\forall x : S_x = \begin{cases} 1 & \text{при } x \equiv 0; \\ 0 & \text{при } x > 0. \end{cases} \tag{10}$$

В выражениях 9, и 10:

- x – индекс входного разряда устройства инкремента;
- D – набор входных разрядов устройства инкремента;
- F – набор входных разрядов первого операнда сумматора;
- S – набор входных разрядов второго операнда сумматора;

Описание устройства инкремента в разрабатываемой библиотеке представлено в листинге 13.

```
class DeviceInc(Device):
    """Increment device"""
    mandatory_signals = ('strobe_signals', 'data_signals', 'output_signals',)
    mandatory_signals_using_subs = ('strobe_signals',)
    truth_table_signals = ('strobe_signals', 'data_signals', 'output_signals',)
    constraints = {
        'strobe_signals': {
            'min': 1,
            'max': 10
        },
        'data_signals': {
            'min': 1,
            'max': 5
        },
        'output_signals': {
            'min': 1,
            'max': 7
        }
    }
```

Листинг 13 — Программное описание класса инкремента

В листинге 14 представлен код для программного синтеза устройства инкремента с 2 входными разрядами данных (data_signals='d:2'), 1 прямым входным разрядом строб-сигнала (strobe_signals='z:1') и 2 выходными разрядами (output_signals='o:2').

```
>>> from circuitry.devices.adder import DeviceInc
>>> pprint(
...     DeviceInc(strobe_signals='z:1', data_signals='a:2',
...               output_signals='o:2', strobe_signals_subs=dict(z0=1))
... )
{'data_signals': (a0, a1),
 'functions': [And(Not(a0), z0),
               And(Or(And(Not(a0), a1), And(Not(a1), a0)), z0)],
 'output_signals': (o0, o1),
 'strobe_signals': (z0,),
 'strobe_signals_function': z0,
 'strobe_signals_subs': {'z0': 1},
 'strobe_signals_truth_table': [1],
 'truth_table': [([1], [0, 0], [1, 0]),
                 ([1], [1, 0], [0, 1]),
                 ([1], [0, 1], [1, 1]),
                 ([1], [1, 1], [0, 0])]}
```

Листинг 14 — Программный синтез устройства инкремента

На рисунке 14 представлено условно-графическое обозначение синтезированного устройства инкремента.

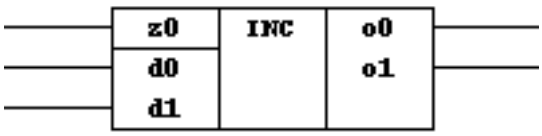

Рисунок 14 — Условно-графическое обозначение устройства инкремента

На рисунке 15 представлено устройство синтезированного инкремента в виде модели Simulink.

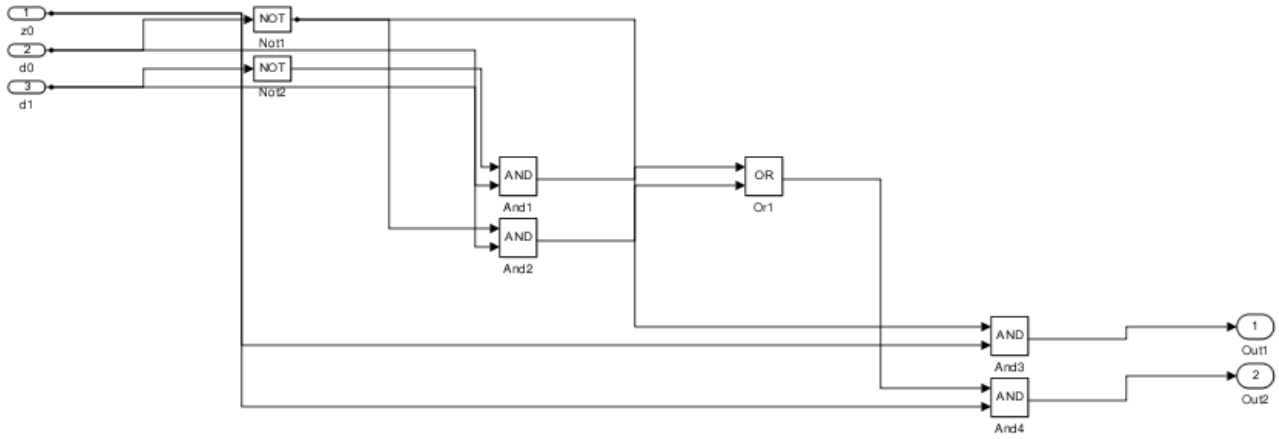

Рисунок 15 — Представление инкремента в виде модели Simulink

5.3.3 Устройство декремента

Устройство декремента описывается следующим образом на базе параллельного сумматора с использованием формул из выражений 6, 7 и 8:

$$\forall x : F_x = D_x \tag{11}$$

$$\forall x : S_x = 1 \tag{12}$$

В выражениях 11, и 12:

- x – индекс входного разряда устройства декремента;
- D – набор входных разрядов устройства декремента;
- F – набор входных разрядов первого операнда сумматора;
- S – набор входных разрядов второго операнда сумматора;

Описание устройства инкремента в разрабатываемой библиотеке представлено в листинге 15.

```
class DeviceDec(Device):
    """Decrement device"""
    mandatory_signals = ('strobe_signals', 'data_signals', 'output_signals',)
    mandatory_signals_using_subs = ('strobe_signals',)
    truth_table_signals = ('strobe_signals', 'data_signals', 'output_signals',)
    constraints = {
        'strobe_signals': {
            'min': 1,
            'max': 10
        },
        'data_signals': {
            'min': 1,
            'max': 5
        },
        'output_signals': {
            'min': 1,
            'max': 7
        }
    }
```

Листинг 15 — Программное описание класса декремента

В листинге 16 представлен код для программного синтеза устройства декремента с 2 входными разрядами данных (data_signals='d:2'), 1 прямым входным разрядом строб-сигнала (strobe_signals='z:1') и 2 выходными разрядами (output_signals='o:2').

```
>>> from circuitry.devices.adder import DeviceDec
>>> pprint(
...     DeviceDec(strobe_signals='z:1', data_signals='a:2',
...               output_signals='o:2', strobe_signals_subs=dict(z0=1))
... )
{'data_signals': (a0, a1),
 'functions': [And(Not(a0), z0), And(Or(Not(a0), a1), Or(Not(a1), a0), z0)],
 'output_signals': (o0, o1),
 'strobe_signals': (z0,),
 'strobe_signals_function': z0,
 'strobe_signals_subs': {'z0': 1},
 'strobe_signals_truth_table': [1],
 'truth_table': [([1], [0, 0], [1, 1]),
                 ([1], [1, 0], [0, 0]),
                 ([1], [0, 1], [1, 0]),
                 ([1], [1, 1], [0, 1])]}
```

Листинг 16 — Программный синтез устройства декремента

На рисунке 16 представлено условно-графическое обозначение синтезированного устройства декремента.

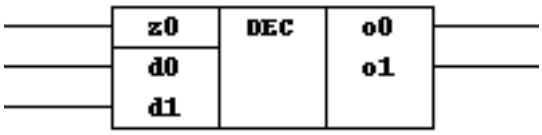

Рисунок 16 — Условно-графическое обозначение устройства декремента

На рисунке 17 представлено устройство синтезированного декремента в виде модели Simulink.

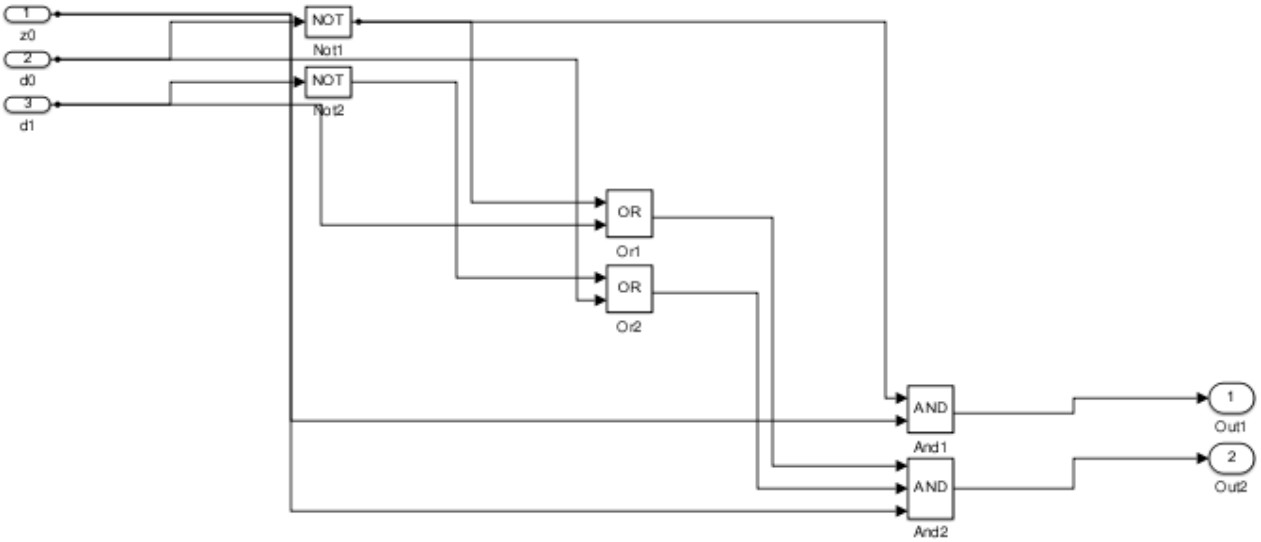

Рисунок 17 — Представление декремента в виде модели Simulink

5.3.4 Устройство отрицания в дополнительном коде

Устройство отрицания в дополнительном коде описывается следующим образом на базе двух устройств поразрядной инверсии, устройства инкремента и устройства декремента с использованием формул из выражений 1, 9, 10, 11 и 12:

$$\forall x : O_x = \begin{cases} inc_x(\overline{D}) & \text{при } D_{n-1} \equiv 0; \\ \overline{dec_x(D)} & \text{при } D_{n-1} \equiv 1. \end{cases} \tag{13}$$

В выражении 13:

- x – индекс входного разряда устройства отрицания;
- n – количество входных разрядов устройства отрицания;
- inc – функция устройства инкремента;
- dec – функция устройства декремента;
- D – набор входных разрядов устройства отрицания;

Описание устройства отрицания в разрабатываемой библиотеке представлено в листинге 17.

```
class DeviceNeg(Device):
    """Negation for two's complement"""
    mandatory_signals = ('strobe_signals', 'data_signals', 'output_signals',)
    mandatory_signals_using_subs = ('strobe_signals',)
    truth_table_signals = ('strobe_signals', 'data_signals', 'output_signals',)
    constraints = {
        'strobe_signals': {
            'min': 1,
            'max': 10
        },
        'data_signals': {
            'min': 1,
            'max': 5
        },
        'output_signals': {
            'min': 1,
            'max': 5
        }
    }
```

Листинг 17 — Программное описание устройства отрицания

В листинге 18 представлен код для программного синтеза устройства отрицания в дополнительном коде с 2 входными разрядами данных (data_signals='d:2'), 1 прямым входным разрядом строб-сигнала (strobe_signals='z:1') и 2 выходными разрядами (output_signals='o:2').

```
>>> from circuitry.devices.adder import DeviceNeg
>>> pprint(
...     DeviceNeg(strobe_signals='z:1', data_signals='d:2',
...               output_signals='o:2', strobe_signals_subs=dict(z0=1))
... )
{'data_signals': (d0, d1),
 'functions': [And(Or(And(Not(d1), Or(Not(z0), d0)), And(d0, d1, z0)), z0),
               And(Or(And(Not(d1), Or(And(Not(d0), d1),
                                      And(Not(d1), d0), Not(z0))),
                      And(Or(And(Not(d0), d1), And(Not(d1), d0)), d1, z0)),
                      z0)],
 'output_signals': (o0, o1),
 'strobe_signals': (z0,),
 'strobe_signals_function': z0,
 'strobe_signals_subs': {'z0': 1},
 'strobe_signals_truth_table': [1],
 'truth_table': [([1], [0, 0], [0, 0]),
                 ([1], [1, 0], [1, 1]),
                 ([1], [0, 1], [0, 1]),
                 ([1], [1, 1], [1, 0])]}
```

Листинг 18 — Программный синтез устройства отрицания

На рисунке 18 представлено условно-графическое обозначение синтезированного устройства отрицания в дополнительном коде.

Рисунок 18 — Условно-графическое обозначение устройства отрицания

На рисунке 19 представлено устройство синтезированного отрицания в виде модели Simulink.

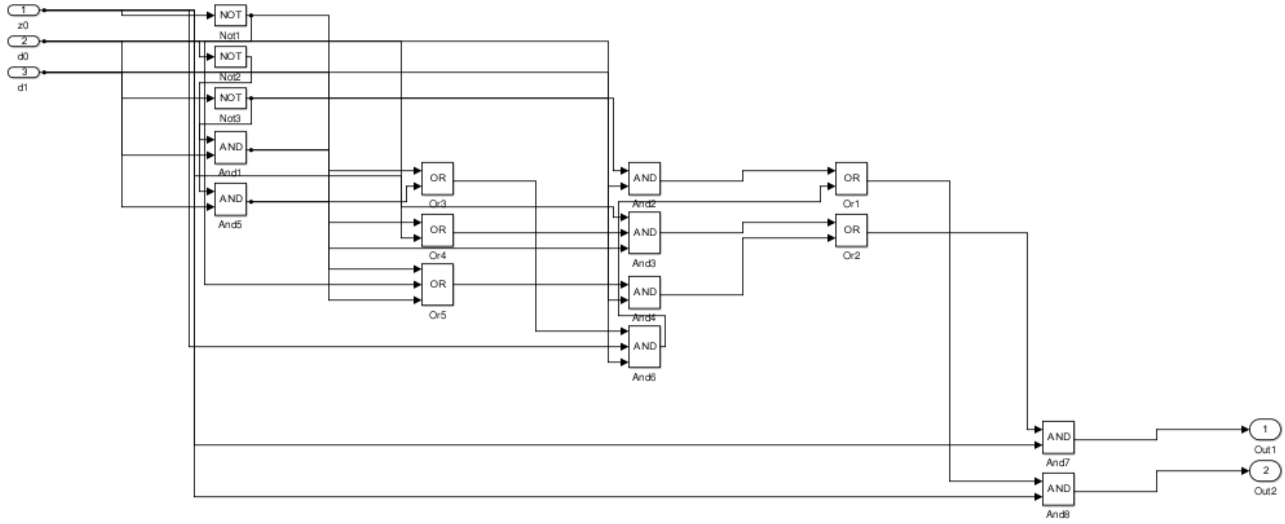

Рисунок 19 — Представление отрицания в виде модели Simulink

5.4 Цифровые компараторы

Цифровой компаратор или компаратор амплитуд является электронным устройством, берущим два числа в двоичном виде и определяющим, является ли первое число меньшим, большим или равным второму числу. Чтобы определить наибольшее из двух двоичных чисел, рассматриваются отношение величин пар значащих цифр, начиная с наиболее значащих битов, последовательно продвигаясь к младшим значащим битам до нахождения неравенства [9].

Вся кодовая база цифровых компараторов находится в пакете devices.cmp и наследуется от абстрактного класса Device.

5.4.1 Логическая эквиваленция

Устройство поразрядной эквиваленции описывается следующим образом:

$$\forall x : E_x = \overline{F_x \oplus S_x} \tag{14}$$

В выражении 14:

- x – индекс входного разряда первого и второго операнда;
- F – набор входных разрядов первого операнда;
- S – набор входных разрядов второго операнда;
- E – набор выходных разрядов.

Описание поразрядной эквиваленции в разрабатываемой библиотеке представлено в листинге 19.

```
class DeviceEq(Device):
    """Equality device"""
    mandatory_signals = ('strobe_signals', 'first_signals',
                         'second_signals', 'output_signals',)
    mandatory_signals_using_subs = ('strobe_signals',)
    truth_table_signals = ('strobe_signals', 'first_signals',
                           'second_signals', 'output_signals',)
    constraints = {
        'strobe_signals': {
            'min': 1,
            'max': 10
        },
        'first_signals': {
            'min': 1,
            'max': 7
        },
        'second_signals': {
            'min': 1,
            'max': 7
        },
        'output_signals': {
            'min': 1,
            'max': 7
        }
    }
```

Листинг 19 — Программное описание класса эквиваленции

В листинге 20 представлен код для программного синтеза устройства эквиваленции с 2 входными разрядами первого операнда (first_signals='f:2'), 2 входными разрядами второго операнда (second_signals='s:2'), 1 прямым входным разрядом строб-сигнала (strobe_signals='z:1') и 2 выходными разрядами (output_signals='o:2').

```
>>> from circuitry.devices.cmp import DeviceEq
>>> pprint(
...     DeviceEq(strobe_signals='z:1', first_signals='f:2',
...              second_signals='s:2', output_signals='o:2',
...              strobe_signals_subs=dict(z0=1))
... )
{'first_signals': (f0, f1),
 'functions': [And(Or(Not(f0), s0), Or(Not(s0), f0), z0),
               And(Or(Not(f1), s1), Or(Not(s1), f1), z0)],
 'output_signals': (o0, o1),
 'second_signals': (s0, s1),
 'strobe_signals': (z0,),
 'strobe_signals_function': z0,
 'strobe_signals_subs': {'z0': 1},
 'strobe_signals_truth_table': [1],
 'truth_table': [([1], [0, 0], [0, 0], [1, 1]),
                 ([1], [0, 0], [1, 0], [0, 1]),
                 ([1], [0, 0], [0, 1], [1, 0]),
                 ([1], [0, 0], [1, 1], [0, 0]),
                 ([1], [1, 0], [0, 0], [0, 1]),
                 ([1], [1, 0], [1, 0], [1, 1]),
                 ([1], [1, 0], [0, 1], [0, 0]),
                 ([1], [1, 0], [1, 1], [1, 0]),
                 ([1], [0, 1], [0, 0], [1, 0]),
                 ([1], [0, 1], [1, 0], [0, 0]),
                 ([1], [0, 1], [0, 1], [1, 1]),
                 ([1], [0, 1], [1, 1], [0, 1]),
                 ([1], [1, 1], [0, 0], [0, 0]),
                 ([1], [1, 1], [1, 0], [1, 0]),
                 ([1], [1, 1], [0, 1], [0, 1]),
                 ([1], [1, 1], [1, 1], [1, 1])]}
```

Листинг 20 — Программный синтез устройства эквиваленции

На рисунке 20 представлено условно-графическое обозначение синтезированного устройства эквиваленции.

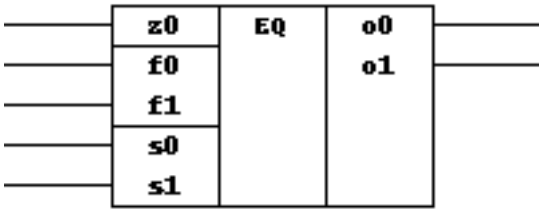

Рисунок 20 — Условно-графическое обозначение устройства эквиваленции

На рисунке 21 представлено устройство синтезированной эквиваленции в виде модели Simulink.

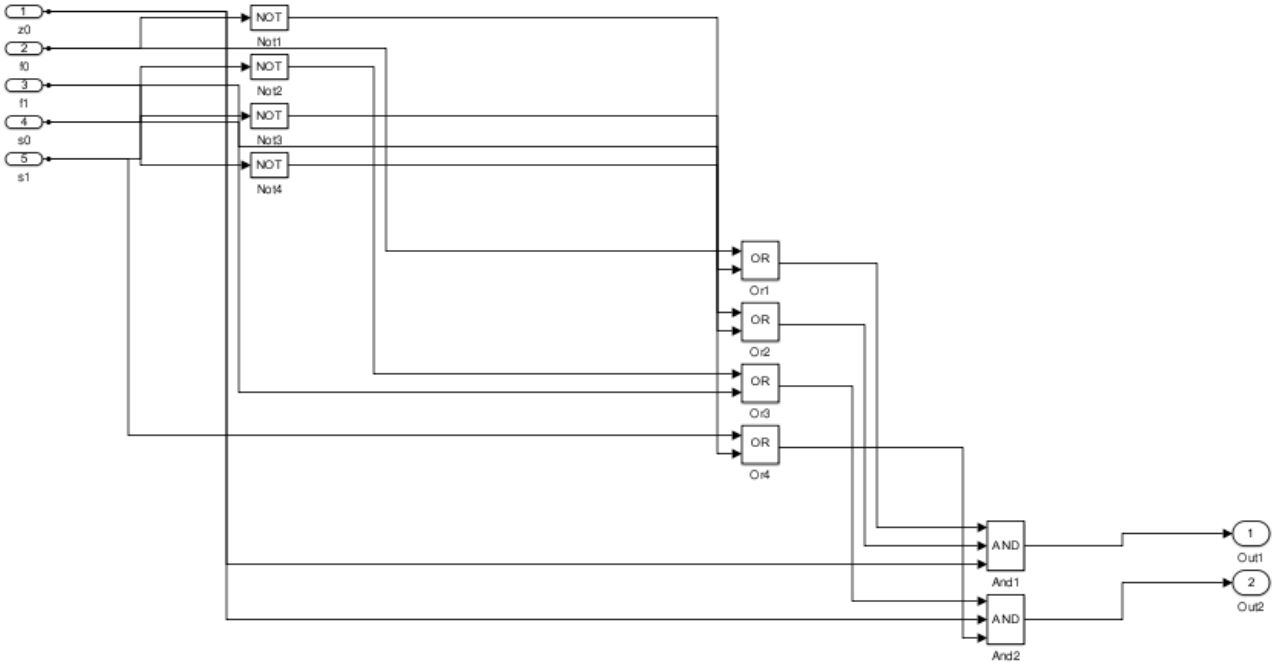

Рисунок 21 — Представление эквиваленции в виде модели Simulink

5.4.2 Цифровой компаратор

Устройство цифрового компаратора использует в качестве базового устройство эквиваленции и описывается следующим образом, используя выражение 14:

$$L_0 = (\overline{F_0} \wedge S_0 \wedge E_1) \vee (\overline{F_1} \wedge S_1 \wedge E_2) \vee \ldots \vee (\overline{F_n} \wedge S_n) \tag{15}$$

$$Q_0 = E_0 \wedge E_1 \wedge \ldots \wedge E_n \tag{16}$$

$$G_0 = (F_0 \wedge \overline{S_0} \wedge E_1) \vee (F_1 \wedge \overline{S_1} \wedge E_2) \vee \ldots \vee (F_n \wedge \overline{S_n}) \tag{17}$$

$$O = (L_0, Q_0, G_0) \tag{18}$$

В выражениях 15, 16, 17 и 18:

- n – количество входных разрядов первого и второго операнда;
- F – набор входных разрядов первого операнда;
- S – набор входных разрядов второго операнда;
- L_0 – разряд, показывающий, что первый операнд меньше второго;
- Q_0 – разряд, показывающий, что первый операнд равен второму;
- G_0 – разряд, показывающий, что первый операнд больше второго;
- O – набор выходных разрядов.

Описание цифрового компаратора в разрабатываемой библиотеке представлено в листинге 21.

```
1 class DeviceCmp(DeviceEq):
2     """Digital comparator device"""
```

Листинг 21 — Программное описание класса цифрового компаратора

В листинге 22 представлен код для программного синтеза цифрового компаратора с 2 входными разрядами первого операнда (first_signals='f:2'), 2 входными разрядами второго операнда (second_signals='s:2'), 1 прямым входным разрядом строб-сигнала (strobe_signals='z:1') и 3 выходными разрядами (output_signals='o:3').

```
>>> pprint(
...     DeviceCmp(strobe_signals='z:1', first_signals='f:2',
...               second_signals='s:2', output_signals='o:3',
...               strobe_signals_subs=dict(z0=1))
... )
{'first_signals': (f0, f1),
 'functions': [And(Or(And(Not(f0), Or(Not(f1), s1),
                          Or(Not(s1), f1), s0, z0),
                      And(Not(f1), s1)), z0),
               And(Or(Not(f0), s0), Or(Not(f1), s1),
                   Or(Not(s0), f0), Or(Not(s1), f1), z0),
               And(Or(And(Not(s0), Or(Not(f1), s1),
                          Or(Not(s1), f1), f0, z0),
                      And(Not(s1), f1)), z0)],
 'output_signals': (o0, o1, o2),
 'second_signals': (s0, s1),
 'strobe_signals': (z0,),
 'strobe_signals_function': z0,
 'strobe_signals_subs': {'z0': 1},
 'strobe_signals_truth_table': [1],
 'truth_table': [([1], [0, 0], [0, 0], [0, 1, 0]),
                 ([1], [0, 0], [1, 0], [1, 0, 0]),
                 ([1], [0, 0], [0, 1], [1, 0, 0]),
                 ([1], [0, 0], [1, 1], [1, 0, 0]),
                 ([1], [1, 0], [0, 0], [0, 0, 1]),
                 ([1], [1, 0], [1, 0], [0, 1, 0]),
                 ([1], [1, 0], [0, 1], [1, 0, 0]),
                 ([1], [1, 0], [1, 1], [1, 0, 0]),
                 ([1], [0, 1], [0, 0], [0, 0, 1]),
                 ([1], [0, 1], [1, 0], [0, 0, 1]),
                 ([1], [0, 1], [0, 1], [0, 1, 0]),
                 ([1], [0, 1], [1, 1], [1, 0, 0]),
                 ([1], [1, 1], [0, 0], [0, 0, 1]),
                 ([1], [1, 1], [1, 0], [0, 0, 1]),
                 ([1], [1, 1], [0, 1], [0, 0, 1]),
                 ([1], [1, 1], [1, 1], [0, 1, 0])]}
```

Листинг 22 — Программный синтез цифрового компаратора

На рисунке 22 представлено условно-графическое обозначение синтезированного устройства цифрового компаратора.

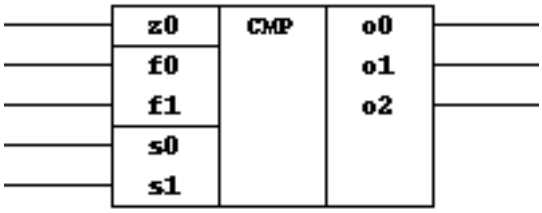

Рисунок 22 — Условно-графическое обозначение цифрового компаратора

На рисунке 23 представлено устройство синтезированного цифрового компаратора в виде модели Simulink.

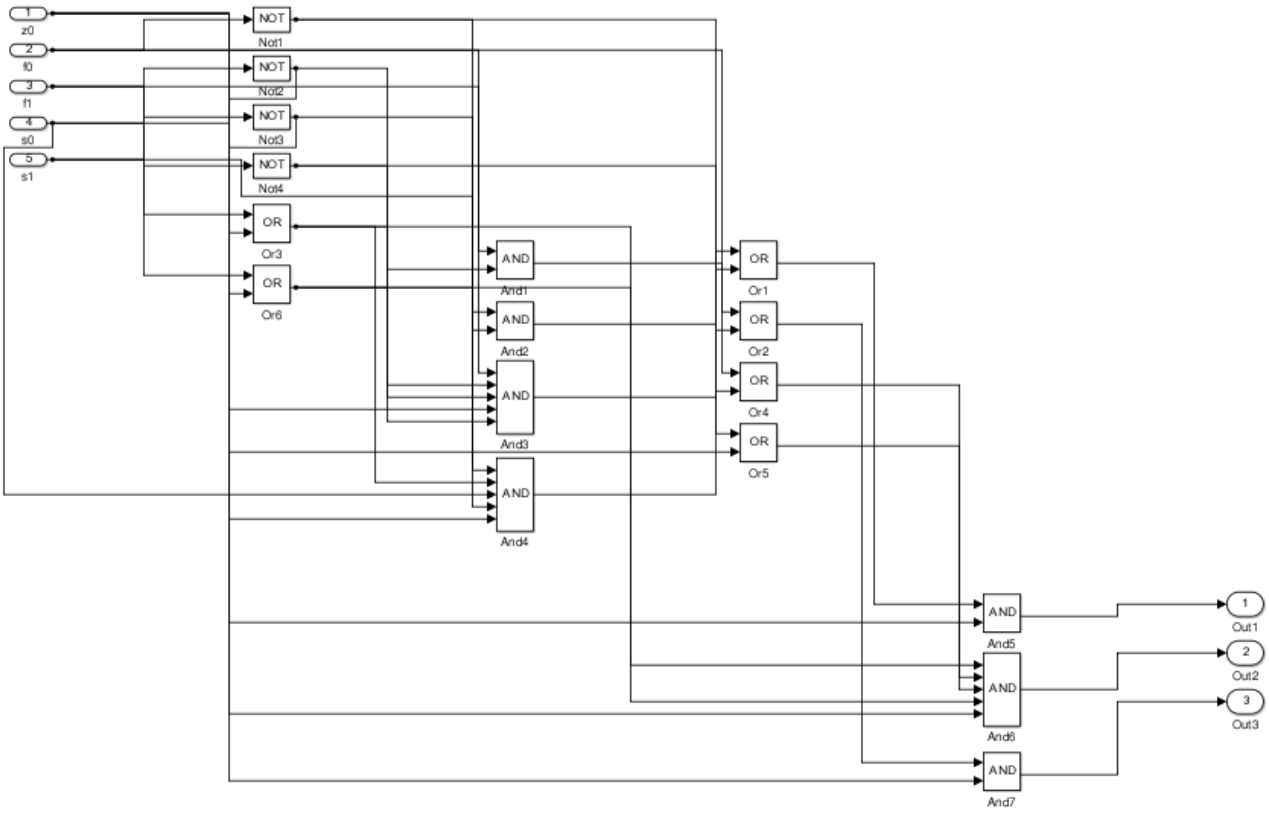

Рисунок 23 — Представление цифрового компаратора в виде модели Simulink

6 Форматы отображения схемотехнических устройств

В разделе представлены способы использования библиотеки для формирования синтезируемых устройств в различных видах посредством задействования механизма адаптеров. Основные два способа отображения синтезированных устройств – текстовый и графический.

6.1 Отображение в формате MATLAB / Simulink

Адаптеры, отвечающие за формирование кода модели Simulink из синтезируемых устройств, именуются MatlabAdapter и ExtendedMatlabAdapter для генерации отдельного блока и полноценной модели, соответственно. Описание первого представлено в листинге 23.

```
1 class MatlabAdapter(AbstractAdapter):
2     public_methods = ('matlab_code',)
3     default_method = lambda self: '\n'.join(self.matlab_code())
```

Листинг 23 — Программное описание класса адаптера MATLAB

Описание расширенного адаптера для генерации моделей MATLAB / Simulink, представленное в листинге 24, не сильно отличается от первого, так как расширяет только функционал (здесь опущен), а не сигнатуры.

```
1 class ExtendedMatlabAdapter(MatlabAdapter):
2     pass
```

Листинг 24 — Программное описание класса расширенного адаптера MATLAB

Пример применения расширенного адаптера к устройству (в данном случае, мультиплексором) хорошо иллюстрируется кодом, представленным в листинге 25.

```
>>> from circuitry.devices.mux import DeviceMux
>>> from circuitry.adapters.matlab.extended import ExtendedMatlabAdapter
>>>
>>> device_mux = DeviceMux(strobe_signals='v:1', address_signals='a:2',
...                        data_signals='d:4', output_signals='o:1',
...                        strobe_signals_subs=dict(v0=1),
...                        output_signals_subs=dict(o0=1))
>>>
>>> matlab_code = ExtendedMatlabAdapter(device_mux).default_method()
>>>
```

Листинг 25 — Генерация кода MATLAB

По окончании выполнения кода, в переменной matlab_code будет содержаться автоматически сгенерированный код MATLAB, представленный в листингах 26, 27 и 28.

```
sys = 'newModel1'
new_system(sys)
open_system(sys)
sys = 'newModel1/straight'
pos = [100 0 100 + 70 0 + 120]
add_block('built-in/SubSystem', sys, 'Position', pos)
pos = [400 0 400 + 30 0 + 20]
add_block('built-in/Logical Operator', [sys '/Not1'], 'Position', pos, ...
          'Operator', 'NOT', 'Number of input ports', '1')
pos = [400 40 400 + 30 40 + 20]
add_block('built-in/Logical Operator', [sys '/Not2'], 'Position', pos, ...
          'Operator', 'NOT', 'Number of input ports', '1')
pos = [200 0 200 + 30 0 + 10]
add_block('built-in/Inport', [sys '/v0'], 'Position', pos)
pos = [400 80 400 + 30 80 + 20]
add_block('built-in/Logical Operator', [sys '/Not3'], 'Position', pos, ...
          'Operator', 'NOT', 'Number of input ports', '1')
pos = [400 120 400 + 30 120 + 20]
add_block('built-in/Logical Operator', [sys '/Not4'], 'Position', pos, ...
          'Operator', 'NOT', 'Number of input ports', '1')
pos = [400 160 400 + 30 160 + 20]
add_block('built-in/Logical Operator', [sys '/Not5'], 'Position', pos, ...
          'Operator', 'NOT', 'Number of input ports', '1')
pos = [400 200 400 + 30 200 + 20]
add_block('built-in/Logical Operator', [sys '/Not6'], 'Position', pos, ...
          'Operator', 'NOT', 'Number of input ports', '1')
pos = [200 30 200 + 30 30 + 10]
add_block('built-in/Inport', [sys '/a0'], 'Position', pos)
pos = [200 60 200 + 30 60 + 10]
add_block('built-in/Inport', [sys '/a1'], 'Position', pos)
pos = [200 90 200 + 30 90 + 10]
add_block('built-in/Inport', [sys '/d0'], 'Position', pos)
pos = [200 120 200 + 30 120 + 10]
add_block('built-in/Inport', [sys '/d1'], 'Position', pos)
pos = [200 150 200 + 30 150 + 10]
add_block('built-in/Inport', [sys '/d2'], 'Position', pos)
pos = [200 180 200 + 30 180 + 10]
add_block('built-in/Inport', [sys '/d3'], 'Position', pos)
pos = [600 270 600 + 30 270 + 70]
add_block('built-in/Logical Operator', [sys '/And1'], 'Position', pos, ...
          'Operator', 'AND', 'Number of input ports', '6')
pos = [600 360 600 + 30 360 + 70]
add_block('built-in/Logical Operator', [sys '/And2'], 'Position', pos, ...
          'Operator', 'AND', 'Number of input ports', '6')
pos = [600 450 600 + 30 450 + 70]
add_block('built-in/Logical Operator', [sys '/And3'], 'Position', pos, ...
          'Operator', 'AND', 'Number of input ports', '6')
pos = [800 270 800 + 30 270 + 50]
add_block('built-in/Logical Operator', [sys '/Or1'], 'Position', pos, ...
          'Operator', 'OR', 'Number of input ports', '4')
pos = [600 540 600 + 30 540 + 70]
add_block('built-in/Logical Operator', [sys '/And4'], 'Position', pos, ...
          'Operator', 'AND', 'Number of input ports', '6')
pos = [1000 660 1000 + 30 660 + 30]
add_block('built-in/Logical Operator', [sys '/And5'], 'Position', pos, ...
          'Operator', 'AND', 'Number of input ports', '2')
```

Листинг 26 — Сгенерированный код MATLAB – часть 1

```
pos = [1200 660 1200 + 30 660 + 20]
add_block('built-in/Outport', [sys '/Out1'], 'Position', pos)
add_line(sys, 'And5/1', 'Out1/1', 'autorouting','on')
add_line(sys, 'a1/1', 'Not1/1', 'autorouting','on')
add_line(sys, 'a1/1', 'And2/1', 'autorouting','on')
add_line(sys, 'a1/1', 'And1/1', 'autorouting','on')
add_line(sys, 'Not1/1', 'And4/1', 'autorouting','on')
add_line(sys, 'Not1/1', 'And3/1', 'autorouting','on')
add_line(sys, 'Not2/1', 'And4/2', 'autorouting','on')
add_line(sys, 'Not2/1', 'And2/2', 'autorouting','on')
add_line(sys, 'And1/1', 'Or1/1', 'autorouting','on')
add_line(sys, 'And2/1', 'Or1/2', 'autorouting','on')
add_line(sys, 'And3/1', 'Or1/3', 'autorouting','on')
add_line(sys, 'v0/1', 'And5/1', 'autorouting','on')
add_line(sys, 'a0/1', 'Not2/1', 'autorouting','on')
add_line(sys, 'a0/1', 'And3/2', 'autorouting','on')
add_line(sys, 'a0/1', 'And1/2', 'autorouting','on')
add_line(sys, 'Not3/1', 'And4/3', 'autorouting','on')
add_line(sys, 'Not3/1', 'And3/3', 'autorouting','on')
add_line(sys, 'Not3/1', 'And1/3', 'autorouting','on')
add_line(sys, 'Not4/1', 'And2/3', 'autorouting','on')
add_line(sys, 'Not4/1', 'And4/4', 'autorouting','on')
add_line(sys, 'Not4/1', 'And3/4', 'autorouting','on')
add_line(sys, 'Or1/1', 'And5/2', 'autorouting','on')
add_line(sys, 'Not5/1', 'And2/4', 'autorouting','on')
add_line(sys, 'Not5/1', 'And3/5', 'autorouting','on')
add_line(sys, 'Not5/1', 'And1/4', 'autorouting','on')
add_line(sys, 'Not6/1', 'And4/5', 'autorouting','on')
add_line(sys, 'Not6/1', 'And2/5', 'autorouting','on')
add_line(sys, 'Not6/1', 'And1/5', 'autorouting','on')
add_line(sys, 'And4/1', 'Or1/4', 'autorouting','on')
add_line(sys, 'd2/1', 'Not3/1', 'autorouting','on')
add_line(sys, 'd2/1', 'And2/6', 'autorouting','on')
add_line(sys, 'd3/1', 'Not4/1', 'autorouting','on')
add_line(sys, 'd3/1', 'And1/6', 'autorouting','on')
add_line(sys, 'd0/1', 'Not5/1', 'autorouting','on')
add_line(sys, 'd0/1', 'And4/6', 'autorouting','on')
add_line(sys, 'd1/1', 'Not6/1', 'autorouting','on')
add_line(sys, 'd1/1', 'And3/6', 'autorouting','on')
```

Листинг 27 — Сгенерированный код MATLAB – часть 2

```
sys = 'newModel1'
pos = [20 20 20 + 20 20 + 20]
add_block('built-in/Constant', [sys '/v0'], 'Position', pos, ...
          'Value', '1', 'OutDataTypeStr', 'boolean')
add_line(sys, 'v0/1', 'straight/1', 'autorouting','on')
pos = [20 60 20 + 20 60 + 20]
add_block('built-in/Constant', [sys '/a0'], 'Position', pos, ...
          'Value', '0', 'OutDataTypeStr', 'boolean')
add_line(sys, 'a0/1', 'straight/2', 'autorouting','on')
pos = [20 100 20 + 20 100 + 20]
add_block('built-in/Constant', [sys '/a1'], 'Position', pos, ...
          'Value', '0', 'OutDataTypeStr', 'boolean')
add_line(sys, 'a1/1', 'straight/3', 'autorouting','on')
pos = [20 140 20 + 20 140 + 20]
add_block('built-in/Constant', [sys '/d0'], 'Position', pos, ...
          'Value', '1', 'OutDataTypeStr', 'boolean')
add_line(sys, 'd0/1', 'straight/4', 'autorouting','on')
pos = [20 180 20 + 20 180 + 20]
add_block('built-in/Constant', [sys '/d1'], 'Position', pos, ...
          'Value', '0', 'OutDataTypeStr', 'boolean')
add_line(sys, 'd1/1', 'straight/5', 'autorouting','on')
pos = [20 220 20 + 20 220 + 20]
add_block('built-in/Constant', [sys '/d2'], 'Position', pos, ...
          'Value', '0', 'OutDataTypeStr', 'boolean')
add_line(sys, 'd2/1', 'straight/6', 'autorouting','on')
pos = [20 260 20 + 20 260 + 20]
add_block('built-in/Constant', [sys '/d3'], 'Position', pos, ...
          'Value', '0', 'OutDataTypeStr', 'boolean')
add_line(sys, 'd3/1', 'straight/7', 'autorouting','on')
pos = [250 20 250 + 70 20 + 30]
add_block('built-in/Display', [sys '/Display_straight_1'], 'Position', pos)
add_line(sys, 'straight/1', 'Display_straight_1/1', 'autorouting','on')
```

Листинг 28 — Сгенерированный код MATLAB – часть 3

Модель Simulink, получаемая в результате выполнения этого кода представлена на рисунке 24.

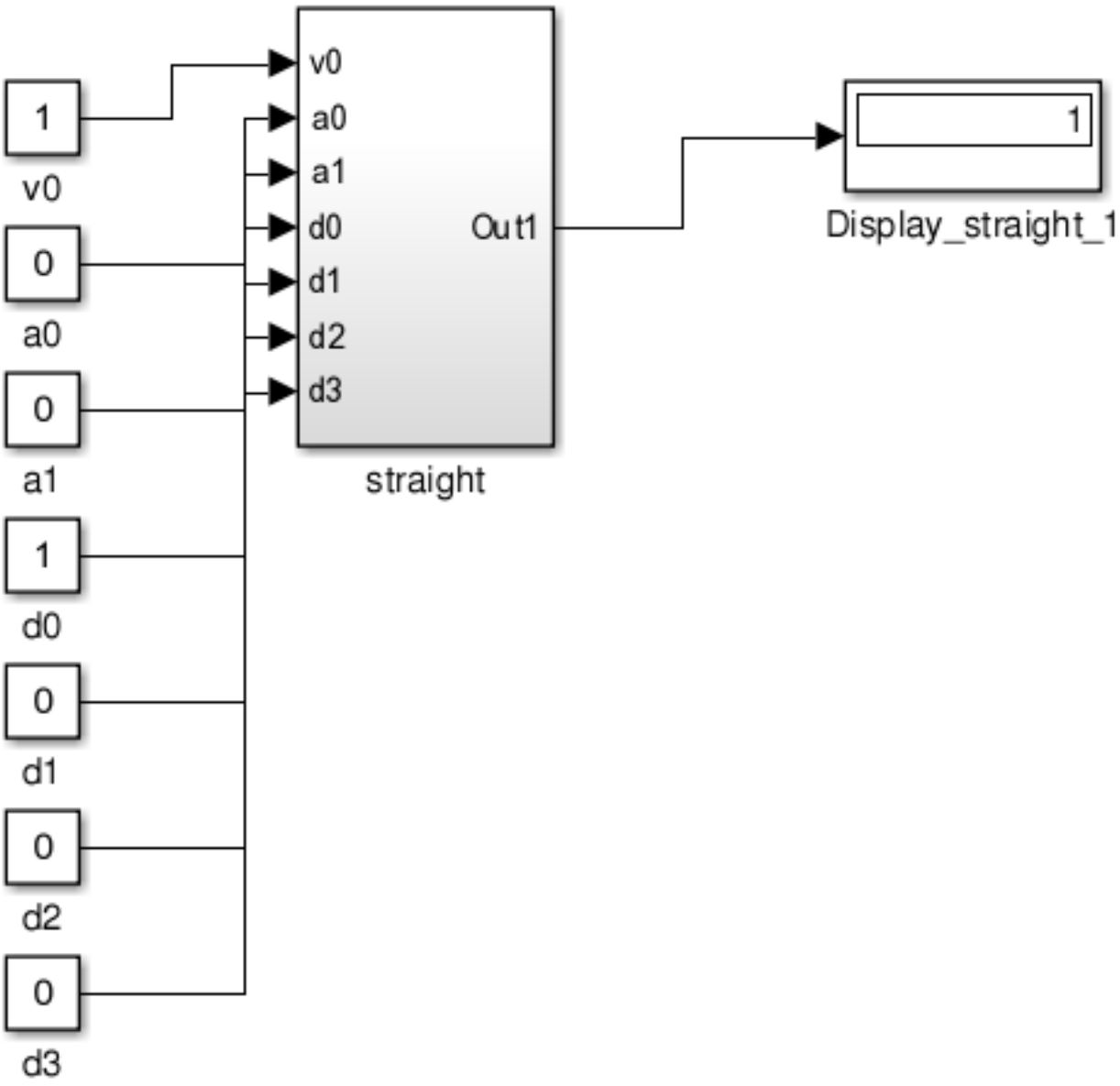

Рисунок 24 — Модель мультиплексора в Simulink

Блок Simulink представляет из себя схему устройства, построенную с помощью адаптера графов (рисунок 25).

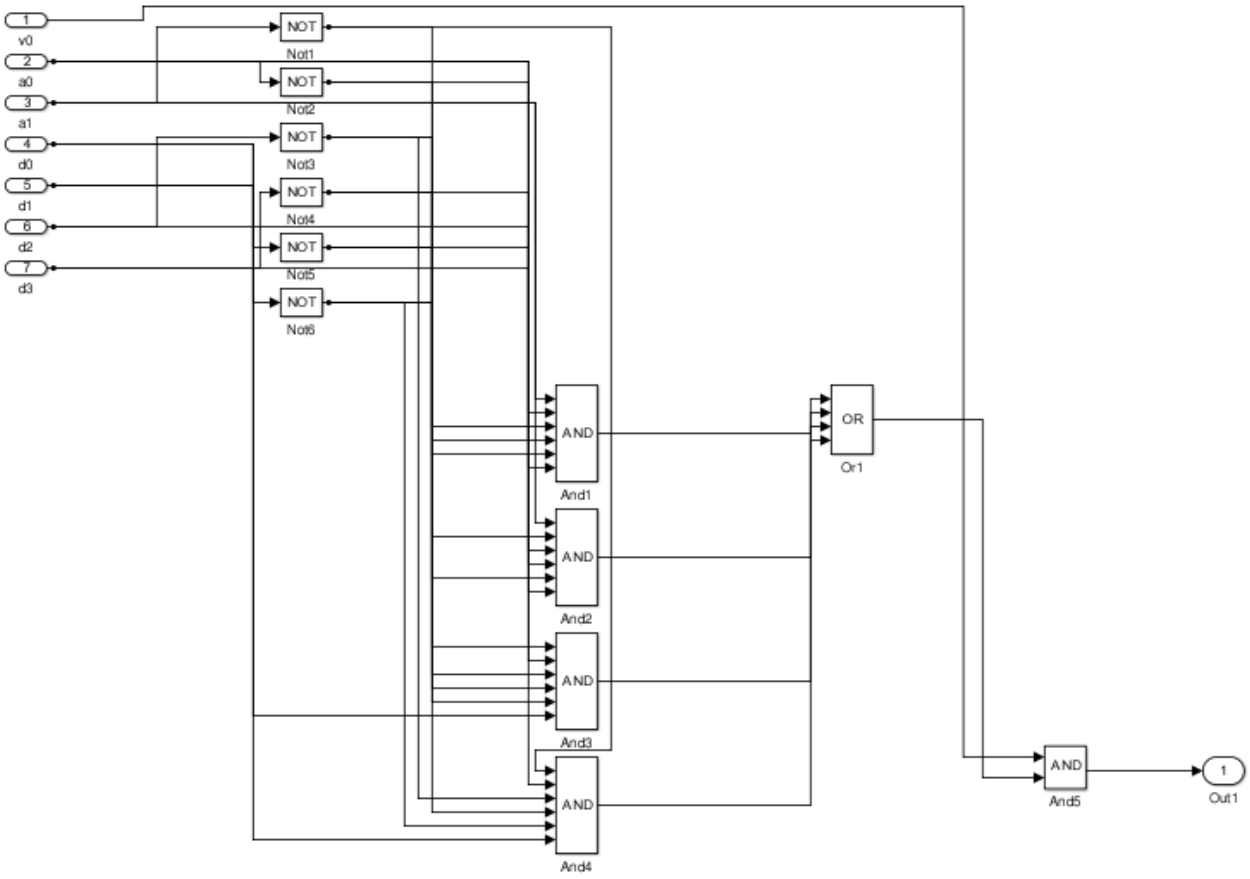

Рисунок 25 — Модель мультиплексора в Simulink - внутреннее представление блока

6.2 Отображение в виде графа

Адаптер формирующий граф из примитивов символьной алгебры логики именуется GraphAdapter. Его описывание представлено в листинге 29.

```
class GraphAdapter(AbstractAdapter):
    public_properties = ('graph', 'max_distance')
    default_content_type = 'image/png'

    def default_method(self):
        graph = self.graph
        labels_dict = dict()
        for graph_node in graph.nodes_iter():
            graph_type = graph.node[graph_node]['type']
            if graph_type == 'input':
                labels_dict[graph_node] = graph_node
            elif graph_type == 'output':
                labels_dict[graph_node] = 'output'
        pyplot.figure(figsize=(6, 4), dpi=100)
        graph_pos = random_layout(graph)
        draw_networkx_nodes(graph, pos=graph_pos, node_size=640)
        draw_networkx_edges(graph, pos=graph_pos, width=0.5, alpha=0.3)
        draw_networkx_labels(graph, pos=graph_pos, font_size=8,
                             labels=labels_dict)

        pyplot.xlim(-0.05, 1.05)
        pyplot.ylim(-0.05, 1.05)
        pyplot.axis('off')
        _, tmpfile = mkstemp(suffix='.png')
        pyplot.savefig(tmpfile)
        return tmpfile
```

Листинг 29 — Программное описание класса адаптера графов

Пример применения адаптера построения графов к устройству (в данном случае, эквиваленция) иллюстрируется кодом, представленным в листинге 30.

```
>>> from circuitry.adapters.graph import GraphAdapter
>>> from circuitry.devices.cmp import DeviceEq
>>>
>>> device_eq = DeviceEq(strobe_signals='v:1', first_signals='f:2',
...                      second_signals='s:2', output_signals='o:2',
...                      strobe_signals_subs=dict(v0=1))
>>>
>>> GraphAdapter(device_eq).default_method()
'/tmp/tmpNs3x52.png'
```

Листинг 30 — Генерация графа

Файл, полученный применением адаптера представлен на рисунке 26.

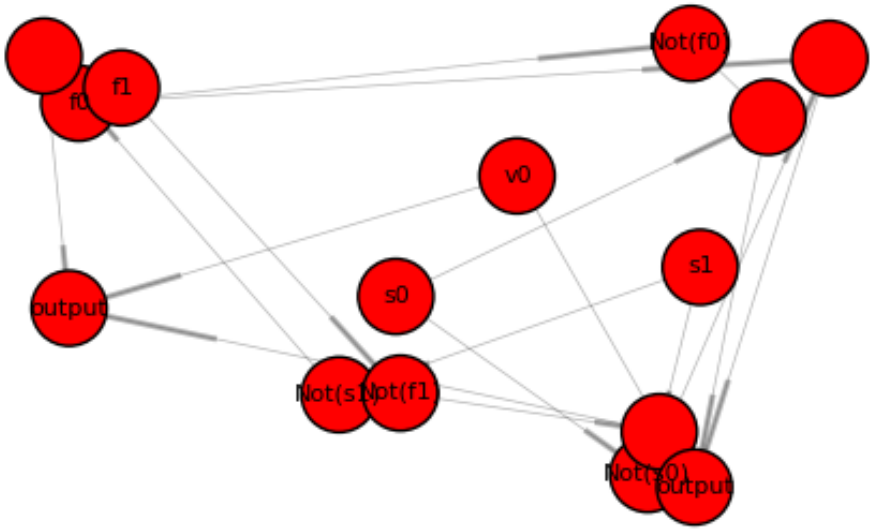

Рисунок 26 — Представление устройства эквиваленции в виде графа

6.3 Условно-графическое обозначение

Адаптер, позволяющий выводить условно графические изображения, именуется ElectronicSymbolAdapter и имеет определение, представленное в листинге 31.

```
class ElectronicSymbolAdapter(AbstractAdapter):
    public_properties = ('image',)
    default_content_type = 'image/png'

    def default_method(self):
        image = self.image
        large_image = Image.new('RGB', (600, 400), self._options['background'])
        large_image.paste(image,
                          (300 - self._options['width'] / 2,
                           200 - self._options['height'] / 2))
        _, tmpfile = mkstemp(suffix='.png')
        large_image.save(tmpfile)
        return tmpfile
```

Листинг 31 — Программное описание класса адаптера УГО

Пример применения адаптера УГО к устройству (в данном случае, демультиплексор) иллюстрируется кодом, представленном в листинге 32.

```
>>> from circuitry.adapters.visual.symbol import ElectronicSymbolAdapter
>>> from circuitry.devices.mux import DeviceDemux
>>>
>>> device_demux = DeviceDemux(strobe_signals='v:1', address_signals='a:2',
...                            data_signals='d:1', output_signals='o:4',
...                            strobe_signals_subs=dict(v0=1),
...                            data_signals_subs=dict(d0=1),
...                            output_signals_subs=dict(o0=1, o1=1, o2=1, o3=1))
>>>
>>> ElectronicSymbolAdapter(device_demux).default_method()
'/tmp/tmp8y85m2.png'
```

Листинг 32 — Генерация условно-графического обозначения

Файл, полученный применением адаптера представлен на рис. 27.

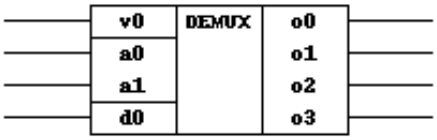

Рисунок 27 — Условно-графическое обозначение демультиплексора

6.4 Таблицы истинности в текстовом представлении

Адаптеры, отвечающие за генерацию таблиц истинности, именуются ConsoleTruthTableAdapter и TwosComplementConsoleTruthTableAdapter для генерации простой таблицы истинности и таблицы истинности с отображением десятичных представлений в обратном и дополнительном коде. Описание первого представлено в листинге 33.

```
class ConsoleTruthTableAdapter(AbstractAdapter):
    public_properties = ('console_table',)
    default_method = lambda self: self.console_table
```

Листинг 33 — Программное описание класса адаптера таблиц истинности

Описание более функционального адаптера представлено в листинге 34.

```
class TwosComplementConsoleTruthTableAdapter(ConsoleTruthTableAdapter):
    def default_method(self):
        min_len = len(self._device.output_signals)
        for signals_name in self._device.mandatory_signals:
            if signals_name not in ['output_signals', 'strobe_signals']:
                if len(self._device[signals_name]) < min_len:
                    min_len = len(self._device[signals_name])
        self._options['digits'] = min_len
        return self.console_table
```

Листинг 34 — Программное описание класса адаптера таблиц истинности в дополнительном коде

Пример применения более функционального адаптера к устройству (в данном случае, отрицание в дополнительном коде) хорошо иллюстрируется кодом, представленном в листинге 35.

```
>>> from circuitry.adapters.console import \
...         TwosComplementConsoleTruthTableAdapter
>>> from circuitry.devices.adder import DeviceNeg
>>>
>>> device_neg = DeviceNeg(strobe_signals='v:1', data_signals='d:3',
...                        output_signals='o:3',
...                        strobe_signals_subs=dict(v0=1))
>>>
>>> print TwosComplementConsoleTruthTableAdapter(device_neg).default_method()
1 (1, 1)     000 (0, 0)     000 (0, 0)
1 (1, 1)     001 (1, 1)     111 (-7, -1)
1 (1, 1)     010 (2, 2)     110 (-6, -2)
1 (1, 1)     011 (3, 3)     101 (-5, -3)
1 (1, 1)     100 (-4, -4)   100 (-4, -4)
1 (1, 1)     101 (-5, -3)   011 (3, 3)
1 (1, 1)     110 (-6, -2)   010 (2, 2)
1 (1, 1)     111 (-7, -1)   001 (1, 1)
>>>
```

Листинг 35 — Генерация таблицы истинности

6.5 Таблицы истинности в формате LaTeX

Адаптеры, отвечающие за генерацию таблиц истинности в формате LaTeX, именуются LatexTruthTableAdapter и DeviceMuxLatexTruthTableAdapter для генерации простой таблицы истинности и таблицы истинности для мультиплексоров и демультиплексоров, соответственно. Описание первого представлено в листинге 36.

```
class LatexTruthTableAdapter(AbstractAdapter):
    public_properties = ('latex_columns', 'latex_columns_names', 'latex_table')

    default_method = lambda self: \
        (r'\\begin{tabular}\{%(latex_columns)s}' +
         r'\\\hline\\%(latex_columns_names)s\\' +
         r'\hline\\%(latex_table)s\hline\\\end{tabular}') % \
        {'latex_columns': self.latex_columns,
         'latex_columns_names': self.latex_columns_names,
         'latex_table': self.latex_table}
```

Листинг 36 — Программное описание класса адаптера LaTeX

Описание адаптера для обработки мультиплексоров и демультиплексоров, представленное в листинге 37, не сильно отличается от первого, ибо только расширяет его функционал, не меняя сигнатур.

```
class DeviceMuxLatexTruthTableAdapter(LatexTruthTableAdapter):
    pass
```

Листинг 37 — Программное описание класса адаптера LaTeXдля мультиплексоров

Пример применения второго адаптера к устройству (в данном случае, мультиплексор) хорошо иллюстрируется следующим кодом, представленным в листинге 38.

```
>>> from circuitry.adapters.latex.mux import DeviceMuxLatexTruthTableAdapter
>>> from circuitry.devices.mux import DeviceMux
>>>
>>> device_mux = DeviceMux(strobe_signals='v:1', address_signals='a:2',
...                        data_signals='d:4', output_signals='o:1',
...                        strobe_signals_subs=dict(v0=1),
...                        output_signals_subs=dict(o0=1))
>>>
>>> print DeviceMuxLatexTruthTableAdapter(device_mux).default_method()
\begin{tabular}{|c|cc|cccc|c|}
\hline
$V_{0}$&$A_{0}$&$A_{1}$&$D_{0}$&$D_{1}$&$D_{2}$&$D_{3}$&$O_{0}$ \\
\hline 0&x&x&x&x&x&x&0 \\
1&0&0&1/0&x&x&x&1/0 \\
1&1&0&x&1/0&x&x&1/0 \\
1&0&1&x&x&1/0&x&1/0 \\
1&1&1&x&x&x&1/0&1/0 \\
\hline
\end{tabular}
>>>
```

Листинг 38 — Генерация кода LaTeX

Таблица истинности, сгенерированная адаптером, при включении в данный документ, приведена в таблице 1.

V_0	A_0	A_1	D_0	D_1	D_2	D_3	O_0
0	x	x	x	x	x	x	0
1	0	0	1/0	x	x	x	1/0
1	1	0	x	1/0	x	x	1/0
1	0	1	x	x	1/0	x	1/0
1	1	1	x	x	x	1/0	1/0

Таблица 1 — Таблица истинности мультиплексора

7 Внутренняя реализация программной библиотеки

7.1 Зависимости от внешних компонентов

В рамках проекта было решено не разрабатывать базовые сущности (такие как логические элементы символьной алгебры и графы) с нуля, а взять готовые программные библиотеки со свободными лицензиями. Зависимости проекта на текущий момент включают в себя следующие составные компоненты:

- SymPy (`http://sympy.org/`) – библиотека для работы с символьной алгеброй, содержащая примитивы для описания логических выражений [6]; является базовой для синтеза схемотехнических устройств проекта.
- Pillow (`http://python-imaging.github.io/`) (поддерживаемый форк PIL) – библиотека для работы с графическими изображениям [10]; используется для реализации адаптера вывода условно-графических обозначений устройств.
- NumPy (`http://www.numpy.org/`) – библиотека для математических расчетов [11]; используется библиотекой SymPy.
- NetworkX (`http://networkx.github.io/`) – библиотека для работы с графами [12]; используется в реализации адаптера отображения схем в виде графов и адаптера построения моделей Simulink.
- MatplotLib (`http://matplotlib.org/`) – библиотека для визуального отображения связанных сущностей [13]; используется в реализации адаптера отображения схем в виде графов.

7.2 Генерация новых типов синтезируемых устройств

Способ описания синезируемых устройств проекта базируется на нескольких основных принципах:

- Класс описываемого устройства наследуется от абстрактного базового класса circuitry.devices.Device.
- Для устройства делается стандартное описание в формате PyDoc.
- Переопределяется кортеж mandatory_signals, содержащий типы сигналов (типы strobe_signals и output_signals являются обязательными).
- Переопределяется кортеж mandatory_signals_using_subs, содержащий типы сигналов, использующих подстановки (входы и выходы могут быть прямыми и инвертированными, тип strobe_signals является обязательным).
- Переопределяется кортеж truth_table_signals, используемый всеми адаптерами генерации таблиц истинности и содержащий типы сигналов для включения в таблицы.
- Переопределяется словарь constraints, содержащий ключами наименования всех типов сигналов, перечисляемых в кортеже mandatory_signals, а значениям словари с ключами min и max, определяющими ограничения по количеству для каждого из типов сигналов.
- В секции инициализации первым идет вызов конструктора родительского класса.
- Далее заполяется список функций functions, состоящий из комбинированных примитивов библиотеки символьной алгебры с именами сигналов или наборов других функций в качестве входных параметров.
- При необходимости автоматической генерации таблиц истинности для работы адаптеров, в конце секции инициализации вызывается функция _generate_through_truth_table с кортежом сигналов signals_list, которые необходимо включить в сгенерированную таблицу истинности.

7.3 Генерация новых типов адаптеров

Способ описания адаптеров для обработки синтезируемых устройств проекта базируется на нескольких основных принципах:

- Класс адаптера наследуется от абстрактного базового класса circuitry.adapters.AbstractAdapter.
- Переопределяется кортеж public_properties, содержащий наименования методов класса, доступных для вызова.
- Переопределяется функция default_method, возвращающая результат обработки синтезируемого устройства с параметрами по-умолчанию (в случае графических изображений, возвращается имя файла с изображением).
- Переопределяется строка default_content_type, содержащая тип содержимого, возвращаемого методом по-умолчанию (text/plain для текста, image/png для графических изображений).

8 Выводы

В ходе выполнения курсовой работы были изучены логические методы синтеза комбинационных схем и произведены следующие действия по достижению поставленных целей:

- Были изучены программные продукты, позволяющие упростить синтез комбинационных схем.
- Была обоснована разработка программного продукта для синтеза комбинационных логических схемотехнических устройств с возможностями отображения синтезируемых устройств в различных форматах.
- В результате реализации проекта, для синтеза схемотехнических устройств разработана библиотека CircuitryLib, которая позволяет автоматически синтезировать комбинационные схемы, а также является расширяемой благодаря изначально проведенной декомпозиции используемых сущностей.
- Помимо библиотеки, был разработан интерфейс пользователя CircuitryLib-web, который позволяет произвести ограниченное методами по-умолчанию адаптеров изучение программной библиотеки.

Дальнейшие работы по проекту предполагают расширение количества синтезируемых комбинационных схем (например, умножители), доработку библиотеки для возможности задания базиса и ограничения на количество входов для логических элементов, рассмотрение потенциально возможной интеграции с системой симуляции схем по времени MyHDL.

Общий прогресс разработки и само программное обеспечение можно получить из системы контроля версий GitHub по следующим адресам:

- `https://github.com/profitware/circuitrylib` – библиотека схемотехнического моделирования CircuitryLib;
- `https://github.com/profitware/circuitrylib-web` – веб-интерфейс пользователя библиотеки.

Список использованных источников

1. Коновалов И.В., Коновалов В.Н., Белов А.А., Нежельский П.Н. Автоматизированный синтез комбинационных логических схем / Материалы Всероссийской НТК «Прогрессивные технологии, конструкции и системы в приборо- и машиностроении». М.: МГТУ им. Н.Э. Баумана, 2005. Т. 1. С. 190.

2. Jan Decaluwe Overview – MyHDL 0.8 documentation [Электронный ресурс] // URL: `http://docs.myhdl.org/en/latest/manual/preface.html`, (Дата обращения: 03.06.2014).

3. Jonathan P Dawson Introduction – Chips v0.1 documentation [Электронный ресурс] // URL: `http://dawsonjon.github.io/chips/introduction/index.html`, (Дата обращения: 05.06.2014).

4. BinPy Development Team Home – BinPy/BinPy Wiki [Электронный ресурс] // URL: `https://github.com/BinPy/BinPy/wiki`, (Дата обращения: 03.06.2014).

5. Wikimedia Foundation, Inc. Логический вентиль – Википедия [Электронный ресурс] // URL: `http://ru.wikipedia.org/?oldid=59843325`, (Дата обращения: 05.06.2014).

6. SymPy Development Team Welcome to SymPy's documentation! – SymPy 0.7.5 documentation [Электронный ресурс] // URL: `http://docs.sympy.org/latest/index.html`, (Дата обращения: 03.06.2014).

7. Wikimedia Foundation, Inc. Мультиплексор (электроника) – Википедия [Электронный ресурс] // URL: `http://ru.wikipedia.org/?oldid=61083564`, (Дата обращения: 05.06.2014).

8. Wikimedia Foundation, Inc. Сумматор – Википедия [Электронный ресурс] // URL: `http://ru.wikipedia.org/?oldid=63231329`, (Дата обращения: 06.06.2014).

9. Wikimedia Foundation, Inc. Цифровой компаратор – Википедия [Электронный ресурс] // URL: http://ru.wikipedia.org/?oldid=53665348, (Дата обращения: 06.06.2014).

10. Secret Labs AB, Fredrik Lundh, Alex Clark Pillow – Pillow v2.4.0 (PIL fork) [Электронный ресурс] // URL: http://pillow.readthedocs.org/en/latest/, (Дата обращения: 03.06.2014).

11. SciPy developers Numpy and Scipy Documentation – Numpy and Scipy documentation [Электронный ресурс] // URL: http://docs.scipy.org/doc/, (Дата обращения: 03.06.2014).

12. NetworkX developer team Overview – NetworkX [Электронный ресурс] // URL: https://networkx.github.io/, (Дата обращения: 03.06.2014).

13. John Hunter, Darren Dale, Eric Firing, Michael Droettboom and the matplotlib development team matplotlib: python plotting – Matplotlib 1.3.1 documentation [Электронный ресурс] // URL: http://matplotlib.org/1.3.1/index.html, (Дата обращения: 03.06.2014).