САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Архитектура ЭВМ

Отчет

по домашней работе № 2

Построение сложных логических схем

Выполнил: Левицкий Иван Михайлович

Номер ИСУ: 334916

студ. гр. M3135

Санкт-Петербург

2021

Цель работы: моделирование сложных логических схем

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

Теоретическая часть

Счетчик - устройство для подсчета числа входных импульсов, в простейшем виде представляющее собой несколько связанных между собой триггеров и элемент вывода (индикатор значения счетчика).

Счетчики бывают разных типов: суммирующие, вычитающие, реверсивные (по направлению счета); с последовательным или же параллельным переносом (по способу построения цепи переноса); синхронные и асинхронные (по способу переключения триггеров).

Разберем один отдельный тип, а именно асинхронный суммирующий счетчик с последовательным переносом. Объясним каждое слово из его названия: “асинхронный с последовательным переносом” - значит то, что триггеры в нем соединены последовательно и соответственно переключаются последовательно; “суммирующий” - значит то, что он пробегает значения модуля счета слева - направо (по их возрастанию))

Если говорить строже, то асинхронный суммирующий счетчик с последовательным переносом - это последовательностная схема, преобразующая поступающие на вход импульсы в код Q по формуле $Q = (d + i)\, mod\, n$,где d- начальное значение счетчика (в рамках

диапазона модуля счета), i - количество (сумма) поданных импульсов на данный момент, n - модуль счета.

То есть такой счетчик при включенном генераторе импульсов просто должен выводить циклично значения: $d,\ (d+1),\ (d+2),\ \dots,\ (n-1),\ 0,\ 1,\ \dots,\ (d-1),\ d$ и т. д.

Внутренняя же реализация такого счетчика, очевидно, может быть устроена по разному, например с помощью последовательно включенных JK-триггеров (как конкретно - будет изложено ниже, в практической части).

Практическая часть

Вариант №1

1. Асинхронный суммирующий счетчик с последовательным переносом по модулю 10

Будем реализовывать наш счетчик с помощью JK-триггеров (с доп.входом инициализации нулем, т.к. изначально в logisim evolution триггеры не проинициализированы, а также нам потребуется возможность сброса триггера в 0, т.к. наш модуль счета не степень двойки), т.к. этот вид триггера удобен тем, что при $J = K = 1$ он инвертирует свое состояние. Для этого сначала реализуем RS-триггер (см. рис 1), являющийся компонентом JK-триггера. Аналогично реализуем в нем доп. вход инициализации нулем (для опять же самой инициализации и наших задач) и для удобства доп.вход инициализации единицей; сделаем это с помощью подведения к конечным элементам “или”, выдающим Q и инверсное Q ,

соответственно reset_0 и reset_1 (см.рис 1).

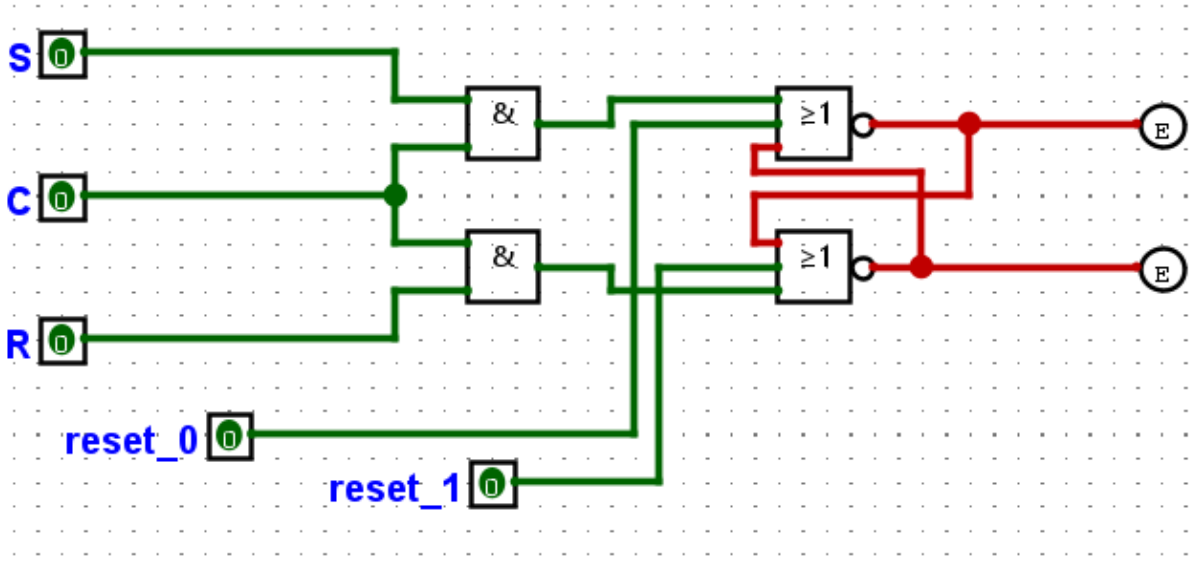

Рисунок 1 – RS-триггер (с внешней инициализацией 0 и 1)

Далее на основе нашего RS-триггера реализуем наш JK-триггер (см.рис 2).

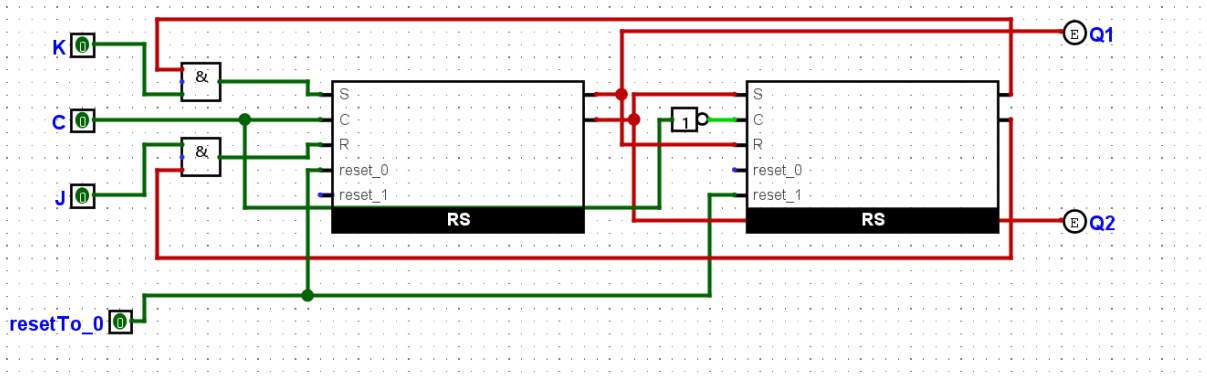

Рисунок 2 – JK-триггер (с внешней инициализацией 0)

Он устроен как классический JK-триггер с добавлением входа resetTo_0, инициализирующего его 0 с помощью реализации на рис. 2 (зануляем RS, выводящий ответ, и инициализируем 1 второй RS, чтобы предотвратить возможные ошибки)

На основе такого JK-триггера и создадим наш счетчик. Основная идея состоит в том, что у нас есть 4 JK-триггера, каждый отвечающий за соответствующий бит выводящегося на индикаторе числа, и к J и K входам

везде подведена константная единица (то есть при включении синхронизации триггер всегда инвертирует хранящее в себе значение), остается лишь так подвести к ним синхронизацию, чтобы у триггера, отвечающего за 0-ой бит числа, она менялась каждый такт, у триггера, отвечающего за 2-ой бит, она менялась каждые 2 такта и т.д. Таким образом мы, очевидно, последовательно пробежим все значения $Q = 0000$ $Q = 0001$, ..., $Q = 1111$, а нам же нужно, чтобы после значения $Q = 9 = 1001$значение счетчика стало равным 0, т.к. счетчик по модулю 10, а не 16, - значит в нашем счетчике еще должна быть логическая схема, искусственно зануляющая все триггеры, несмотря на их синхронизацию, в момент $Q = 10 = 1010$. Сделаем такую схему (см. рис 3)

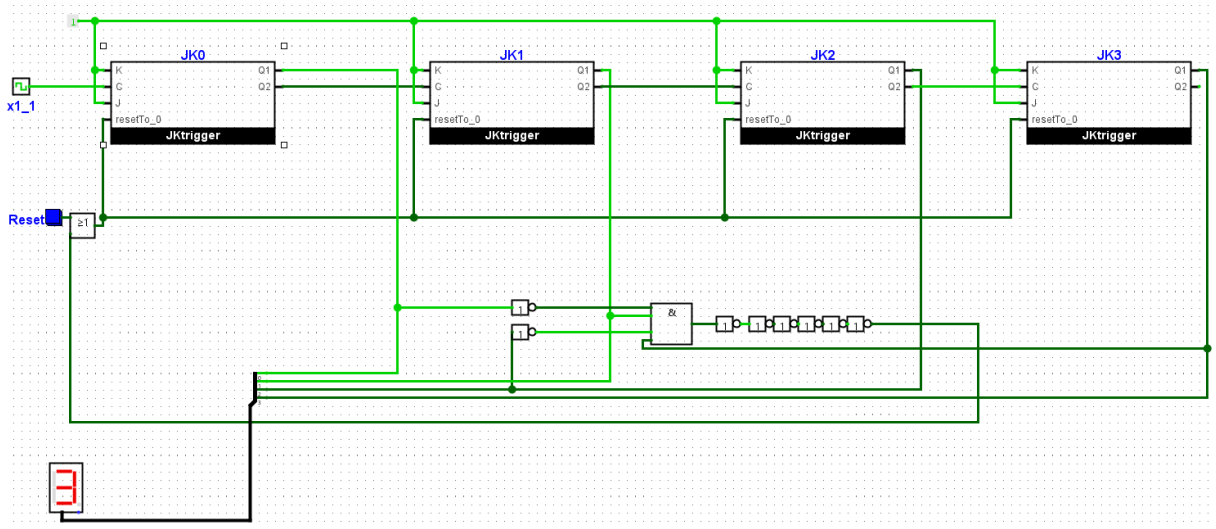

Рисунок 3 – Counter (on mod 10)

(см. рис 3) Подключим ко всем J и K константную единицу и подадим на вход синхронизации (C) каждого последующего JK-триггера инверсный выход предыдущего (а на JK0, отвечающий за 0-ой бит, подадим синхронизацию напрямую из генератора x1_1), заметим, что тогда мы добьемся ровно того, что хотели, т.к. значения JK0 будут меняться каждый такт, из чего следует то, что значения JK1 будут меняться каждые два такта (только при смене синхронизации 0 на 1, то есть, пропуская моменты смены 1 на 0) и т.д. значения JK4 будут меняться каждые 8 тактов (по тем же причинам). Логика инициализации, заключена в кнопке Reset (нажимающейся перед работой счетчика и подающей сигнал на входы обнуления (ResetTo_0) каждого из триггеров), а логика обнуления к моменту $Q = 10 = 1010$, заключена в элементе “и”, к которому

подведены выходы или их инверсии со всех триггеров и который выдает 1 только один раз (как раз в момент $Q = 10$); обратим внимание на стоящее после этого элемента “и” четное количество элементов “не”, они не выполняют в силу своей четности никакой логической нагрузки, но замедляют поступление 1 к обнулению JK-триггеров, что предотвращает ошибку одновременного прихода сигналов и в инверсию триггеров, и в их обнуление в один такт в момент $Q = 10$ (без этой модификации обнуление на некоторых триггерах не происходит). Для удобства логику инициализации и обнуления к моменту $Q = 10 = 1010$ несколько совместим с помощью элемента “или” и уже от него подведем общие провода к ResetTo_0. Вывод значений Q1 с каждого из триггеров JK0, JK1, JK2, JK3 подведем к четверному разветвителю (соответственно ко входам отвечающим за нулевой бит, первый бит, второй бит и третий бит), а с него подведем провод к шестнадцатеричному индикатору. Таким образом, исходя из своего устройства наш счетчик работает следующим образом: после нажатия кнопки Reset он включается (триггеры инициализируется), далее каждый такт генератора счетчик увеличивается на 1 (показание на индикаторе), и после значения 9 счетчик переходит в значение 0 и далее зацикливается. То есть наш счетчик работает ровно так, как и должен работать асинхронный суммирующий счетчик с последовательным переносом по модулю 10.

Временная диаграмма (см.рис 4, пояснение: JK0, JK1, JK2, JK3 - соответственно выходы Q1 из JK-триггеров; выходы Q2 из них же не отображены, т.к. это не имеет смысла в силу их инверсности относительно Q1; Reset нажимается просто за некоторое время до запуска генератора тактов x1_1; во время начала 10 такта (за малые доли этого такта если конкретнее) значения на триггерах на малые доли такта все-таки становятся равные ситуации $Q = 10$, но сигнал с логики обнуления доходит быстрее и триггеры не успевают послать индикатору значение 10 и выводят 0; после прохода “обнуления” ситуация, очевидно, просто зацикливается)

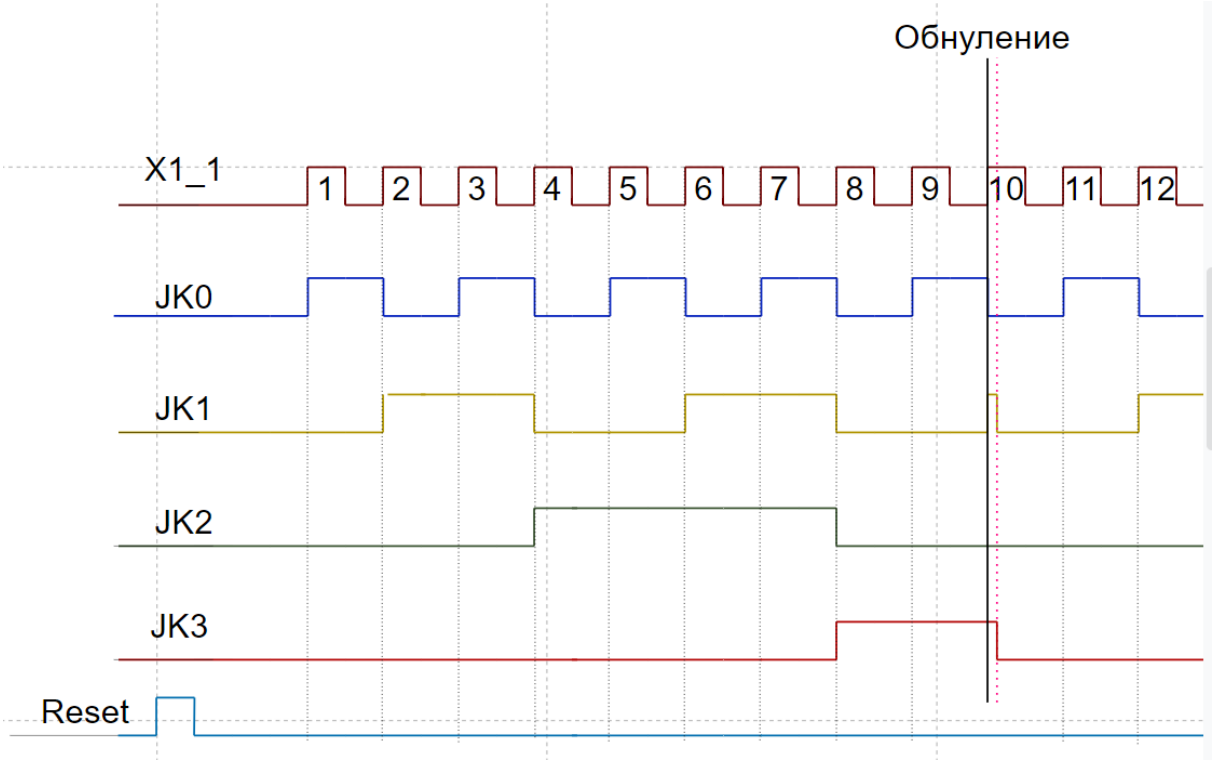

Рисунок 4 – временная диаграмма

2.Квадратный корень.

Для взятия квадратного корня (с округлением вниз) построим некоторую версию вычитателя/сравнителя двух чисел на основе и идее сумматора. Для его построения соответственно построим еще две подсхемы. Первой будет полный сумматор (FullAdder, см. рис 5).

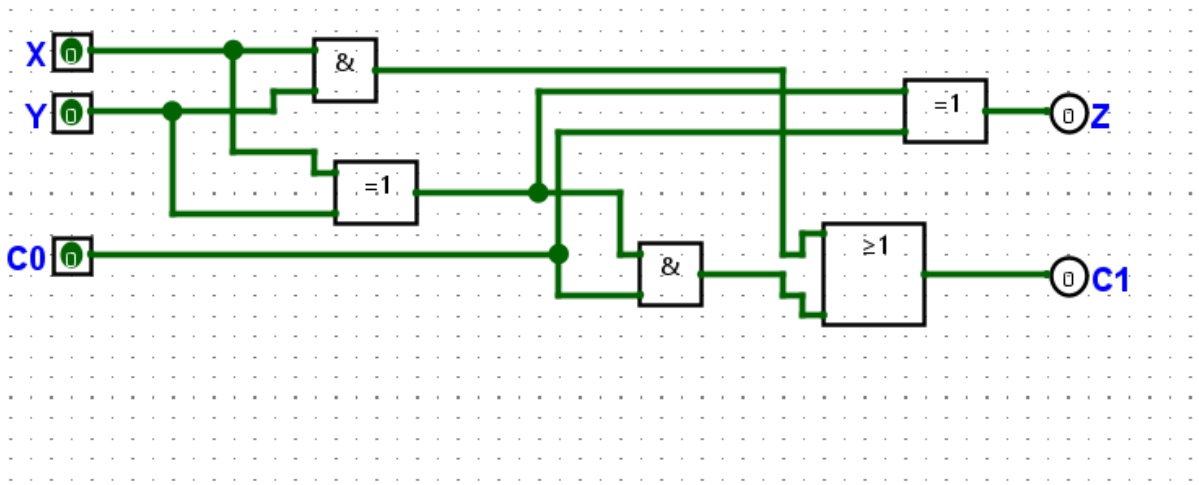

Рисунок 5 – FullAdder (полный сумматор)

X некоторый бит числа x, Y соответствующий бит числа y, C0 - бит предыдущего переноса. Z - получающийся бит разрядности равной разрядности X и Y, C1 - получающийся бит переноса.

Второй доп. подсхемой будет следующий элемент (minicomporator, см.рис 6), его конкретное применение мы рассмотрим в контексте уже самого сумматора, а вообще он просто выводит lastBit (см.рис 6) если highBit = 0 и выводит newBit в обратном случае

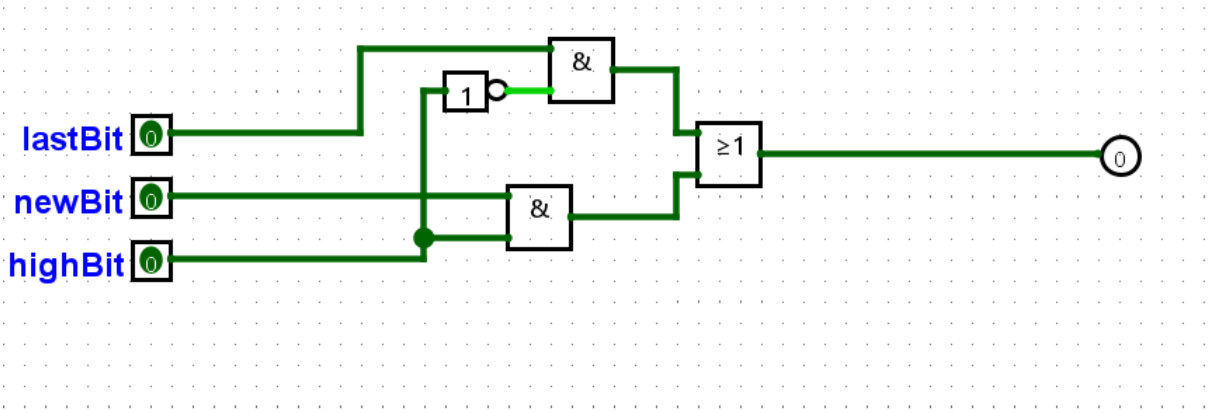

Рисунок 6 – minicomporator

Сам наш вычитатель/сравнитель (DeductorAndComparator) 8-битных чисел будет устроен следующим образом: это будет последовательно (каскадно) соединенные 8 полных сумматоров (см.рис 7) которым на входы битов второго числа будут подаваться инвертированные биты второго числа, на бит входного переноса в первый полный сумматор будет подана константная единица, а выходной бит переноса из последнего сумматора будет выводить доп. 9-ый бит, отвечающий за то, можем ли мы вычесть из первого числа второе. Таким образом мы получим почти адаптированный каскадный вычитатель (вычитатель с функцией сравнителя): если второе число меньше либо равно первого, то он действительно вычтет из первого второе (по сути механически воплощая формулу $a - b = a + (-b) = a + \overline{b} + 1$) и всегда выведет единицу 9-ым битом (это утверждение несложно доказывается по индукции: база для однобитных чисел очевидна, тогда пусть для k-битных чисел наше утверждение верно, тогда докажем для k+1-битных; если старшие биты у них одинаковы то применим предположение индукции, а т.к. у второго числа старший бит инвертируется, а у первого нет, то в последнем

сумматоре будет единичка из переноса по предположению индукции и ровно одна из входных чисел, если же старшие биты не равны, то у X бит единичка а у Y нолик (т.к. X > Y) из чего следует, то что после инверсии бит Y тоже будет 1 а значит и бит переноса будет 1). Если же второе число больше первого, то 9-ый бит будет всегда 0 (доказывается аналогично обратному случаю, описанному выше), и мы хотим, чтобы в этом случае вывелась не разность этих чисел, а осталось прежнее число (то есть X) для чего подведем входные биты (xi), биты получающиеся после вычитания (Z) и 9-ый бит (по факту сравнивающий два числа) в minicomparator-ы и уже из них будем выводить биты окончательного числа (v0, v1, …, v7). Таким образом после ввода 8-битных чисел X и Y в наш DeductorAndComparator мы получим число X - Y (если X >= Y) и получим число X в обратном случае, а также поймем с помощью значения v8, что мы сделали (смогли ли вычесть)

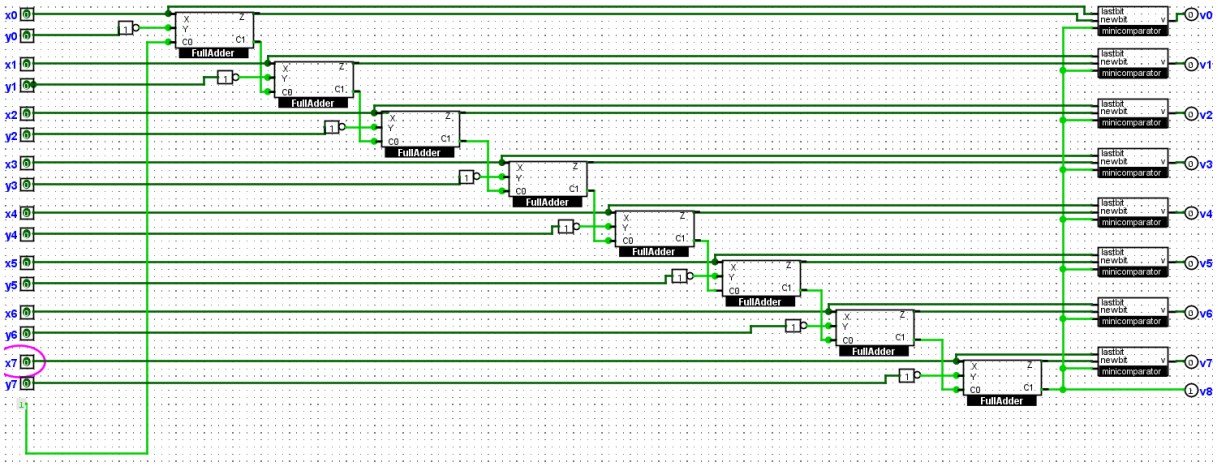

Рисунок 6 – DeductorAndComparator

На основе DeductorAndComparator построим SQRT (схему, вычисляющую собственно корень): по сути просто воплотим в logisim уже изученный на занятии алгоритм взятия корня с округлением вниз. Введем (см. рис 7) число с помощью восьми контактов x0, … x7, обозначающих соответственные биты числа. Заведем 2 старших бита в нужном порядке (обратном, т.к. входы начинаются с меньшего) в первый DeductorAndComparator и попытаемся вычесть подведенную константную единицу. В ходе данного алгоритма получим 9-ым битом либо 0, либо 1, выведем ее старшим битом в итоговый ответ. Далее будем действовать следующим образом, каждый раз будем заводить выводы из предыдущего

по использованию DeductorAndComparator в соответственные разряды X следующего компоратора (со сдвигом на два), а также младшими битами (то есть в старшие входы X) подведем два следующих еще неиспользованных бита входного числа (проводя параллели с алгоритмом в столбик, “снесем следующую пару чисел”) , а в Y запишем уже найденное число умноженное на 4 и увеличенное затем на 1 (с помощью константы 1 в младший бит Y), затем DeductorAndComparator сравнит эти числа и если можно вычтет, а 9-ый бит запишет как ответ в следующий незанятый бит 4-битного итогового числа (корень из 8 бит, очевидно, 4-битный). Подводя итог мы просто четырьмя итерациями (четырьмя DeductorAndComparator) воплотим уже изученный алгоритм взятия корня.

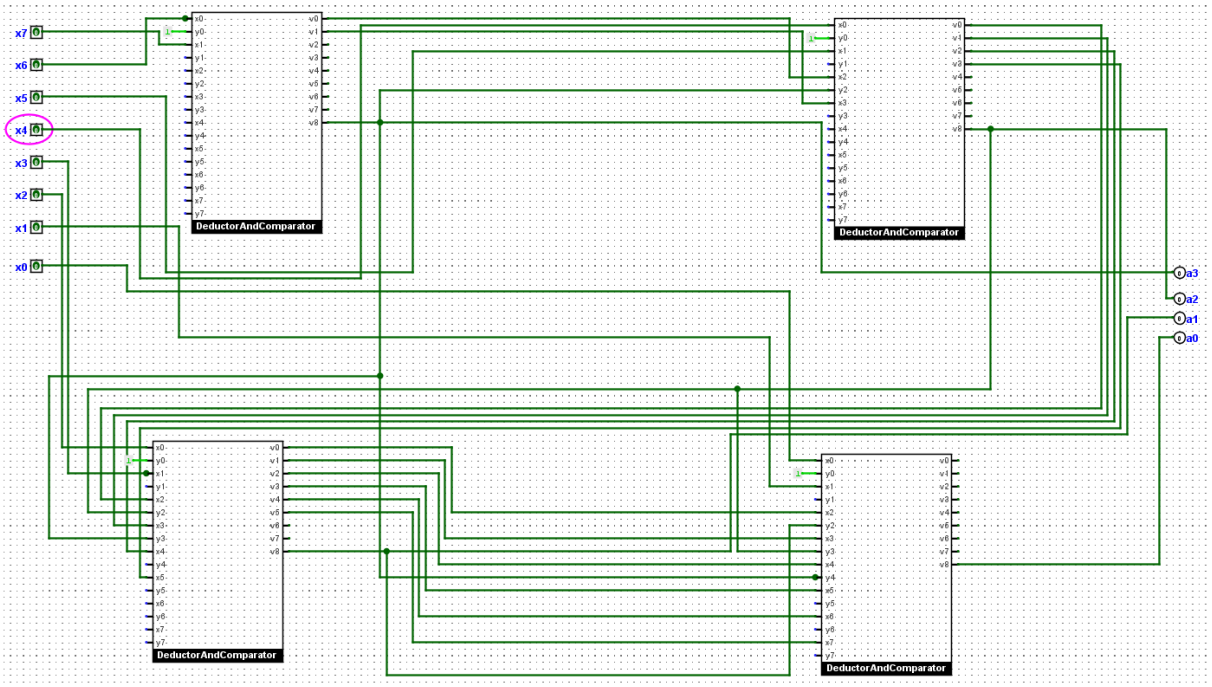

Рисунок 6 – SQRT (для 8 битного числа с округлением вниз)