АННОТАЦИЯ

Суть данной курсовой работы заключается в построении модели логического устройства сочетающего в себе функциональные узлы делителя частоты, фильтра дребезга контактов кнопки, конечного автомата генератора последовательности, матричного индикатора и генератора широтно-импульсных сигналов, а также верификация данной модели.

Работа состоит из пояснительной записки и графической части. Графическая часть представляет собой описание реализуемого программного решения.

Пояснительная записка содержит 5 разделов.

В первом разделе освещается проектирование моделей комбинационной логической схемы, которые будут использованы в дальнейших разделах.

Второй раздел посвящен построению конечного автомата–генератора последовательности.

В третьем разделе освещается процесс разработки функционального узла, применяющегося для анализа входной последовательности.

Четвертый раздел отведен под описание процесса разработки узла для управления матричным индикатором.

В пятом разделе описана реализация широтно-импульсного генератора и 64-разрядного сдвигового регистра.

Работу выполнил студент группы ИВБО-02-19 Денисов К.Ю.

Руководитель — профессор кафедры вычислительной техники, доцент технических наук Потехин Д. С.

Консультант — старший преподаватель кафедры вычислительной техники Борисенко Н. В.

Количество страниц — 75, количество рисунков — 31, количество таблиц — 7, количество приложений — 2. В работе использованы 10 источников.

Первое приложение содержит в себе исходные коды модулей на языке Verilog, разрабатываемых в ходе курсовой работы.

Второе приложение содержит результаты симуляции цифрового устройства в САПР ISE Design Suite.

СОДЕРЖАНИЕ

ПЕРЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

МДНФ — Минимальная дизъюнктивная нормальная форма

МКНФ — Минимальная конъюнктивная нормальная форма

САПР — Система автоматизированного проектирования

PWM — Широтно-импульсная модуляция

ВВЕДЕНИЕ

Целью данной курсовой работы является построение логического устройства сочетающего в себе функциональные узлы делителя частоты, фильтра дребезга контактов кнопки, конечного автомата генератора последовательности, блока управления матричного индикатора и генератора широтно-импульсных сигналов, а также верификация данной модели.

1 ПРОЕКТИРОВАНИЕ МОДЕЛЕЙ КОМБИНАЦИОННОЙ ЛОГИЧЕСКОЙ СХЕМЫ

1.1 Постановка задачи

Спроектировать синтезируемые модели комбинационной схемы 4x4, описанной Таблицей истинности согласно варианту задания, тремя различными способами:

1. На вентильном уровне, методом карт Карно в виде МДНФ, в схемотехническом редакторе Schematic editor САПР Xilinx ISE Design Suite.

2. На вентильном уровне, методом карт Карно в виде МКНФ, на языке описания аппаратуры Verilog.

3. На поведенческом уровне, на языке описания аппаратуры Verilog. Реализовать на языке Verilog тестовое окружение и провести верификацию спроектированных моделей при помощи симулятора iSim из состава САПР Xilinx ISE Design Suite.

Провести апробацию моделей при помощи отладочной платы Digilent Nexys 4 на ПЛИС Xilinx Artix 7 XC7A100T-1CSG324. Комбинации на входах комбинационных схем должны задаваться при помощи движковых переключателей отладочной платы, комбинации на выходах комбинационных схем должны отображаться светодиодами отладочной платы.

1.2 Реализация логической схемы

Спроектировать синтезируемые модели комбинационной схемы 4x4 согласно Таблице истинности 1.1 и вектор-функции.

Таблица 1.1 — Вектор-функция

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
0	4	4	8	3	0	7	2	2	D	7	C	5	2	A	C

1.3 Построение таблицы истинности

Построим таблицу истинности по заданному вектору. Входы обозначим X_3, X_2, X_1, X_0, а выходы Q_3, Q_2, Q_1, Q_0 соответственно (см. Таблицу 1.2).

Таблица 1.2 — Таблица истинности

№	X_3	X_2	X_1	X_0	F	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0	c	1	1	0	0
1	0	0	0	1	a	1	0	1	0
2	0	0	1	0	2	0	0	1	0
3	0	0	1	1	5	0	1	0	1
4	0	1	0	0	c	1	1	0	0
5	0	1	0	1	7	0	1	1	1
6	0	1	1	0	0	1	1	0	1
7	0	1	1	1	2	0	0	1	0
8	1	0	0	0	2	0	0	1	0
9	1	0	0	1	7	0	1	1	1
10	1	0	1	0	0	0	0	0	0
11	1	0	1	1	3	0	0	1	1
12	1	1	0	0	8	1	0	0	0
13	1	1	0	1	4	0	1	0	0
14	1	1	1	0	4	0	1	0	0
15	1	1	1	1	0	0	0	0	0

1.4 Построение карт Карно

Построим карты Карно для 4 переменных X_3, X_2, X_1, X_0 для каждой из бинарных функций Q_3, Q_2, Q_1, Q_0 (см. Рисунок 1.1).

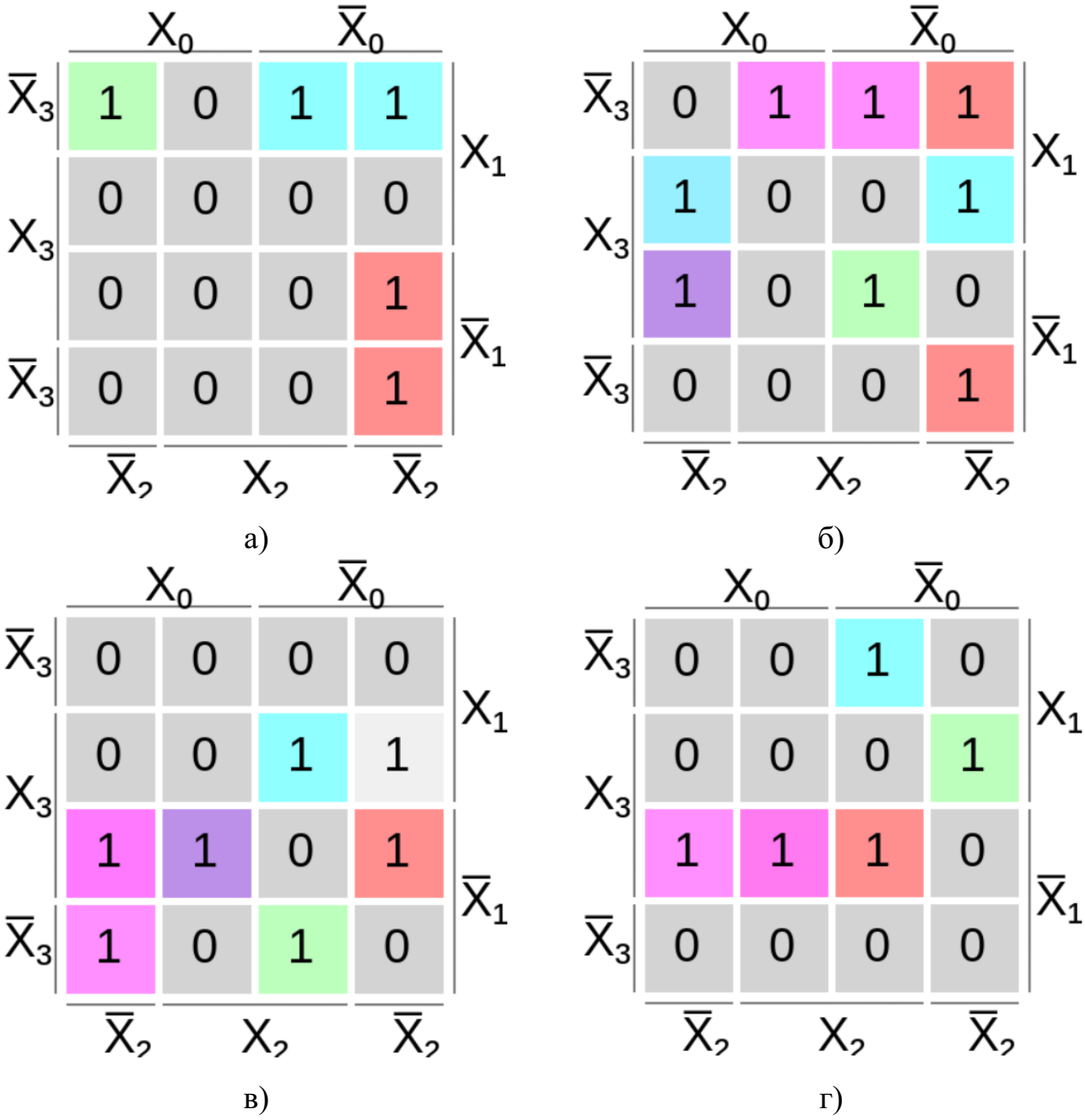


Рисунок 1.1 — Карты Карно для 4-х переменных для функций а)Q_3, б)Q_2, в)Q_2, г)Q_1, д)Q_0

1.5 Минимизация булевых функций

По построенным картам Карно опишем МДНФ для реализации данных функций на вентильном уровне в редакторе Schematic editor САПР Xilinx ISE Design Suite.

$$Q_3 = \bar{X}_3\bar{X}_2\bar{X}_1 \vee X_2\bar{X}_1\bar{X}_0 \vee \bar{X}_3X_2\bar{X}_0 \tag{1.1}$$

$$Q_2 = \bar{X}_3\bar{X}_1\bar{X}_0 \vee \bar{X}_3\bar{X}_2X_1X_0 \vee X_2\bar{X}_1X_0 \vee X_2X_1\bar{X}_0 \vee X_3\bar{X}_1X_0 \tag{1.2}$$

$$Q_1 = \bar{X}_3\bar{X}_1X_0 \vee \bar{X}_3\bar{X}_2X_1\bar{X}_0 \vee \bar{X}_3X_2X_0 \vee X_3\bar{X}_2\bar{X}_1 \vee X_3\bar{X}_2X_0 \tag{1.3}$$

$$Q_0 = \bar{X}_2X_1X_0 \vee \bar{X}_3X_2\bar{X}_1X_0 \vee \bar{X}_3X_2X_1\bar{X}_0 \vee X_3\bar{X}_2X_0 \tag{1.4}$$

Также опишем МКНФ для реализации булевых функций средствами VHDL в САПР Xilinx ISE Design Suite.

$$Q_3 = \left(\bar{X}_3 \vee \bar{X}_1\right) \cdot \left(X_2 \vee \bar{X}_1\right) \cdot \left(\bar{X}_2 \vee \bar{X}_0\right) \tag{1.5}$$

$$Q_2 = \left(X_2 \vee X_2 \vee X_1 \vee \bar{X}_0\right) \cdot \left(\bar{X}_3 \vee X_2\bar{X}_1\right) \cdot \left(X_2 \vee \bar{X}_1 \vee X_0\right) \cdot \tag{1.6}$$

$$\left(\bar{X}_2 \vee \bar{X}_1 \vee \bar{X}_0\right) \cdot \left(\bar{X}_3 \vee X_1 \vee X_0\right) \tag{1.7}$$

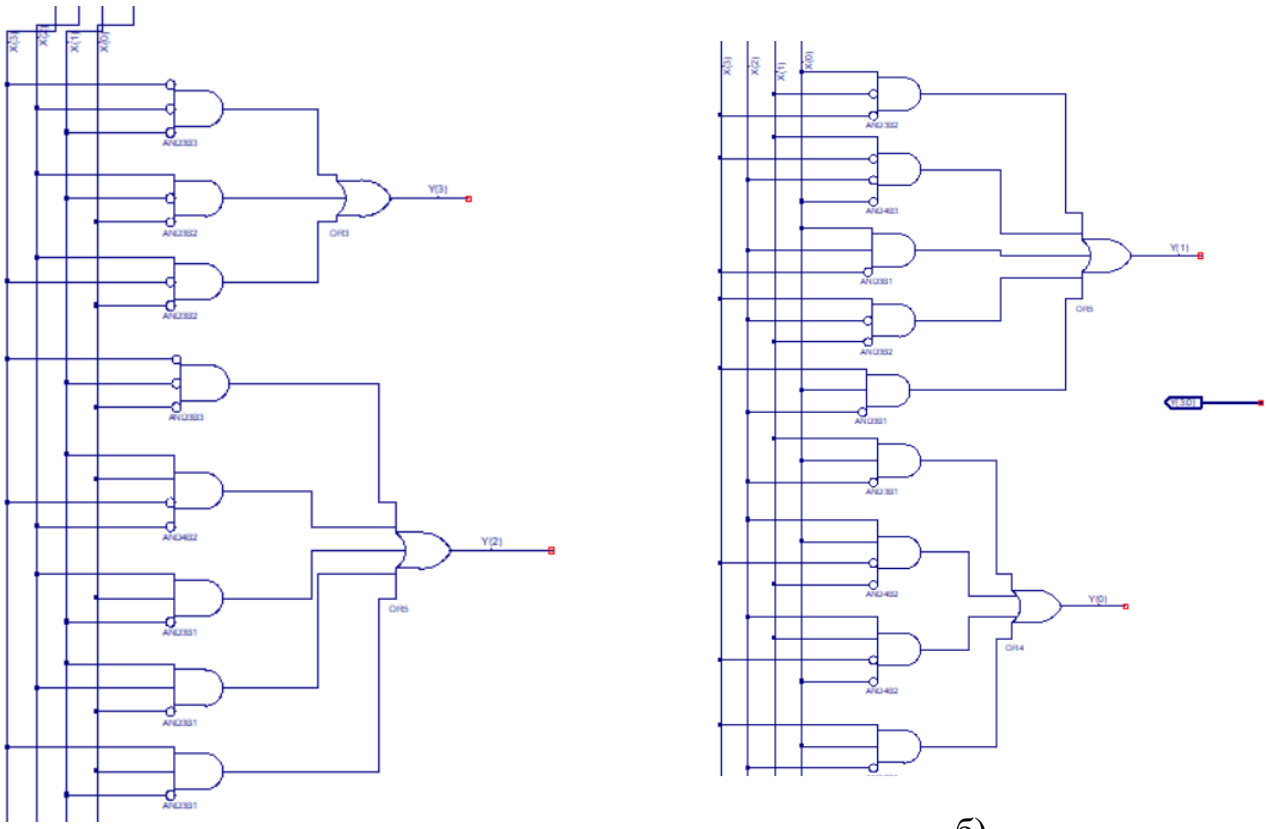
$$Q_1 = (X_3 \vee X_1 \vee X_0) \cdot \left(\bar{X}_3 \vee X_1\bar{X}_0\right) \cdot \left(\bar{X}_3 \vee \bar{X}_2\right) \cdot \tag{1.8}$$

$$\left(X_3 \vee X_2 \vee \bar{X}_1 \vee \bar{X}_0\right) \cdot \left(\bar{X}_2 \vee X_0\right) \tag{1.9}$$

$$Q_0 = (X_3 \vee X_2 \vee X_1) \cdot (X_2 \vee X_0) \cdot (X_1 \vee X_0) \cdot \left(\bar{X}_3 \vee \bar{X}_2\right) \cdot \left(\bar{X}_2 \vee \bar{X}_1 \vee \bar{X}_0\right) \tag{1.10}$$

1.6 Реализация функций в схемотехническом редакторе

Опишем функции Q_3, Q_2, Q_1, Q_0 на вентильном уровне в схемотехническом редакторе Schematic editor САПР Xilinx ISE Design Suite (см. Рисунок 1.2).



а) б)

Рисунок 1.2 — Схемотехнический редактор а) Лист 1, б) Лист 2

1.7 Реализация функций на вентильном уровне

На основании МКНФ опишем функции Q_3, Q_2, Q_1, Q_0 на вентильном уровне с помощью языка Verilog.

Исходный код данного модуля приведен в Приложении А на Листинге А.1.

1.8 Реализация функций на поведенческом уровне

На основании построенной ранее Таблицы истинности 1.2 опишем функции Q_3, Q_2, Q_1, Q_0 на поведенческом уровне с помощью языка Verilog.

Исходный код данного модуля приведен в Приложении А на Листинге А.2.

1.9 Создание проекта САПР Xilinx ISE

Опишем файл верхнего уровня проекта САПР Xilinx ISE Design Suite, в котором подключим все остальные модули, укажем входные и выходные сигналы. Исходный код данного модуля приведен в Приложении А на Листинге А.3.

1.10 Тестирование и отладка средствами симулятора iSim

После компоновки проекта, подключения модуля верхнего уровня, проведем верификацию спроектированных моделей с помощью симулятора iSim из состава САПР Xilinx ISE Design Suite. Результаты тестирования можно видеть в Приложении Б на Рисунке Б.1.

1.11 Вывод

В данном разделе нами были получены общие навыки работы с программным обеспечением Xilinx ISE Design Suite, изучены основы языка Verilog. С помощью полученных знаний была спроектированы синтезируемые модели комбинационной схемы 4x4, описанной тремя различными способами.

2 ПОСТРОЕНИЕ ГЕНЕРАТОРА ПОСЛЕДОВАТЕЛЬНОСТИ

2.1 Постановка задачи

Требуется описать конечный автомат, представляющий собой генератор фиксированной последовательности логических сигналов, в виде синтезируемой модели на языке Verilog HDL. Автомат должен иметь интерфейс, представленный на Рисунке 2.1.

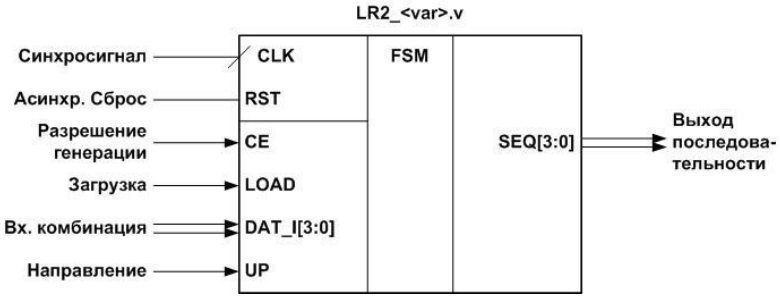


Рисунок 2.1 — Интерфейс цифрового автомата

Автомат является синхронным цифровым узлом, срабатывающим по восходящим фронтам синхросигнала *CLK*. Исключение составляет асинхронный вход сброса *RST*, принудительно устанавливающий регистр автомата в исходное состояние (определяется вариантом). Автомат должен реагировать на входные воздействия согласно Таблице 2.1.

Таблица 2.1 — Таблица функционирования автомата

RST	CLK	LOAD	CE	UP	Действие
1	X	X	X	X	Асинхронный сброс SEQ <= Func(4'h0)
0	posedge	1	X	X	Загрузка SEQ <= Func(DAT_I)
0	posedge	0	1	0	Обратная генерация SEQ <= Func(i-1)
0	posedge	0	1	1	Прямая генерация SEQ <= Func(i+1)
0	posedge	0		X	Хранение SEQ <= SEQ

Последовательность генерируемых сигналов определяется функцией *Func(i)*, где *i* — четырехразрядный двоичный индекс, представляющий собой номер элемента последовательности. Инкремент индекса соответствует пря-

мой генерации последовательности. Декремент индекса соответствует обратной генерации последовательности.

Последовательность для каждого варианта выполнения работы определяется из таблицы вариантов следующим образом: индекс *i* задан входными комбинациями от *F* до 0 в верхней строке таблицы, а выходные комбинации *Func(i)*, формируемые на выходах *SEQ[3:0]*, заданы строкой таблицы, соответствующей выбранному варианту. Допускается использовать различные варианты кодировки состояний автомата. Автомат может иметь организацию согласно абстрактным моделям Мили или Мура.

2.2 Реализация конечного автомата

Требуется описать конечный автомат, представляющий собой генератор фиксированной последовательности логических сигналов, в виде синтезируемой модели на языке Verilog HDL согласно данной таблице истинности и вектор-функции (см. Таблицу 2.2).

Таблица 2.2 — Вектор-функция

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
0	4	4	8	3	0	7	2	2	D	7	C	5	2	A	C

2.3 Структурная схема автомата

Построим структурную схему цифрового устройства. Используем делитель частоты для снижения частоты тактового генератора, фильтр дребезга для использования кнопок в качестве устройств ввода (см. Рисунок 2.2).

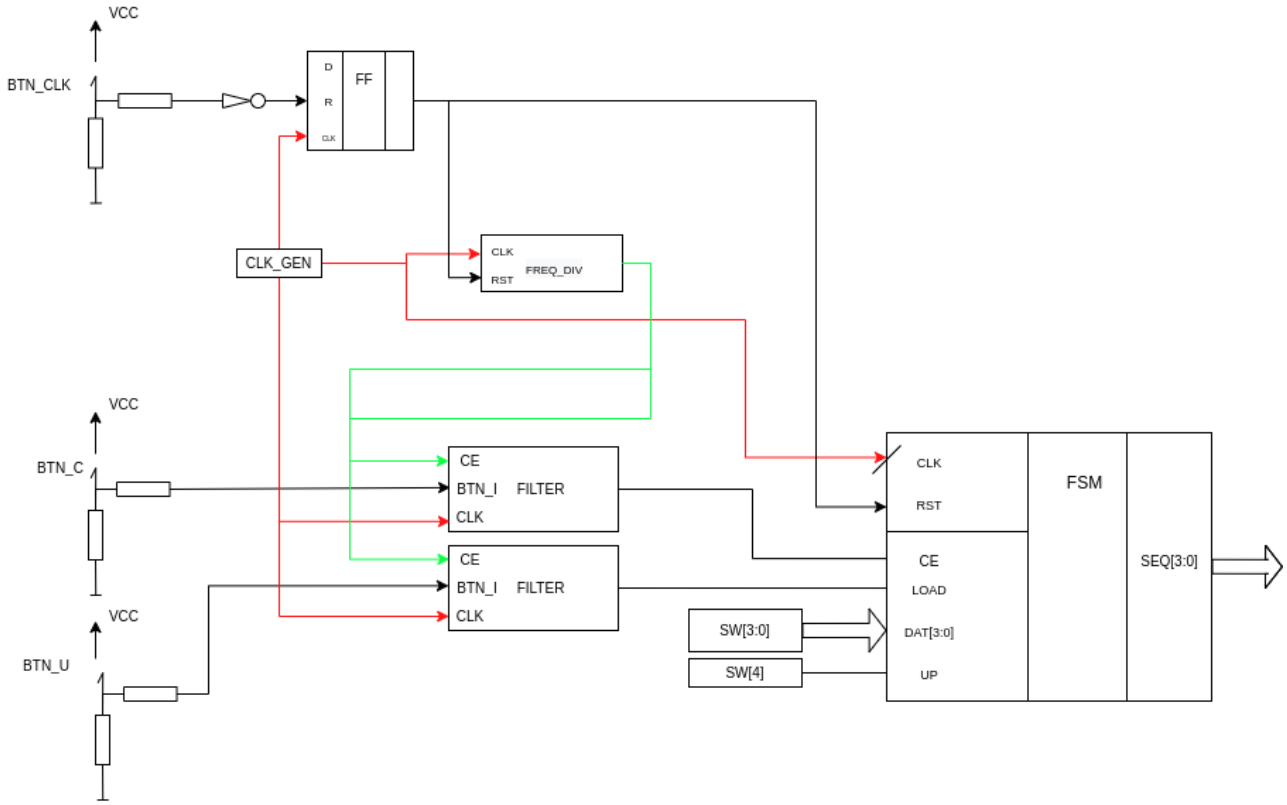


Рисунок 2.2 — Структурная схема устройства

2.4 Кодировка состояний автомата в двоичной и шестнадцатиричной системах

Опишем модуль behaviour.v, указав в нем состояния автомата, приведенные в шестнадцатеричной системе. Исходный код данного модуля приведен в Приложении А на Листинге А.2.

2.5 Граф состояний

Опишем граф перехода цифрового автомата согласно указанным режимам работы (переход в следующее или предыдущее состояние, загрузка состояния, хранение, сброс) (см. Рисунок 2.3).

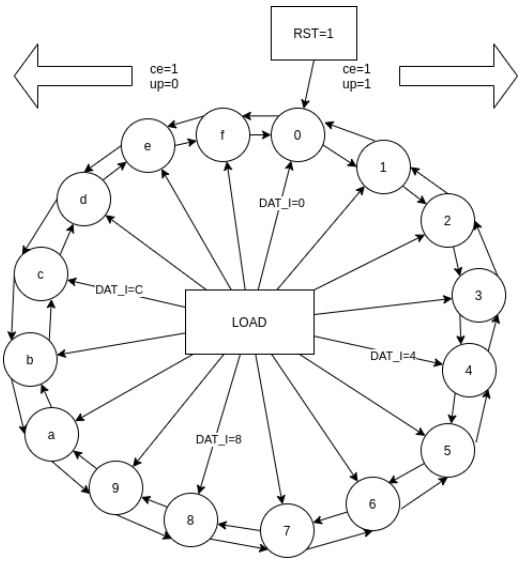


Рисунок 2.3 — Граф переходов

2.6 Создание проекта САПР Xilinx ISE

Автомат генератор последовательности. Реализуем на языке Verilog модуль, описывающий цифровой автомат. Исходный код данного модуля приведен в Приложении А на Листинге А.4.

Делитель частоты. Реализуем на языке Verilog модуль, описывающий делитель частоты. Исходный код данного модуля приведен в Приложении А на Листинге А.5.

Фильтр дребезга кнопок. Реализуем на языке Verilog модуль, описывающий фильтр дребезга. Исходный код данного модуля приведен в Приложении А на Листинге А.6.

Модуль верхнего уровня. Реализуем на языке Verilog модуль верхнего уровня. Исходный код данного модуля приведен в Приложении А на Листинге А.8.

Тестовое окружение. Реализуем на языке Verilog модуль, описывающий тестовое окружение, описывающее входные воздействия для данной модели. Исходный код данного модуля приведен в Приложении А на Листинге А.7.

2.7 Тестирование и отладка средствами симулятора iSim

После компоновки проекта, подключения модуля верхнего уровня, проведем верификацию спроектированных моделей с помощью симулятора iSim из состава САПР Xilinx ISE Design Suite. Результаты тестирования можно видеть в Приложении Б на Рисунках Б.2. Б.3.

2.8 Вывод

В данном разделе нами были получены общие навыки работы с программным обеспечением Xilinx ISE Design Suite, изучены основы языка Verilog.

С помощью полученных знаний был спроектирован конечный автомат, представляющий собой генератор фиксированной последовательности логических сигналов, в виде синтезируемой модели.

3 РАЗРАБОТКА АНАЛИЗАТОРА ПОСЛЕДОВАТЕЛЬНОСТИ

3.1 Постановка задачи

Требуется разработать цифровой узел на основе отладочной платы Digilent Nexys 4, представляющий собой анализатор фиксированной последовательности логических сигналов. Узел должен обеспечивать индикацию ожидаемых и вводимых элементов последовательности посредством входящих в состав отладочной платы семисегментных индикаторов согласно данному заданию.

Узел должен быть реализован в виде синтезируемой модели на языке Verilog HDL.

Автомат должен иметь интерфейс, представленный на Рисунке 3.1.

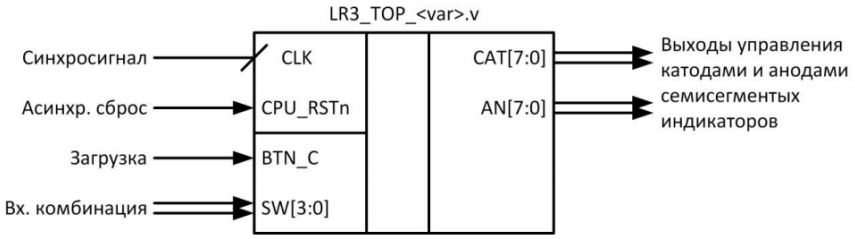


Рисунок 3.1 — Интерфейс модели цифрового узла

Разрабатываемое устройство является синхронным цифровым узлом, срабатывающим по восходящим фронтам синхросигнала CLK. Исключение составляет асинхронный вход сброса CPU_RSTn, принудительно устанавливающий все регистры узла в исходное состояние. Подача сигнала сброса на вход узла осуществляется посредством соответствующей кнопки (CPU_RSTn) отладочной платы.

Распознавание элементов последовательности осуществляется четверками, т.е. необходимо обеспечить последовательную загрузку в узел элементов Y с номерами 0-3, 4- 7, 8-B, C-F для успешного распознавания последовательности. При осуществлении ввода значения, не соответствующего текущему ожидаемому элементу последовательности, необходимо повторить ввод всей четверки элементов заново.

Индикация работы узла посредством двух блоков семисегментных индикаторов подчиняется следующим правилам:

1. Левый блок семисегментных индикаторов отображает ожидаемый (младший разряд) и введенные (три старших разряда) элементы последовательности в объеме распознаваемой четверки.
2. Правый блок семисегментных индикаторов отображает предысторию ввода комбинаций последовательности. Последняя введенная комбинация отображается в младшем разряде блока.
3. Не задействованные в текущий момент времени семисегментные индикаторы на обоих блоках должны находится в выключенном состоянии.
4. Обновление отображаемых значений на обоих блоках семисегментных индикаторов рекомендуется выполнять с частотой от 60Гц до 200Гц.

3.2 Моделирование цифрового устройства

Реализуем устройство, являющегося синхронным цифровым узлом. Его работу организуем по восходящему фронту синхросигнала, асинхронный сброс — по нисходящему фронту сигнала CPU_RSTn.

Приведем таблицу состояний устройства (см. Таблицу 3.1). Текущее состояние цифрового устройства зависит от последнего введенного пользователем с помощью движковых переключателей элемента последовательности.

Таблица 3.1 — Состояния цифрового устройства

Состояние	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Значение	0	4	4	8	3	0	7	2	2	D	7	C	5	2	A	C	–
Разряд	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	–

Перечислим функциональные узлы, которые уже были реализованы ранее.

1. Делитель частоты. Реализован в разделе 2.
2. Фильтр дребезга кнопок. Реализован в разделе 2.

3. Конечный автомат генератор последовательности. Реализован в разделе 2.

Приведем граф состояний цифрового устройства (см. Рисунок 3.2). Переход в следующее состояние происходит только в случае ввода верного элемента последовательности. При неправильном вводе цифровое устройство переходит в ближайшее пройденное состояние, номер которого кратен 4. Номер состояния, в которое перейдет устройство в случае некорректного ввода можно вычислить по формуле $S_{i+1} = (S_i \div 4) \cdot 4$.

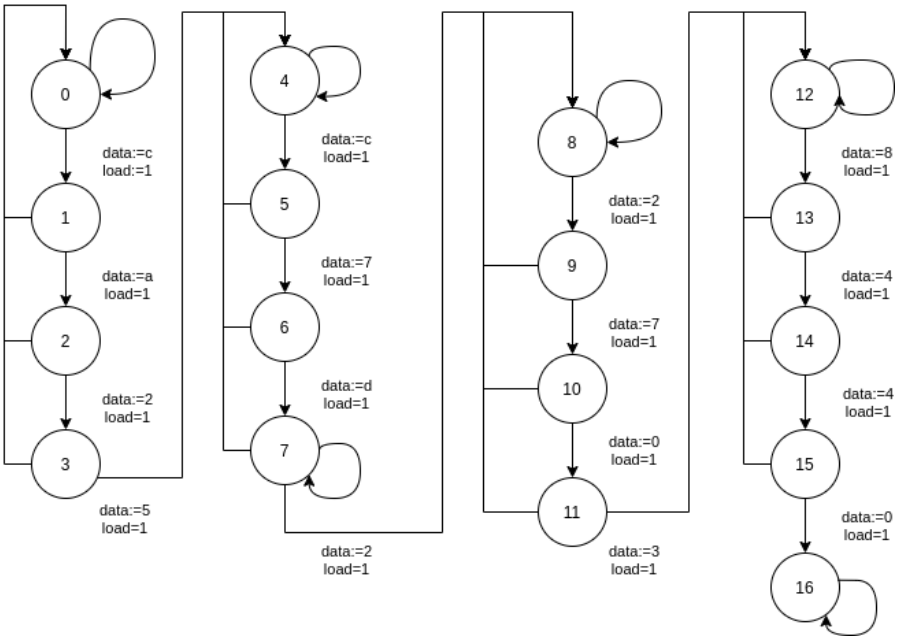


Рисунок 3.2 — Граф состояний

Приведем блок схему работы устройства (см. Рисунок 3.3). Значение X представляет собой номер элемента цифровой последовательности, ввод значения которого (Y) ожидается. Значения Y каждого элемента цифровой последовательности определяются вариантом задания.

Приведем структурную схему синхронного цифрового узла (см. Рисунок 3.4).

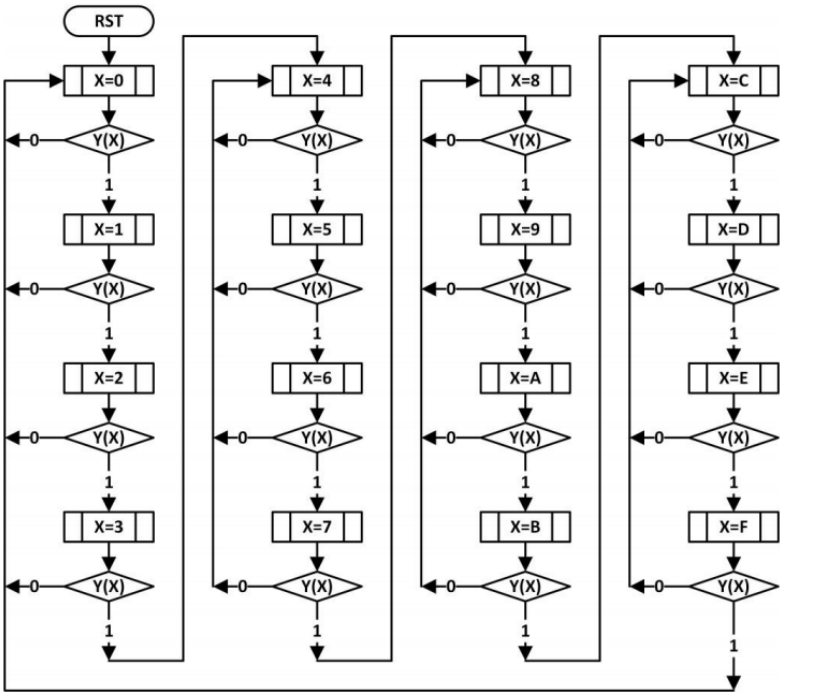


Рисунок 3.3 — Алгоритм распознавания последовательности

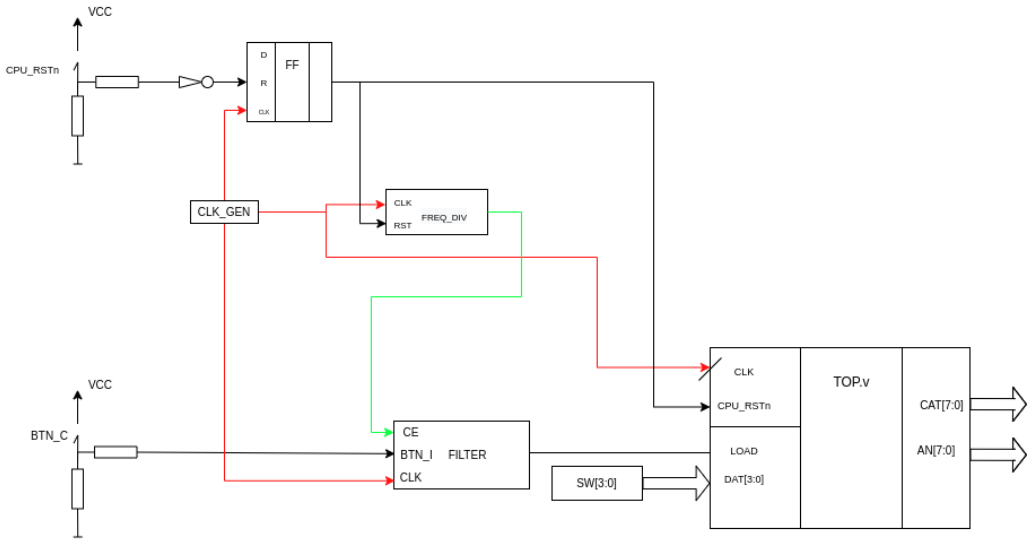


Рисунок 3.4 — Структурная схема узла

3.3 Описание принципа работы

Опишем главный алгоритм работы цифрового устройства — контроль ввода шестнадцатеричных чисел с возможностью повторного задания последовательности.

Данный алгоритм описан в файле seqAuto.v. Его содержание приведено в приложении А на Листинге А.9.

Сначала производим первоначальную инициализацию переменных, затем, если есть необходимость обновить изображение на дисплее (переменная `updateDisplay`), начинаем анализировать текущее состояние дисплея и пользовательский ввод.

Если мы находимся в состоянии, номер которого кратен четырем, то отображаем только левый разряд на левом дисплее. Выводим на это место значение соответствующее значение функции, а в остальные 3 разряда левого дисплея выводим нули.

Если же мы находимся в каком–то другом состоянии, то следует перезаписать все разряды на левом дисплее на один разряд влево, записать в освободившийся разряд записываем очередное значение вектор–функции.

В конце устанавливаем переменную `updateDisplay` в нуль.

Если пользователь подал сигнал на загрузку числа, произведем следующие действия:

Перезапишем все разряды на правом дисплее на один разряд влево, в освободившийся разряд запишем заданное пользователем число.

Осуществим проверку введенного числа — сравним его с соответствующим значением вектор–функции. Если пользователь произвел верный ввод, инструментируем переменную состояния автомата (переведем его в следующее состояние). В случае ввода неправильного числа возвращаемся в ближайшее пройденное состояние, номер которого кратен 4.

В конце устанавливаем переменную `updateDisplay` в единицу.

3.4 Создание проекта САПР Xilinx ISE

Автомат анализатор входной последовательности. Приведем содержание файла seqAuto.v, реализующего главный алгоритм управления цифровым устройством, который был описан в предыдущей секции (см. Приложение А Листинг А.9).

Фильтр дребезга кнопок. Используем ранее созданный модуль, реализующий фильтр дребезга кнопок. Исходный код данного модуля приведен в Приложении А на Листинге А.6.

Делитель частоты. Используем ранее созданный модуль, реализующий делитель частоты, применяемый в данном проекте для снижения выходных характеристик частотного генератора для подбора оптимального режима работы с семисегментными индикаторами. Исходный код данного модуля приведен в Приложении А на Листинге А.5.

Драйвер для работы с семисегментыми индикаторами. Приведем содержание файла NexysDisplay.v, описывающего работу с семисегментыми индикаторами. Исходный код данного модуля приведен в Приложении А на Листинге А.10.

Пользовательская функция. Приведем содержание файла outFunc.v, описывающего представление бинарной вектор–функции, значения которой сравниваются с пользовательским вводом. Исходный код данного модуля приведен в Приложении А на Листинге А.2.

Сигнальный дешифратор. Приведем содержание файла SevenSegDec.v, описывающего работу сигнального дешифратора для работы с семисегментыми индикаторами. Исходный код данного модуля приведен в Приложении А на Листинге А.11.

Модуль верхнего уровня. Приведем содержание файла top.v, описывающего работу модуля верхнего уровня, объединяющего все файлы и организующего работу с устройством. Исходный код данного модуля приведен в Приложении А на Листинге А.12.

3.5 Тестирование и отладка средствами симулятора ISim

Произведем тестирование работы спроектированного цифрового устройства средствами средствами САПР Xilinx ISE 14.

Результаты тестирования приведены в Приложении Б на Рисунках Б.4-Б.8.

3.6 Вывод

В данном разделе нами были получены общие навыки работы с программным обеспечением Xilinx ISE Design Suite, изучены основы языка Verilog.

С помощью полученных знаний был спроектирован и разработан цифровой узел на основе отладочной платы Digilent Nexys 4, представляющий собой анализатор фиксированной последовательности логических сигналов.

4 МОДЕЛИРОВАНИЕ ФУНКЦИОНАЛЬНОГО УЗЛА УПРАВЛЕНИЯ МАТРИЧНЫМ ДИСПЛЕЕМ

4.1 Постановка задачи

Требуется разработать модель цифрового логического устройства объёме ПЛИС Spartan-3E XC3S500E4PQ208C, сочетающую в себе функциональные узлы делителя частоты 1кГц, фильтра дребезга контактов кнопки и конечного автомата генератора последовательности.

Цифровой узел должен выводить на матричный дисплей размером восемь строк на восемь столбцов шестнадцатеричную цифру, значение которой определяется данной таблицы истинности и вектор-функции (Таблица 4.1).

Таблица 4.1 — Вектор-функция

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
0	4	4	8	3	0	7	2	2	D	7	C	5	2	A	C

Организовать на языке Verilog тестовый модуль, осуществляющий верификацию написанных модулей. Тестовое окружение должно моделировать работу устройства на частоте 1 МГц. Функциональные узлы делителя частоты и фильтра в тестировании могут не участвовать.

4.2 Моделирование цифрового устройства

Приступим к реализации модели цифрового узла. Приведем структурную схему синтезируемой части проекта (см. Рисунок 4.1).

Перечислим функциональные узлы, которые уже были реализованы ранее.

1. Делитель частоты. Реализован в разделе 2.
2. Фильтр дребезга кнопок. Реализован в разделе 2.
3. Конечный автомат генератор последовательности. Реализован в разделе 2.

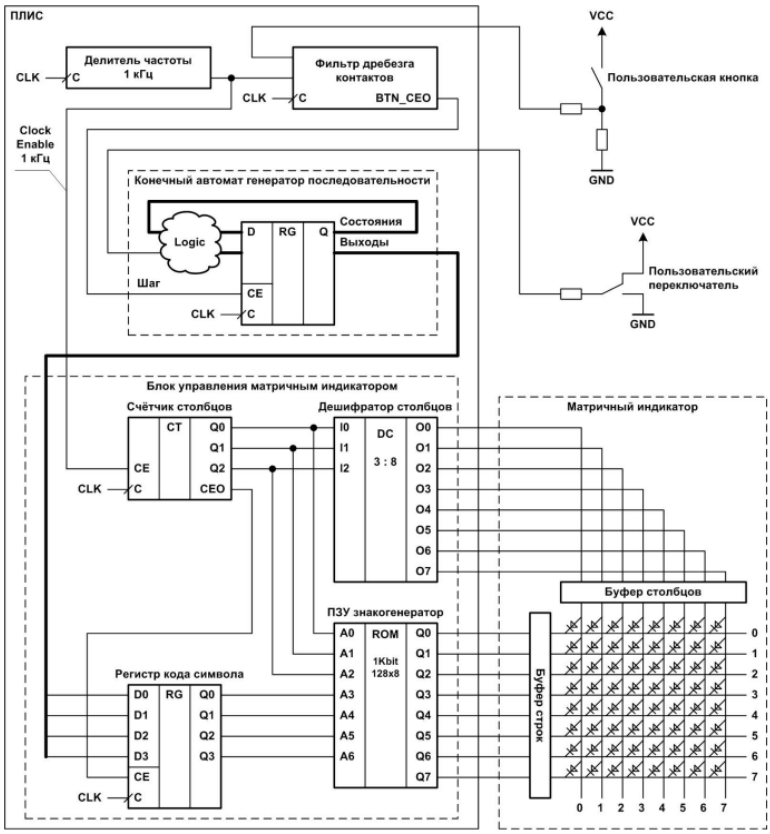


Рисунок 4.1 – Структурная схема синтезируемой части проекта

Процесс создания данных модулей не будет освещаться в настоящем разделе.

4.3 Принцип работы матричного индикатора

Рассмотрим принцип работы матричного индикатора. Вывод организуем по столбцам. В определённый момент времени активным является столбец, и любые из восьми светодиодов активного столбца могут подсвечиваться одновременно.

Это даёт возможность увидеть одноцветное изображение, выводимое на матричный дисплей, непосредственно на временной диаграмме. Принцип вывода информации на матричный индикатор на примере символа «9» показан на Рисунке 4.2.

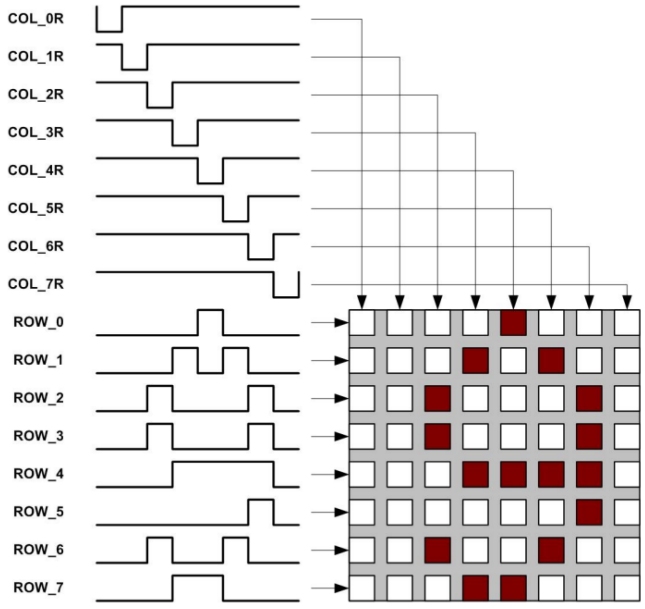


Рисунок 4.2 — Вывод символа на матричный индикатор

4.4 Создание проекта САПР Xilinx ISE

Блок управления матричным индикатором . Реализуем модель функционального узла, выполняющего роль блока управления матричным дисплеем размеров восемь строк на восемь столбцов. Его работу организуем по восходящему фронту синхросигнала CLK, сброс — по восходящему фронту сигнала RST.

Устройство должно содержать в себе счетчик столбцов, дешифратор столбцов, выбирающий в определенный момент времени только один активный столбец, а также элемент памяти (ПЗУ) хранящий шаблоны всех шестнадцатеричных чисел. ПЗУ будет использоваться знакогенератором для формирования символа из четырехразрядного двоичного входа, поданного в качестве входного сигнала.

Число шаблонов определяется числом символов и равно 16. Итого объём ПЗУ знакогенератора составляет 1 кбит или 128 байт.

Младшие три разряда адреса ПЗУ выбирают текущий столбец в рамках одного шаблона. Старшие четыре разряда адреса ПЗУ выбирают текущий шаблон.

Шаблоны шестнадцатеричных чисел будут сформированы вручную и занесены в файл. В ходе моделирования работы устройства ПЗУ блока управ-

ления индикатором будет проинициализировано значениями, указанными в файле.

Исходный код данного функционального блока приведен в Приложении А в Листинге А.13.

Модуль верхнего уровня. Опишем модуль верхнего уровня, в котором подключим функциональные узлы моделируемого цифрового устройства согласно схеме, приведенной на Рисунке 4.1.

В данном модуле будем использовать делитель частоты DIV_1MHz, преобразующий сигнал частотой 48МГц в сигнал частотой 1МГц.

Подключим автомат генератор последовательности fsm к драйверу матричного дисплея lcd, для того, чтобы иметь выводить выходное состояние автомата на индикатор.

Исходный код данного модуля приведен в Приложении А в Листинге А.14.

Тестовый модуль. Приступим к реализации модуля, который будет организовывать тестовое окружение рассматриваемой части цифрового логического устройства. В модуле верхнего уровня опишем генератор тактового сигнала частотой 48 МГц. Который будет использован в качестве тактового сигнала для всех синхронных функциональных узлов в составе устройства.

Сигнал с частотой 1МГц будет использоваться как сигнал разрешения счета для конечного автомата генератора последовательности, блока управления матричным индикатором, фильтра дребезга кнопок.

Исходный код данного модуля приведен в Приложении А в Листинге А.15.

4.5 Тестирование и отладка средствами симулятора iSim

После компоновки проекта, подключения модуля верхнего уровня, проведем верификацию спроектированных моделей с помощью симулятора iSim из состава САПР Xilinx ISE Design Suite. Результаты тестирования можно видеть в Приложении Б на Рисунках Б.9, на котором мы видим повернутые на

180 градусов шестнадцатеричные числа, соответствующие исходной вектор-функции (см. Рисунок 4.1).

4.6 Вывод

В данном разделе нами были получены общие навыки работы с программным обеспечением Xilinx ISE Design Suite, изучены основы языка Verilog.

С помощью полученных знаний был спроектирован функциональный узел по управлению матричным индикатором, который был использован для вывода текущего состояния автомата–генератора последовательности.

5 РЕАЛИЗАЦИЯ ГЕНЕРАТОРА ШИМ

5.1 Постановка задачи

Требуется разработать модель цифрового логического устройства объёме ПЛИС Spartan-3E XC3S500E4PQ208C, сочетающую в себе функциональные узлы делителя частоты 1кГц, фильтра дребезга контактов кнопки и конечного автомата генератора последовательности.

Центральным узлом схемы является 64-разрядный сдвиговый регистр. Исходное состояние сдвигового регистра определяется заданной в виде вектор-функции таблицей истинности (см. Таблицу 4.1).

Таблица 5.1 — Вектор-функция

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
0	4	4	8	3	0	7	2	2	D	7	C	5	2	A	C

Сдвиговый регистр после снятия сигнала асинхронного сброса RST должен содержать значения, соответствующие варианту, как показано на Рисунке 5.1.

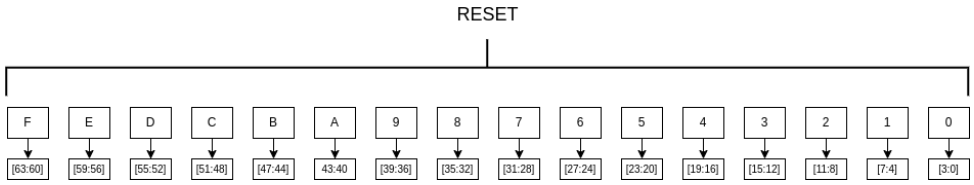


Рисунок 5.1 — Исходное состояние сдвигового регистра

Во время штатного функционирования, при деактивированном сигнале сброса RST сдвиговый регистр способен выполнять две операции. Первая операция — циклический сдвиг влево на 4 разряда, выполняется при «1» на входе SHIFT_4B_L, как показано на Рисунке 5.2.

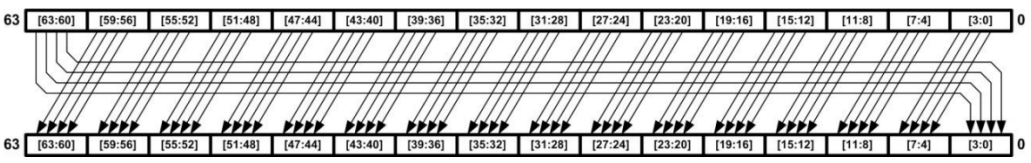


Рисунок 5.2 — Операция сдвига влево

Вторая операция – циклический сдвиг вправо на 4 разряда, выполняется при «0» на входе SHIFT_4B_L и «1» на входе SHIFT_4B_R, как показано на Рисунке 5.3.

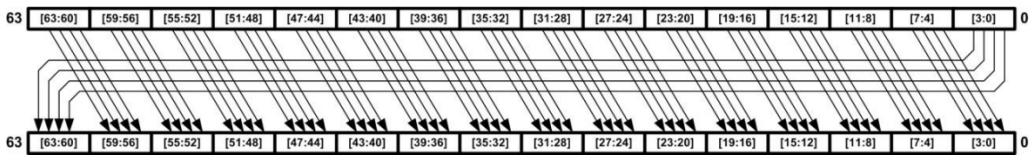


Рисунок 5.3 — Операция сдвига вправо

Сигналы, управляющие сдвиговым регистром, формируются фильтрами дребезга контактов по нажатию на две пользовательские кнопки. Выходы сдвигового регистра подаются на два блока индикации.

Первый блок использует светодиоды общего назначения, которые управляются в режиме регулировки яркости свечения с помощью широтноимпульсного сигнала.

Второй блок индикации использует матричный индикатор, на который выводятся все 64 разряда сдвигового регистра. Блок управления матричным индикатором коммутирует разряды сдвигового регистра согласно Рисунку 5.4. Такая организация вывода позволяет увидеть изображение, выводимое на индикатор, непосредственно на временной диаграмме.

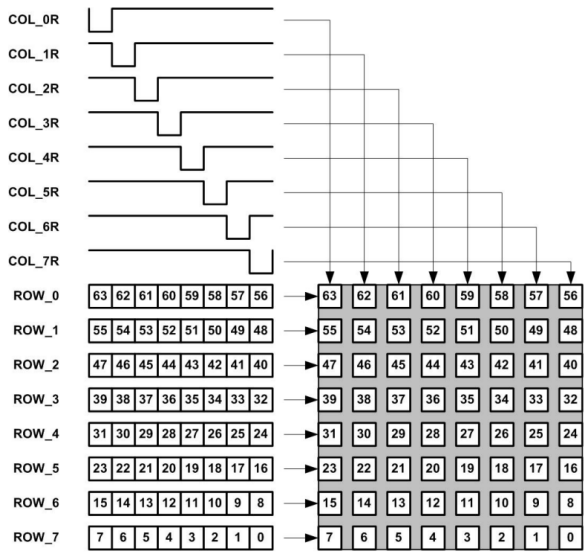


Рисунок 5.4 — Вывод содержимого регистра на матричный индикатор

В процессе выполнения задания необходимо описать на языке Verilog модель генератора широтно-импульсного сигнала, имеющий интерфейс, приведенный на Рисунке 5.5.

Рисунок 5.5 — Интерфейс генератора широтно-импульсного сигнала

5.2 Принцип широтно-импульсной модуляции

Принцип широтно-импульсной модуляции (Pulse-Width Modulation — PWM) заключается в использовании двух уровней сигнала: низкого и высокого, или «0» и «1», соответственно, чередующихся с фиксированным периодом, но с различной длительностью каждого уровня в объёме периода.

5.3 Реализация генератора ШИМ

Синтезируемая модель генератора ШИМ-сигналов имеет параметр UDW, определяющий разрядность входной шины и число состояний управляющего автомата в объёме периода сигнала (частоту дискретизации).

Именно этот параметр обеспечивает универсальность модели.

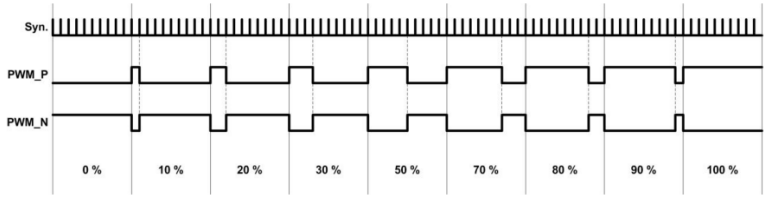


Рисунок 5.6 — Принцип широтно-импульсной модуляции

Число возможных входных комбинаций PWM_IN, задающих коэффициент заполнения периода активным уровнем сигнала («1» – для PWM_P, «0» – для PWM_N), составляет $2UDW$ штук. При этом комбинация из всех «0» задаёт коэффициент 0%, а комбинация из всех «1» – коэффициент 100%.

Период выходного сигнала делится на $(2UDW-1)$ равных тактов, заданных частотой сигнала разрешения синхронизации CE, либо равных периоду

синхросигнала CLK, если на вход CE подана константа «1». Таким образом, частота выходного ШИМ-сигнала получается из частоты дискретизации (частоты CE или CLK при CE = «1») путём деления на коэффициент ($2UDW-1$).

Граф переходов конечного автомата показан на Рисунке 5.7. Состояние входного регистра изменяется в последнем такте предпоследнего состояния автомата $(2UDW-2)$.

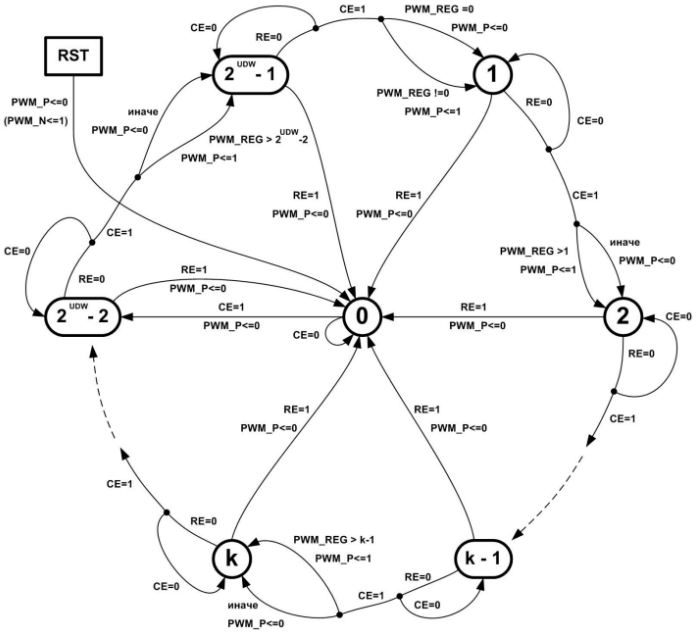


Рисунок 5.7 — Граф переходов конечного автомата

Входной регистр позволяет закончить период генерации ШИМ-сигнала на основе неизменной управляющей комбинации, зафиксированной перед началом текущего периода.

Период генерации начинается в состоянии $2UDW-1$ и заканчивается в состоянии $2UDW-2$. Таким образом, изменения комбинации на входе PWM_IN не приведут к искажению формы выходного сигнала.

Вход синхронного сброса RE предназначен для возврата автомата в исходное состояние — 0 и для перевода выходов в пассивное состояние. Синхронный сброс является приоритетной операцией и воздействует независимо от разрешения синхронизации CE.

На входной регистр PWM_REG синхронный сброс не оказывает воздействия. Временная диаграмма начала цикла работы генератора после синхронного сброса показана на Рисунке 5.8.

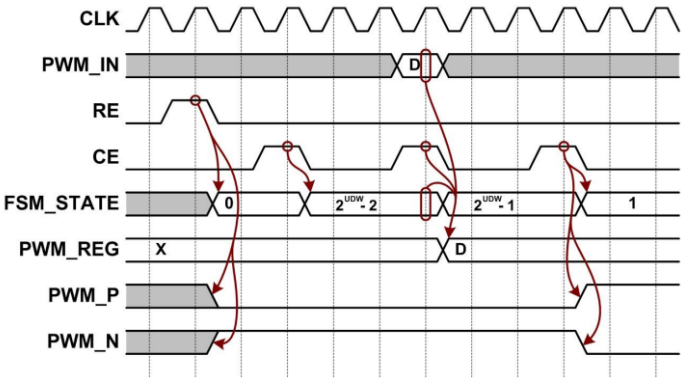


Рисунок 5.8 – Синхронный сброс генератора

5.4 Создание проекта САПР Xilinx ISE

Приступим к реализации модели цифрового узла. Функциональная схема генератора ШИМ-сигналов приведена на Рисунке 5.9.

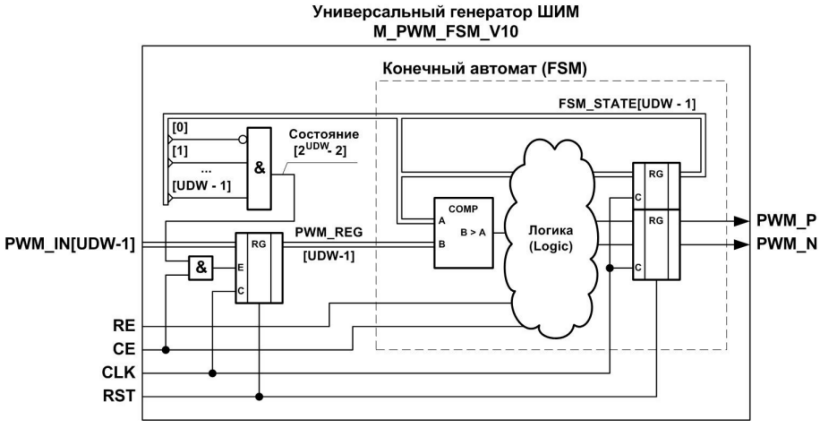


Рисунок 5.9 – Функциональная схема генератора

Перечислим функциональные узлы, которые уже были реализованы ранее.

1. Делитель частоты. Реализован в разделе 2.
2. Фильтр дребезга кнопок. Реализован в разделе 2.
3. Конечный автомат генератор последовательности. Реализован в разделе 2.
4. Блок управления матричным индикатором. Реализован в разделе 4.

ШИМ генератор. Опишем на языке программирования Verilog модуль, реализующий ШИМ генератор. Исходный код данного функционального блока приведен в Приложении А в Листинге А.16.

Сдвиговый регистр. Опишем на языке программирования Verilog 64-разрядный сдвиговый регистр. Данный модуль реализует две операции:

1. Циклический сдвиг влево на 4 разряда.
2. Циклический сдвиг вправо на 4 разряда

Исходный код данного функционального блока приведен в Приложении А в Листинге А.17.

Блок управления матричным индикатором. Изменим модуль, реализующий блок управления матричным индикатором — установим новый способ отображения входных сигналов на дисплей согласно Рисунку 5.4.

Листинг 5.1 — Измененный способ коммутации

```
1     always@*
2     begin
3         rows = {data[63-column_ctr],
4                 data[55-column_ctr],
5                 data[47-column_ctr],
6                 data[39-column_ctr],
7                 data[31-column_ctr],
8                 data[23-column_ctr],
9                 data[15-column_ctr],
10                data[7-column_ctr]
11                };
12    end
```

Исходный код данного функционального блока приведен в Приложении А в Листинге А.18.

Модуль верхнего уровня. Реализуем на языке Verilog модуль верхнего уровня. Исходный код данного модуля приведен в Приложении А на Листинге А.19.

Тестовый модуль. Приступим к реализации модуля, который будет организовывать тестовое окружение рассматриваемой части цифрового логического устройства. В модуле верхнего уровня опишем генератор тактового сиг-

нала частотой 48 МГц. Который будет использован в качестве тактового сигнала для всех синхронных функциональных узлов в составе устройства.

Сигнал с частотой 1МГц будет использоваться как сигнал разрешения счета для генераторов ШИМ-сигналов, счетчика столбцов.

Исходный код данного модуля приведен в Приложении А в Листинге А.20.

5.5 Тестирование и отладка средствами симулятора ISim

После компоновки проекта, подключения модуля верхнего уровня, проведем верификацию спроектированных моделей с помощью симулятора iSim из состава САПР Xilinx ISE Design Suite. Результаты тестирования можно видеть в Приложении Б на Рисунках Б.10. Б.11.

5.6 Вывод

В данном разделе нами были получены общие навыки работы с программным обеспечением Xilinx ISE Design Suite, изучены основы языка Verilog.

С помощью полученных знаний был спроектирован функциональный узел–генератор ШИМ сигналов, который был использован для управления светодиодными индикаторами, а также 64-рязрядный сдвиговый регистр.

ЗАКЛЮЧЕНИЕ

Результатом работы над данной курсовой работой является модель логического устройства сочетающего в себе функциональные узлы делителя частоты 1кГц, фильтра дребезга контактов кнопки, конечного автомата генератора последовательности, блока управления матричного индикатора и генератора широтно-импульсных сигналов.

Также в ходе выполнения данной работы были проделаны мероприятия по симуляции и тестированию данной модели средствами симулятора ISim в составе САПР Xilinx.

Цели и задачи по реализации устройства для ресинхронизации данных выполнены в полном объеме.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Зотов, Валерий*. Проектирование встраиваемых микропроцессорных систем на основе ПЛИС фирмы Xilinx / Валерий Зотов // *М.: Горячая линия–Телеком*. — 2006.

2. *Ушенина, Инна*. Использование временных ограничений PERIOD и OFFSET при проектировании цифровых устройств на ПЛИС фирмы Xilinx / Инна Ушенина // *Компоненты и технологии*. — 2013. — № 5. — С. 97–106.

3. *Patharkar, Ankush Suresh*. Performance analysis of synchronizer and measurement of metastability / Ankush Suresh Patharkar // 2015 International Conference on Computing Communication Control and Automation / IEEE. — 2015. — Pp. 494–498.

4. *Cummings, Clifford E*. Simulation and synthesis techniques for asynchronous FIFO design / Clifford E Cummings // SNUG 2002 (Synopsys Users Group Conference, San Jose, CA, 2002) User Papers. — 2002.

5. *Тарасов, Илья Евгеньевич*. Язык описания аппаратуры Verilog / Илья Евгеньевич Тарасов. — 2011.

6. *Попов, АЮ*. Проектирование цифровых устройств с использованием ПЛИС / АЮ Попов. — 2009.

7. *Кузелин, Михаил*. ПЛИС фирмы Xilinx: семейство SpartanTM-II / Михаил Кузелин // *Компоненты и технологии*. — 2001. — no. 12. — Pp. 84–86.

8. *Том, Леклидер*. Погружаясь в ПЛИС / Леклидер Том // *Компоненты и технологии*. — 2006. — no. 65. — Pp. 56–60.

9. *Стемпковский, АЛ*. Основы логического синтеза средствами САПР Synopsys с использованием Verilog HDL: учеб. пособие / АЛ Стемпковский, МЮ Семенов // *М.: МИЭТ*. — 2005.

10. *Харрис, Дэвид*. Цифровая схемотехника и архитектура компьютера / Дэвид Харрис, Сара Харрис. — Litres, 2021.

ПРИЛОЖЕНИЯ

Приложение А — листинги реализованных модулей

Приложение Б — результаты симуляции и тестирования модулей

Приложение А Листинг

Листинг А.1 — Реализация функций на вентильном уровне

```
‘timescale 1ns / 1ps
module mknf(
    input[3:0]  X,
    output[3:0] Y
    );
   assign Y[3] = (~X[3]|~X[1])&(X[2]|~X[1])&(~X[2]|~X[0])&(~X[3]|X[2]);
   assign Y[2] = (X[3]|X[2]|X[1]|~X[0])&(~X[3]|X[2]|~X[1])&(X[2]|~X[1]|X[0])
                                    &(~X[2]|~X[1]|~X[0])&(~X[3]|X[1]|X[0]);
   assign Y[1] = (X[3]|X[1]|X[0])&(~X[3]|~X[1]|X[0])&(~X[3]|~X[2])
                                    &(X[3]|X[2]|~X[1]|~X[0])&(~X[2]|X[0]);
   assign Y[0] = (X[3]|X[2]|X[1])&(X[2]|X[0])&(X[1]|X[0])
                                    &(~X[3]|~X[2])&(~X[2]|~X[1]|~X[0]);
endmodule
```

Листинг А.2 — Реализация функций на поведенческом уровне

```
‘timescale 1ns / 1ps

module beheviour(
    input[3:0]  X,
    output reg [3:0] Y
    );
     always@(X)
        case(X)
            4’h0: Y<=4’hc;
            4’h1: Y<=4’ha;
            4’h2: Y<=4’h2;
            4’h3: Y<=4’h5;
            4’h4: Y<=4’hc;
            4’h5: Y<=4’h7;
            4’h6: Y<=4’hd;
            4’h7: Y<=4’h2;
            4’h8: Y<=4’h2;
            4’h9: Y<=4’h7;
            4’ha: Y<=4’h0;
            4’hb: Y<=4’h3;
            4’hc: Y<=4’h8;
            4’hd: Y<=4’h4;
            4’he: Y<=4’h4;
            4’hf: Y<=4’h0;
        default: Y<=4’h0;

```

```
27      endcase
28  endmodule
```

Листинг А.3 — Реализация логических функций. Файл верхнего уровня

```
1   `timescale 1ns / 1ps
2   module top(
3       input CLK,
4       input CPU_RSTn,
5       input BTN_C,
6       input  [3:0] SW,
7       output [7:0] CAT,
8       output [7:0] AN
9       );
10
11      reg rst;
12
13      always @(posedge CLK) begin
14          rst <= ~CPU_RSTn;
15      end
16      wire [31:0] hex;
17      wire [7:0] disp_en;
18
19      NexysDisplay Disp(
20          .CLK(CLK),
21          .RST(rst),
22          .HEX_IN(hex),
23          .DISP_EN(disp_en),
24          .CA(CAT[0]),
25          .CB(CAT[1]),
26          .CC(CAT[2]),
27          .CD(CAT[3]),
28          .CE(CAT[4]),
29          .CF(CAT[5]),
30          .CG(CAT[6]),
31          .DP(CAT[7]),
32          .AN(AN)
33      );
34      wire divClk;
35      freq_div DIV(
36      .rst(rst),
37      .clk(CLK),
38      .co(divClk)
39      );
40      wire loadDebounced;
41      wire loadDebounced1;
42        M_BTN_FILTER_V10 Filter(
```

```
43      .CLK(CLK),
44     .CE(divClk),
45     .BTN_IN(BTN_C),
46     .RST(rst),
47     .BTN_OUT(),
48     .BTN_CEO(loadDebounced)
49     );
50     seqAuto Auto(
51     .clk(CLK),
52     .rst(rst),
53     .load(loadDebounced),
54     .data(SW),
55     .display(hex),
56     .displayEnable(disp_en)
57     );
58 endmodule
```

Листинг А.4 — Цифровой автомат

```
1  `timescale 1ns / 1ps
2  //
3  module fsm(input           rst,
4                input         clk,
5                input         ce,
6                input         load,
7                input         up,
8                input  [3:0] data,
9
10               output [3:0] seq);
11
12 reg [3:0] state;
13
14 behaviour beh (.X(state),
15                    .Y(seq));
16
17 always @(posedge clk, posedge rst)
18     begin
19         if (rst)
20             state <= 4'h0;
21         else
22             begin
23                 if (load)
24                     state <= data;
25                 if (ce && up)
26                     state <= state + 4'h1;
27                 if (ce && !up)
28                     state <= state - 4'h1;
```

```
29                  if (!ce && !load)
30                      state <= state;
31              end
32      end
33  endmodule
```

Листинг А.5 — Делитель частоты

```
1   `timescale 1ns / 1ps
2   module freq_div(input   rst,
3                   input   clk,
4
5                   output reg  co);
6       reg [16:0] counter;
7       parameter divisior = 17'd10000;
8
9       always @(posedge clk, posedge rst) begin
10          if (rst)
11              begin
12                  counter <= 0;
13                  co <= 0;
14              end
15          else
16                  if (counter >= (divisior - 17'b1))
17                      begin
18                          counter <= 17'b0;
19                          co <= 1;
20                      end
21                  else
22                  begin
23                      co <= 0;
24                      counter <= counter + 17'b1;
25                  end
26      end
27  endmodule
```

Листинг А.6 — Фильтр дребезга

```
1   `timescale 1ns / 1ps
2   module M_BTN_FILTER_V10(
3   input CLK,
4   input CE,
5   input BTN_IN,
6   input RST,
7   output BTN_OUT,
8   output reg BTN_CEO
9   );
10   parameter [3:0] CNTR_WIDTH = 4; // Internal Counter Width
```

```
// Internal signals declaration:
 reg [CNTR_WIDTH - 1:0] FLTR_CNT;
 reg BTN_D, BTN_S1, BTN_S2;
//----------------------------------------// Main Counter:
 always @ (posedge CLK, posedge RST)
if(RST) FLTR_CNT <= {CNTR_WIDTH{1'b0}};
else
 if(!(BTN_S1 ^ BTN_S2)) // if BTN_S1 = BTN_S2
 FLTR_CNT <= {CNTR_WIDTH{1'b0}}; // Return to Zero
 else if(CE) // else if Clock Enable
 FLTR_CNT <= FLTR_CNT + 1; // Increment
//----------------------------------------// Input Synchronizer:
 always @ (posedge CLK, posedge RST)
if(RST)
 begin
 BTN_D <= 1'b0;
 BTN_S1 <= 1'b0;
 end
else
 begin
 BTN_D <= BTN_IN;
 BTN_S1 <= BTN_D;
 end
//----------------------------------------// Output Register:
 always @ (posedge CLK, posedge RST)
if(RST) BTN_S2 <= 1'b0;
else if(&(FLTR_CNT) & CE) BTN_S2 <= BTN_S1;
//----------------------------------------// Output Front Detector Clock
   Enable:
 always @ (posedge CLK, posedge RST)
if(RST) BTN_CEO <= 1'b0;
else BTN_CEO <= &(FLTR_CNT) & CE & BTN_S1;
//----------------------------------------assign BTN_OUT = BTN_S2;
//----------------------------------------endmodule
```

Листинг А.7 — Тестирование цифрового автомата

```
`timescale 1ns / 1ps

module test_fsm;

    // Inputs
    reg          rst;
    reg          clk;
    reg          ce;
    reg          load;
    reg          up;
```

```
11      reg [3:0] data;
12
13      // Outputs
14      wire [3:0] seq;
15
16      // Instantiate the Unit Under Test (UUT)
17      fsm uut (
18          .rst(rst),
19          .clk(clk),
20          .ce(ce),
21          .load(load),
22          .up(up),
23          .data(data),
24          .seq(seq)
25      );
26      always
27          #5 clk=~clk;
28
29      initial begin
30          // Initialize Inputs
31          rst = 1;
32          clk = 0;
33          ce = 0;
34          load = 0;
35          up = 0;
36          data = 0;
37
38          // Wait 100 ns for global reset to finish
39          #100;
40
41          // count forward
42
43          rst=0;
44          ce=1;
45          up=1;
46          #165;
47
48          //count bashward
49
50          rst=0;
51          ce=1;
52          up=0;
53          #180;
54
55          //one state (store mode)
56
```

```
57          rst=0;
58          ce=0;
59          up=0;
60          #100;
61
62
63          rst=0;
64          ce=0;
65          up=0;
66          load=1;
67
68          //load mode
69          data=4'h0;
70          #20;
71          data=4'h1;
72          #20;
73          data=4'h2;
74          #20;
75          data=4'h3;
76          #20;
77          data=4'h4;
78          #20;
79          data=4'h5;
80          #20;
81          data=4'h6;
82          #20;
83          data=4'h7;
84          #20;
85          data=4'h8;
86          #20;
87          data=4'h9;
88          #20;
89          data=4'ha;
90          #20;
91          data=4'hb;
92          #20;
93          data=4'hc;
94          #20;
95          data=4'hd;
96          #20;
97          data=4'he;
98          #20;
99          data=4'hf;
100         //#20;
101
102         $stop;
```

```
103
104         // Add stimulus here
105
106     end
107 endmodule
```

Листинг А.8 — Цифровой автомат. Файл верхнего уровня

```
1  `timescale 1ns / 1ps
2
3  module top(
4      input           clk,
5      input  [4:0]    SW,
6      input           CPU_RESET,
7      input           BTNC,
8      input           BTNU,
9      output [3:0]    LED
10     );
11
12
13      reg             RST_I;
14      wire            RST;
15      wire            CO;
16      wire            BTNC_CEO;
17      wire            BTNU_CEO;
18
19      always @ (posedge clk, negedge CPU_RESET)
20        begin
21           if(~CPU_RESET)
22             RST_I <= 1'b1;
23          else
24             RST_I <= 1'b0;
25         end
26      assign RST=RST_I;
27      freq_div #(10000) FREQ_CO (
28      .rst (RST),
29      .clk (clk),
30      .co  (CO)
31      );
32
33      M_BTN_FILTER_V10 b_f_u(
34      .CLK     (clk),
35     .CE      (CO),
36     .BTN_IN  (BTNU),
37     .RST     (RST_I),
38      .BTN_CEO (BTNU_CEO)
39      );
```

```
40
41      M_BTN_FILTER_V10 b_f_c(
42      .CLK     (clk),
43     .CE      (co),
44     .BTN_IN  (BTNC),
45     .RST     (RST_I),
46      .BTN_CEO (BTNC_CEO)
47      );
48
49
50      fsm FSM1(
51      .rst   (RST_I),
52      .clk   (clk),
53     .ce    (BTNC_CEO),
54     .load  (BTNU_CEO),
55     .up    (SW[4]),
56     .data  (SW[3:0]),
57      .seq   (LED)
58      );
59 endmodule
```

Листинг А.9 — Описание главного алгоритма

```
1  timescale 1ns / 1ps
2
3  module seqAuto(
4      input clk,
5      input rst,
6      input load,
7      input [3:0] data,
8      output reg [31:0] display,
9      output reg [7:0] displayEnable
10     );
11     reg [3:0] state;
12     reg updateDisplay;
13
14     wire[3:0] expected;
15
16     outFunc CL(
17       .in(state),
18       .out(expected)
19     );
20     always @(posedge clk, posedge rst)
21         begin
22             if (rst)
23                 begin
24                     state <= 0;
```

```
25                      display <= 0;
26                      displayEnable <= 0;
27                      updateDisplay <= 1;
28                  end
29              else
30                  begin
31                      if (updateDisplay)
32                          begin
33                              // If we are in the first state in the four
34                              if (state[1:0] == 2'b00)
35                                  begin
36                                      // display only the left digit on the
                                           left display
37                                      displayEnable[7:4] <= 4'h1;
38                                      // we output the corresponding value of
                                           the function to this place
39                                      display[19:16] <= expected;
40                                      // we output zeros to the remaining 3
                                           digits of the left display
41                                      display[31:20] <= 0;
42                                  end
43                              else
44                                  begin
45                                  // shift the digits on the left display by
                                       1 digit
46                                      displayEnable[7:4] <=
                                           {displayEnable[6:4], 1'b1};
47                                      display[31:16] <= {display[27:16],
                                           expected};
48                                  end
49                                  updateDisplay <= 0;
50                          end
51                      // if the download signal is given
52                      if (load)
53                          begin
54                              // we shift the digits on the right dipley by 1
                                   digit
55                              displayEnable[3:0] <= {displayEnable[2:0],
                                   1'b1};
56                              display[15:0 ]<= {display[11:0], data};
57                              // if the correct digit is entered
58                              if (data == expected)
59                                  begin
60                                  // go to the next state
61                                      state <= state + 1'b1;
62                                  end
```

```
                            else
                                // going back to a multiple of the 4th
                                begin
                                    state <= state & 4'b1100;
                                end
                            updateDisplay <= 1;
                        end
                end
        end
endmodule
```

Листинг А.10 — Описание модуля работы с семисегментыми индикаторами алгоритма

```
`timescale 1ns / 1ps

module NexysDisplay(
input CLK,
input RST,
input [31:0] HEX_IN,
input [7:0] DISP_EN,
output CA,
output CB,
output CC,
output CD,
output CE,
output CF,
output CG,
output DP,
output [7:0] AN
);
// Internal signals declaration:
//----------------------------------------
reg [9:0] CLK_DIV_H, CLK_DIV_L;
reg CE0_DIV_H, CE0_DIV_L;
reg [2:0] DIGIT_CNT;
reg [3:0] I_CODE;
wire O_SEG_A, O_SEG_B, O_SEG_C, O_SEG_D, O_SEG_E, O_SEG_F, O_SEG_G;
reg [7:0] ANODE_DC;
//----------------------------------------
// 1048576 Clock Divider:
always @ (posedge CLK, posedge RST)
if(RST)
    begin
        CLK_DIV_H <= 10'h000;
        CLK_DIV_L <= 10'h000;
        CE0_DIV_H <= 1'b0;
        CE0_DIV_L <= 1'b0;
```

```
35      end
36  else
37      begin
38          if(CLK_DIV_H == 10'h063) // 3FF - :1024 , 063 - :100
39              begin
40                  CLK_DIV_H <= 10'h000;
41                  CEO_DIV_H <= 1'b1;
42              end
43          else
44              begin
45                  CLK_DIV_H <= CLK_DIV_H + 1;
46                  CEO_DIV_H <= 1'b0;
47              end
48          if(CEO_DIV_H)
49              begin
50                  if(&(CLK_DIV_L))
51                      CLK_DIV_L <= 10'h000;
52                  else
53                      CLK_DIV_L <= CLK_DIV_L + 1;
54              end
55          if(&(CLK_DIV_L) & CEO_DIV_H)
56              CEO_DIV_L <= 1'b1;
57          else
58              CEO_DIV_L <= 1'b0;
59      end
60  //----------------------------------------
61  // Display Digit Counter:
62  always @ (posedge CLK, posedge RST)
63  if(RST) DIGIT_CNT <= 3'd0;
64  else if(CEO_DIV_L) DIGIT_CNT <= DIGIT_CNT + 1;
65  //----------------------------------------
66  // Display Digit Multiplexer:
67  always @ (DIGIT_CNT, HEX_IN, DISP_EN)
68  case(DIGIT_CNT)
69  3'd0:
70  begin
71  I_CODE <= HEX_IN[3:0];
72  ANODE_DC <= DISP_EN[0] ? 8'd1 : 0;
73  end
74  3'd1:
75  begin
76  I_CODE <= HEX_IN[7:4];
77  ANODE_DC <= DISP_EN[1] ? 8'd2 : 0;
78  end
79  3'd2:
80  begin
```

```
I_CODE <= HEX_IN[11:8];
ANODE_DC <= DISP_EN[2] ? 8'd4 : 0;
end
3'd3:
begin
I_CODE <= HEX_IN[15:12];
ANODE_DC <= DISP_EN[3] ? 8'd8 : 0;
end
3'd4:
begin
I_CODE <= HEX_IN[19:16];
ANODE_DC <= DISP_EN[4] ? 8'd16 : 0;
end
3'd5:
begin
I_CODE <= HEX_IN[23:20];
ANODE_DC <= DISP_EN[5] ? 8'd32 : 0;
end
3'd6:
begin
I_CODE <= HEX_IN[27:24];
ANODE_DC <= DISP_EN[6] ? 8'd64 : 0;
end
default:
begin
I_CODE <= HEX_IN[31:28];
ANODE_DC <= DISP_EN[7] ? 8'd128 : 0;
end
endcase
//-----------------------------------------
// 7-Segment Decoder:
SevenSegDec DISP_DEC (
.I_CODE(I_CODE),
.O_SEG_A(O_SEG_A),
.O_SEG_B(O_SEG_B),
.O_SEG_C(O_SEG_C),
.O_SEG_D(O_SEG_D),
.O_SEG_E(O_SEG_E),
.O_SEG_F(O_SEG_F),
.O_SEG_G(O_SEG_G));
//-----------------------------------------
assign AN = ~ANODE_DC | {8{RST}};
assign CA = ~O_SEG_A;
assign CB = ~O_SEG_B;
assign CC = ~O_SEG_C;
assign CD = ~O_SEG_D;
```

```
127 assign CE = ~O_SEG_E;
128 assign CF = ~O_SEG_F;
129 assign CG = ~O_SEG_G;
130 assign DP = 0;
131 //----------------------------------------
132 endmodule
```

Листинг А.11 — Описание сигнального дешифратора

```
1  `timescale 1ns / 1ps
2
3  module SevenSegDec(
4  input [3:0] I_CODE, // Input HEX-Digit
5  output reg O_SEG_A, // Segment-A Active High
6  output reg O_SEG_B, // Segment-B Active High
7  output reg O_SEG_C, // Segment-C Active High
8  output reg O_SEG_D, // Segment-D Active High
9  output reg O_SEG_E, // Segment-E Active High
10 output reg O_SEG_F, // Segment-F Active High
11 output reg O_SEG_G // Segment-G Active High
12 );
13 //----------------------------------------
14 // Decoder Comb. Logic:
15 always @ (I_CODE)
16 case(I_CODE)
17 4'h0:
18 begin
19 O_SEG_A <= 1'b1;
20 O_SEG_B <= 1'b1;
21 O_SEG_C <= 1'b1;
22 O_SEG_D <= 1'b1;
23 O_SEG_E <= 1'b1;
24 O_SEG_F <= 1'b1;
25 O_SEG_G <= 1'b0;
26 end
27 4'h1:
28 begin
29 O_SEG_A <= 1'b0;
30 O_SEG_B <= 1'b1;
31 O_SEG_C <= 1'b1;
32 O_SEG_D <= 1'b0;
33 O_SEG_E <= 1'b0;
34 O_SEG_F <= 1'b0;
35 O_SEG_G <= 1'b0;
36 end
37 4'h2:
38 begin
```

```
39 O_SEG_A <= 1'b1;
40 O_SEG_B <= 1'b1;
41 O_SEG_C <= 1'b0;
42 O_SEG_D <= 1'b1;
43 O_SEG_E <= 1'b1;
44 O_SEG_F <= 1'b0;
45 O_SEG_G <= 1'b1;
46 end
47 4'h3:
48 begin
49 O_SEG_A <= 1'b1;
50 O_SEG_B <= 1'b1;
51 O_SEG_C <= 1'b1;
52 O_SEG_D <= 1'b1;
53 O_SEG_E <= 1'b0;
54 O_SEG_F <= 1'b0;
55 O_SEG_G <= 1'b1;
56 end
57 4'h4:
58 begin
59 O_SEG_A <= 1'b0;
60 O_SEG_B <= 1'b1;
61 O_SEG_C <= 1'b1;
62 O_SEG_D <= 1'b0;
63 O_SEG_E <= 1'b0;
64 O_SEG_F <= 1'b1;
65 O_SEG_G <= 1'b1;
66 end
67 4'h5:
68 begin
69 O_SEG_A <= 1'b1;
70 O_SEG_B <= 1'b0;
71 O_SEG_C <= 1'b1;
72 O_SEG_D <= 1'b1;
73 O_SEG_E <= 1'b0;
74 O_SEG_F <= 1'b1;
75 O_SEG_G <= 1'b1;
76 end
77 4'h6:
78 begin
79 O_SEG_A <= 1'b1;
80 O_SEG_B <= 1'b0;
81 O_SEG_C <= 1'b1;
82 O_SEG_D <= 1'b1;
83 O_SEG_E <= 1'b1;
84 O_SEG_F <= 1'b1;
```

```
85  O_SEG_G <= 1'b1;
86  end
87  4'h7:
88  begin
89  O_SEG_A <= 1'b1;
90  O_SEG_B <= 1'b1;
91  O_SEG_C <= 1'b1;
92  O_SEG_D <= 1'b0;
93  O_SEG_E <= 1'b0;
94  O_SEG_F <= 1'b0;
95  O_SEG_G <= 1'b0;
96  end
97  4'h8:
98  begin
99  O_SEG_A <= 1'b1;
100 O_SEG_B <= 1'b1;
101 O_SEG_C <= 1'b1;
102 O_SEG_D <= 1'b1;
103 O_SEG_E <= 1'b1;
104 O_SEG_F <= 1'b1;
105 O_SEG_G <= 1'b1;
106 end
107 4'h9:
108 begin
109 O_SEG_A <= 1'b1;
110 O_SEG_B <= 1'b1;
111 O_SEG_C <= 1'b1;
112 O_SEG_D <= 1'b1;
113 O_SEG_E <= 1'b0;
114 O_SEG_F <= 1'b1;
115 O_SEG_G <= 1'b1;
116 end
117 4'hA:
118 begin
119 O_SEG_A <= 1'b1;
120 O_SEG_B <= 1'b1;
121 O_SEG_C <= 1'b1;
122 O_SEG_D <= 1'b0;
123 O_SEG_E <= 1'b1;
124 O_SEG_F <= 1'b1;
125 O_SEG_G <= 1'b1;
126 end
127 4'hB:
128 begin
129 O_SEG_A <= 1'b0;
130 O_SEG_B <= 1'b0;
```

```
131 O_SEG_C <= 1'b1;
132 O_SEG_D <= 1'b1;
133 O_SEG_E <= 1'b1;
134 O_SEG_F <= 1'b1;
135 O_SEG_G <= 1'b1;
136 end
137 4'hC:
138 begin
139 O_SEG_A <= 1'b1;
140 O_SEG_B <= 1'b0;
141 O_SEG_C <= 1'b0;
142 O_SEG_D <= 1'b1;
143 O_SEG_E <= 1'b1;
144 O_SEG_F <= 1'b1;
145 O_SEG_G <= 1'b0;
146 end
147 4'hD:
148 begin
149 O_SEG_A <= 1'b0;
150 O_SEG_B <= 1'b1;
151 O_SEG_C <= 1'b1;
152 O_SEG_D <= 1'b1;
153 O_SEG_E <= 1'b1;
154 O_SEG_F <= 1'b0;
155 O_SEG_G <= 1'b1;
156 end
157 4'hE:
158 begin
159 O_SEG_A <= 1'b1;
160 O_SEG_B <= 1'b0;
161 O_SEG_C <= 1'b0;
162 O_SEG_D <= 1'b1;
163 O_SEG_E <= 1'b1;
164 O_SEG_F <= 1'b1;
165 O_SEG_G <= 1'b1;
166 end
167 default:
168 begin
169 O_SEG_A <= 1'b1;
170 O_SEG_B <= 1'b0;
171 O_SEG_C <= 1'b0;
172 O_SEG_D <= 1'b0;
173 O_SEG_E <= 1'b1;
174 O_SEG_F <= 1'b1;
175 O_SEG_G <= 1'b1;
176 end
```

```
endcase
//----------------------------------------
endmodule
```

Листинг А.12 — Реализация анализатора последовательности. Файл верхнего уровня

```
`timescale 1ns / 1ps
module top(
    input CLK,
    input CPU_RSTn,
    input BTN_C,
    input  [3:0] SW,
    output [7:0] CAT,
    output [7:0] AN
    );

    reg rst;

    always @(posedge CLK) begin
        rst <= ~CPU_RSTn;
    end
    wire [31:0] hex;
    wire [7:0] disp_en;

    NexysDisplay Disp(
        .CLK(CLK),
        .RST(rst),
        .HEX_IN(hex),
        .DISP_EN(disp_en),
        .CA(CAT[0]),
        .CB(CAT[1]),
        .CC(CAT[2]),
        .CD(CAT[3]),
        .CE(CAT[4]),
        .CF(CAT[5]),
        .CG(CAT[6]),
        .DP(CAT[7]),
        .AN(AN)
    );
    wire divClk;
    freq_div DIV(
    .rst(rst),
    .clk(CLK),
    .co(divClk)
    );
    wire loadDebounced;
    wire loadDebounced1;
```

```
42        M_BTN_FILTER_V10 Filter(
43       .CLK(CLK),
44      .CE(divClk),
45      .BTN_IN(BTN_C),
46      .RST(rst),
47      .BTN_OUT(),
48      .BTN_CEO(loadDebounced)
49      );
50      seqAuto Auto(
51      .clk(CLK),
52      .rst(rst),
53      .load(loadDebounced),
54      .data(SW),
55      .display(hex),
56      .displayEnable(disp_en)
57      );
58  endmodule
```

Листинг А.13 — Описание блока управления матричным индикатором

```
1  `timescale 1ns / 1ps
2
3  module LR4_MATRIX_DISP_V10(
4    input       [3:0] data,
5    input             CLK,
6    input             CE,
7    input             RST,
8    output reg [7:0] col_select,
9    output reg [7:0] rows
10  );
11
12      reg [2:0] column_ctr;
13
14      reg [3:0] digit;
15
16      reg [7:0] mem [127:0];
17
18      initial
19      begin
20          digit = 0;
21          column_ctr = 0;
22          col_select = 8'hFF;
23          $readmemb("lcd_digits.mem", mem);
24      end
25
26      wire digit_reg_ceo;
27      assign digit_reg_ceo = column_ctr == 7;
```

```
28
29      always@*
30      begin
31          case(column_ctr)
32              3'd0: col_select <= 8'b01111111;
33              3'd1: col_select <= 8'b10111111;
34              3'd2: col_select <= 8'b11011111;
35              3'd3: col_select <= 8'b11101111;
36              3'd4: col_select <= 8'b11110111;
37              3'd5: col_select <= 8'b11111011;
38              3'd6: col_select <= 8'b11111101;
39              3'd7: col_select <= 8'b11111110;
40          endcase
41      end
42
43      always@*
44      begin
45          rows = mem[{digit, column_ctr}];
46      end
47
48      always@(posedge CLK, posedge RST)
49      begin
50          if (RST)
51              begin
52                digit = 0;
53                column_ctr = 0;
54                col_select = 8'hFF;
55          end
56          else
57              begin
58                if(CE)
59                begin
60                      column_ctr <= column_ctr + 1;
61                end
62                if(digit_reg_ceo)
63                begin
64                      digit <= data;
65                end
66            end
67      end
68  endmodule
```

Листинг А.14 — Управление матричным индикатором. Модуль вернхнего уровня

```
1  `timescale 1ns / 1ps
2
3  module top(
```

```
4     input          CLK,
5     input          SYS_NRST,
6     input          STEP,
7     input          UP,
8
9     output [7:0]   rows,
10    output [7:0]   cols
11
12    );
13
14  reg         RST = 0;
15    wire [3:0] seq;
16    wire        CE_1MHZ;
17
18    always@(posedge  CLK)
19       RST = SYS_NRST;
20
21    // Divider 1MHz:
22    M_CLOCK_DIVIDER #(
23    .DIVIDER(48),
24    .CNT_WDT(6)
25    ) DIV_1MHZ (
26    .CLK(CLK),
27    .RST(RST),
28    .CEO(CE_1MHZ));
29
30    wire BTN_STEP, BTN_UP;
31    // Filtered input signals
32    M_BTN_FILTER_V10 FLTR_STEP(.BTN_IN(STEP),.CLK(CLK)
33    ,.CE(CE_1MHZ),.RST(RST),.BTN_CEO(BTN_STEP));
34    M_BTN_FILTER_V10 FLTR_UP(.BTN_IN(UP),.CLK(CLK)
35    ,.CE(CE_1MHZ),.RST(RST),.BTN_OUT(BTN_UP));
36
37    // FSM sequence generator
38    LR2_SEQ_GEN_FSM fsm(.RST(RST), .CLK(CLK), .CE(STEP), .load(0), .up(UP),
          .data(0), .seq(seq));
39
40    LR4_MATRIX_DISP_V10 lcd(.data(seq), .CE(CE_1MHZ), .CLK(CLK),
          .col_select(cols), .rows(rows), .RST(RST));
41
42  endmodule
```

Листинг А.15 — Управление матричным индикатором. Тестовый модуль

```
1  `timescale 1ns / 1ps
2
3  module test_top;
```

```
4
5       // Clock Generator - 48 MHz
6       parameter PERIOD_CLK = 20.8; // 20.8ns
7       parameter DUTY_CYCLE_CLK = 0.4;
8
9       reg         CLK, SYS_NRST;
10      reg         STEP;
11      reg         UP;
12      wire [7:0] rows;
13      wire [7:0] cols;
14
15      initial
16      forever
17          begin
18              CLK = 1'b0;
19              #(PERIOD_CLK-(PERIOD_CLK*DUTY_CYCLE_CLK)) CLK = 1'b1;
20              #(PERIOD_CLK*DUTY_CYCLE_CLK);
21          end
22
23      // Init. Reset startUP pulse (100ns POR)
24      initial
25      begin
26          SYS_NRST = 0;
27          #100;
28          SYS_NRST = 1;
29          #100;
30          SYS_NRST = 0;
31      end
32
33      top utt(
34          .CLK(CLK),
35          .SYS_NRST(SYS_NRST),
36          .STEP(STEP),
37          .UP(UP),
38          .rows(rows),
39          .cols(cols)
40            );
41
42
43          assign COL_7R=cols[7];
44
45
46          initial begin
47          // Initialize Inputs
48          STEP = 0;
49         UP   = 1;
```

```
50          // Wait 100 ns for global reset to finish
51          #100;
52
53
54          @(posedge COL_7R);
55          @(posedge CLK);
56          #(PERIOD_CLK*0.2);
57          STEP = 1'b1;
58          #(PERIOD_CLK);
59          STEP = 1'b0;
60
61
62      // .....
63      // input sequece
64
65          @(posedge COL_7R);
66          @(posedge CLK);
67          #(PERIOD_CLK*0.2);
68          STEP = 1'b1;
69          #(PERIOD_CLK);
70          STEP = 1'b0;
71
72      end
73  endmodule
```

Листинг А.16 — Генератор ШИМ

```
1   `timescale 1ns / 1ps
2   module PWM_module #(parameter UDW = 4)
3       (
4       input CLK,
5       input RST,
6       input CE,
7       input [UDW - 1:0] pwm_in,
8       output reg pwm_p,
9        output reg [UDW - 1:0] pwm_reg, fsm_state
10      );
11
12      assign pwm_n = ~pwm_p;
13
14      initial
15      begin
16          pwm_reg <= 0;
17          fsm_state <= 0;
18          pwm_p <= 0;
19      end
20
```

```
21
22      always@(posedge CLK, posedge RST)
23      begin
24          if(RST)
25          begin
26              pwm_reg <= 0;
27          end
28          else if(CE && fsm_state == {{(UDW - 1){1'b1}}, {1'b0}})
29          begin
30              pwm_reg <= pwm_in;
31          end
32      end
33
34      always@(posedge CLK, posedge RST)
35      begin
36          if(RST)
37          begin
38              pwm_reg <= 0;
39              fsm_state <= 0;
40              pwm_p <= 0;
41          end
42          else if (CE)
43          begin
44              case(fsm_state)
45                  0:
46                  begin
47                      fsm_state <= {{(UDW - 1){1'b1}}, {1'b0}};
48                      pwm_p <= 0;
49                  end
50                  {(UDW){1'b1}}:
51                  begin
52                      fsm_state <= 1;
53                      pwm_p <= pwm_reg == {(UDW){1'b0}} ? 0 : 1;
54                  end
55                  default:
56                  begin
57                      fsm_state <= fsm_state + 1;
58                      pwm_p <= pwm_reg > fsm_state ? 1 : 0;
59                  end
60              endcase
61          end
62          else
63          begin
64              fsm_state <= fsm_state;
65          end
66      end
```

```
endmodule
```

Листинг А.17 — Сдвиговый регистр

```
‘timescale 1ns / 1ps

module SR64_S4B(
     input [63:0] data_in,
     input RST,
    input CLK,
    input shift_4b_l,
    input shift_4b_r,
    output reg [63:0] data_out
    );


    always@(posedge CLK, posedge RST)
    begin
        if (RST)
            begin
                data_out <=data_in;
            end
        else
        begin
            if(shift_4b_l)
            begin
                data_out <= {data_out[59:0], data_out[63:60]};
            end
            else if (shift_4b_r)
            begin
                data_out <= {data_out[3:0], data_out[63:4]};
            end
            else
            begin
                data_out <= data_out;
            end
        end
    end
endmodule
```

Листинг А.18 — Блок управления матричным индикатором

```
‘timescale 1ns / 1ps

module LCDMatrixDriverNew(
    input [63:0] data,
     input CLK,
    input CE,
```

```
7       input RST,
8       output reg [7:0] col_select,
9       output reg [7:0] rows
10      );
11
12      reg [2:0] column_ctr;
13
14      reg [3:0] digit;
15
16      reg [7:0] mem [127:0];
17
18      initial
19      begin
20          digit = 0;
21          column_ctr = 0;
22          col_select = 8'hFF;
23      end
24
25      wire digit_reg_ceo;
26      assign digit_reg_ceo = column_ctr == 7;
27
28      // Column decoder
29      always@*
30      begin
31          case(column_ctr)
32              3'd0: col_select <= 8'b01111111;
33              3'd1: col_select <= 8'b10111111;
34              3'd2: col_select <= 8'b11011111;
35              3'd3: col_select <= 8'b11101111;
36              3'd4: col_select <= 8'b11110111;
37              3'd5: col_select <= 8'b11111011;
38              3'd6: col_select <= 8'b11111101;
39              3'd7: col_select <= 8'b11111110;
40          endcase
41      end
42
43      always@*
44      begin
45          rows = {data[63-column_ctr],
46                  data[55-column_ctr],
47                  data[47-column_ctr],
48                  data[39-column_ctr],
49                  data[31-column_ctr],
50                  data[23-column_ctr],
51                  data[15-column_ctr],
52                  data[7-column_ctr]
```

```
53            };
54      end
55
56      always@(posedge CLK, posedge RST)
57      begin
58          if (RST)
59              begin
60                digit = 0;
61                column_ctr = 0;
62                col_select = 8'hFF;
63              end
64              else
65                  begin
66                    if(CE)
67                    begin
68                          column_ctr <= column_ctr + 1;
69                    end
70                    if(digit_reg_ceo)
71                    begin
72                          digit <= data;
73                    end
74                end
75      end
76  endmodule
```

Листинг А.19 — Генератор ШИМ. Модуль верхнего уровня

```
1   `timescale 1ns / 1ps
2
3   module top(
4       input          CLK,
5       input          SYS_NRST,
6       input          LEFT,
7       input          RIGHT,
8
9       output [7:0]   rows,
10      output [7:0]   cols
11      );
12      reg         RST = 0;
13      wire [3:0] seq;
14      wire        CE_1MHZ;
15
16      always@(posedge  CLK)
17         RST = SYS_NRST;
18
19      reg [63:0] func;
20      wire [63:0] shift_func  ;
```

```
21      wire [15:0] pwm_p;
22
23      reg [15:0] pwm_n;
24
25      initial func<=64'hca25c7d227038440;
26
27      // Divider 1MHz:
28      M_CLOCK_DIVIDER #(
29      .DIVIDER(20),
30      .CNT_WDT(6)
31      ) DIV_1KHZ (
32      .CLK(CLK),
33      .RST(RST),
34      .CEO(CE_1MHZ));
35
36      M_BTN_FILTER_V10 FLTR_STEP(.BTN_IN(LEFT),.CLK(CLK)
37                                  ,.CE(CE_1MHZ),.RST(RST),.BTN_CEO(BTN_LEFT));
38      M_BTN_FILTER_V10 FLTR_UP(.BTN_IN(RIGHT),.CLK(CLK)
39                               ,.CE(CE_1MHZ),.RST(RST),.BTN_OUT(BTN_RIGHT));
40
41      SR64_S4B shifter (
42                               .data_in(func),
43                               .RST(RST),
44                               .CLK(CLK),
45                               .shift_4b_l(LEFT),
46                               .shift_4b_r(RIGHT),
47                               .data_out(shift_func)
48      );
49      genvar i;
50      generate
51          for (i=0;i<16;i=i+1) begin
52              PWM_module pwm_i(
53                           .CLK(CLK),
54                           .RST(SYS_NRST),
55                           .CE(CE_1MHZ),
56                           .pwm_in(shift_func[4*(i+1)-1:4*i]),
57                           .pwm_p(pwm_p[i])
58                           );
59              end
60      endgenerate
61      LCDMatrixDriverNew lcd(.data(shift_func), .RST(RST), .CE(CE_1MHZ)
62                              , .CLK(CLK), .col_select(cols), .rows(rows));
63
64
65  endmodule
```

Листинг А.20 — Генератор ШИМ. Тестовый модуль

```
1  `timescale 1ns / 1ps
2
3  module test_top;
4
5      // Clock Generator - 48 MHz
6      parameter PERIOD_CLK = 20.8; // 20.8ns
7      parameter DUTY_CYCLE_CLK = 0.4;
8
9      reg        CLK, SYS_NRST;
10     reg        LEFT;
11     reg        RIGHT;
12     wire [7:0] rows;
13     wire [7:0] cols;
14
15     initial
16     forever
17         begin
18             CLK = 1'b0;
19             #(PERIOD_CLK-(PERIOD_CLK*DUTY_CYCLE_CLK)) CLK = 1'b1;
20             #(PERIOD_CLK*DUTY_CYCLE_CLK);
21         end
22
23     // Init. Reset startRIGHT pulse (100ns POR)
24     initial
25     begin
26         SYS_NRST = 0;
27         #100;
28         SYS_NRST = 1;
29         #100;
30         SYS_NRST = 0;
31     end
32     top utt(
33                 .CLK(CLK)
34               ,.SYS_NRST(SYS_NRST)
35               ,.LEFT(LEFT)
36               ,.RIGHT(RIGHT)
37               ,.rows(rows)
38               ,.cols(cols)
39
40           );
41         assign COL_7R=cols[7];
42         initial begin
43         // Initialize Inputs
44         LEFT = 0;
45        RIGHT   = 0;
```

```
46          // Wait 100 ns for global reset to finish
47          #100;
48
49          @(posedge COL_7R);
50          @(posedge CLK);
51          #(PERIOD_CLK*0.2);
52          LEFT = 1'b1;
53          #(PERIOD_CLK);
54          LEFT = 1'b0;
55          @(posedge COL_7R);
56          @(posedge CLK);
57          #(PERIOD_CLK*0.2);
58          LEFT = 1'b1;
59          #(PERIOD_CLK);
60          LEFT = 1'b0;
61
62          // --- input signals
63
64
65
66      end
67  endmodule
```

Приложение Б Результаты симуляций

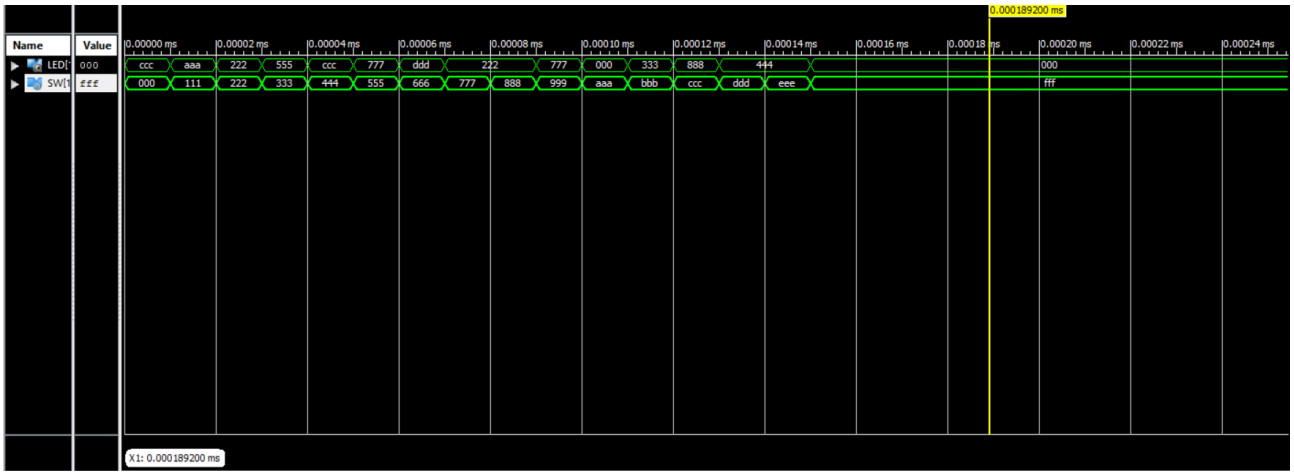


Рисунок Б.1 — Реализация комбинационной логической схемы

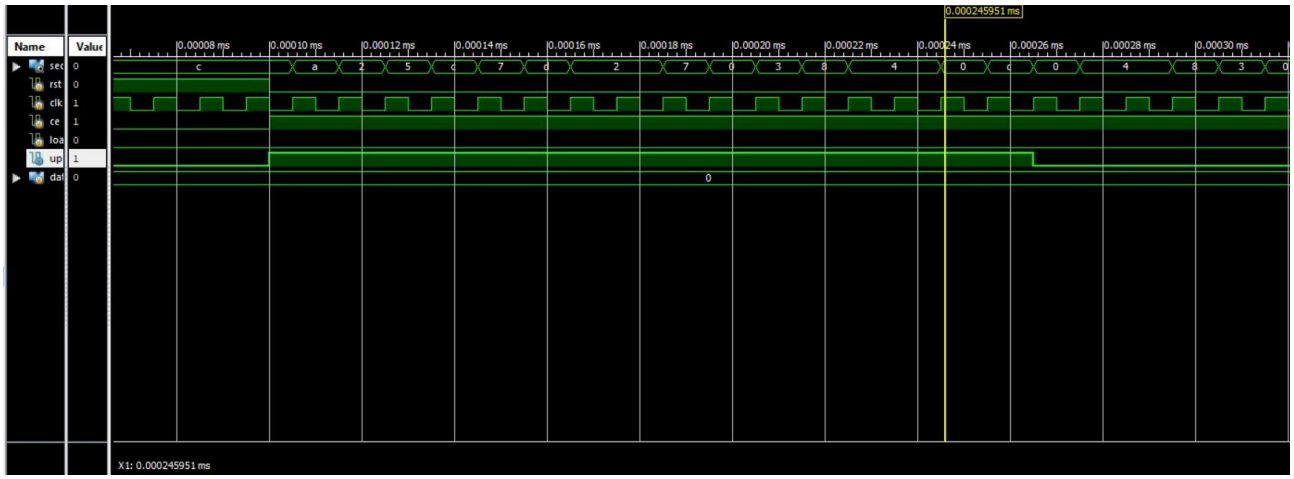


Рисунок Б.2 — Реализация цифрового автомата. Часть 1

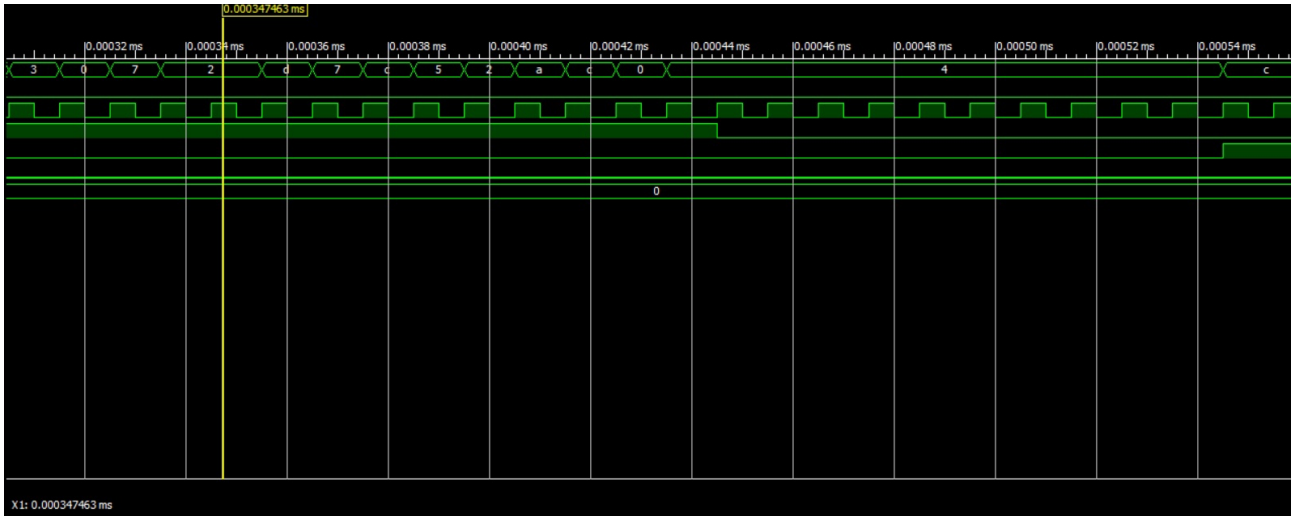


Рисунок Б.3 — Реализация цифрового автомата. Часть 2

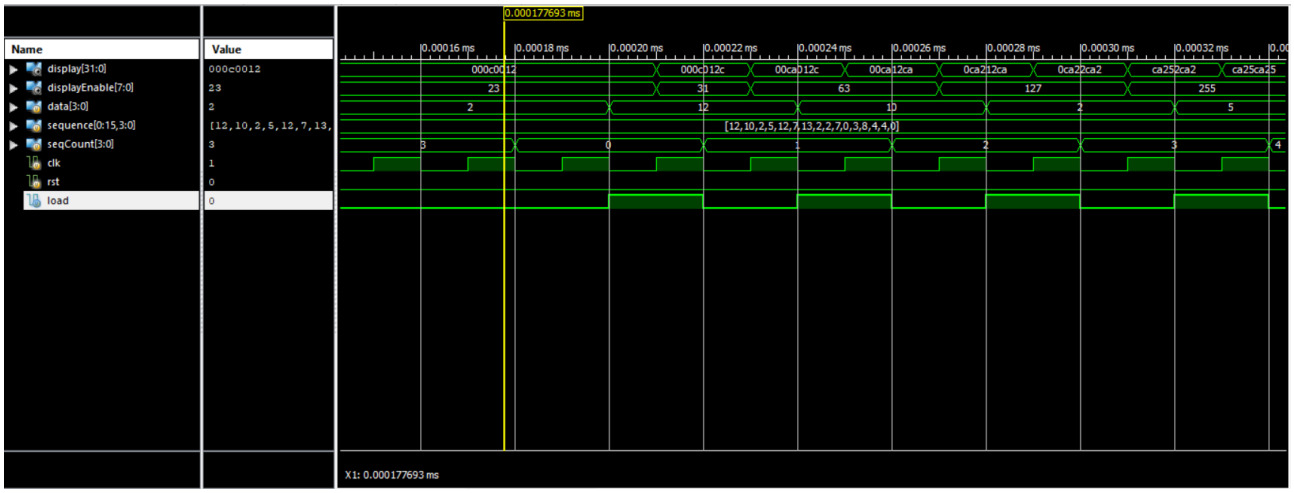


Рисунок Б.4 — Реализация анализатора последовательности. Часть 1

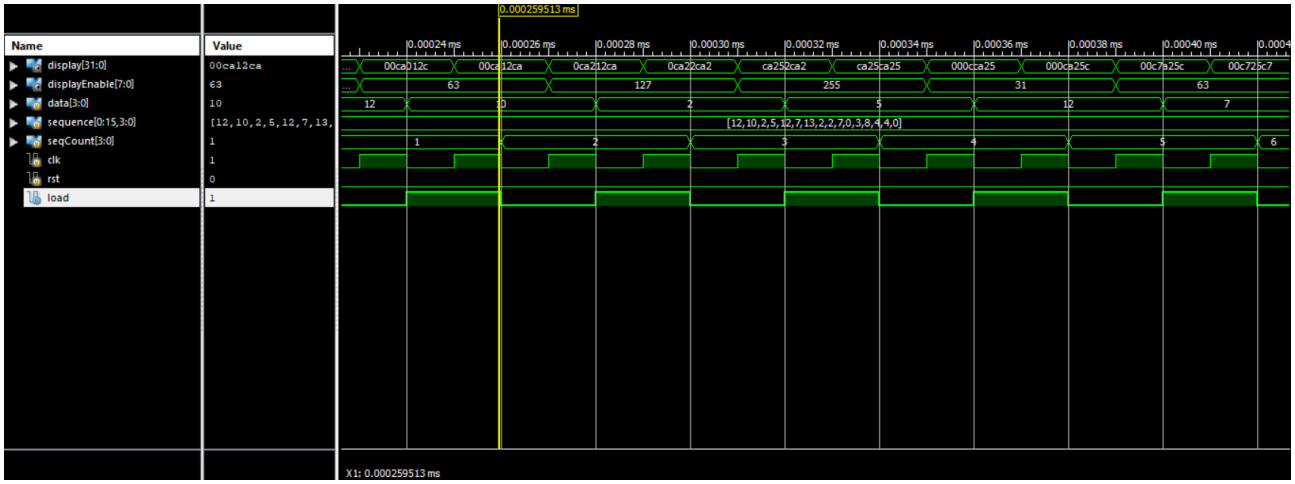


Рисунок Б.5 — Реализация анализатора последовательности. Часть 2

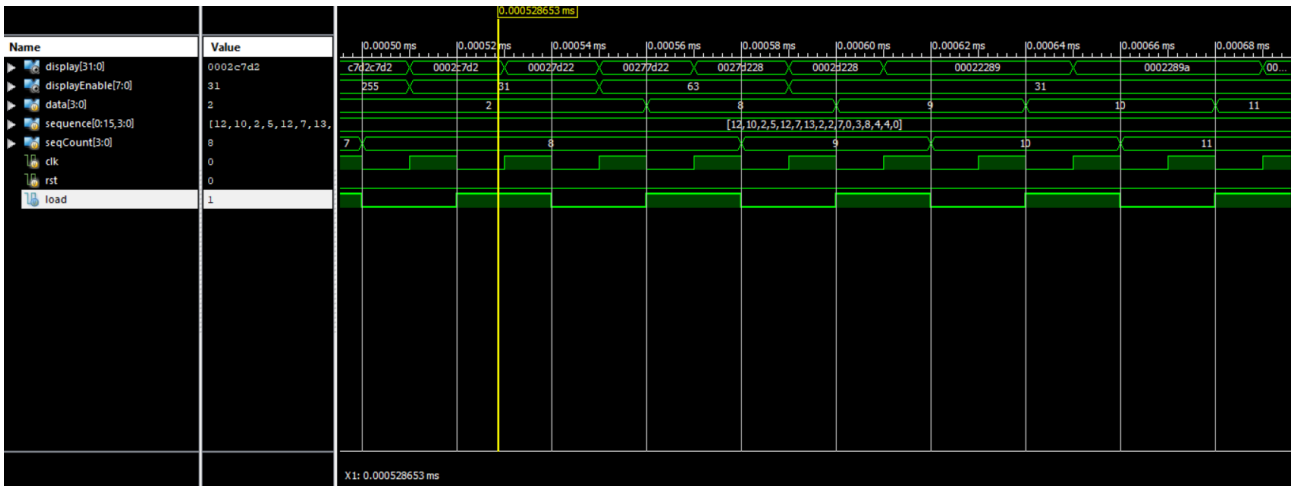


Рисунок Б.6 — Реализация анализатора последовательности. Часть 3

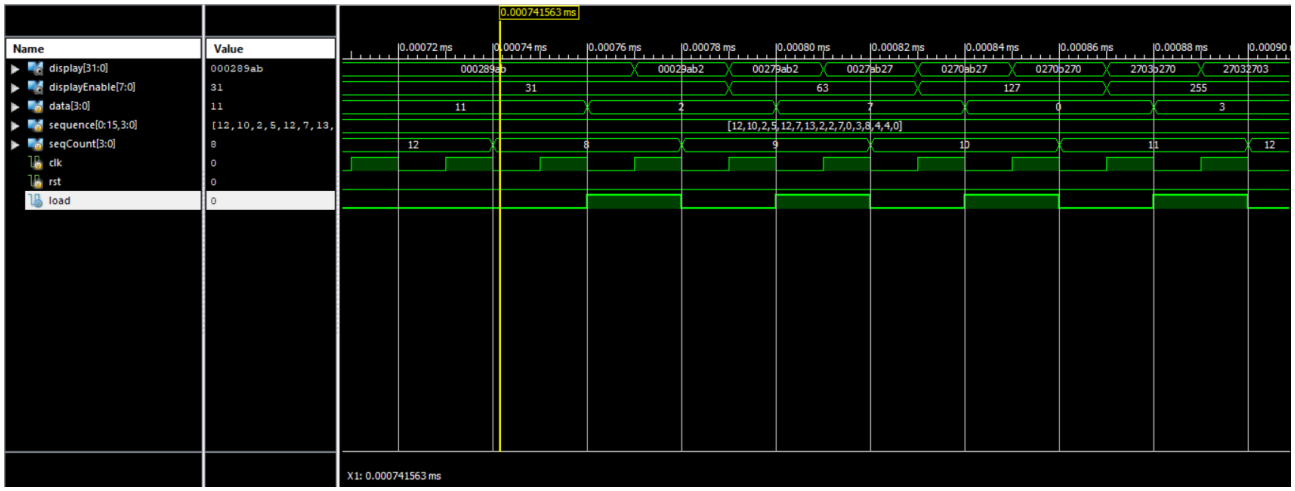


Рисунок Б.7 — Реализация анализатора последовательности. Часть 4

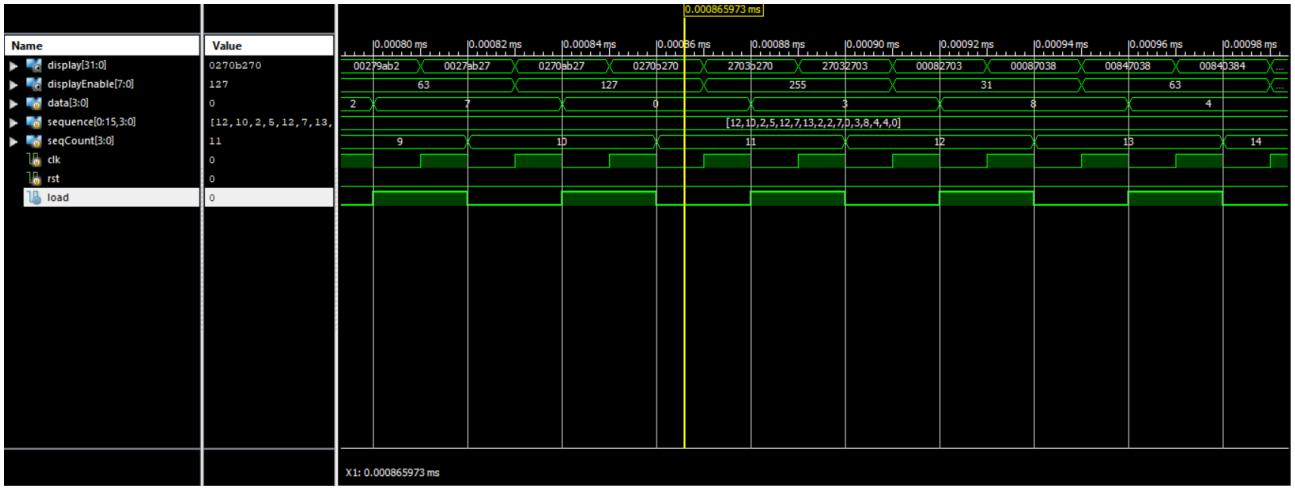


Рисунок Б.8 — Реализация анализатора последовательности. Часть 5

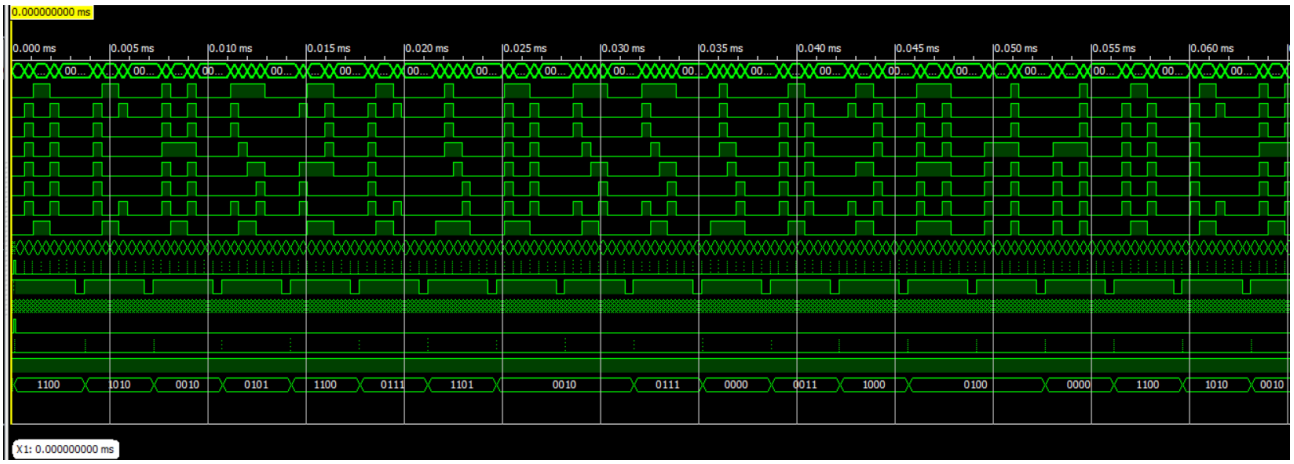


Рисунок Б.9 — Управление матричным индикатором. Результат симуляции

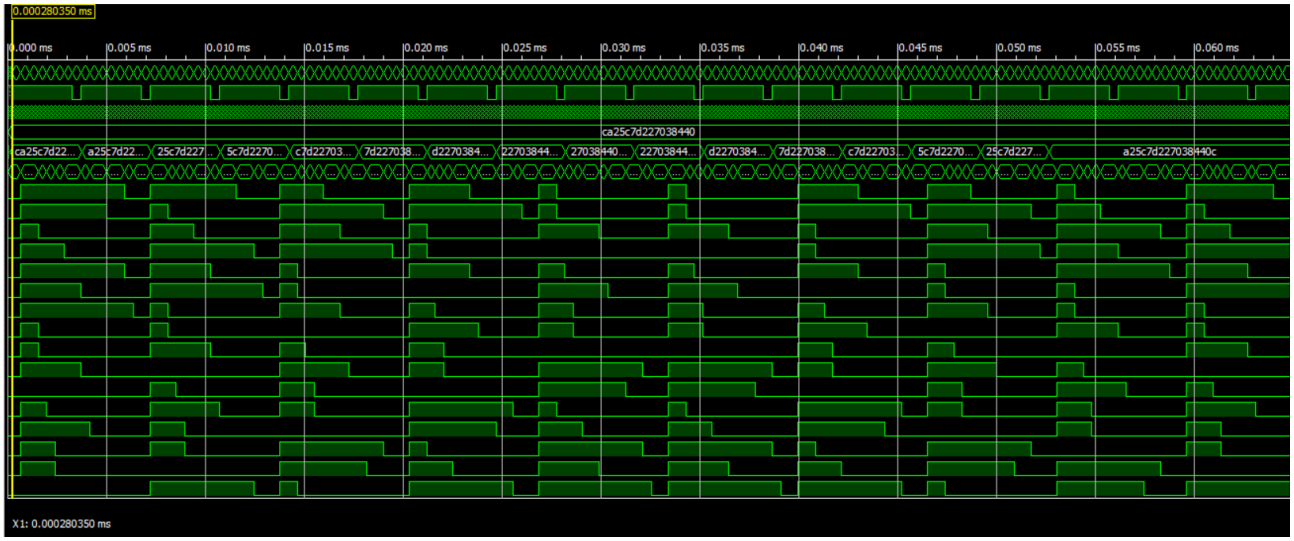


Рисунок Б.10 — Генерация ШИМ. Результат симуляции. Часть 1

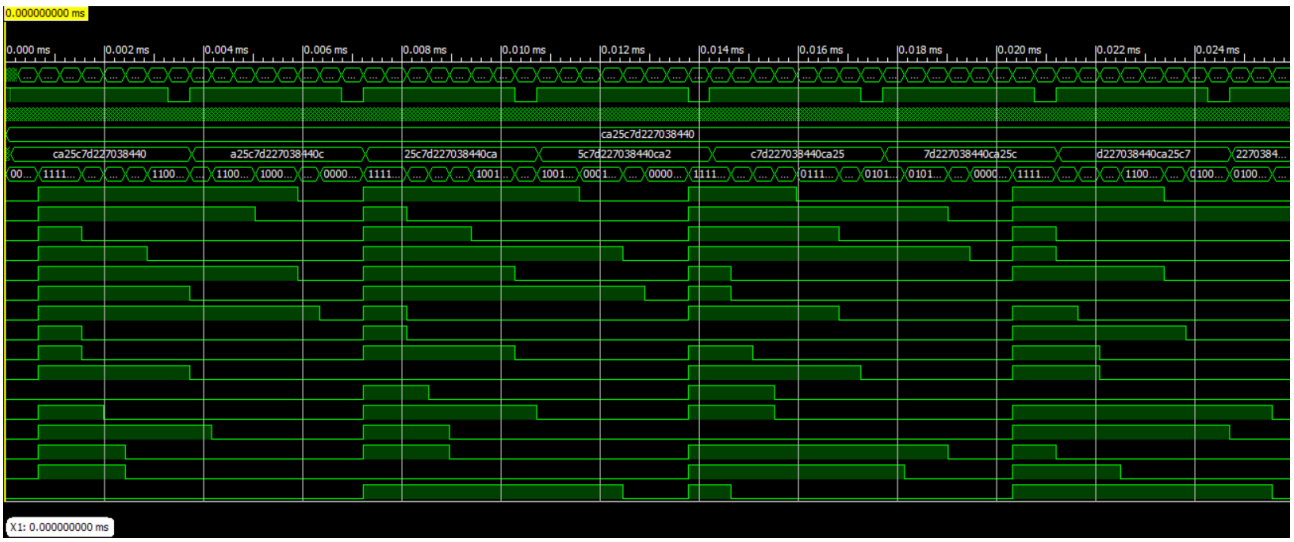


Рисунок Б.11 — Генерация ШИМ. Результат симуляции. Часть 2