УДК 004.94

УНИФИКАЦИЯ ОПИСАНИЯ ПРОГРАММНЫХ И АППАРАТНЫХ МОДЕЛЕЙ В БАЗИСЕ СБИС

Кряхтунов Г.М., Боронников А.С., Платонова О.В.

Федеральное государственное бюджетное образовательное учреждение высшего образования «МИРЭА — Российский технологический университет», 119454, Российская Федерация, г. Москва, проспект Вернадского, 78, e-mail: gleb.kryakhtunov@yandex.ru, boronnikov-anton@mail.ru, oplatonova@gmail.com

Проектирование в базисе сверхбольших интегральных схем (СБИС) предполагает модульность, интеграцию библиотек готовых функциональных блоков и стандартизированные интерфейсы. В статье выдвигается гипотеза о том, что создание унифицированного формата для описания аппаратных моделей позволит ускорить время разработки и тестирования аппаратных решений в базисе СБИС. Унификация описания аппаратных моделей является важной частью для совместимости, упрощения разработки, тестирования, обновлений и ускорения вывода продукта на рынок. Выявлены основные недостатки рассмотренных существующих решений и методов. Определены аспекты аппаратных моделей на примере устройства, принимающего данные по протоколу UART. Выделены основные функциональные узлы аппаратных моделей: комбинационные схемы, простые схемы с памятью и конечные автоматы. Рассмотрены реализация каждого функционального узла на языке описания аппаратуры Verilog HDL и на языке программирования высокого уровня Python. Представлены симуляции разработанных моделей на системном и аппаратном уровне. Выявлены сходства описания аппаратных и программных моделей. Предложен маршрут проектирования аппаратных моделей в базисе СБИС с использованием унифицированного описания. Предложена структура аппаратной модели, которая не зависит от интегрированной среды разработки и средств автоматизированного проектирования. Работа закладывает основу для стандартизации методов проектирования вычислительных систем, ориентированных на разработку в базисе СБИС. Перспективные пути развития данной гипотезы могут включать следующие направления: интеграция с искусственным интеллектом и машинным обучением, расширение области тестирования аппаратных моделей, автоматизация процесса трансляции моделей в целевые языки, разработка систем визуализации для проектирования. Эти направления могут значительно расширить возможности применения предложенной гипотезы и способствовать более эффективному проектированию вычислительных систем в базисе СБИС. Предполагается проведение исследований в области автоматического анализа и оптимизации тестирования аппаратных моделей, что позволит повысить эффективность проектирования и упростить интеграцию новых технологий в существующие вычислительные системы и их элементы.

Ключевые слова: описание, программные модели, аппаратные модели, языки программирования, языки описания аппаратуры, СБИС, модель, Verilog HDL, Python, тестирование, верификация.

UNIFICATION OF THE DESCRIPTION OF SOFTWARE AND HARDWARE MODELS BASED ON VLSI

Kryakhtunov G.M., Boronnikov A.S., Platonova O.V.

Federal State Budgetary Educational Institution of Higher Education «MIREA – Russian Technological University», 119454, Russian Federation, Moscow, Vernadskogo Ave., 78, e-mail: gleb.kryakhtunov@yandex.ru, boronnikov-anton@mail.ru, oplatonova@gmail.com

Design based on very-large-scale integrated circuits (VLSI) involves modularity, integration of libraries of ready-made functional blocks and standardized interfaces. The article hypothesizes that the creation of a unified format for describing hardware models will speed up the development and testing of hardware solutions based on VLSI. Unification of hardware model descriptions is an important part for compatibility, simplification of development, testing, updates and acceleration of product launch to the market. The main disadvantages of the considered existing solutions and methods are revealed. Aspects of hardware models are defined using the example of a device receiving data using the UART protocol. The main functional units of the hardware models are highlighted: combinational circuits, simple circuits with memory, and finite automata. The implementation of each functional node in the hardware description language Verilog HDL and in the high-level programming language Python is considered. Simulations of the developed models at the system and hardware levels are presented. The similarities of the description of hardware and software models are revealed. A route for designing hardware models based on VLSI using a unified description is proposed. The structure of the hardware model is proposed, which does not depend on the integrated development environment and computer-aided design tools. The work lays the

foundation for the standardization of design methods for computing systems focused on development in the VLSI framework. Promising ways to develop this hypothesis may include the following areas: integration with artificial intelligence and machine learning, expanding the field of testing hardware models, automating the process of translating models into target languages, and developing visualization systems for design. These areas can significantly expand the possibilities of applying the proposed hypothesis and contribute to more efficient design of computing systems based on VLSI. It is planned to conduct research in the field of automatic analysis and optimization of hardware model testing, which will improve design efficiency and simplify the integration of new technologies into existing computing systems and their elements.

Keywords: description, software models, hardware models, programming languages, hardware description languages, VLSI, model, Verilog HDL, Python, testing, verification.

Введение

При проектировании вычислительных систем и их элементов встает вопрос о том, какие инструментальные средства необходимо использовать для описания аппаратных моделей, а также для их моделирования и верификации. Существует множество стандартов и методов, которые позволяют описать аппаратные модели с учетом их реализации под конкретную предметную область.

Для образовательных целей используется программное обеспечение, как Logisim или Logicly, поддерживающее проектирование и моделирование простых цифровых схем [1-2].

Для описания аппаратных моделей и их верификаций в составе сверхбольших интегральных схем (СБИС) используются языки описания аппаратуры – Verilog HDL, VHDL, SystemVerilog [3-5].

Для высокоуровневого моделирования систем на кристалле (СнК) используется библиотека SystemC, которая является расширением языка программирования C++. SystemC позволяет описать симуляции цифровых схем на высоком уровне абстракции, включая верификацию [6-9].

Для верификации аппаратных моделей широко применяются различные симуляторы, обеспечивающие проверку корректности работы проектируемых систем. Наиболее популярными инструментами в этой области являются iSim, Modelsim, Icarus, Verilator и другие [10]. В дополнение к симуляторам активно используется методология UVM (Universal Verification Methodology), которая предоставляет стандартизированную структуру для создания тестовых модулей [11].

Современные методы проектирования и моделирования аппаратных моделей охватывают широкий набор инструментальных средств, но их основной недостаток заключается в том, что нет единого формата или стандарта, который позволял бы переносить аппаратные модели из одной формы описания в другую.

Актуальным направлением информационных технологий является проектирование аппаратных средств в базисе СБИС, которые позволяют реализовывать аппаратные модели или системы различной сложности.

При этом сложность и уровень аппаратных систем с каждым годом растет, поэтому время проектирования и тестирования разрабатываемых решений существенно увеличивается. В дополнении к вышеизложенному, со временем появляются новые стандарты и методы, которые не поддерживают уже существующие разработанные IP-блоки, поэтому могут возникать сложности при переносе описания аппаратных моделей под более новые стандарты.

Гипотеза

Создание унифицированного формата для описания аппаратных моделей позволит решить выше рассмотренные проблемы. Такой подход уберет зависимость описания аппаратной модели от среды разработки и ускорит время проектирования аппаратных решений, так как процесс моделирования и верификации будет вынесен на системный уровень, а интерпретация модели будет на целевом языке описания аппаратуры.

Постановка задачи

Унификация описания моделей является важной задачей в проектировании в базисе СБИС, поскольку она повышает эффективность разработки аппаратных решений и упрощает процесс их тестирования.

В ходе поиска существующих решений по данной предметной области были найдены научные работы и материалы, которые охватывают стык программной и аппаратной части разрабатываемых вычислительных систем.

В диссертации [12] представлена унифицированная языковая модель проектирования Bluespec Codesign Language (BCL), которая позволяет описать программно-аппаратные решения. Идея языка в том, что модель обеспечивает простой способ определения того, какие части проекта должны быть реализованы на аппаратном уровне, а какие на программном.

В [13] описывается язык Kanagawa, дающий возможность программистам создавать аппаратные решения с использованием языковой модели, которая содержит функции языков программирования.

На более высоком уровне абстракции описания аппаратных моделей в начале 2000-х годов появились языки программирования Handel-C [14] и Impulse-C. Они предоставляли возможность описывать сложные параллельные аппаратные процессы в привычной для программистов форме в виде похожего синтаксиса языка программирования C.

В ходе обзора выявлен главный недостаток, а именно высокая сложность внедрения – найденные языки для описания моделей не используются в современных системах проектирования, что может свидетельствовать о том, что предложенные стандарты не нашли распространения в индустрии. Это может быть связано с отсутствием совместимости с существующими инструментальными средствами или недостаточной поддержкой со стороны разработчиков.

Эти ограничения подчеркивают необходимость исследований в области унификации описания аппаратных моделей, а также разработки новых подходов и методов, которые позволят специалисту данной области абстрагироваться и не привязываться к существующим средствам разработки. Также значимость темы подчеркивается растущей сложностью проектируемых вычислительных систем и потребностью в сокращении времени их разработки, что требует создания новых стандартов унификаций описания аппаратных моделей.

Таким образом, в данной работе поставлены следующие задачи:

– рассмотреть ключевые аспекты описания аппаратных моделей на примере существующей системы с целью унификации методов и подходов к их описанию;

– сравнить описание аппаратных моделей с использованием языка описания аппаратуры и языка программирования высокого уровня для определения общих параметров моделей;

– сформировать структуру унифицированного формата и предложить с его использованием маршрут проектирования аппаратных моделей.

Пример аппаратной системы

Рассмотрим аспекты аппаратных моделей на примере устройства, принимающего данные по протоколу UART. Протокол UART – физический протокол передачи данных. Обмен данными происходит с помощью кадров, которые в общем случае имеют структуру: стартовый бит, биты данных, бит четности, стоповые биты. В качества примера возьмем наиболее распространенный формат кадра данных, который состоит из следующей последовательности: стартовый бит, восемь битов данных и один стоповый бит (рис. 1).

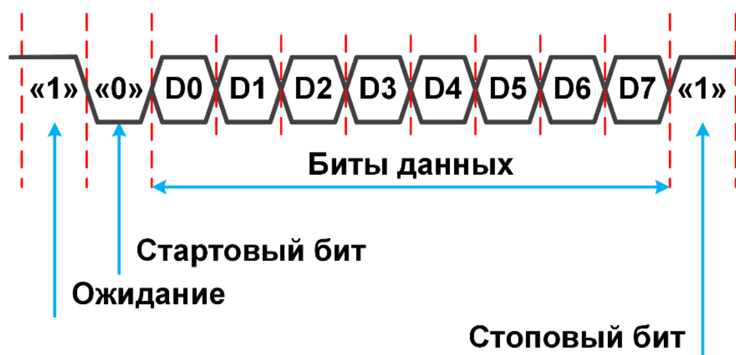

Рисунок 1 – Формат кадра данных протокола UART

В начальный момент времени инициатор обмена протокола UART устанавливает линию передачи данных в логическое состояние «1» – это означает, что данные не передаются. Обмен данными между инициатором и исполнителем начинается, когда инициатор переводит линию передачи данных из логического состояния «1» в «0» для указания начала кадра данных (стартовый бит). Затем инициатор передает 8 битов данных, передача которых осуществляется с младшего бита. Далее инициатор переводит линию в логическое состояние «1», которое задает конец кадра данных (стоповый бит). Передача битов кадра данных происходит с фиксированной скоростью, наиболее распространенные из них: 9600, 19200, 115200 бод/с.

Сэмплирование (выборка) битов кадра данных UART происходит в середине битового интервала, когда сигнал принимаемых данных наиболее стабильный (рис. 2). Для обеспечения выборки битов данных в середине битового интервала используют частоту дискретизации, которая обычно в 8 или 16 раз выше, чем скорость передачи данных.

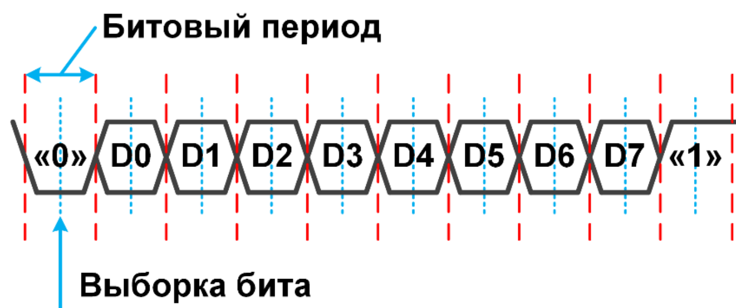

Рисунок 2 – Сэмплирование битов кадра данных протокола UART

Функциональная схема приемника данных протокола UART представлена на рис. 3. Схема состоит из следующих функциональных блоков:

– Синхронизатор – блок, синхронизирующий асинхронный сигнал принимаемых данных «rxd»;

– Делитель частоты – блок, реализующий деление тактовой частоты, согласно частоте дискретизации битов данных кадра UART;

– Счетчик сэмплов – блок, формирующий признак выбора бита данных кадра UART в середине битового интервала;

– Принимающий автомат – конечный автомат, реализующий прием данных по протоколу UART согласно формату, представленном на рис. 1.

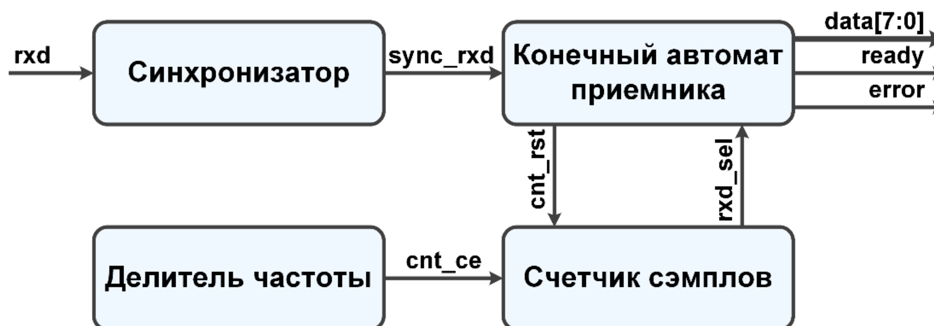

Рисунок 3 – Функциональная схема приемника UART

Описание сигналов представлено ниже:

– rxd – принимаемые данные;
– sync_rxd – синхронизированные принимаемые данные «rxd»;
– cnt_ce – сигнал разрешения синхронизации счетчика сэмплов;
– cnt_rst – синхронный сброс счетчика сэмплов;
– rxd_sel – сигнал разрешения выбора бита данных кадра UART;
– data[7:0] – 8-разрядные принятые данные;
– ready – сигнал готовности принятых данных «data»;
– error – признак принятых данных «data» с ошибкой.

Рассмотрим каждый функциональный блок приемника UART более подробно. Начнем с синхронизатора входной линии приема данных «rxd». Он состоит из двух последовательно соединенных D-триггеров, где выход первого триггера подключен к входу второго (рис. 4). Такой подход позволяет реализовать синхронизацию асинхронного входного сигнала «rxd».

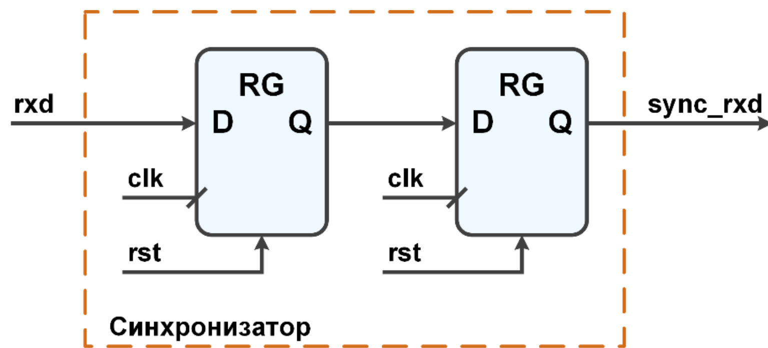

Рисунок 4 – Структурная схема синхронизатора

Делитель частоты формирует сигнал разрешения синхронизации «cnt_ce» счетчика сэмплов. Делитель состоит из счетчика и комбинационной схемы, которая сравнивает текущее значение счетчика с указанным значением коэффициента деления. Пример структурной схемы делителя для приемника UART, у которого тактовая частота синхросигнала равна 100 МГц, скорость обмена составляет 9600 бод/с, а частота дискретизации в 16 раз выше, чем скорость передачи данных представлена на рис.5.

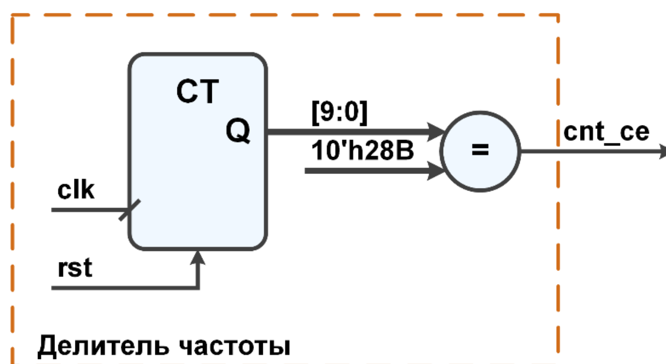

Рисунок 5 – Структурная схема делителя частоты

Счетчик сэмплов кадра данных UART имеет схожую структурную схему (рис. 6) с делителем частоты. Отличие заключаются в том, что у счетчика есть управляющий вход для синхронного сброса «RE» в состояние «0», когда обмена данными по протоколу UART нет и сигнал разрешения синхронизации «CE».

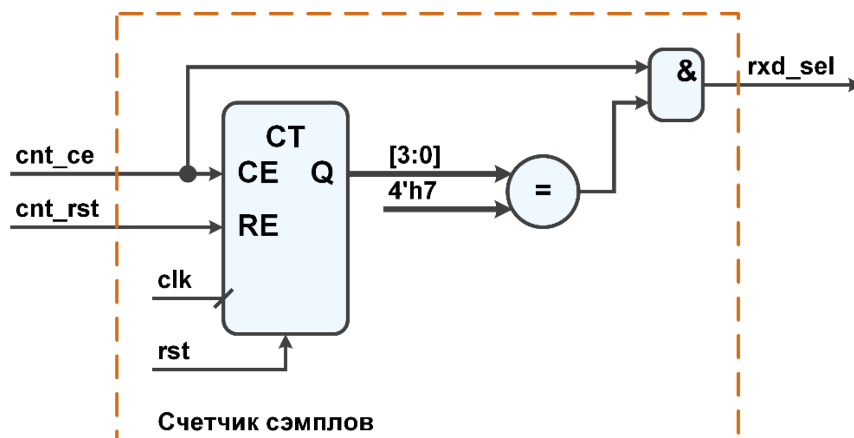

Рисунок 6 – Структурная схема счетчика сэмплов

Анализ последовательности битов, принимаемых по протоколу UART, реализует конечный автомат, граф переходов которого представлен на рис.7.

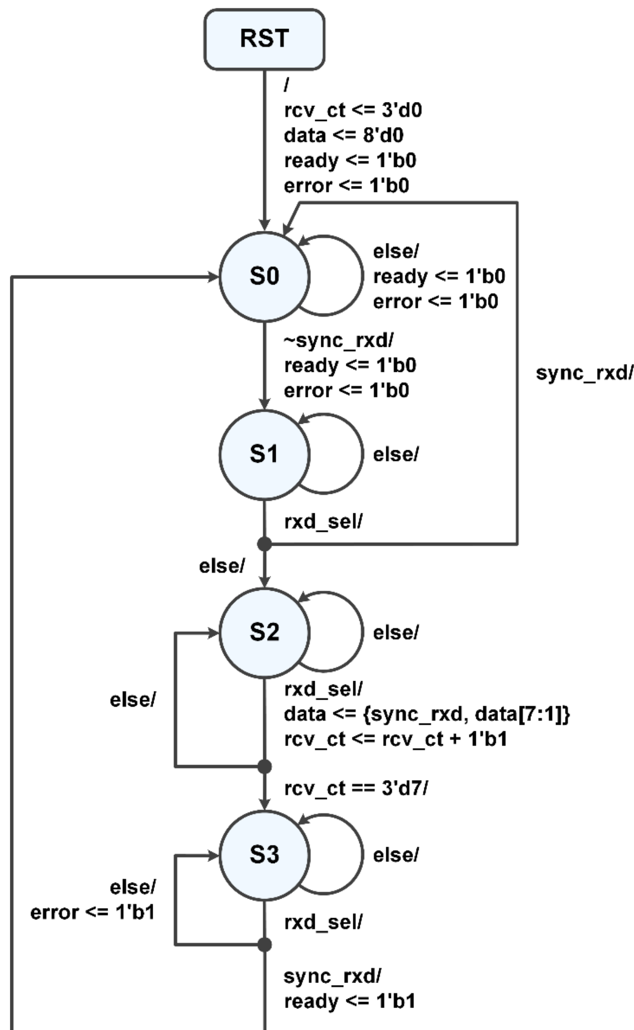

Рисунок 7 – Граф конечного автомата приемника

Граф переходов состоит из 4 состояний, где каждое состояние отвечает за определенный этап распознавания кадра данных UART. Описание состояний графа представлено в табл.1.

Таблица 1 – Описание состояний конечного автомата приемника

Состояние	Код состояния	Описание
S0	00b	Ожидание начала кадра данных UART
S1	01b	Проверка стартового бита кадра данных
S2	10b	Прием 8-разрядных данных UART
S3	11b	Проверка стопового бита кадра данных

Счетчик «rcv_ct» отвечает за количество принятых битов кадра протокола UART. Разрядность счетчика составляет 3 бита, так как осуществляется прием 8-разрядных данных. При достижении значения «7» автомат переходит в состояние «S3», что означает – все 8 битов данных кадра UART приняты.

Структурная схема конечного автомата приемника данных UART представлена на рис. 8. Состояние «state» отвечает за текущее состояние конечного автомата, в зависимости от входных признаков и текущего состояния, поступающих на комбинационную схему, формируется следующее состояние, в которое переходит конечный автомат. Аналогично остальные регистры обновляют значения согласно комбинационной схеме, которая формирует новые данные и управляющие сигналы в зависимости от текущего состояния и входных признаков конечного автомата.

Сигнал «cnt_rst» формируется с помощью комбинационной схемы, которая анализирует состояние конечного автомата на равенство состоянию «S0».

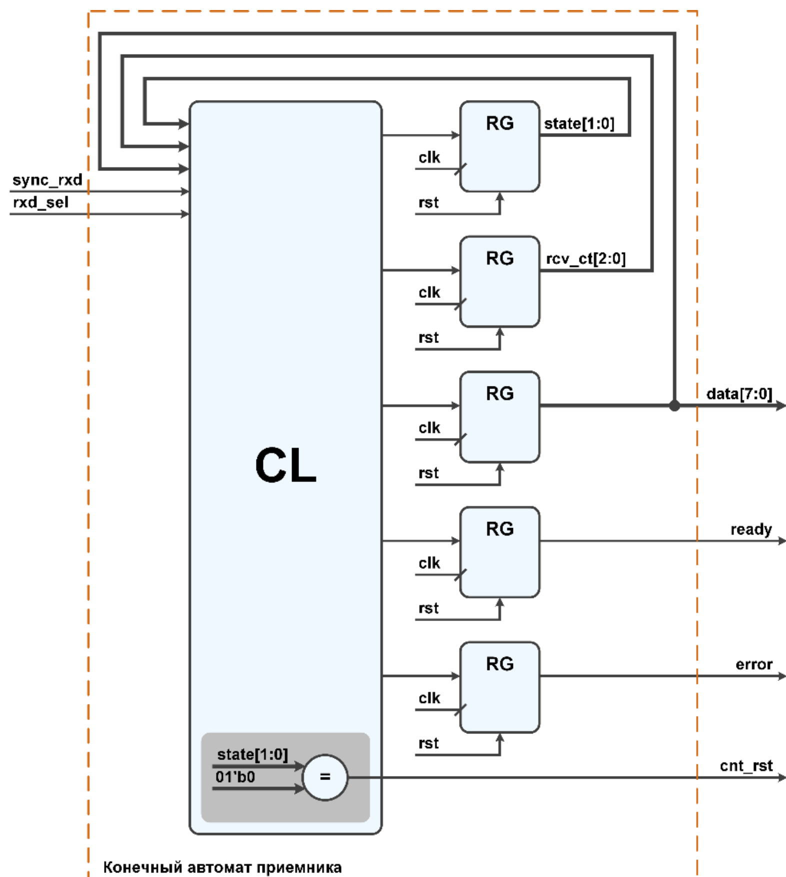

Рисунок 8 – Функциональная схема конечного автомата приемника

Аспекты аппаратных моделей

Проектирование в базисе СБИС имеет модульный характер это означает, что один и тот же разработанный функциональный блок может использоваться в разных проектах или повторяться несколько раз в рамках одного. Пример функциональной схемы модуля представлен на рис. 9. На нем представлены следующие элементы такие, как узлы – принимающие и передающие потоки данных, связи – формирующие взаимодействие между двумя узлами, модули – блоки, который могут содержать в себе комбинационные схемы, регистры и другие модули.

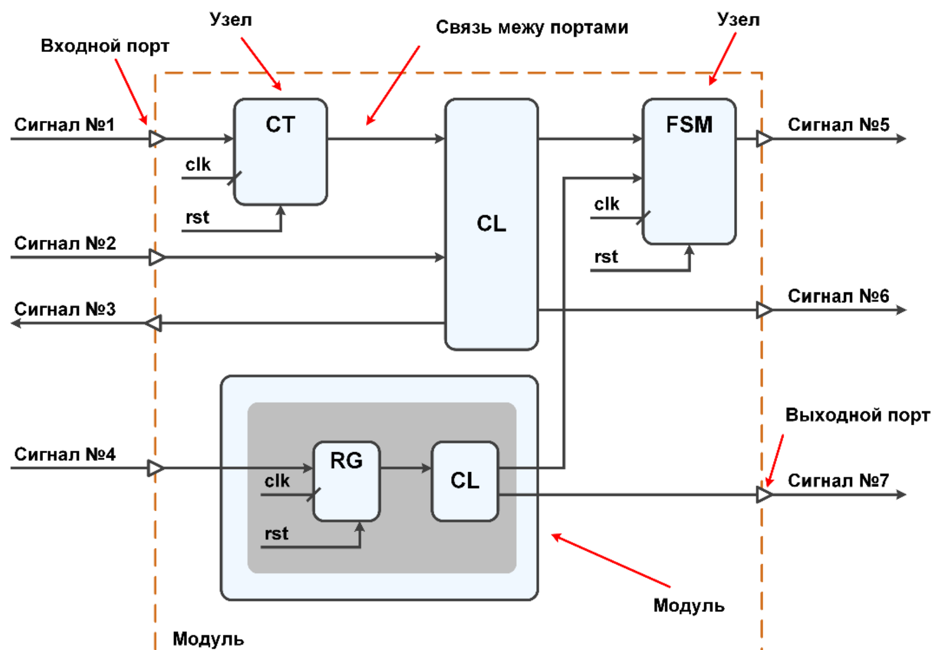

Рисунок 9 – Функциональная схема модуля

Классифицировать элементы функционального блока можно на следующие категории:

– Комбинационные схемы – схемы без элементов памяти, выходные значения сигналов формируются в зависимости от входных;

– Простые схемы с памятью – схемы, реализующие примитивную логику переходов между состояниями. В большинстве случаев описываются с помощью таблиц функционирования;

– Конечные автоматы – схемы с памятью, реализующие сложную логику переходов между состояниями. Обычно используются для приема, передачи и обработки информации.

Комбинационные схемы строятся на основе базовых элементов: «НЕ», «ИЛИ-НЕ», «И-НЕ». Различные комбинации этих элементов описывают различные комбинационные узлы и блоки: «И», «ИЛИ», «Искл. ИЛИ», мультиплексоры, дешифраторы, преобразователи кодов и т.д. Условно графические обозначения основных элементов и узлов, а также их таблица истинности представлены на рис.10.

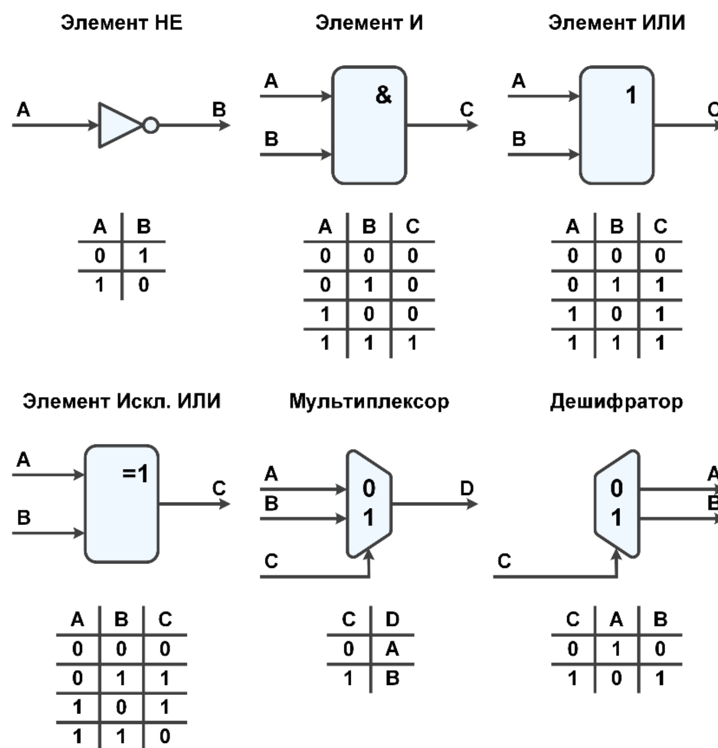

Рисунок 10 – Основные элементы комбинационной логики

Количество входов и выходов у комбинационных узлов и блоков может отличаться, аналогично может отличаться и разрядность обрабатываемых или коммутируемых данных, поэтому конфигурация элемента комбинационной логики берется под конкретную поставленную задачу.

Простые схемы с памятью строятся на взаимодействии комбинационной логики и D-триггеров (рис. 11). К таким элементам относятся регистры с управляющими входами, сдвиговые регистры, счетчики.

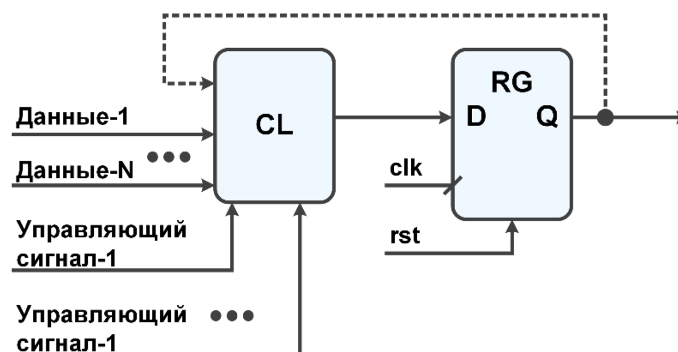

Рисунок 11 – Простая схема с памятью

Простая схема с памятью состоит из двух частей – комбинационной логики и регистра (набор параллельно соединенных D-триггеров). Основной частью комбинационной схемы обычно выступает мультиплексор,

который выбирает записываемые данные в зависимости от управляющих сигналов. Регистр хранит полученное значение комбинационной схемой.

Количество управляющих входов и их разрядность может отличаться по аналогии с комбинационными схемами.

Рассмотрим реализацию схемы счетчика, который поддерживает следующие режимы работы: хранение, загрузка, инкремент, декремент и синхронный сброс. Функционирование счетчика с приоритетами сигналов представлено в табл. 2.

Таблица 2 – Функционирование счетчика

RE	LD	EN	DIR	Описание
1	x	x	x	Синхронный сброс счетчика (Q <= 0)
0	1	x	x	Загрузка значения счетчика с входа данных D (Q <= D)
0	0	1	0	Инкремент содержимого счетчика (Q <= Q + 1)
0	0	1	1	Декремент содержимого счетчика (Q <= Q - 1)
0	0	0	0	Хранение

Структурная схема счетчика согласно рис. 11. и табл. 2. представлена на рис. 12. На схеме представлен каскад мультиплексоров, который в зависимости от управляющих сигналов формирует значения на входе данных D-триггера.

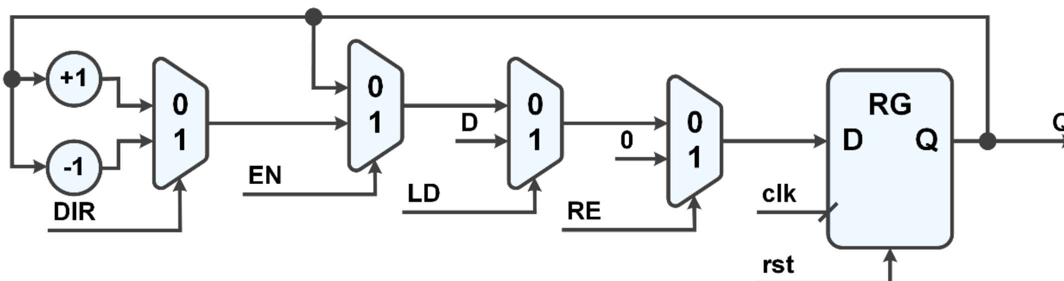

Рисунок 12 – Структурная схема счетчика

Аналогично, как и простые схемы с памятью, конечные автоматы состоят из комбинационной схемы и регистров. В отличие от выше рассмотренных простых схем с памятью, конечные автоматы могут выполнять разный набор действий при одном и том же наборе управляющих сигналов в зависимости от текущего состояния и формировать управляющие сигналы для других функциональных блоков системы. Архитектурами конечных автоматов являются автомат Мили и автомат Мура [15].

На рис. 13 представлена структурная схема конечного автомата. Новое значение состояния конечного автомата формируется на основании текущего состояния, входных признаков, сформированных заранее, и данных, которые анализируются комбинационной схемой.

Конечный автомат может формировать управляющие сигналы двух видов: первый – когда управляющий сигнал формируется комбинационной схемой, второй – когда конечный автомат генерирует управляющие сигналы для простых схем с памятью, которые формируют управляющие сигналы для остальных функциональных блоков системы.

Пример функциональной схемы конечного автомата и его графа переходов представлен на рис. 7 и рис. 8.

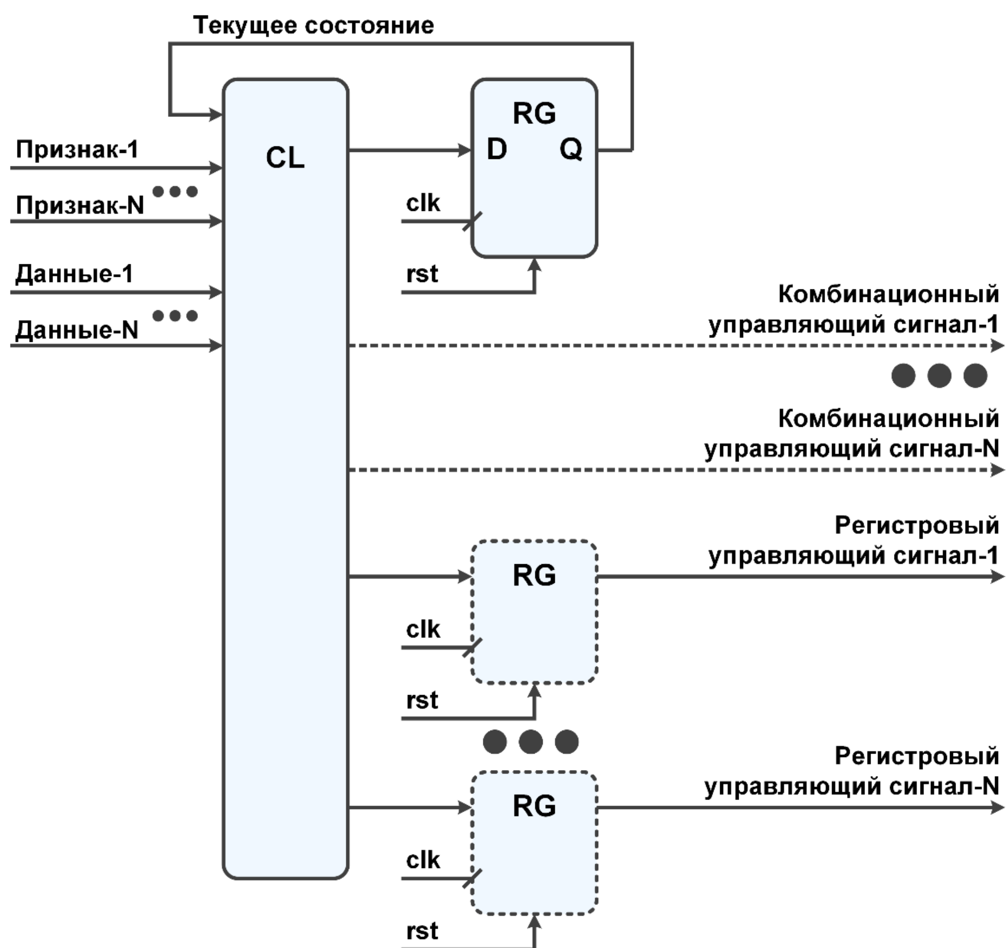

Рисунок 13 – Схема конечного автомата

В ходе рассмотрения системы приемника данных по протоколу UART выявлены необходимые аспекты для описания аппаратных моделей. Для описания аппаратной модели достаточно использовать три типа функциональных узлов: комбинационные схемы, простые схемы с памятью и конечные автоматы. Каждый узел может имеет набор входных и выходных портов, которые соединяются связями с портами других узлов, тем самым образововывая функциональные блоки.

Сравним реализации аппаратных моделей на языке проектирования аппаратуры и языке программирования высокого уровня для выявления сходства описания.

Определение сходств описания аппаратных и программных моделей

Рассмотрим первый пример (рис. 14), на котором представлена комбинационная схема, реализующая сравнение двух 4-разрядных чисел с последующей конъюнкцией с одноразрядным сигналом «A».

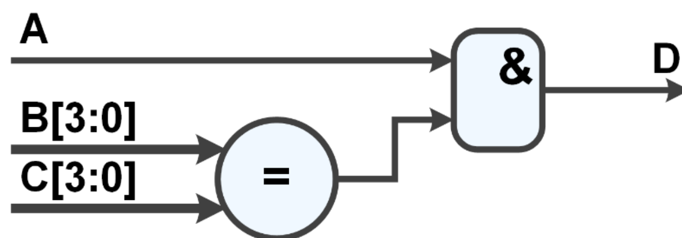

Рисунок 14 – Простая комбинационная схема

Опишем модуль данной комбинационной схемы (листинг 1) на языке описания аппаратуры Verilog HDL и его тестовый модуль (листинг 2).

Листинг 1 – Модуль комбинационной схемы (Verilog HDL)

```
module cl(
 input       A,
 input [3:0] B,
 input [3:0] C,
 output      D
);
 assign D = A & (B == C);
endmodule
```

Листинг 2 – Тестовый модуль комбинационной схемы (Verilog HDL)

```
module tb_cl;

 // Inputs
reg A;
reg [3:0] B;
reg [3:0] C;

 // Outputs
wire D;

 // Instantiate the Unit Under Test (UUT)
cl uut (.A(A), .B(B), .C(C), .D(D));

 // Tests
initial begin
 A = 1'b0; B = 4'h0; C = 4'h0; #100;
 A = 1'b0; B = 4'h5; C = 4'hA; #100;
 A = 1'b0; B = 4'hD; C = 4'hD; #100;
 A = 1'b1; B = 4'h5; C = 4'hA; #100;
 A = 1'b1; B = 4'hD; C = 4'hD; #100;
 $stop;
End

endmodule
```

Симуляция разработанного модуля комбинационной схемы представлена на рис. 15. На временной диаграмме представлен перебор комбинаций, когда сигнал «A» равен «0»/«1» и число «B» равно/неравно «C».

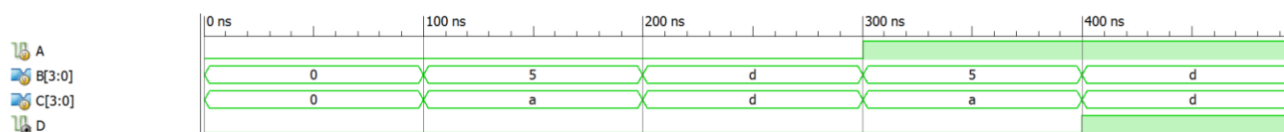

Рисунок 15 – Временная диаграмма симуляции модуля комбинационной схемы

Теперь, опишем модуль комбинационной схемы (листинг 3) на языке программирования Python и его тестовый модуль (листинг 4).

Листинг 3 – Модуль комбинационной схемы (Python)

```
class cl:
 def __init__(self):
   self.out = 0
 def apply(self, a, b, c):
  self.out = a and (b == c)
```

Листинг 4 – Тестовый модуль комбинационной схемы (Python)

```
tb_cl = cl()
a_set = [0x0, 0x0, 0x0, 0x1, 0x1]
b_set = [0x0, 0x5, 0xd, 0x5, 0xd]
c_set = [0x0, 0xa, 0xd, 0xa, 0xd]
print("   A   |   B   |   C   |   D   ")
for i in range(5):
 tb_cl.apply(a_set[i], b_set[i], c_set[i])
 print("  {0:#x}  |  {1:#x}  |  {2:#x}  |  {3:#x}  ".format(a_set[i], b_set[i],
c_set[i], tb_cl.out))
```

Результат симуляции тестового модуля комбинационной схемы в консоль, представлен на рис. 16.

```
 A   |  B   |  C   |  D
0x0  | 0x0  | 0x0  | 0x0
0x0  | 0x5  | 0xa  | 0x0
0x0  | 0xd  | 0xd  | 0x0
0x1  | 0x5  | 0xa  | 0x0
0x1  | 0xd  | 0xd  | 0x1
```

Рисунок 16 – Вывод в консоль симуляции модуля комбинационной схемы

Рассмотрим пример реализации простой схемы с памятью (рис. 17), в качестве которой будет выступать счетчик с управляющими входами «CE» и «RE».

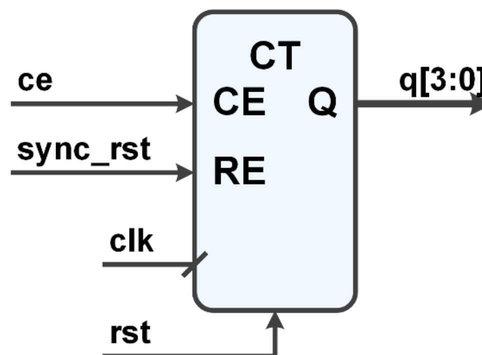

Рисунок 17 – Интерфейс счетчика с управляющими входами

Счетчик может работать в трех режимах: инкремент, синхронный сброс и хранение. Описание управляющих сигналов представлено в табл. 3.

Таблица 3 – Функционирование счетчика с управляющими входами

RE	CE	Описание
1	x	Синхронный сброс счетчика (Q <= 0)
0	1	Инкремент содержимого счетчика (Q <= Q + 1)
0	0	Хранение

Аналогично, опишем модуль счетчика (листинг 5) на языке проектирования аппаратуры Verilog HDL и его тестовый модуль (листинг 6).

Листинг 5 – Модуль счетчика с управляющими входами (Verilog HDL)

```
module ct(
 input           CLK,
 input           RST,
 input           CE,
 input           RE,
 output reg [3:0] Q
);
always @(posedge CLK, posedge RST)
 if(RST)     Q <= 4'h0;
 else if(RE) Q <= 4'h0;
 else if(CE) Q <= Q + 1’b1;
endmodule
```

Листинг 6 – Модуль верификации счетчика с управляющими входами (Verilog HDL)

```
module tb_ct;
 // Inputs
 reg CLK;
 reg RST;
 reg CE;
 reg RE;
 // Outputs
 wire [3:0] Q;
 // Instantiate the Unit Under Test (UUT)
 ct uut (
      .CLK(CLK),
      .RST(RST),
      .CE(CE),
      .RE(RE),
      .Q(Q)
 );
 always begin CLK = 0; #10; CLK = 1; #10; end
 initial begin
      RST = 1; RE = 0; CE = 0; #25;
      RST = 0; RE = 0; CE = 1; #115;
      RST = 0; RE = 0; CE = 0; #35;
      RST = 0; RE = 1; CE = 0; #25;
      RST = 0; RE = 0; CE = 0; #25;
      $stop;
 end
endmodule
```

Симуляция разработанного модуля счетчика с управляющими входами представлена на рис. 18.

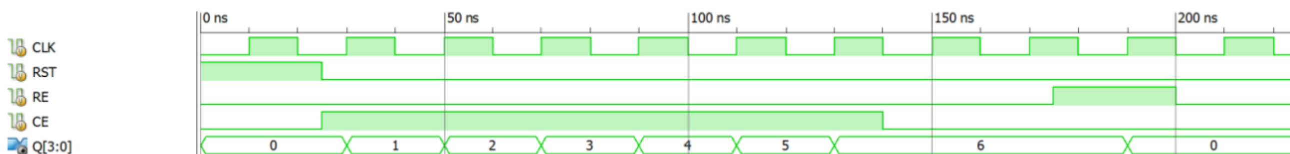

Рисунок 18 – Временная диаграмма симуляции счетчика с управляющими входами

Счетчик – простая с схема с памятью, поэтому в классе объявляются свойства «q» и «last_clk», которые отвечают за текущее значение счетчика и предыдущее значение синхросигнала «clk». Свойство «last_clk» необходимо для определения восходящего фронта синхросигнала «clk» в методе «apply» Программная модель счетчика представлена в листинге 7.

Листинг 7 – Модуль счетчика с управляющими входами (Python)

```
class ct:
 def __init__(self):
   self.q = 0
   self.last_clk = 0

 def apply(self, rst, clk, re, ce):
  if rst: self.q = 0
  elif not self.last_clk and clk:
   if re: self.q = 0
   elif ce: self.q = (self.q + 1) & 0xF
  self.last_clk = clk
```

Так как счетчик не комбинационная схема, а схема с памятью, которая работает по переднему фронту тактового синхросигнала, то в тестовом модуле необходимо предусмотреть генерацию этого сигнала.

Тестовый модуль на языке программирования Python представлен в листинге 8.

Листинг 8 – Тестовый модуль счетчика с управляющими входами (Python)

```
tb_ct = ct()

signals_set = [
  (1, 0, 0),
  (0, 0, 1),
  (0, 0, 0),
  (0, 1, 0),
  (0, 0, 0)
]

change_points = [0, 25, 140, 175, 200]
signals_set_index = 0
points = [10, 30, 50, 70, 90, 110, 130, 150, 170, 190, 210]
switch_clk = 10
rst, clk, ce, sync_rst = 0, 0, 0, 0
print("    T     |   R   |  CE   |   Q   ")

for i in range(225):
 switch_clk -= 1
 if switch_clk == 0:
  clk = not clk
  switch_clk = 10

 if i in change_points:
   rst, sync_rst, ce = signals_set[signals_set_index]
   signals_set_index += 1

 example.apply(rst, clk, sync_rst, ce)

 if i in points:
  print("  {0: >3} нс  |  {1:#x}  |  {2:#x}  |  {3:#x}  ".format(i, sync_rst,
ce, tb_ct.q))
```

Результат симуляции тестового модуля счетчика с управляющими входами описанного на языке программирования Python представлен на рис. 19.

T	RE	CE	Q
10 нс	0x0	0x0	0x0
30 нс	0x0	0x1	0x1
50 нс	0x0	0x1	0x2
70 нс	0x0	0x1	0x3
90 нс	0x0	0x1	0x4
110 нс	0x0	0x1	0x5
130 нс	0x0	0x1	0x6
150 нс	0x0	0x0	0x6
170 нс	0x0	0x0	0x6
190 нс	0x1	0x0	0x0
210 нс	0x0	0x0	0x0

Рисунок 19 – Вывод в консоль симуляции тестового модуля счётчика

Рассмотрим более подробно схему на рис.6, если выделить блоки (рис. 20), составляющих данную схему, можно заметить, что счетчик и комбинационная схема уже были реализованы с помощью программных и аппаратных моделей ранее.

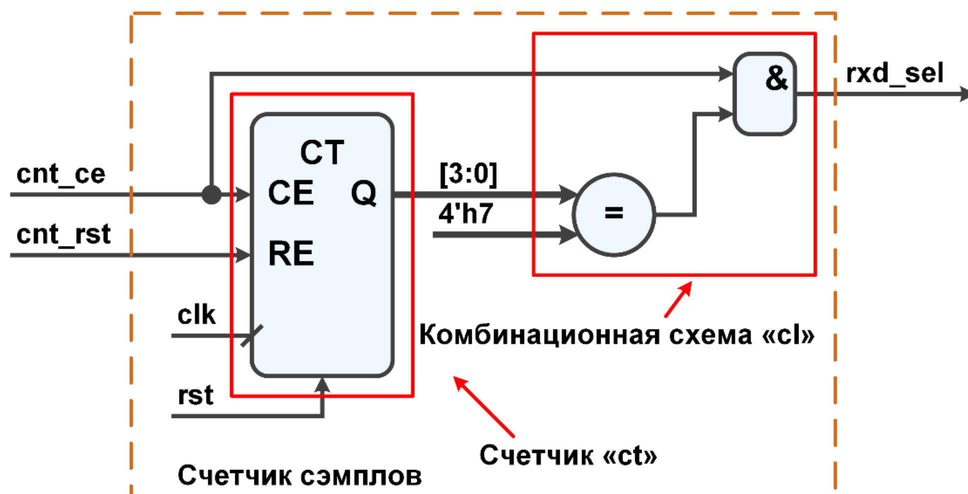

Рисунок 20 – Функциональные узлы счетчика сэмплов

Описание аппаратной модели функционального блока счетчика сэмплов на языке описания аппаратуры Verilog HDL представлено в листинге 9, а его тестовый модуль в листинге 10.

Листинг 9 – Модуль счетчика сэмплов (Verilog HDL)

```
module sample_ct(
 input clk, input rst,
 input cnt_ce, input cnt_rst, output rxd_sel
);

wire [3:0] cnt;
ct ct(
 .CLK(clk), .RST(rst),
 .CE(cnt_ce), .RE(cnt_rst), .Q(cnt));
cl cl (
 .A(cnt_ce), .B(cnt),
 .C(4'h7), .D(rxd_sel));

endmodule
```

Листинг 10 – Тестовый модуль счетчика сэмплов (Verilog HDL)

```
module tb_sample_ct;
 // Inputs
 reg clk;
 reg rst;
 reg cnt_ce;
 reg cnt_rst;
 // Outputs
 wire rxd_sel;
 // Instantiate the Unit Under Test (UUT)
 sample_ct uut (
      .clk(clk),
      .rst(rst),
      .cnt_ce(cnt_ce),
      .cnt_rst(cnt_rst),
      .rxd_sel(rxd_sel)
 );
 always begin
  clk = 0; #10;
  clk = 1; #10;
 end
 initial begin
      rst = 1; cnt_rst = 0; cnt_ce = 0; #25;
      rst = 0; cnt_rst = 0; cnt_ce = 1; #215;
      rst = 0; cnt_rst = 1; cnt_ce = 1; #55;
      rst = 0; cnt_rst = 0; cnt_ce = 1; #25;
      rst = 0; cnt_rst = 0; cnt_ce = 0; #40;
      $stop;
 end
endmodule
```

Симуляция разработанного модуля счетчика сэмплов представлена на рис. 21.

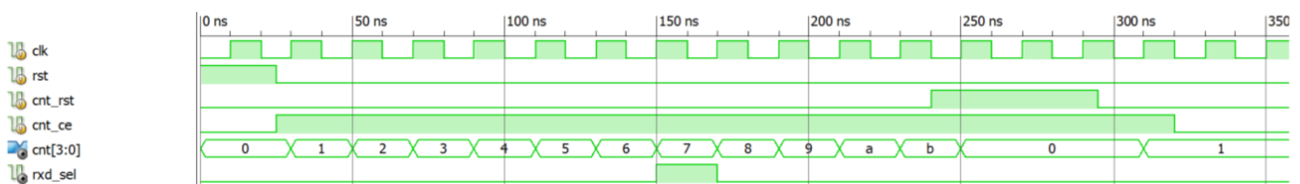

Рисунок 21 – Временная диаграмма симуляции счетчика сэмплов

Используя классы описанные на языке программирования Python, опишем счетчик сэмплов с помощью программной модели (листинг 11) и его тестовый модуль (листинг 12)

Листинг 11 – Модуль счетчика сэмплов (Python)

```
class sample_ct:
 def __init__(self):
  self.ct = ct()
  self.cl = cl()
  self.rxd_sel = 0

 def apply(self, rst, clk, cnt_rst, cnt_ce):
  self.ct.apply(rst, clk, cnt_rst, cnt_ce)
  self.cl.apply(cnt_ce, self.ct.q, 7)
  self.rxd_sel = self.cl.out
```

Листинг 12 – Тестовый модуль счетчика сэмплов (Python)

```
tb_sample_ct = sample_ct()

signals_set = [
  (1, 0, 0),
  (0, 0, 1),
  (0, 1, 1),
  (0, 0, 1),
  (0, 0, 0)
]

change_points = [0, 25, 240, 295, 320, 360]
signals_set_index = 0
points = [10, 30, 50, 70, 90, 110, 130, 150, 170, 190, 210]
switch_clk = 10
rst, clk, sync_rst, ce = 0, 0, 0, 0
print("    T     |   R   |  CE   |  cnt  |  rxd_sel")

for i in range(320):
 switch_clk -= 1
 if switch_clk == 0:
  clk = not clk
  switch_clk = 10

 if i in change_points:
  rst, sync_rst, ce = signals_set[signals_set_index]
  signals_set_index += 1

 tb_sample_ct.apply(rst, clk, sync_rst, ce)

 if i in points:
  print("  {0: >3} нс   |  {1:#x}   |  {2:#x}   |  {3:#x}   |  {4:#x}".format(i,
sync_rst, ce, tb_sample_ct.ct.q,  tb_sample_ct.cl.out))
```

Результат симуляции тестового модуля счетчика сэмплов описанного на языке программирования Python представлен на рис. 22.

```
  T     | RE   | CE   | cnt  | rxd_sel
 10 нс  | 0x0  | 0x0  | 0x0  | 0x0
 30 нс  | 0x0  | 0x1  | 0x1  | 0x0
 50 нс  | 0x0  | 0x1  | 0x2  | 0x0
 70 нс  | 0x0  | 0x1  | 0x3  | 0x0
 90 нс  | 0x0  | 0x1  | 0x4  | 0x0
110 нс  | 0x0  | 0x1  | 0x5  | 0x0
130 нс  | 0x0  | 0x1  | 0x6  | 0x0
150 нс  | 0x0  | 0x1  | 0x7  | 0x1
170 нс  | 0x0  | 0x1  | 0x8  | 0x0
190 нс  | 0x0  | 0x1  | 0x9  | 0x0
210 нс  | 0x0  | 0x1  | 0xa  | 0x0
```

Рисунок 22 – Вывод в консоль симуляции тестового модуля счётчика сэмплов

Рассмотрим реализацию графа конечного автомата, представленного на рис. 7, на языке описания аппаратуры Verilog HDL (листинг 13).

Листинг 13 – Модуль конечного автомата (Verilog HDL)

```
module fsm(
 input            clk,
 input            rst,
 input            sync_rxd,
 input            rxd_sel,
 output reg [7:0] data,
 output reg       ready,
 output reg       error,
 output           cnt_rst
);

parameter [1:0] S0 = 2'b00;
parameter [1:0] S1 = 2'b01;
parameter [1:0] S2 = 2'b10;
parameter [1:0] S3 = 2'b11;

reg [1:0] state;
reg [2:0] rcv_ct;

always @(posedge clk, posedge rst)
 if(rst) begin
   state  <= S0;
   rcv_ct <= 3'b000;
   data   <= 8'h00;
   ready  <= 1'b0;
   error  <= 1'b0;
  end
 else
  case(state)
   S0: begin
 ready <= 1'b0;
 error <= 1'b0;
 if(~sync_rxd) state <= S1;
end
S1: if(rxd_sel)
 if(sync_rxd) state <= S0;
 else state <= S2;
S2: if(rxd_sel) begin
 data <= {sync_rxd, data[7:1]};
 rcv_ct <= rcv_ct + 1'b1;
 if(rcv_ct == 3'h7) state <= S3;
end
S3: if(rxd_sel) begin
 if(sync_rxd) begin
  state <= S0;
  ready <= 1'b1;
 end
 else error <= 1'b1;
end
  endcase

 assign cnt_rst = state == S0;
endmodule
```

Тестовый модуль конечного автомата приемника UART, проверяющий прием кадра данных, соответствующего формату, и прием кадра с ошибкой представлен в листинге 14.

Листинг 14 – Тестовый модуль конечного автомата (Verilog HDL)

```
module tb_fsm;
 // Inputs
 reg clk;
 reg rst;
 reg sync_rxd;
 reg rxd_sel;
 // Outputs
 wire [7:0] data;
 wire       ready;
 wire       error;
 wire       cnt_rst;
 // Instantiate the Unit Under Test (UUT)
 fsm uut (
     .clk(clk),
     .rst(rst),
     .sync_rxd(sync_rxd),
     .rxd_sel(rxd_sel),
     .data(data),
     .ready(ready),
     .error(error),
     .cnt_rst(cnt_rst)
 );
 always begin
  clk = 0; #10;
  clk = 1; #10;
 end
 initial begin
     rst = 1; sync_rxd = 1; rxd_sel = 0; #20
     rst = 0; sync_rxd = 0; rxd_sel = 0; #20

     rst = 0; sync_rxd = 0; rxd_sel = 1; #20
     rst = 0; sync_rxd = 0; rxd_sel = 0; #20

     rst = 0; sync_rxd = 1; rxd_sel = 1; #20
     rst = 0; sync_rxd = 1; rxd_sel = 0; #20

     rst = 0; sync_rxd = 0; rxd_sel = 1; #20
     rst = 0; sync_rxd = 0; rxd_sel = 0; #20
     rst = 0; sync_rxd = 1; rxd_sel = 1; #20
     rst = 0; sync_rxd = 1; rxd_sel = 0; #20
     rst = 0; sync_rxd = 1; rxd_sel = 1; #20
     rst = 0; sync_rxd = 1; rxd_sel = 0; #20
     rst = 0; sync_rxd = 0; rxd_sel = 1; #20
     rst = 0; sync_rxd = 0; rxd_sel = 0; #20
     rst = 0; sync_rxd = 1; rxd_sel = 1; #20
     rst = 0; sync_rxd = 1; rxd_sel = 0; #20
     rst = 0; sync_rxd = 1; rxd_sel = 1; #20
     rst = 0; sync_rxd = 1; rxd_sel = 0; #20
     rst = 0; sync_rxd = 0; rxd_sel = 1; #20
     rst = 0; sync_rxd = 0; rxd_sel = 0; #20
```

Продолжение Листинга 14

```
        rst = 0; sync_rxd = 1; rxd_sel = 1; #20
        rst = 0; sync_rxd = 1; rxd_sel = 0; #20

        rst = 0; sync_rxd = 0; rxd_sel = 1; #20
        rst = 0; sync_rxd = 0; rxd_sel = 1; #20
        rst = 0; sync_rxd = 0; rxd_sel = 0; #20

        rst = 0; sync_rxd = 1; rxd_sel = 1; #20
        rst = 0; sync_rxd = 1; rxd_sel = 0; #20

        rst = 0; sync_rxd = 0; rxd_sel = 1; #20
        rst = 0; sync_rxd = 0; rxd_sel = 0; #20

        rst = 0; sync_rxd = 1; rxd_sel = 1; #20
        rst = 0; sync_rxd = 1; rxd_sel = 0; #20

        rst = 0; sync_rxd = 1; rxd_sel = 1; #20
        rst = 0; sync_rxd = 1; rxd_sel = 0; #20

        rst = 0; sync_rxd = 0; rxd_sel = 1; #20
        rst = 0; sync_rxd = 0; rxd_sel = 0; #20

        rst = 0; sync_rxd = 1; rxd_sel = 1; #20
        rst = 0; sync_rxd = 1; rxd_sel = 0; #20

        rst = 0; sync_rxd = 1; rxd_sel = 1; #20
        rst = 0; sync_rxd = 1; rxd_sel = 0; #20

        rst = 0; sync_rxd = 0; rxd_sel = 1; #20
        rst = 0; sync_rxd = 0; rxd_sel = 0; #20

        rst = 0; sync_rxd = 0; rxd_sel = 1; #20
        rst = 0; sync_rxd = 0; rxd_sel = 0; #20

        rst = 0; sync_rxd = 1; rxd_sel = 1; #20
        rst = 0; sync_rxd = 1; rxd_sel = 0; #20
        $stop;
 end

endmodule
```

Симуляция разработанного модуля конечного автомата представлена на рис. 23.

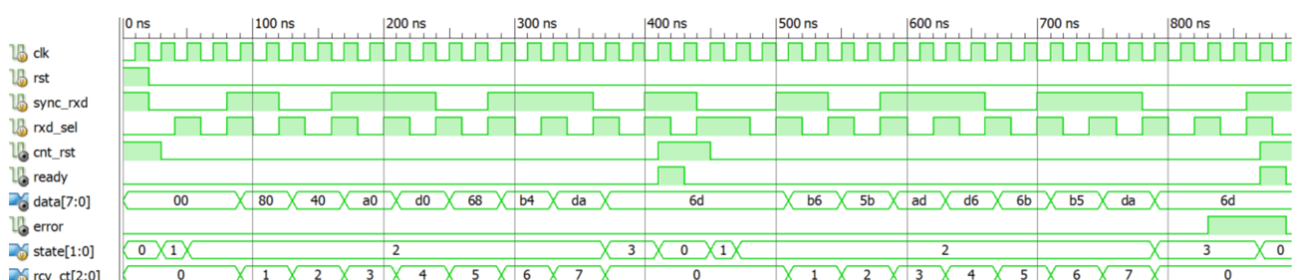

Рисунок 23 – Временная диаграмма симуляции модуля конечного автомата

Опишем конечный автомат с помощью программной модели (листинг 15) на языке программирование Python и его тестовый модуль (листинг 16)

Листинг 15 – Модуль конечного автомата (Python)

```
class fsm:

 def __init__(self):
  self.last_clk = 0
  self.state    = S0
  self.rcv_ct   = 0
  self.data     = 0
  self.ready    = 0
  self.error    = 0
  self.cnt_rst  = 0

 def apply(self, rst, clk, sync_rxd, rxd_sel):
  if rst:
   self.rcv_ct = 0
   self.data = 0
   self.ready = 0
   self.error = 0

  if not self.last_clk and clk:
   if self.state == S0:
    self.ready = 0
    self.error = 0
    if not sync_rxd: self.state = S1

   elif self.state == S1:
    if rxd_sel:
     if sync_rxd: self.state = S0
     else: self.state = S2

   elif self.state == S2:
    if rxd_sel:
     self.data = (self.data >> 1) | (0x80 * sync_rxd)
     if self.rcv_ct == 7: self.state = S3
     self.rcv_ct = (self.rcv_ct + 1) & 0x7

   elif self.state == S3:
    if rxd_sel:
     if sync_rxd:
      self.ready = 1
      self.state = S0
     else: self.error = 1;

  self.cnt_rst = self.state == S0
  self.last_clk = clk
```

Листинг 16 – Тестовый модуль конечного автомата (Python)

```
tb_fsm = fsm()

signals_set = [
(1, 1, 0), (0, 0, 0), (0, 0, 1), (0, 0, 0),
(0, 1, 1), (0, 1, 0), (0, 0, 1), (0, 0, 0),
(0, 1, 1), (0, 1, 0), (0, 1, 1), (0, 1, 0),
(0, 0, 1), (0, 0, 0), (0, 1, 1), (0, 1, 0),
(0, 1, 1), (0, 1, 0), (0, 0, 1), (0, 0, 0),
(0, 1, 1), (0, 1, 0), (0, 0, 1), (0, 0, 1),
(0, 0, 0), (0, 1, 1), (0, 1, 0), (0, 0, 1),
(0, 0, 0), (0, 1, 1), (0, 1, 0), (0, 1, 1),
(0, 1, 0), (0, 0, 1), (0, 0, 0), (0, 1, 1),
(0, 1, 0), (0, 1, 1), (0, 1, 0), (0, 0, 1),
(0, 0, 0), (0, 0, 1), (0, 0, 0), (0, 1, 1),
(0, 1, 0) ]

change_points = [i for i in range(0, 900, 20)]
signals_set_index = 0
points = [i for i in range(10, 900, 20)]
switch_clk = 10
clk, rst, sync_rxd, rxd_sel = 0, 0, 0, 0
print("    T     | sync_rxd | rxd_sel | cnt_rst | ready | data | error | state
| rcv_ct")

for i in range(900):
 switch_clk -= 1
 if switch_clk == 0:
  clk = not clk
  switch_clk = 10

 if i in change_points:
  rst, sync_rxd, rxd_sel = signals_set[signals_set_index]
  signals_set_index += 1

 tb_fsm.apply(rst, clk, sync_rxd, rxd_sel)

 if i in points:
    print("  {0: >3} нс  |    {1:#x}   |   {2:#x}   |   {3:#x}   |  {4:#x}  |
0x{5:0>2x} |  {6:#x}  |  {7:#x}  |  {8:#x}".format(
     i,  sync_rxd,  rxd_sel,  tb_fsm.cnt_rst,    tb_fsm.ready,  tb_fsm.data,
tb_fsm.error, tb_fsm.state, tb_fsm.rcv_ct))
```

Результат симуляции тестового модуля счетчика конечного автомата описанного на языке программирования Python представлен на рис. 24.

T	sync_rxd	rxd_sel	cnt_rst	ready	data	error	state	rcv_ct
10 нс	0x1	0x0	0x1	0x0	0x00	0x0	0x0	0x0
30 нс	0x0	0x0	0x0	0x0	0x00	0x0	0x1	0x0
50 нс	0x0	0x1	0x0	0x0	0x00	0x0	0x2	0x0
70 нс	0x0	0x0	0x0	0x0	0x00	0x0	0x2	0x0
90 нс	0x1	0x1	0x0	0x0	0x80	0x0	0x2	0x1
110 нс	0x1	0x0	0x0	0x0	0x80	0x0	0x2	0x1
130 нс	0x0	0x1	0x0	0x0	0x40	0x0	0x2	0x2
150 нс	0x0	0x0	0x0	0x0	0x40	0x0	0x2	0x2
170 нс	0x1	0x1	0x0	0x0	0xa0	0x0	0x2	0x3
190 нс	0x1	0x0	0x0	0x0	0xa0	0x0	0x2	0x3
210 нс	0x1	0x1	0x0	0x0	0xd0	0x0	0x2	0x4
230 нс	0x1	0x0	0x0	0x0	0xd0	0x0	0x2	0x4
250 нс	0x0	0x1	0x0	0x0	0x68	0x0	0x2	0x5
270 нс	0x0	0x0	0x0	0x0	0x68	0x0	0x2	0x5
290 нс	0x1	0x1	0x0	0x0	0xb4	0x0	0x2	0x6
310 нс	0x1	0x0	0x0	0x0	0xb4	0x0	0x2	0x6
330 нс	0x1	0x1	0x0	0x0	0xda	0x0	0x2	0x7
350 нс	0x1	0x0	0x0	0x0	0xda	0x0	0x2	0x7
370 нс	0x0	0x1	0x0	0x0	0x6d	0x0	0x3	0x0
390 нс	0x0	0x0	0x0	0x0	0x6d	0x0	0x3	0x0
410 нс	0x1	0x1	0x1	0x1	0x6d	0x0	0x0	0x0
430 нс	0x1	0x0	0x1	0x0	0x6d	0x0	0x0	0x0
450 нс	0x0	0x1	0x0	0x0	0x6d	0x0	0x1	0x0
470 нс	0x0	0x1	0x0	0x0	0x6d	0x0	0x2	0x0
490 нс	0x0	0x0	0x0	0x0	0x6d	0x0	0x2	0x0
510 нс	0x1	0x1	0x0	0x0	0xb6	0x0	0x2	0x1
530 нс	0x1	0x0	0x0	0x0	0xb6	0x0	0x2	0x1
550 нс	0x0	0x1	0x0	0x0	0x5b	0x0	0x2	0x2
570 нс	0x0	0x0	0x0	0x0	0x5b	0x0	0x2	0x2
590 нс	0x1	0x1	0x0	0x0	0xad	0x0	0x2	0x3
610 нс	0x1	0x0	0x0	0x0	0xad	0x0	0x2	0x3
630 нс	0x1	0x1	0x0	0x0	0xd6	0x0	0x2	0x4
650 нс	0x1	0x0	0x0	0x0	0xd6	0x0	0x2	0x4
670 нс	0x0	0x1	0x0	0x0	0x6b	0x0	0x2	0x5
690 нс	0x0	0x0	0x0	0x0	0x6b	0x0	0x2	0x5
710 нс	0x1	0x1	0x0	0x0	0xb5	0x0	0x2	0x6
730 нс	0x1	0x0	0x0	0x0	0xb5	0x0	0x2	0x6
750 нс	0x1	0x1	0x0	0x0	0xda	0x0	0x2	0x7
770 нс	0x1	0x0	0x0	0x0	0xda	0x0	0x2	0x7
790 нс	0x0	0x1	0x0	0x0	0x6d	0x0	0x3	0x0
810 нс	0x0	0x0	0x0	0x0	0x6d	0x0	0x3	0x0
830 нс	0x0	0x1	0x0	0x0	0x6d	0x1	0x3	0x0
850 нс	0x0	0x0	0x0	0x0	0x6d	0x1	0x3	0x0
870 нс	0x1	0x1	0x1	0x1	0x6d	0x1	0x0	0x0
890 нс	0x1	0x0	0x1	0x0	0x6d	0x0	0x0	0x0

Рисунок 24 – Вывод в консоль симуляции тестового модуля конечного автомата

На основании рассмотренных описаний аппаратных моделей составлена сравнительная характеристика, которая представлена в табл. 3.

Таблица 3 – Сравнительная характеристика описания аппаратных моделей

Язык описания аппаратуры Verilog HDL	**Язык программирования Python**
Комбинационная логика	

`module cl(`
` input p0,`
` input p1,`
` …`
` input pn,`
` output f`
`);`
`assign f = логическая_функция(p0,…,pn);`
`endmodule`
	`class cl:`
	` def __init__(self):`
	`  self.f = 0`
	` def apply(self, p0, p1, …, pn):`
	`  self.f = логическая_функция(p0,…,pn)`

Продолжение Таблицы 3

Простые схемы с памятью

```
module simple_rg(
 input rst,
 input clk,
 input p0,
 input p1,
 …
 input pn,
 input data0,
 input data1,
 …
 input datan,
 output q
);
always @(posedge clk, posedge rst)
 if(rst) q <= начальное_значение;
 else
  if(p0) q <= data0;
  else if(p1) q <= data1;
  …
  else if(pn) q <= datan;
endmodule
```

```
class simple_rg:
 def __init__(self):
  self.q = 0
  self.last_clk = 0
 def apply(self, rst, clk, p0, p1, …,
pn, data0, data1, …, datan):
  if rst: self.q = начальное_значение
  elif not self.last_clk and clk:
   if p0: self.q = data0
   elif p1: self.q = data1
   …
   elif pn: self.q = datan
  self.last_clk = clk
```

Конечный автомат

```
module fsm(
 input rst,
 input clk,
 input p0,
 input p1,
 …
 input pn,
 output cl_q0,
 output cl_q1,
 …
 output cl_qn,
 output reg_q0,
 output reg_q1,
 …
 output reg_qn
);
reg состояние_автомата;
always @(posedge clk, posedge rst)
 if(rst) begin
  состояние_автомата <=
начальное_состояние;
  reg_q0 <= начальное_состояние;
  reg_q1 <= начальное_состояние;
  …
  reg_qn <= начальное_состояние;
 end
else
 case(состояние_автомата)
  состояние-0: begin
```

```
class fsm:
 def __init__(self):
  self.state = начальное_состояние
  self.cl_q0 = начальное_состояние
  self.cl_q1 = начальное_состояние
  ...
  self.cl_qn = начальное_состояние
  self.reg_q0 = начальное_состояние
  self.reg_q1 = начальное_состояние
  ...
  self.reg_qn = начальное_состояние
  self.last_clk = 0
 def apply(self, rst, clk, p0, p1, …,
pn):
  if rst:
   self.state =  начальное_состояние
   self.reg_q0 = начальное_состояние
   self.reg_q1 = начальное_состояние
   ...
   self.reg_qn = начальное_состояние
  elif not self.last_clk and clk:
   if self.state == состояние-0:
    if p0:
     self.state  = следующее_состояние
     self.reg_q0 = новое_состояние
     self.reg_q1 = новое_состояние
     ...
     self.reg_qn = новое_состояние
```

Продолжение Таблицы 3

```
if(p0) begin
     состояние_автомата <=
следующее_состояние;
     reg_q0 <= новое_состояние;
     reg_q1 <= новое_состояние;
     …
     reg_qn <= новое_состояние;
   end
   else if(p1) ...
   ...
   else if(pn) ...
  end
  состояние-1: begin
   if(p0) begin
     состояние_автомата <=
следующее_состояние;
     reg_q0 <= новое_состояние;
     reg_q1 <= новое_состояние;
     …
     reg_qn <= новое_состояние;
   end
   else if(p1) ...
   ...
   else if(pn) ...
  end
  ...
  состояние-n: begin
   if(p0) begin
     состояние_автомата <=
следующее_состояние;
     reg_q0 <= новое_состояние;
     reg_q1 <= новое_состояние;
     …
     reg_qn <= новое_состояние;
   end
   else if(p1): ...
   ...
   else if(pn) ...
  end
 endcase
assign cl_q0 = логическая_функция;
assign cl_q1 = логическая_функция;
…
assign cl_qn = логическая_функция;
endmodule
```

```
     elif p1: ...
     ...
     elif pn: ...
    elif self.state == состояние-1:
     if p0:
      self.state  = следующее_состояние
      self.reg_q0 = новое_состояние
      self.reg_q1 = новое_состояние
      ...
      self.reg_qn = новое_состояние
     elif p1: ...
     ...
     elif pn: ...
     ...
     elif self.state == состояние-n:
      if p0:
       self.state  = следующее_состояние
       self.reg_q0 = новое_состояние
       self.reg_q1 = новое_состояние
       ...
       self.reg_qn = новое_состояние
      elif p1: ...
      ...
      elif pn: ...
    self.cl_q0 = логическая_функция
    self.cl_q1 = логическая_функция
    ...
    self.cl_qn = логическая_функция
    self.last_clk = clk
```

В ходе сравнения аппаратных и программных моделей были выявлены их сходства описания. Интерфейс аппаратной модели на языке Verilog HDL описывается в виде входных и выходных портов, а в программной модели на Python в качестве входных портов выступают аргументы методов класса, а в качестве выходных – свойства или возвращаемые значения методов классов.

Аппаратную модель можно описать, используя объектно-ориентированное программирование с помощью классов. Также аппаратную модель можно описать с помощью процедурного программирования, при этом модель будет описываться как набор объявленных и инициализируемых переменных с последующей логикой преобразований их значений.

При описании комбинационных схем, простых схем с памятью и конечных автоматов используются схожие по своей смысловой нагрузке синтаксические конструкции языков описания аппаратуры и программирования. На основание этого выдвигается предположение, что аппаратные и программные модели можно описать с помощью унифицированного стандарта, который можно интерпретировать в целевой язык описания аппаратуры или программирования.

Таким образом использование одинаковых операций и методов в языках программирования и проектирования может позволить добиться унификации описания программных и аппаратных моделей.

Также язык программирования позволяет описать тестовое окружение, которое позволит верифицировать модель на системном уровне без необходимости установки специализированных САПР для проектирования в базисе СБИС.

На основании рассмотренных аспектов и сходств описания программных и аппаратных моделей предлагается маршрут проектирования функциональных блоков в базисе СБИС на основе унифицированного стандарта.

Маршрут проектирования функциональных блоков в базисе СБИС

Маршрут проектирования (рис. 25) функциональных блок в базисе СБИС включает в себя:

1. Описание аппаратной модели с помощью унифицированного стандарта.
2. Интерпретация аппаратной модели в целевой язык программирования (Python, C, C#, C++, Java и т.д.).
3. Моделирование и верификация полученных файлов на целевом языке программирования в интегрированной среде разработки. В случае если моделирование и тестирование прошло успешно, то осуществляется переход к шагу 4, иначе описание аппаратной модели корректируется (шаг 1).
4. Интерпретация аппаратной модели в целевой язык описания аппаратуры (Verilog HDL, VHDL, SystemC и т.д.).
5. Верификация аппаратной модели в симуляторе (iSim, Modelsim, Icarus и т.д.). Если тестирование прошло успешно, то осуществляется переход к шагу 6, иначе описание аппаратной модели корректируется (шаг 1).
6. Собрать проект в САПР (Xilinx ISE 14.7, Vivado, Quartus и т.д.).
7. Провести синтез, трансляцию и реализацию в кристалле СБИС.
8. Сгенерировать файл конфигурации.
9. Сконфигурировать СБИС на основании полученного файла конфигурации.

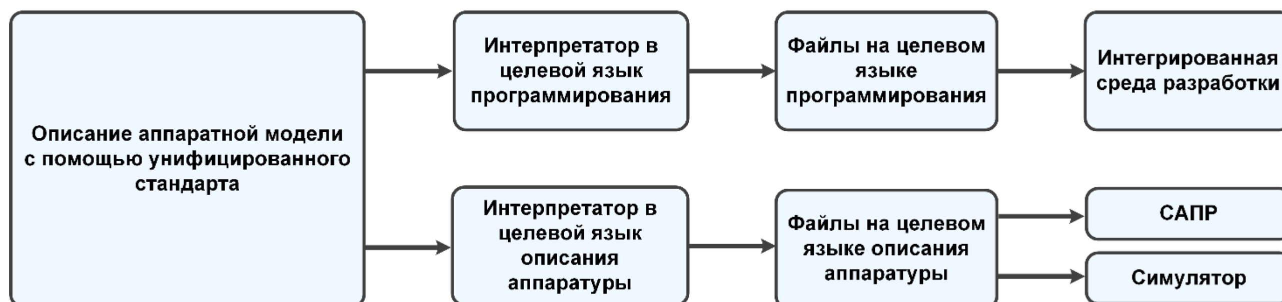

Рисунок 25 – Маршрут проектирования

Представленный маршрут проектирования представляет из себя стандартный маршрут, который используется при проектировании в базисе СБИС. За исключением, что описанную аппаратную модель можно сначала протестировать на системной уровне, а уже потом модель интерпретировать в файл описания аппаратуры с последующей сборкой в САПР.

Структура описания аппаратной модели с помощью унифицированного стандарта

Предлагается следующая структура описания аппаратной модели, которая представлена в листинге 17.

Листинг 17 – Структура описания аппаратной модели

```
- интерфейс
 - входы
 - выходы
- элементы
 - описание комбинационных схем
 - описание простых схем с памятью
 - описание конечных автоматов
 - модули
- связи между элементами
 - связь-1
 - связь-2
 …
 - связь-N
- тестовое окружение
 - тест-1
 …
 - тест-N
```

В интерфейсе описываются входные и выходные порты аппаратной модели. В элементах описываются функциональные узлы или блоки: комбинационные схемы, простые схемы с памятью, конечные автоматы и готовые модули. В связях между элементами задаются связи между функциональными узлами и блоками. В тестовом окружении задается набор тестов для верификации разрабатываемого аппаратного решения.

Такая структура позволит интерпретировать описание аппаратных моделей в целевой язык проектирования и программирования [16].

Заключение

В данной работе выявлены ключевые аспекты описания аппаратных моделей на примере приемника данных по протоколу UART. Выделены основные типы узлов, используемых при описании аппаратной модели: комбинационные схемы, простые схемы с памятью и конечные автоматы.

Проведено сравнение описания аппаратных моделей на языке проектирования Verilog HDL и на языке программирования Python. Определено сходство их описаний.

Предложены структура описания аппаратных моделей и маршрут проектирования в базисе СБИС.

Таким образом, выявленные аспекты, сходства и структура описания аппаратных моделей в дальнейшем лягут в основу стандарта для моделирования и тестирования аппаратных решений в базисе СБИС без ориентирования на конкретную интегрированную среду разработки или САПР.

Список литературы

1. Цыбов Н.Н. Исследование и анализ возможностей программной среды idealCircuit, Logisim, TINA Design, TINA-TI, DoCircuits, DIALux, AutoCAD Electrical для разработки учебных виртуальных электронных лабораторий // Вестник Кыргызского государственного университета строительства, транспорта и архитектуры им. Н.Исанова, 2016. – № 2(52). – С. 123-132.
2. Добыш Д.С., Куксевич В.Ф., Черненко Д.В. Проектирование элементов вычислительных систем в программах моделирования электронных схем // Материалы докладов 55-й международной научно-технической конференции преподавателей и студентов, 2022. – Т. 2. – С. 44-46.
3. Кряхтунов Г.М., Борисенко Н.В., Боронников А.С., Деменкова Т.А. Проектирование и разработка систем на базе программируемых логических интегральных схем. Часть 1: Практикум. – М.: МИРЭА – Российский технологический университет. – 2024. – 138 с.
4. Тарасов И.Е. ПЛИС Xilinx. Языки описания аппаратуры VHDL и Verilog, САПР, приемы проектирования. – М.: Горячая линия – Телеком. – 2022 – 538 с.
5. Басс А.В., Антонов М.А. Работа с ПЛИС с использованием языка описания аппаратуры Verilog // Известия Тульского государственного университета. Технические науки, 2019. – № 3. – С. 19-24.

6. Алехин В.А. Проектирование электронных систем с использованием SystemC и SystemC-AMS // Российский Технологический Журнал, 2020 – Т. 8, № 4(36). – С. 79-95.
7. Деменкова Т.А., Мадиу Б.А., Моделирование цифровых систем на уровне транзакций // Научно-технический вестник Поволжья, 2022. – № 12. – С. 299-302.
8. Bhadani R., Banik S., Tu H., Lukic S., Karsai G. Model-based Design Tool for Cyber-physical Power Systems using SystemC-AMS // 2024 IEEE Workshop on Design Automation for CPS and IoT (DESTION)
9. Turossi D., Baschirotto A. A SystemC-AMS Development Framework for High Power IC Test-Hardware // 2024 IEEE European Test Symposium (ETS).
10. Гаращенко А.В., Лашина Д.С., Никитин С. А., Николаев А. В., Прокопьев Е. А., Путря Ф. М., Цыренжапов Б.Н. Практика и перспективы применения открытых и собственных программных решений в маршруте верификации систем на кристалле // Труды ИСП РАН, 2020. – Т. 34, № 5. – С. 23-42.
11. Tarasov I.E. Application of UVM Methodology to Modeling Precision Digital Signal Processing Devices // PROGRAMMNAYA INGENERIA, 2024. – № 15(11). – P. 570-577.
12. Dave N. A Unified Model for Hardware/Software Codesign // PhD. thesis, MIT, 2011. – 188 P.
13. Pelton B., Sapek A., Eguro K. [et al] Wavefront Threading Enables Effective High-Level Synthesis // Proceedings of the ACM on Programming Languages, 2024. – Vol. 8, № PLDI, Article 190. – pp. 1066-1090.
14. Бухтеев А. Средства ESL-проектирования компании Celoxica: программно-ориентированный подход // ЭЛЕКТРОНИКА: НАУКА, ТЕХНОЛОГИЯ, БИЗНЕС, 2006. – № 8(74). – С. 106-111.
15. Лозовский В.В., Штрекер Е.Н., Боронников А.С., Казанцева Л.В. Теория автоматов: Учебное пособие. – М.: МИРЭА – Российский технологический университет. – 2024. – 455 с.
16. Кряхтунов Г.М., Боронников А.С. Подходы к созданию проприетарного формата представления данных // International Journal of Open Information Technologies. – 2024. – Т. 12, № 5. – С. 141-150.

References

1. Tsybov N.N. Research and analysis of the capabilities of the idealCircuit, Logisim, TINA Design, TINA-TI, DoCircuits, DIALux, AutoCAD Electrical software environment for the development of educational virtual electronic laboratories // Bulletin of the Kyrgyz State University of Construction, Transport and Architecture named after N.Isanov, 2016, No. 2(52). – pp. 123-132.
2. Dobysh D.S., Kuksevich V.F., Chernenko D.V. Design of computer system elements in electronic circuit modeling programs // Proceedings of the 55th International Scientific and Technical Conference of Teachers and Students, 2022. – Vol. 2. – pp. 44-46.
3. Kryakhtunov G.M., Borisenko N.V., Boronnikov A.S., Demenkova T.A. Design and development of systems based on programmable logic integrated circuits. Part 1: Practicum. Moscow: MIREA – Russian Technological University. - 2024. – 138 p.
4. Tarasov I.E. Xilinx FPGA. Hardware description languages VHDL and Verilog, CAD, design techniques. – M.: Hotline – Telecom. – 2022 – 538 p
5. Bass A.V., Antonov M.A. Working with FPGAs using the Verilog hardware description language // Proceedings of Tula State University. Technical Sciences, 2019. No. 3. pp. 19-24.
6. Alekhin V.A. Designing electronic systems using SystemC and SystemC-AMS // Russian Technological Journal, 2020, vol. 8, No. 4(36), pp. 79-95.
7. Demenkova T.A., Madiu B.A., Modeling digital systems at the transaction level // Scientific and Technical Bulletin of the Volga region, 2022. - No. 12. – pp. 299-302.
8. Bhadani R., Banik S., Tu H., Lukic S., Karsai G. Model-based Design Tool for Cyber-physical Power Systems using SystemC-AMS // 2024 IEEE Workshop on Design Automation for CPS and IoT (DESTION)
9. Turossi D., Baschirotto A. A SystemC-AMS Development Framework for High Power IC Test-Hardware // 2024 IEEE European Test Symposium (ETS).
10. Garashchenko A.V., Lashina D.S., Nikitin S. A., Nikolaev A.V., Prokopyev E. A., Putrya F. M., Tsyrenzhapov B.N. Practice and prospects of using open and proprietary software solutions in the verification of systems on a chip // Proceedings of the ISP RAS, 2020. – Vol. 34, No. 5. – Pp. 23-42.
11. Tarasov I.E. Application of UVM Methodology to Modeling Precision Digital Signal Processing Devices // PROGRAMMNAYA ENGINEERING, 2024. – № 15(11). – P. 570-577.
12. Dave N. A Unified Model for Hardware/Software Codesign // PhD. thesis, MIT, 2011. – 188 P.

13. Pelton B., Sapek A., Eguro K. [et al] Wavefront Threading Enables Effective High-Level Synthesis // Proceedings of the ACM on Programming Languages, 2024. – Vol. 8, No. PLDI, Article 190. – pp. 1066-1090.

14. Bukhteev A. Celoxica ESL design tools: a software-oriented approach // ELECTRONICS: SCIENCE, TECHNOLOGY, BUSINESS, 2006. – № 8(74). – Pp. 106-111.

15. Lozovsky V.V., Shtreker E.N., Boronnikov A.S., Kazantseva L.V. Theory of finite state machines: a manual. Moscow: MIREA – Russian Technological University. – 2024. – 455 p.

16. Kryakhtunov G.M., Boronnikov A.S. Approaches to creating a proprietary data representation format // International Journal of Open Information Technologies. – 2024. – Vol. 12, No. 5. – pp. 141-150.