САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Архитектура ЭВМ

Отчет

по домашней работе № 2

«ПОСТРОЕНИЕ СЛОЖНЫХ ЛОГИЧЕСКИХ СХЕМ»

Выполнил: Султанов Мирзомансурхон Махсудович

Номер ИСУ: 311629

студ. гр. М3134

Санкт-Петербург

2021

Цель работы: моделирование сложных логических схем.

Инструментарий и требования к работе: работа выполняется в logisim evolution.

Теоретическая часть

Счётчик – это устройство, которое осуществляет счёт входных сигналов и хранение накапливаемой величины. В основе любого счётчика лежат триггеры, однако в самих счётчиках триггеры связаны более сложным образом. Содержимое счётчика сохраняется только до тех пор, пока включено питание схемы. В цифровых приборах счётчик используется для формирования последовательности чисел, для деления частоты и для подсчёта количества сигналов.

Очевидно, что счётчик используется для счёта входных импульсов. То есть с приходом каждого нового входного импульса двоичный код на выходе счётчика увеличивается на единицу, если счётчик суммирующий, и уменьшается на единицу, если счётчик вычитающий (см. рисунок 1) Срабатывать счётчик может по отрицательному фронту тактового сигнала (как на рисунке) или по положительному фронту.

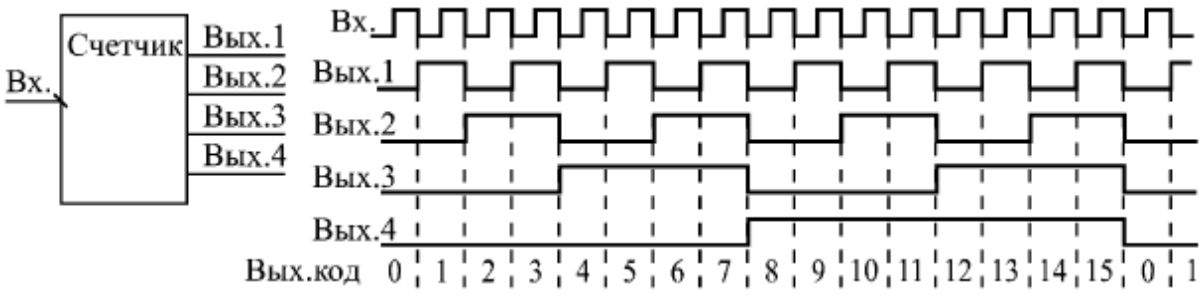

Рисунок 1 – Работа 4-разрядного суммирующего счётчика

Режим счёта обеспечивается использованием внутренних триггеров, работающих в счётном режиме. Выходы счётчика представляют собой выходы триггеров. Каждый выход счётчика представляет собой разряд

двоичного кода, причём разряд, переключающийся чаще других по каждому входному импульсу, будет младшим, а разряд, переключающийся реже других, - старшим. На 1 рисунке видно, что на 1 выходе находится младший разряд, а на 4 выходе - старший.

Число разрядов счётчика определяется максимальной разрядностью числа, которое должно в нём храниться. Двоичный N-разрядный счётчик будет иметь 2^N различных состояний. Каждому состоянию счётчика соответствует двоичное число, начиная от 0 до 2^N-1.

К основным параметрам счётчика кроме $К_{Сч}$ относятся разрешающая способность ($t_р$) и время установления кода ($t_{уст}$). Разрешающая способность – минимально допустимый интервал времени между входными импульсами, при котором еще не происходит сбоя, т.е. пропуска счёта сигналов. Время установки кода – это интервал времени между моментом поступления на вход импульса счета и моментом завершения перехода счётчика в нулевое состояние.

По направлению счёта счётчики делятся на суммирующие, вычитающие и реверсивные. Суммирующие счётчики работают на увеличение выходного кода по каждому входному импульсу; это основной режим, имеющийся во всех счётчиках, он называется режимом прямого счета. Вычитающие счётчики, очевидно, работают на уменьшение выходного кода по каждому входному импульсу; он также называется режимом инверсного счёта. Реверсивный же счётчик, каждый раз доходя до 0 или 2^N-1, меняет направление счёта.

По модулю счёта счётчики делятся на двоичные, двоично-десятичные и с произвольным модулем счёта. Большинство счётчиков работают в обычном двоичном коде (двоичные счётчики), то есть считают от 0 до (2^N– 1), где N - число разрядов выходного кода счётчика. 4-разрядный счётчик в режиме прямого счёта будет считать от 0 (код 0000) до 15 (код 1111). После максимального значения кода счётчик по следующему входному импульсу

переключается опять в 0, то есть работает по кругу. Если же счёт - инверсный, то счётчик считает до нуля, а дальше переходит к максимальному коду 111...1.

В двоично-десятичных счётчиках предельный код на выходе не превышает максимального двоично-десятичного числа, возможного при данном количестве разрядов. Например, 4-разрядный двоично-десятичный счётчик в режиме прямого счета будет считать от 0 (код 0000) до 9 (код 1001), а затем снова от 0 до 9. При инверсном счете двоично-десятичные счётчики считают до нуля, а со следующим входным импульсом переходят к максимально возможному двоично-десятичному числу (то есть 9 - для 4-разрядного счётчика).

Принимая во внимание всё выше сказанное, можно сделать вывод, что верны следующие соотношения:

$$Q = (D + SUM(C)) \bmod M$$

$$CR = (D + SUM(C)) \setminus M$$

В этих формулах: Q - код на выходах счётчика, D - начальное значение записанное в счётчик, SUM(C) - сумма импульсов поступивших на вход в процессе счета и M - модуль счёта или число различных состояний счётчика (число импульсов поступивших на счетный вход, после которых счетчик возвращается в исходное состояние), CR - число импульсов переноса, возникающих при возврате счётчика в исходное состояние на одноименном выходе, mod - операция нахождения остатка при делении на M, \ - операция целочисленного деления .

По способу счёта счётчики могут быть асинхронными и синхронными. В асинхронных счётчиках триггеры переключаются последовательно (асинхронно) от разряда к разряду, а в синхронных одновременно. Один Т-триггер обеспечивает модуль счёта M = 2, а n триггеров дадут $M = 2^n$. При суммировании импульсов необходимо формировать перенос из i-го в (i+1)-ый разряд по следующему правилу:

Правило 1: перенос CR из i-го в (i+1)-ый разряд формируется, если во всех разрядах с i-го по 0-й записана максимальная для данной системы счисления цифра, при этом разряды младше (i+1)-го обнуляются.

На прямых выходах триггеров этих разрядов Q_i формируется отрицательный перепад (см. рисунок 2), а на инверсных - положительный.

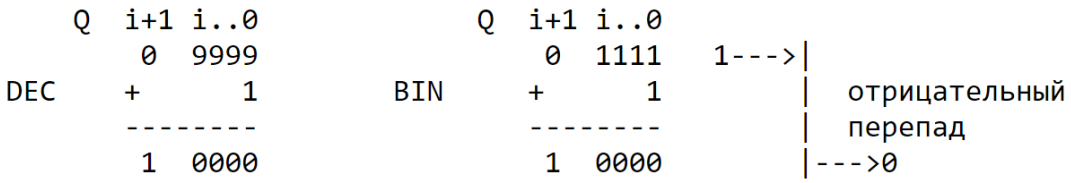

Рисунок 2 – Работа переноса

Правило 2: Если в счётчике используются триггеры с прямым динамическим входом, то сигнал переноса в суммирующем счётчике снимается с инверсных выходов предыдущих триггеров, а если триггеры с инверсным динамическим входом, то сигнал переноса берётся с прямых выходов.

Практическая часть

Кратко опишу план действий. Для начала построим синхронный RS-триггер, затем на его основе построим JK-триггер, затем путём простых преобразований превратим синхронный JK-триггер в синхронный T-триггер и в конце концов построим синхронный суммирующий счётчик с параллельным переносом по модулю 16 и по модулю 12.

RS-триггер построим как на рисунке 3, а JK-триггер как на рисунке 4. Замечу, что в JK-триггер помимо переменных J, C, K ещё будет переменная reset, которая отвечает за включение триггера. Дело в том, что изначально на втором RS-триггере никаких значений нет, из-за этого триггер работает некорректно, так как при любых значениях J, C, K на S и R первого RS-триггера будут подаваться нули. А при нажатии на reset (то есть присваиванию переменной reset 1, а затем 0) мы получим значения J = 0, C

= K = 1, тогда JK-триггер будет хранить 0. При этом замечу, что на reset надо нажимать, только когда значение всех аргументов на входе равны нули, иначе возможны ошибки.

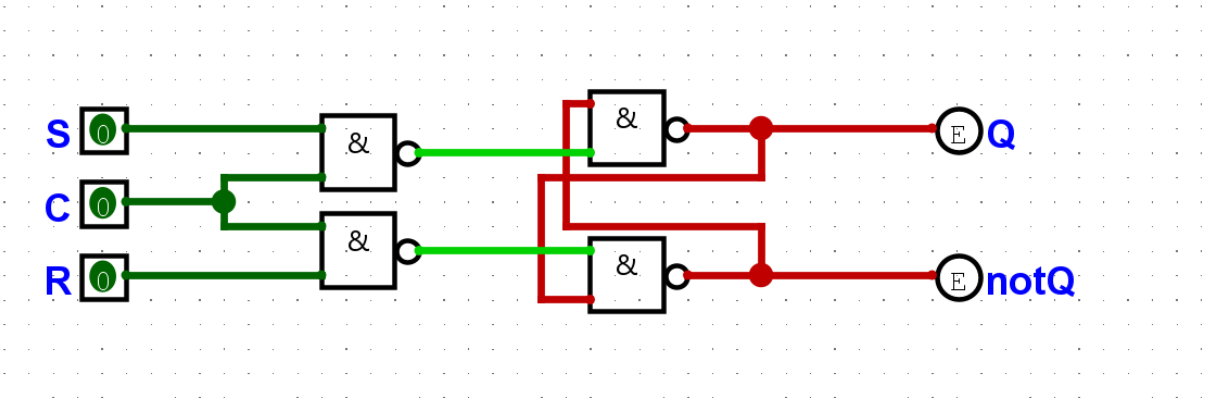

Рисунок 3 – Схема RS-триггера

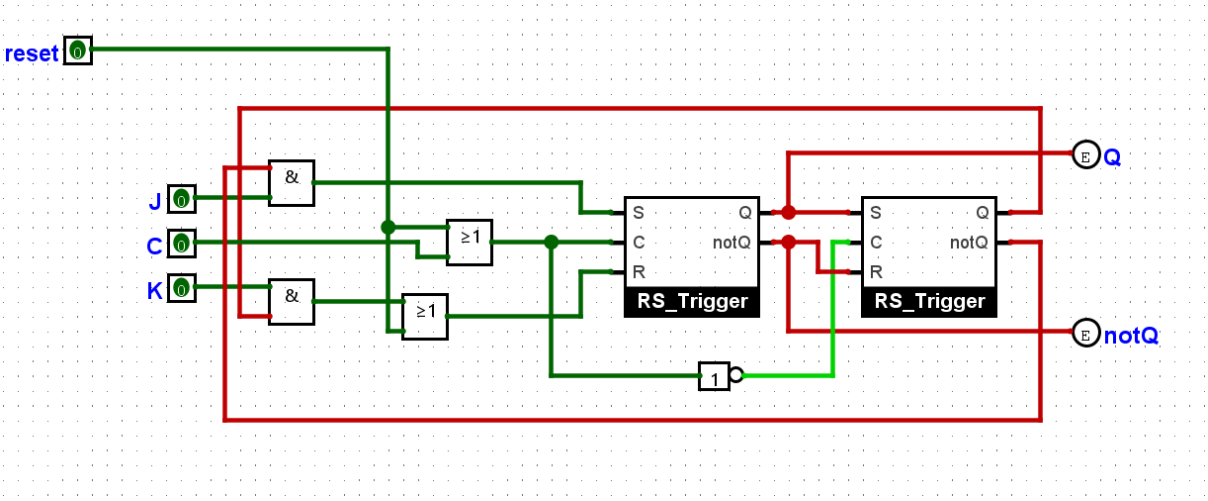

Рисунок 4 – Схема JK-триггера на основе RS-триггера

Для того чтобы наш полученный JK-триггер превратить в T-триггер, достаточно заменить элементы J и K на T, как показано на 5 рисунке. Получается, что если T = 1, то триггер работает, иначе же триггер просто сохраняет своё значение. И при перемене C с 0 на 1 триггер меняет своё значение.

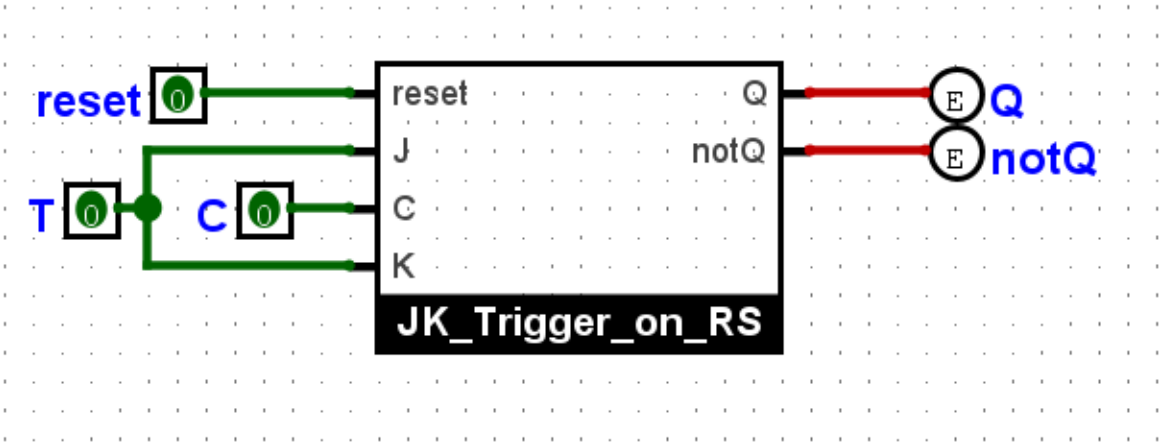

Рисунок 5 – Схема T-триггера на основе JK триггера

Теперь на основе T-триггеров и на самом деле ещё D-триггеров построим синхронный суммирующий счётчик с параллельным переносом. На рисунке 6 можно увидеть эту схему по модулю 16 (при этом старший бит находится снизу, а младший бит сверху). Подводим наш тактовый генератор параллельно ко всем триггерам как переменную C, в то время как к переменной T мы будем подводить конъюнкцию значений предыдущих T-триггеров (по 1 правилу из теории). При этом сначала мы эту конъюнкцию должны запомнить D-триггером, который легко построить. Достаточно D подать на S и его отрицание на R. На схеме D-триггер построен на основе JK-триггера сразу, а не отдельно. Значение конъюнкцию предыдущих битов мы запоминаем, когда C = 0. Соответственно работу счётчика можно поделить на два этапа. При C = 0 мы запоминаем все биты переноса, а при C = 1 увеличиваем счётчик на 1. Без D-триггеров счётчик бы сразу изменил значения всех битов, что не очень хорошо. Таким образом, построен счётчик по основанию 16. Для самого младшего бита очевидным образом нет конъюнкции предыдущих битов, поэтому вместо этого подадим туда CE – carry enable – переменную, которая отвечает за то, включен ли счётчик или нет.

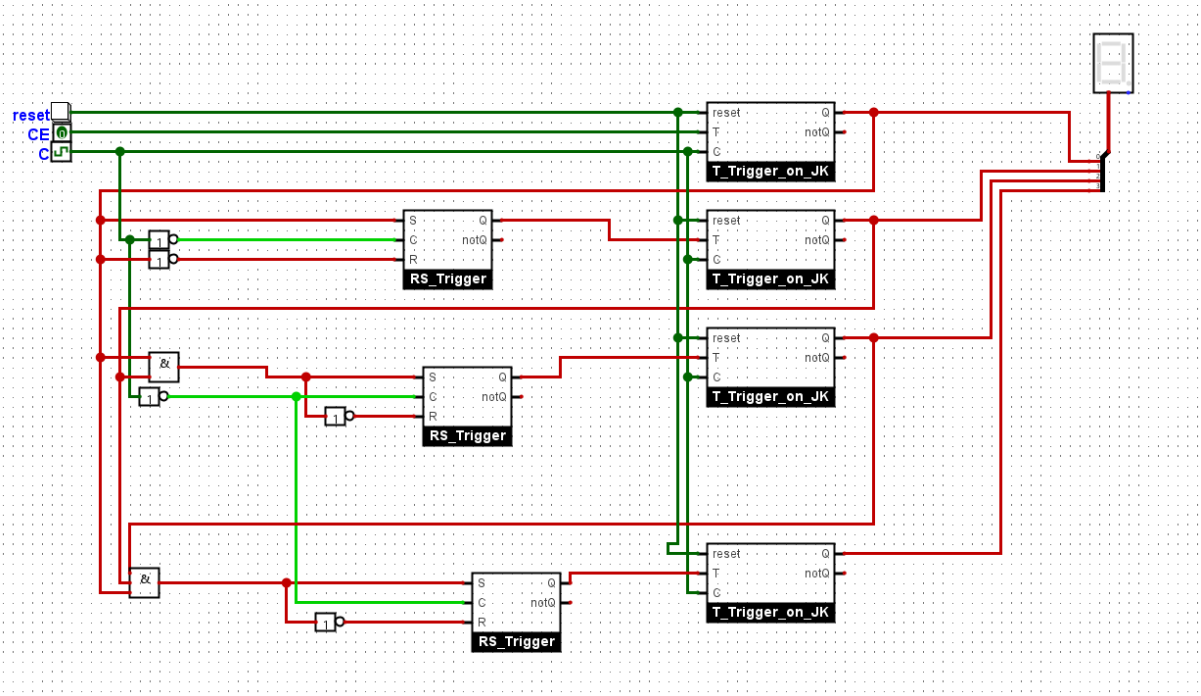

Рисунок 6 – Суммирующий синхронный счётчик с параллельным переносом по модулю 16

Теперь надо его преобразовать в синхронный суммирующий счётчик по модулю 12, то есть когда счётчик доходит до 12, то он сбрасывается до нуля. Для этого возьмём конъюнкцию всех битов таким образом, чтобы выражение давало единицу, только при 11 (код 1011). Затем на тех битах, где стоит единица, к D-триггеру в качестве переменной D подводим дизъюнкцию конъюнкции предыдущих битов и полученную конъюнкцию всех битов. Там же, где стоит ноль, подводим конъюнкцию конъюнкции предыдущих битов и инверсированную полученную конъюнкцию всех битов. На рисунках 7 и 8 можно увидеть синхронный счётчик с параллельным переносом по модулю 12.

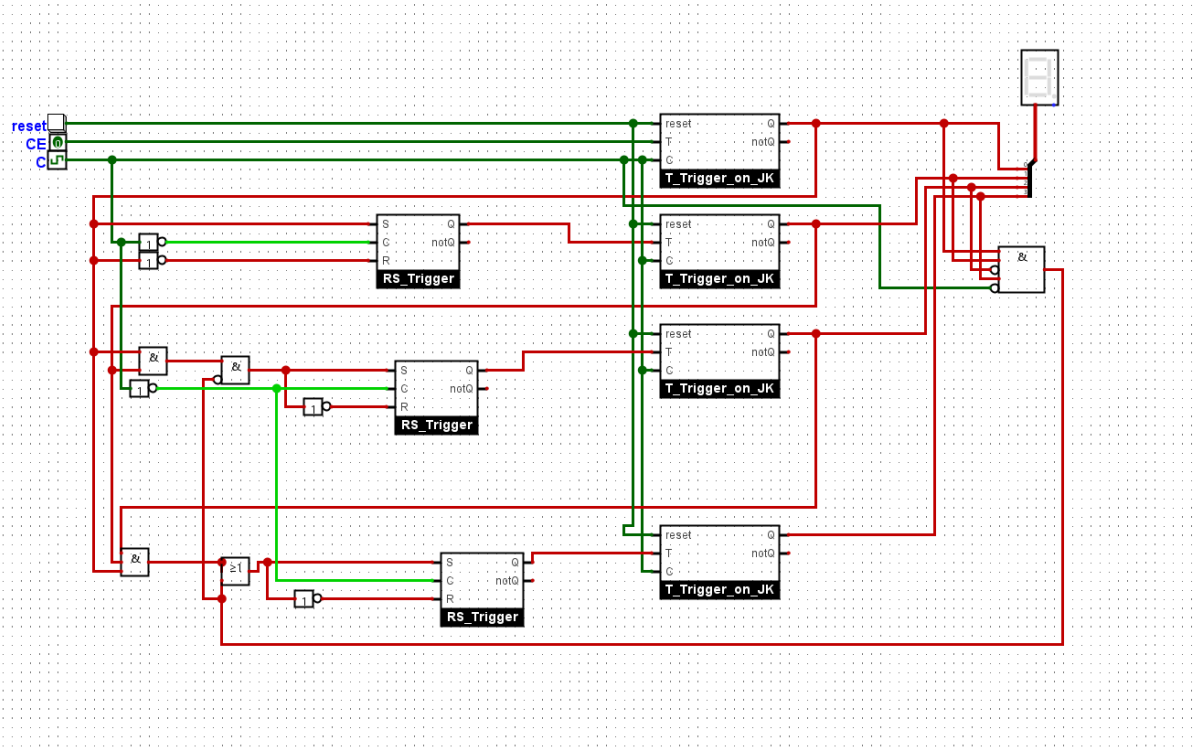

Рисунок 7 – Суммирующий синхронный счётчик с параллельным переносом по модулю 12

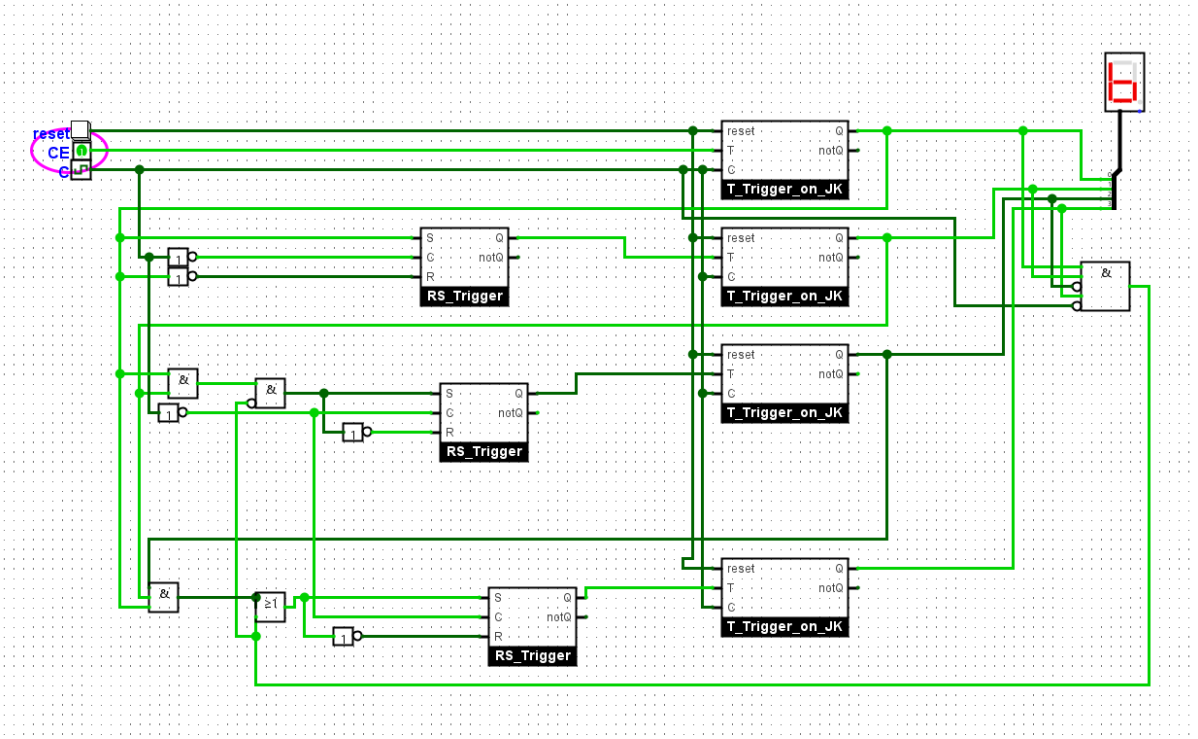

Рисунок 8 – Суммирующий синхронный счётчик с параллельным переносом по модулю 12, когда счётчик равняется 11

Временная диаграмма последней схемы приведена на рисунке 9. Как можно заметить, данный суммирующий счётчик работает по положительному фронту, то есть когда на входе ноль сменяется единицей. В первой строке находится тактовый генератор, а в следующих четырёх выходы Q с триггеров сверху вниз либо же биты нашего числа, начиная с младшего и до старшего, что тоже самое.

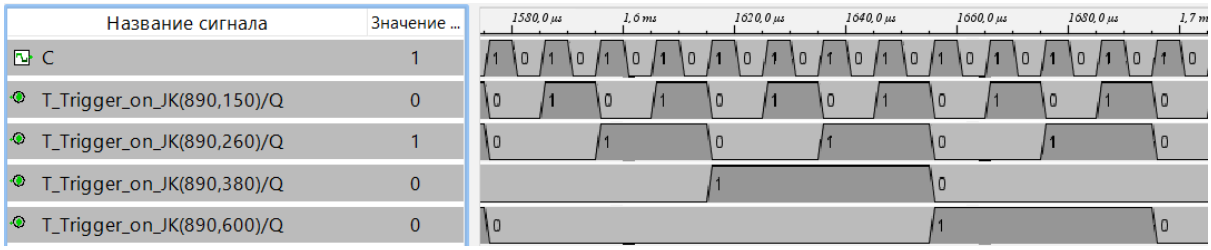

Рисунок 9 – Временная диаграмма суммирующего синхронного счётчика с параллельным переносом по модулю 12

С триггером разобрались, идём дальше. Теперь построим схему для взятия квадратного корня. Здесь будем придерживаться следующего плана. Для начала построим схемы a0, a1, a2, …, a15, которые представляют собой ответ на вопрос: “Будет ли данное число при взятии корня и округлении к нулю равняться i?” для a_i. Заметим, что никогда две разные схемы из этих шестнадцати не будут одновременно равняться 1, так как мы не можем при взятии корня и округления получить два разных числа. Теперь единственное, что нам остаётся, так это в зависимости от того, какая из схем равняется единице, вывести нужный ответ. Очевидно, что для a_i нам надо вывести i.

Для начала разберёмся с “ашками”. Как их построить? Заметим, что по таблице истинности единицы стоят только на аргументах с i^2 включительно и до $(i + 1)^2$ не включительно. Всего выходит $(i + 1)^2 – i^2 = 2i + 1$, это значения будет меньше чем количество всех аргументов x0…x7, а их 256, поэтому имеет смысл построить ДНФ. Будем строить СДНФ и при

возможности объединять конъюнкты. Если будет 2^t конъюнктов, между всеми которыми есть общие 8 - t литералов, то тогда можно оставить конъюнкт из этих общих литералов вместо всех 2^t конъюнктов. Это возможно из того соображения, что мы не зависим от t оставшихся литералов, так как все их комбинации перебраны (их всего 2^t и никакие два из конъюнктов в СДНФ не совпадают). Тогда мы получим ашки, как на рисунке 10.

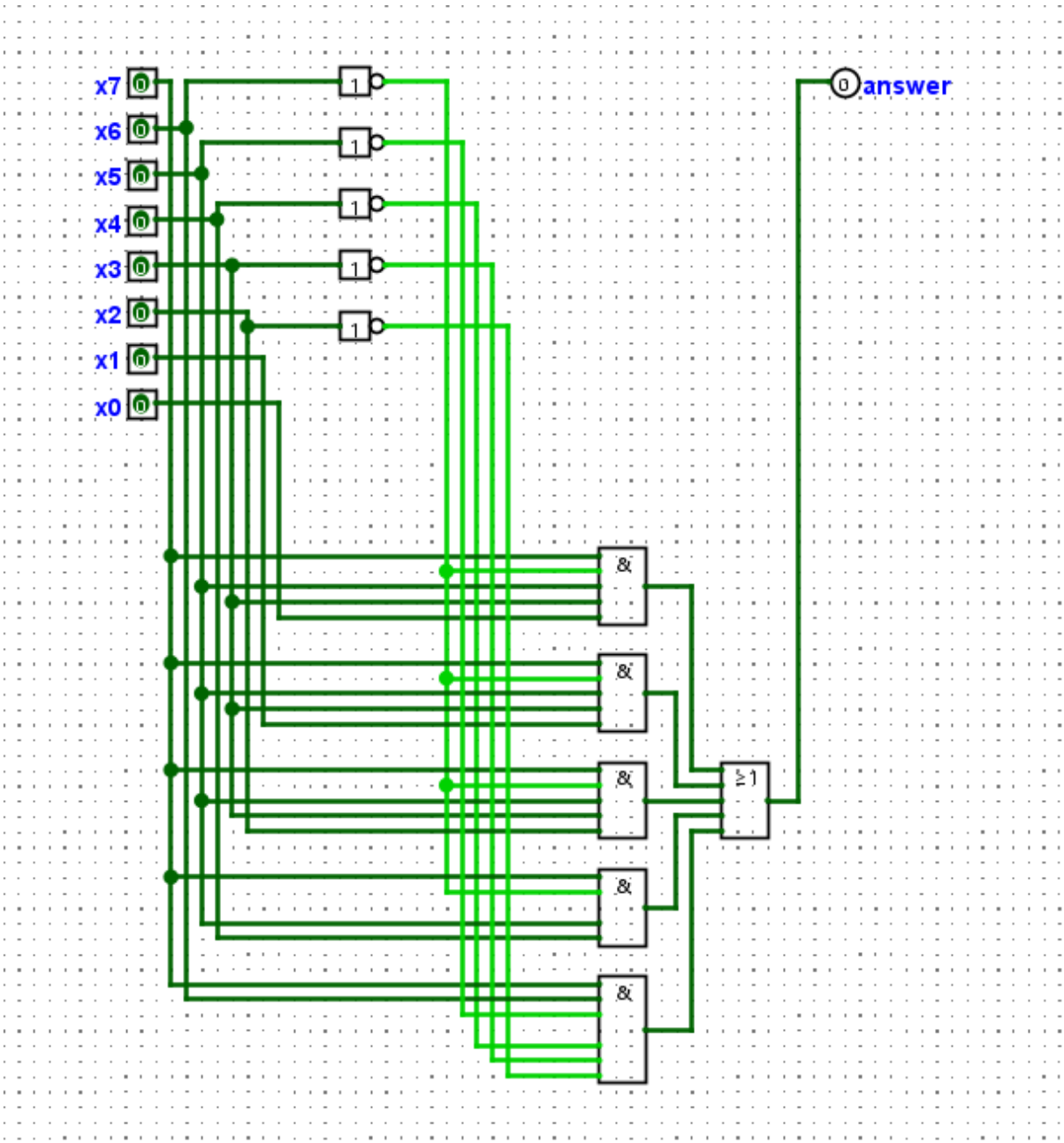

Рисунок 10 – Пример схемы a13

Осталось только сделать соответствие a_i => битовое представление i. Сделать это можно следующим образом. Сделаем четыре дизъюнкции от 8 переменных. Результат каждой дизъюнкции есть один из нужных битов. Условимся, что сверху старший бит, а снизу младший. Но так, как только одна из ашек равняется 1, то битовое представление этой ашки и показывает то, к каким дизъюнкциям мы должны его подвести. Например, a12 (1100) нужно подвести к 4 и 3 биту (4 и 3 биты – самые старшие). Таким образом, получим схему как на рисунке 11. К сожалению, схема вышла большой, поэтому полностью здесь её уместить не вышло.

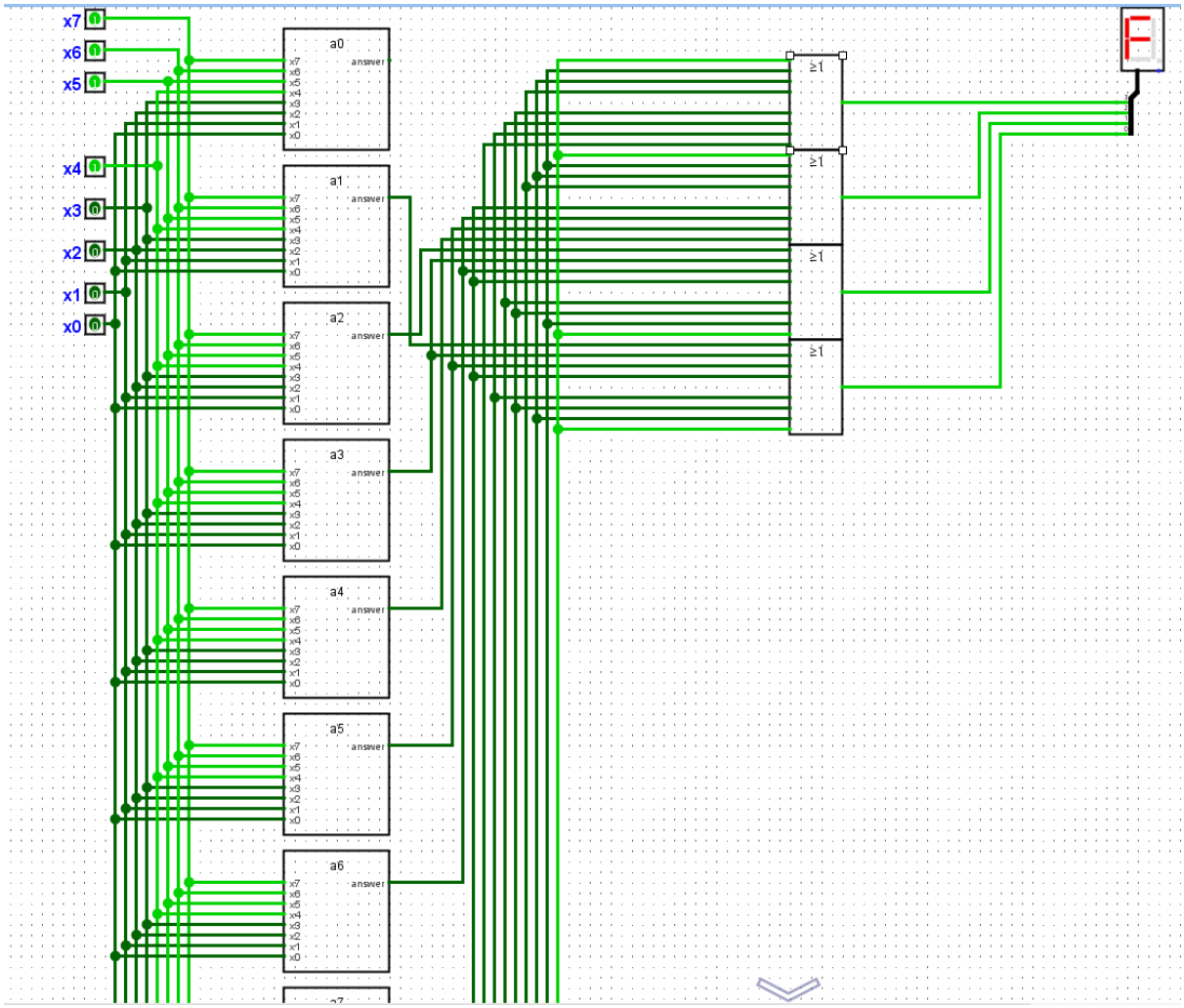

Рисунок 11 – Часть конечной схемы взятия корня