Лабораторная работа №1	М3139	2022
Построение логических схем в среде моделирования	Матвеев Андрей Денисович	

Цель работы: моделирование логических схем на элементах с памятью.

Инструментарий и требования к работе: работа выполняется в среде моделирования Logisim evolution.

Описание

Составить и описать принцип работы двух схем: счётчика и регистра сдвига с линейной обратной связью.

Вариант 76

1) Синхронный вычитающий счетчик с модулем счета 24. Название подсхемы – main.

Перед тем как рассказывать про счётчик, расскажу про JK-Trigger, так как я им пользовался при построении счетчика. JK-Trigger умеет запоминать значение, менять его на 1, на 0, инвертировать и выдавать его на выход. Приведу таблицу истинности (рисунок ниже).

C	K	J	Q(t)	Q(t+1)	Пояснения
0	x	x	0	0	Режим хранения информации
0	x	x	1	1	
1	0	0	0	0	Режим хранения информации
1	0	0	1	1	
1	0	1	0	1	Режим установки единицы J=1
1	0	1	1	1	
1	1	0	0	0	Режим записи нуля K=1
1	1	0	1	0	
1	1	1	0	1	K=J=1 счетный режим триггера
1	1	1	1	0	

Рисунок №1 – Таблица истинности JK-Trigger’a

Здесь не хватает входов R и S. В моём случае вход R устанавливает триггер в 0 в независимости от синхронизации, S делает то же самое, но триггер устанавливается в 1.

Вычитающий счетчик вычитает из числа единицу, если подать сигнал на вход “T”. Также есть вход “C”, отвечающий за синхронизацию, т. е., если C=0, то сигнал со входа “T” никак не повлияет на счетчик. Еще есть вход “R” который ставит счетчик в начальное положение – 24. На “R” надо подать короткий сигнал, перед тем как пользоваться счетчиком. Модуль счетчика – количество импульсов, после которых счетчик придет в начальное положение. В моем случае – это 24, поэтому я сделал счетчик, который считает в диапазоне [1;24], т.е. когда мы в состоянии “1” и мы подаем сигнал на вход “T”, счетчик принимает значение 24 (зацикливается). На выходе у нас число – текущее состояние счетчика из 5-ти битов d0, d1, d2, d3, d4 (d_i – i-ый бит числа), нумерация идёт от младшего бита к старшему.

Я реализовал счетчик с помощью 5-ти JK-Trigger’ов. На вход J и K будем всегда подавать константу 1, тогда JK-Trigger будет работать как Инвертор, т. е. Инвертировать своё, запомненное на предыдущем шаге, значение. А на вход C, будем подавать выход предыдущего JK-Trigger’a, как на рисунке. Если же JK-Trigger отвечает за младший бит (он стоит самый первый в схеме), тогда подаём ему на вход сигнал контакта “T”.

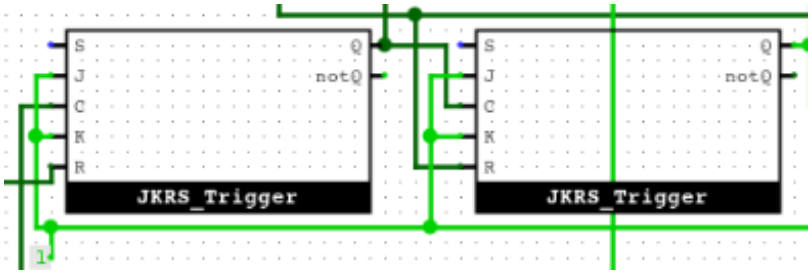

Рисунок №2 – Принцип работы счётчика

Как работает счётчик. Для того, чтобы понять всю схему, достаточно посмотреть, как соединяются первые два триггера (рисунок сверху), остальные соединяются также. Убавление на единицу – в двоичной записи то же самое, что и замена максимального префикса из нулей, на единицы и замена единицы, стоящей сразу за этими нулями, на ноль. Моя схема так же реализована. Весь префикс JK_Trigger’ов из нулей заменится единицами. Первый триггер заменится на единичку, потому что мы подадим T=1, второй заменится, потому что первый стал единичкой, третий заменится, потому что второй стал единичкой, и т. д. JK-Trigger, стоящий сразу за префиксом триггеров, равных нулю, станет нулём, т. к. на него подастся сигнал, и он инвертируется. А дальше, этот JK-Trigger не подаст сигнал, т. к. на выходе он даёт 0 (.

В схеме реализована система сброса (рисунок снизу), когда все биты равны нулю, то схема возвращается в изначальное состояние - 24.

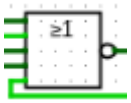

Рисунок №3 – Сброс счётчика

Все выходы ведут в ИЛИ-НЕ Элемент. Т. е. когда все выходы равны нулю, то ИЛИ-НЕ выдаёт на выходе “1”. Этот провод идет во вход “R” (JK-Trigger'ов) для битов d0, d1, d2 и во вход “S” для битов d3, d4. Потому что 24 в двоичной записи = 11000, поэтому “зануляются” первые три бита и устанавливаются единички для оставшихся битов.

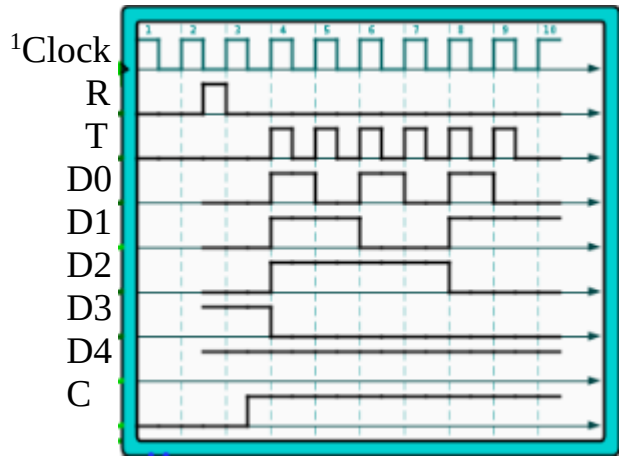

Рисунок №4 - Временная диаграмма счетчика

Окончательная схема счётчика

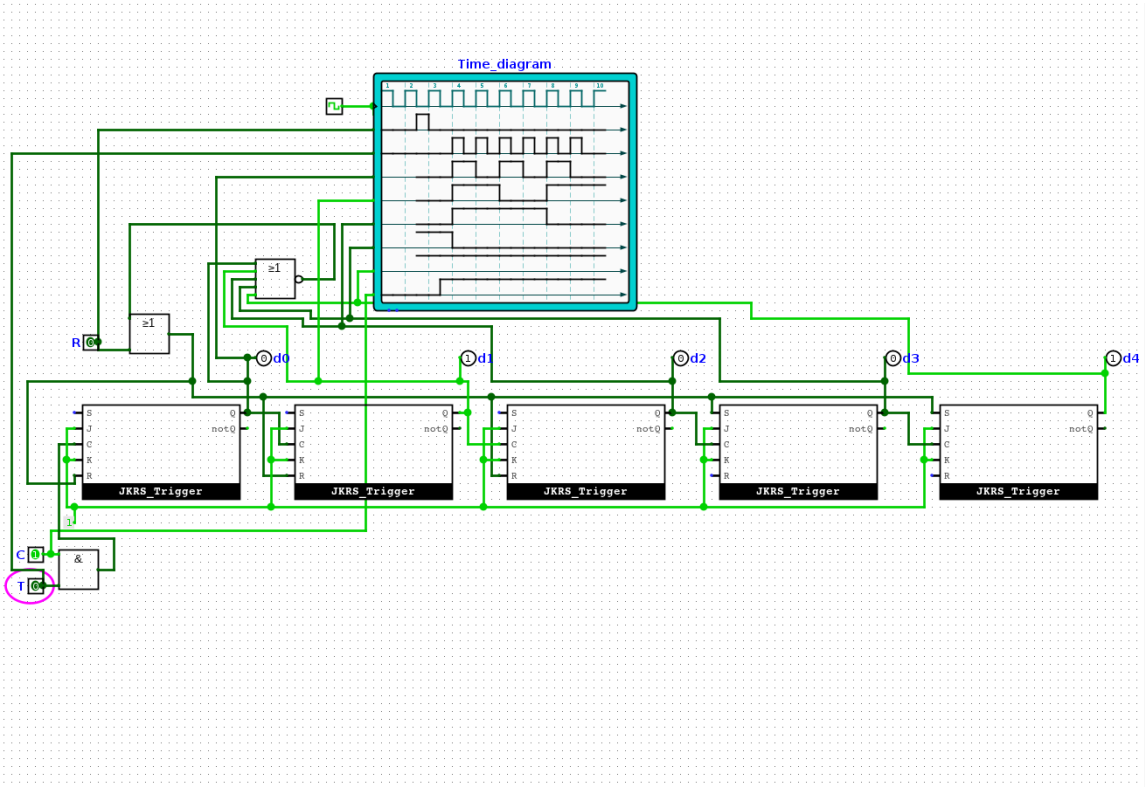

Рисунок №5 – Вычитающий синхронный счётчик

2) Регистр сдвига с линейной обратной связью. Конфигурация Галуа: (6, 5, 3, 2). Название подсхемы – lfsr_galois_myD

Для начала опишу принцип работы D-Trigger'а. D-Trigger запоминает в себе значение и может его либо изменить на 0, либо изменить на 1. На выходе триггер выдаёт запомненное значение. Приведу таблицу истинности (рисунок ниже).

Такт n		Такт $n+1$
C	D	Q^{n+1}
0	0	Q
0	1	Q
1	0	0
1	1	1

Рисунок №6 – Таблица истинности D-Trigger'а

РСЛОС хранит в себе последовательность из нулей и единиц и на каждый такт проводит с ней определённую операцию. В моём случае последовательность менялась по такому правилу: $S_0 = S_{n-1},$; $S_i = S_{i-1} \oplus S_{n-1}$, если $i \in \{7, 6, 4, 3\}$, иначе $S_i = S_{i-1}$. Именно так и работает

конфигурация Галуа: мы двигаем биты циклически вправо, но некоторые из них попутно “ксорим” с выходным битом. Моя конфигурация – (6, 5, 3, 2) – поэтому биты с этими индексами должны “ксориться” с выходным битом, перед тем как записаться в следующий бит.

На вход подаётся два контакта “S” и “T”. “S” – устанавливает схему в начальное положение (все биты равны единице), переключение контакта “T” делает один шаг.

Я реализовал РСЛОС с помощью D-Trigger’ов, каждый D-Trigger отвечает за свой бит. D-Trigger’ы соединены между собой как на рисунке.

Здесь как раз и происходит “проталкивание” бита из прошлого D-Trigger’а в следующий (рисунок ниже).

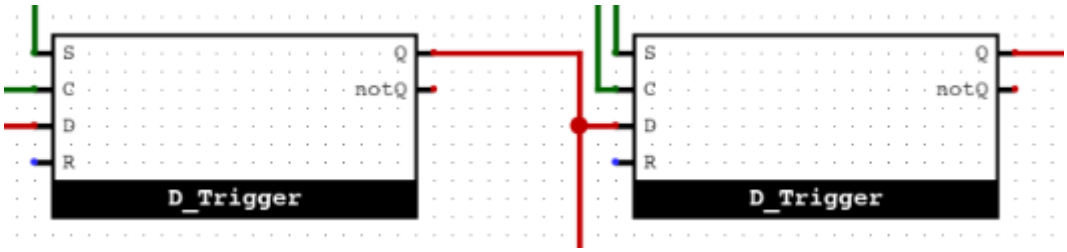

Рисунок №7 – Связка D-Trigger’ов в РСЛОС

Если же надо “заксорить” с выходным битом, то просто ставим элемент ИСКЛЮЧАЮЩЕЕ-ИЛИ (в который идёт выходной бит), а потом уже ведём провод в следующий D-Trigger.

Вход “C” (синхронизация) у D-trigger’ов нужен, чтобы делать шаг. Вход “T” подключается ко всем входам “C” у D-Trigger’ов, поэтому, когда T=1, то все D-Trigger’ы активируются, и происходит один шаг РСЛОСа.

Окончательная схема РСЛОСа

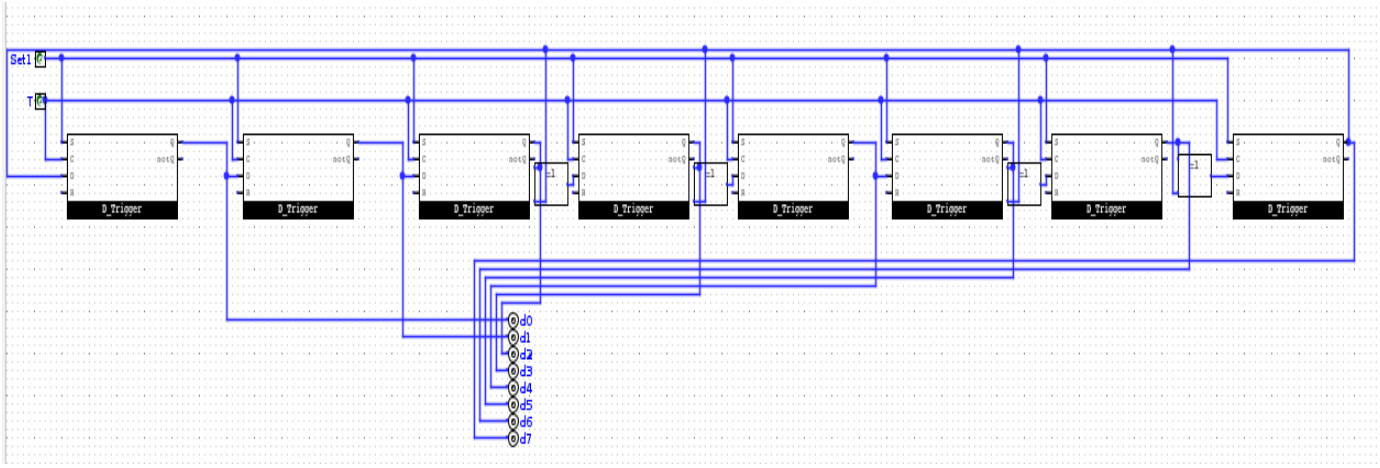

Рисунок №8 - РСЛОС