Приложение B. Линейный сдвиговый регистр с обратной связью

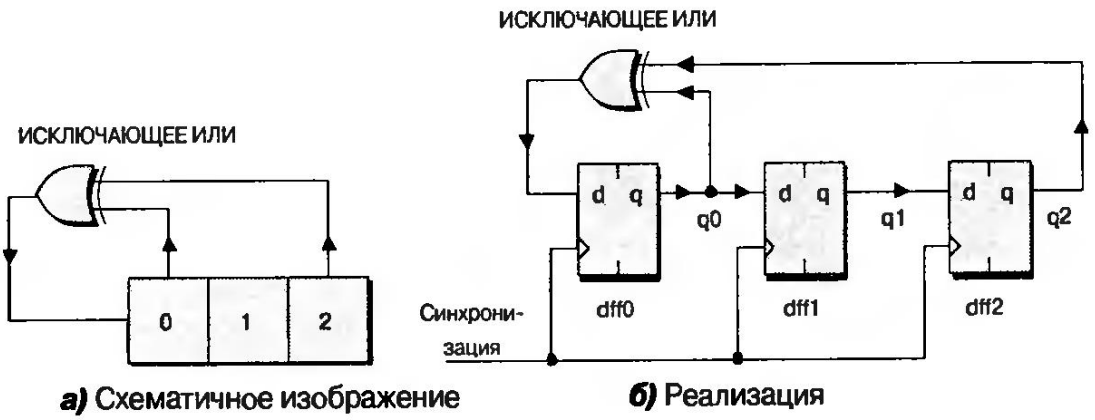
Содержимое этого приложения взято из моей книги «Bebop to the Boolean Boogie (An Unconventional Guild to Electronics, Edition 2)» (ISBN 0-7506-7543-8) с любезного разрешения издателя.

Ауроборос

Ауроборос — это мифологический змей или дракон, пожирающий свой хвост, поэтому его тело образует замкнутый круг. Этот символ использовался в различных древних мировых культурах для обозначения бесконечности или возрождения[1]. В мире электроники прототипом аурооборосa можно назвать *линейный сдвиговый регистр с обратной связью*, в котором выходные значения из стандартного сдвигового регистра определённым образом обрабатываются и подаются обратно на его вход таким образом, чтобы устройство бесконечно вырабатывало последовательность определённых данных.

Реализация вида «многие к одному»

Линейный сдвиговый регистр с обратной связью (или LFSR — *Linear Feedback Shift Registers*) достаточно просто реализуется и полезен во многих приложениях. В общем случае LFSR формируется из простого сдвигового регистра с добавлением обратной связи, подключенной к двум или более точкам в регистровой цепочке, которые называют *отводами* (**Рис. B.1**).



а) Схематичное изображение **б)** Реализация

Рис. B.1. LFSR с элементом ИСКЛЮЧАЮЩЕЕ ИЛИ в цепи обратной связи

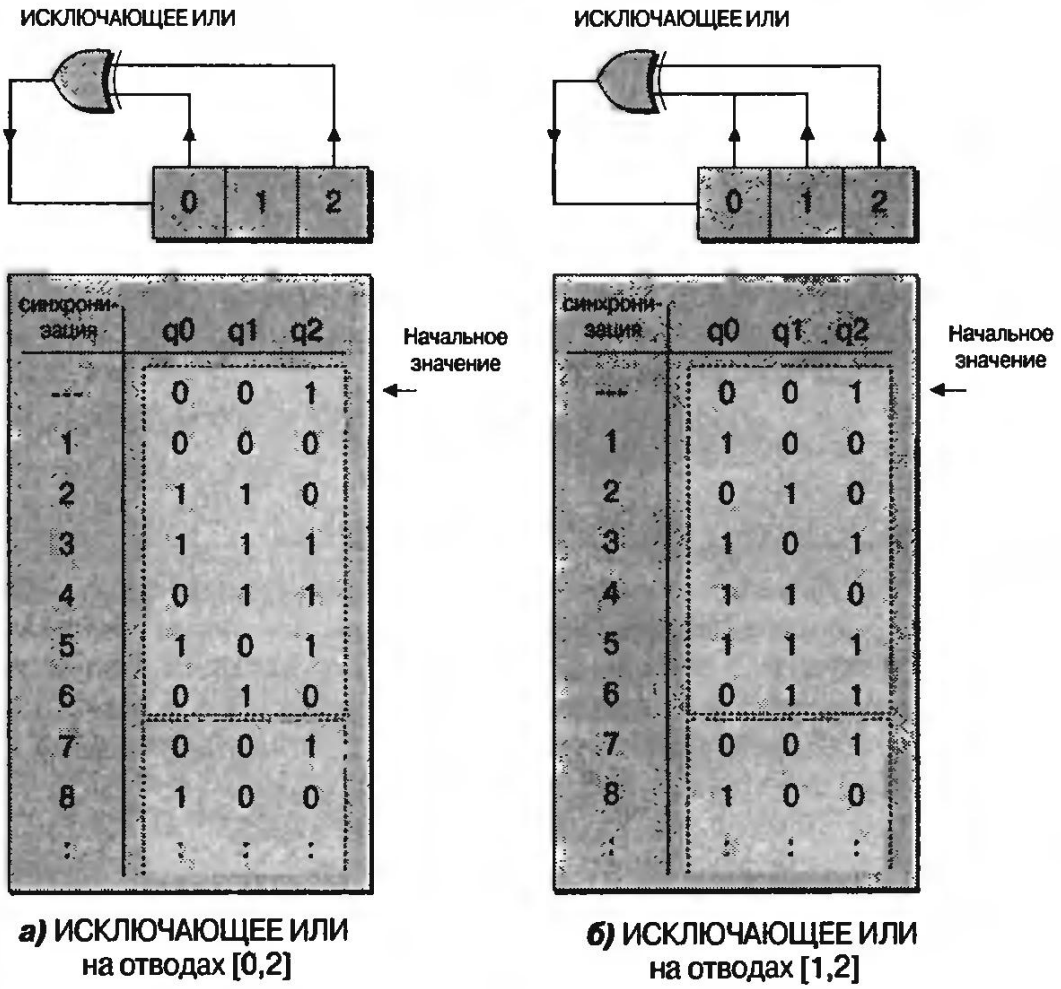
В этом примере в качестве отводов используются биты 0 и 2, и простым способом их представления может служить запись вида [0,2]. Все элементы регистра используют общий сигнал синхронизации, который опущен на схеме регистра для простоты. Данные, поступающие на вход LFSR, формируются логической функцией ИСКЛЮЧАЮЩЕЕ ИЛИ (либо ИСКЛЮЧАЮЩЕЕ ИЛИ-НЕ), подключенной к от-

[1] Не следует путать с двуглавым змеем Амбисфеном из классической мифологии, у которого на хвосте росла вторая голова, благодаря чему он мог передвигаться в обоих направлениях.

1985 г. *Впервые для хранения компьютерных данных использован CD-ROM.*

водам в цепочке регистров, а остальные биты регистра формируются как в обычном сдвиговом регистре.

Последовательность формируемых регистром (LFSR) значений определяется функцией обратной связи (ИСКЛЮЧАЮЩЕЕ ИЛИ либо ИСКЛЮЧАЮЩЕЕ ИЛИ-НЕ), а также расположением отводов в цепочке регистров. Например, рассмотрим два 3-битных регистра, использующих в цепи обратной связи функцию ИСКЛЮЧАЮЩЕЕ ИЛИ, у первого регистра отводы сформированы по схеме [0,2], а у второго — [1,2] (**Рис. В.2**).



а) ИСКЛЮЧАЮЩЕЕ ИЛИ на отводах [0,2]

б) ИСКЛЮЧАЮЩЕЕ ИЛИ на отводах [1,2]

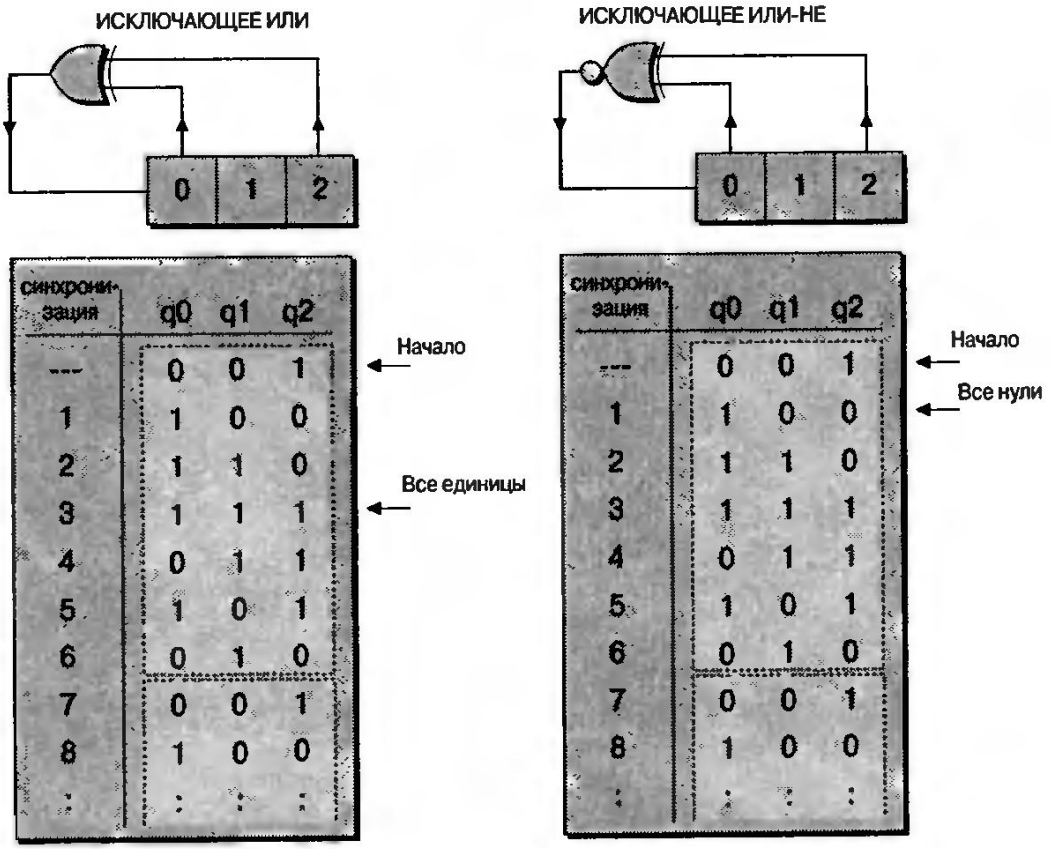
Рис. В.2. Сравнение различных схем реализации отводов

Оба регистра начинают работу с одного и того же начального значения, но с разными отводами для цепи обратной связи. Из-за этого формируемые ими значения довольно быстро, с каждым новым тактом синхронизации, начинают расходиться. В некоторых случаях LFSR перестает формировать циклическую последовательность значений через определённое количество шагов. Однако оба регистра, представленные на **Рис. В.2**, будут формировать *максимально длинную последовательность*, которая будет включать в себя все возможные значения (исключая значения со всеми нулевыми битами) перед тем, как вернуться к начальному положению.

Двоичное поле, состоящее из n бит, может содержать $2n$ уникальных значений, но максимальная длина формируемой LFSR-регистром последовательности будет составлять всего лишь $(2n - 1)$ чисел. Например, 3-битное поле может содержать одно из $2^3 = 8$ значений, но 3-битный LFSR-регистр из **Рис. В.2** поддерживает только $(2^3 - 1) = 7$ чисел. Объяснить это можно тем, что LFSR с функцией ИСКЛЮЧАЮЩЕЕ ИЛИ в цепи обратной связи не поддерживает «запрещённую»

комбинацию, состоящую из последовательности всех нулевых битов, также как и регистр с функцией ИСКЛЮЧАЮЩЕЕ ИЛИ-НЕ не работает с комбинацией со всеми единицами (**Рис. В.3**)[1].

1989 г. *Начал работу волоконно-оптический кабель, проложенный через Тихий океан (поддерживающий до 40000 телефонных разговоров одновременно).*



а) ИСКЛЮЧАЮЩЕЕ ИЛИ на отводах [0,2]

б) ИСКЛЮЧАЮЩЕЕ ИЛИ-НЕ на отводах [0,2]

Рис. В.3. Сравнение функций ИСКЛЮЧАЮЩЕЕ ИЛИ и ИСКЛЮЧАЮЩЕЕ ИЛИ-НЕ в цепи обратной связи

Множество отводов

Каждый LFSR-регистр поддерживает ряд комбинаций отводов, которые обеспечивают формирование последовательности максимальной длины. Проблема заключается в том, чтобы из всех возможных комбинаций отводов исключить те из них, которые позволяют генерировать только ограниченное количество состояний.

Только ради собственного развлечения я написал простую программу на языке C для определения отводов, позволяющих генерировать последовательность максимальной длины для регистров от 2 до 32 бит. Для того чтобы порадовать вас, результаты работы программы представлены на **Рис. В.4** (символом «*» обозначены последовательности, длина которых является простым числом).

Наборы отводов одинаково подходят как для регистров с функцией ИСКЛЮЧАЮЩЕЕ ИЛИ, так и для функции ИСКЛЮЧАЮЩЕЕ ИЛИ-НЕ, хотя последовательности, формируемые этими регистрами, разумеется, будут различны. Как уже говорилось, другие комбинации отводов также могут позволять генерировать последовательность максимальной длины, хотя и в этом случае их вид будет отличаться. На-

[1] Если LFSR обнаружит у себя «запрещённую величину», то он остановится на этом значении, и будет находиться в таком состоянии до тех пор, пока какое-либо внешнее воздействие не выведет его из этого затруднительного состояния.

1990 г. *Швейцария. Британский физик Тим Бернерс-Ли (Tim Berners-Lee) установил первый в мире WWW-сервер.*

1993 г. *Разработан Web-браузер MOSAIC.*

Кол-во бит	Длина последовательности	Отводы
2	3	[0,1]
3	7	[0,2]
4	15	[0,3]
5	31	[1,4]
6	63	[0,5]
7	127	[0,6]
8	255	[1,2,3,7]
9	511	[3,8]
10	1 023	[2,9]
11	2 047	[1,10]
12	4 095	[0,3,5,11]
13	8 191	[0,2,3,12]
14	16 383	[0,2,4,13]
15	32 767	[0,14]
16	65 535	[1,2,4,15]
17	131 071	[2,16]
18	262 143	[6,17]
19	524 287	[0,1,4,18]
20	1 048 575	[2,19]
21	2 097 151	[1,20]
22	4 194 303	[0,21]
23	8 388 607	[4,22]
24	16 777 215	[0,2,3,23]
25	33 554 431	[2,24]
26	67 108 863	[0,1,5,25]
27	134 217 727	[0,1,4,28]
28	268 435 455	[2,27]
29	536 870 911	[1,28]
30	1 073 741 823	[0,3,5,29]
31	2 147 483 647	[2,30]
32	4 294 967 295	[1,5,6,31]

Рис. В.4. Отводы для формирования последовательностей максимальной длины для LFSR от 2 до 32 бит

пример, для 10-битного LFSR-регистра существуют две 2-отводные комбинации ([2,9] и [6,9]), которые позволяют генерировать последовательность максимальной длины. Также существуют двадцать 4-отводных комбинаций, двадцать восемь 6-отводных и десять 8-отводных комбинаций, которые удовлетворяют критерию максимальной длины[1)].

Очень важно! Необходимо заметить, что комбинации отводов, представленных на **Рис. В.4**, могут не самым лучшим образом подходить для решаемых вами задач по некоторым параметрам, например, они могут не быть простыми полиномами или могут не иметь равномерного распределения в случайном пространстве. На рисунке приведены только те данные, которые я выбрал из сгенерированных мной результатов. Если вы будете использовать LFSR для реальных задачах, то книга «Error-Correcting Codes» (авторы W. Wesley Peterson и J. Weldor Jr., издательство «MIT Press») может помочь вам в определении наилучшей комбинации отводов. Также могут быть интересны утилиты проверки с помощью циклического избыточного кода (CRC), рассматриваемых в разделе «Другие необходимые средства и утилиты» в конце 25 главы.

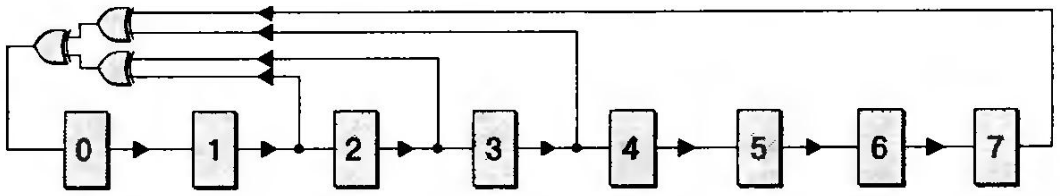
[1)] Более длинная таблица (охватывающая LFSR-регистры длиной до 168 бит) представлена в описании устройства XAPP052 фирмы Xilinx.

Реализация вида «один ко многим»

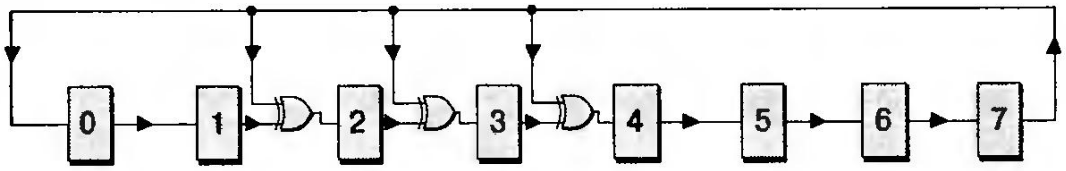
1999 г. *Компания Intel разработала микропроцессор с тактовой частотой 1 ГГц.*

Давайте рассмотрим 8-битный LFSR-регистр, для которого необходимо минимум четыре отвода, чтобы генерировать последовательность максимальной длины. На практике вентиль ИСКЛЮЧАЮЩЕЕ ИЛИ имеет только два входа, поэтому для создания 4-входовой функции необходимо использовать три таких вентиля образующих два уровня логики. Даже в тех случаях, когда LFSR позволяет использовать всего два отвода для формирования максимальной последовательности, по разным причинам может возникнуть необходимость использовать их большее количество, например восемь отводов (для этого потребуется три уровня логики).

Однако увеличение количества логических уровней в цепи обратной связи может отрицательно сказаться на максимальной рабочей частоте LFSR. Решение проблемы заключается в переходе от рассмотренной выше реализации вида «многие к одному» к её аналогам типа «один ко многим» (**Рис. В.5**).



а) «Многие к одному»



б) «Один ко многим»

Рис. В.5. Реализации вида «многие к одному» и «один ко многим»

Традиционная реализация вида «многие к одному» для 8-битного LFSR-регистра использует отводы [1,2,3,7]. Для того чтобы преобразовать это устройство к виду «один ко многим», необходимо наиболее значимый отвод, который всегда будет являться старшим битом (в данном случае 7-й бит), соединить напрямую с самым младшим битом регистра. Также старший бит в индивидуальном порядке через вентили ИСКЛЮЧАЮЩЕЕ ИЛИ подсоединяется к другим отводам (в этом примере это биты [1,2,3]).

Хотя обе реализации будут формировать последовательность максимальной длины, всё же значения этих последовательностей будут различаться. Главным достоинством метода «один ко многим» является то, что в этом случае используется не более одного уровня комбинационной логики в цепи обратной связи независимо от количества задействованных отводов.

Конечно, в ПЛИС на этот счёт есть своя особенность, согласно которой 4-входовая таблица соответствия имеет одинаковое значение за-

держки для 2-, 3- и 4-входовой реализации функции ИСКЛЮЧАЮЩЕЕ ИЛИ. В этом случае метод «многие к одному» даёт определённое преимущество только при использовании LFSR-регистров с числом отводов более четырёх.

Инициализация LFSR

Интересной особенностью LFSR-регистра с логической функцией ИСКЛЮЧАЮЩЕЕ ИЛИ в цепи обратной связи является то, что если он обнаружит у себя внутри значение, состоящее из всех нулей, то он начнёт с удовольствием выдавать эти нули до тех пор, пока кто-нибудь или что-нибудь его не остановит. Так же, регистр с функцией ИСКЛЮЧАЮЩЕЕ ИЛИ-НЕ, обнаружив у себя все единицы, начнёт постоянно выдавать их на выход. Эта особенность может проявить себя при включении питания, когда каждый бит регистра устанавливается случайным образом либо в состояние логического 0, либо в состояние 1, и не исключена вероятность того, что в результате в регистре установиться запрещённая комбинация. Поэтому при использовании LFSR-регистров возникает необходимость в предварительной установке начальных значений.

Интересно заметить, что для регистров с функцией ИСКЛЮЧАЮЩЕЕ ИЛИ-НЕ в цепи обратной связи разрешено использование комбинации, состоящей из всех нулей. Это значит, что для инициализации таких регистров можно использовать сигнал общего сброса, устанавливающий значения всех бит в 0.

Загрузить специфичные данные в регистр можно с помощью его входов установки и сброса (set и reset). Например, один бит регистра может быть подключен к сигналу сброса, а остальные к сигналу установки. При подаче этих управляющих сигналов в LFSR-регистр загрузится аппаратно-определяемое начальное значение. Однако в некоторых приложениях желательно предусмотреть возможность изменения значения инициализации. Решить эту задачу можно, например, с помощью подключения мультиплексора на вход LFSR-регистра (**Рис. В.6**).

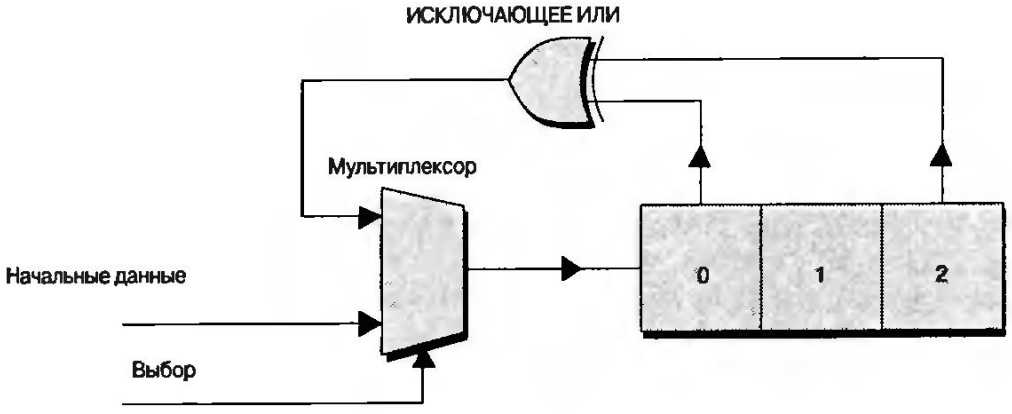


Рис. В.6. Схема загрузки альтернативных начальных значений

Когда на мультиплексор выбран вход для начальной загрузки данных, то устройство работает как обыкновенный сдвиговый регистр и принимает поступающие начальные данные. После загрузки начальных значений мультиплексор переходит в режим передачи данных из цепи обратной связи, и устройство начинает работать уже как LFSR-регистр.

Очереди FIFO

Следует заметить, что формируемые LFSR-регистром необычные последовательности значений находят применения во многих приложениях. Например, давайте рассмотрим 16-словную функцию памяти типа FIFO (*first-in, first-out — первым прибыл, первым вышел*) шириной 4 бита (**Рис. В.7**).

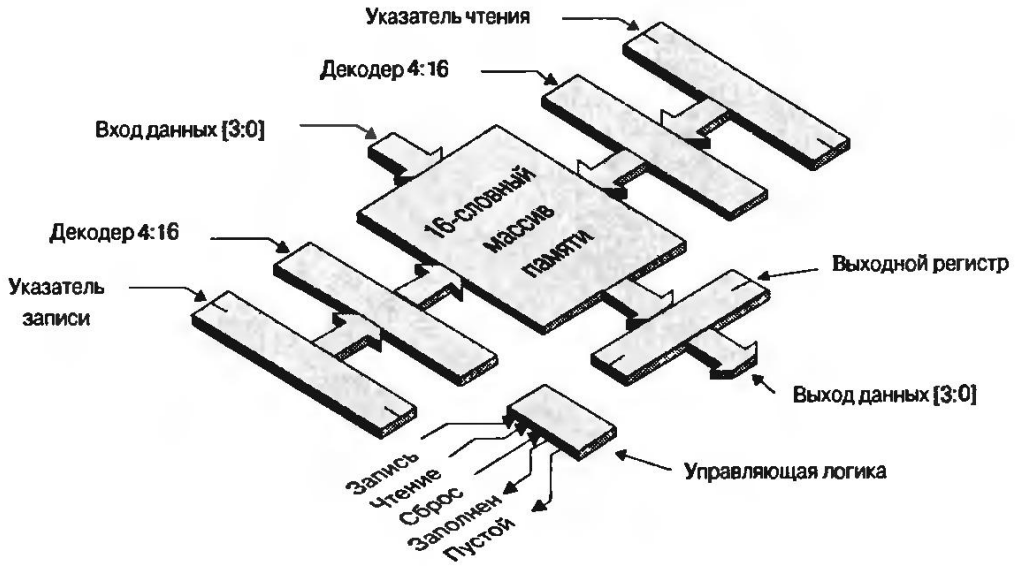


Рис. В.7. FIFO на 16 слов

В дополнение к управляющей логике и выходному регистру, FIFO также содержит указатель чтения и указатель записи, которые представляют собой 4-битные регистры, информация с которых преобразуется дешифратором 4:16 для того, чтобы выбрать 1 из 16 слов, находящихся в массиве памяти.

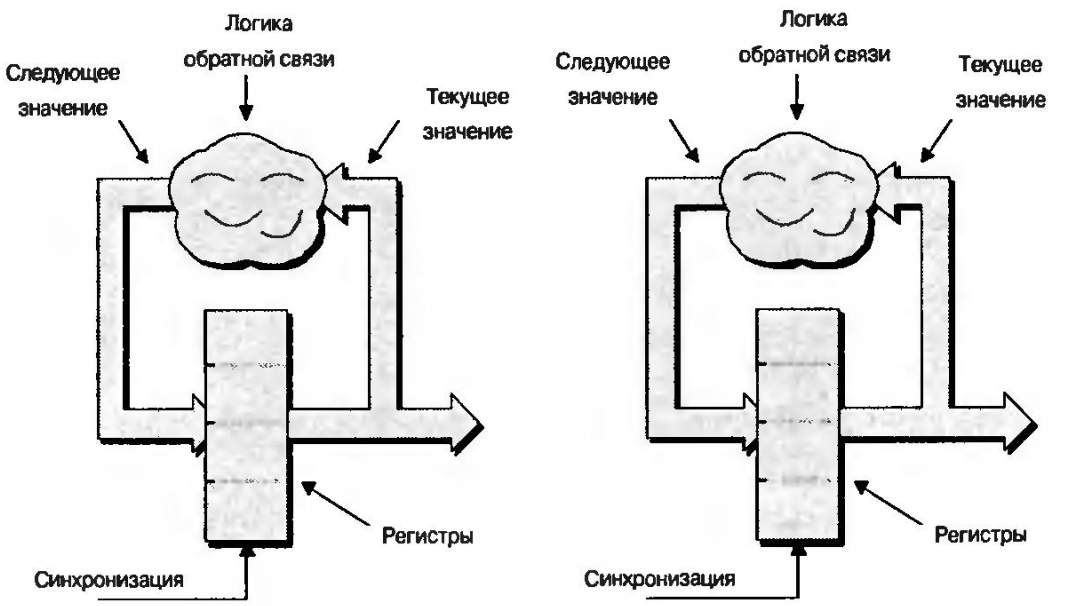
Указатели чтения и записи следуют, друг за другом в бесконечном цикле перебирая адреса массива памяти. При активировании *сигнала записи* данные, находящиеся на входной шине, записываются в слово, адрес которого хранится в указателе записи. После этого указатель записи увеличивает своё значение, и указывает на следующее пустое слово. Соответственно, по *сигналу чтения* данные из слова, адрес которого храниться в регистре чтения, передаются в выходной регистр, после чего значение указателя чтения увеличивается, после чего он будет указывать уже на следующее слово с данными[1]. Также в состав устройства входит логика, обнаруживающая состояния, когда очередь полностью опустошается или заполняется, но её рассмотрение не входит в задачи этого раздела.

Указатели чтения и записи для 16-словной очереди FIFO часто реализуются с помощью 4-битных двоичных счётчиков. Однако для этих целей также можно применить и двоичные последовательности, в частности те, которые генерируются с помощью LFSR-регистра. Как видно из **Рис. В.8**, эти функции работают фактически одинаково.

Из этого рисунка видно, что различие между двоичным счётчиком и LFSR-регистром заключается только в их названии. Идея заключается в том, что для реализации цепи обратной связи для 4-битного двоичного счетчика потребуется несколько логических элементов И и

---

[1] В этом разделе рассматриваются последовательности *запись-увеличение* и *чтение-увеличение*, однако некоторые очереди FIFO используют методы *увеличение-запись* и *увеличение-чтение*.



а) 4-битный двоичный счётчик ***б)*** 4-битный LFSR-регистр

Рис. В.8. Двоичный счётчик и LFSR-регистр

ИЛИ, а цепь обратной связи LFSR-регистра состоит только из одного элемента ИСКЛЮЧАЮЩЕЕ ИЛИ. Это значит, что реализация LFSR-регистра потребуется меньше дорожек и он будет более эффективен с точки зрения занимаемого на кристалле места.

Кроме того, сигнал обратной связи LFSR-регистра проходит только через один уровень логики, а у двоичного счётчика их несколько. Другими словами, новое значение данных в LFSR будет формироваться намного раньше, чем у счетчика, то есть регистр может работать на более высокой частоте. Различие между счётчиком и регистром становится ещё более отчётливым в очередях FIFO с большим количеством слов, для которых требуются многобитные указатели. Следовательно, для некоторых проницательных инженеров LFSR представляет собой довольно интересное решение[1)].

Модификация LFSR для формирования последовательностей длиной 2^n

Отрицательным моментом в рассмотренном выше примере использования 4-битного LFSR-регистра в составе очереди FIFO было то, что генерируемая им последовательность состояла всего из 15 значений ($2^4 - 1$), а счетчик в этом случае выдаёт 16 чисел (2^4). В зависимости от типа приложения, в некоторых случаях разработчики могут не беспокоиться об этой проблеме, особенно когда речь идёт об очередях большой длины. Однако иногда требуется, чтобы формируемая LFSR-регистром последовательность содержала все возможные значения. В этом случае на помощь может придти очень простое решение **(Рис. В.9)**.

Принцип действия такого регистра заключается в том, что числу, состоящему из всех нулей, должно предшествовать значение, в кото-

---

[1)] Также для этих целей используют так называемые счётчики Грея, но в этом приложении мы их рассматривать не будем.

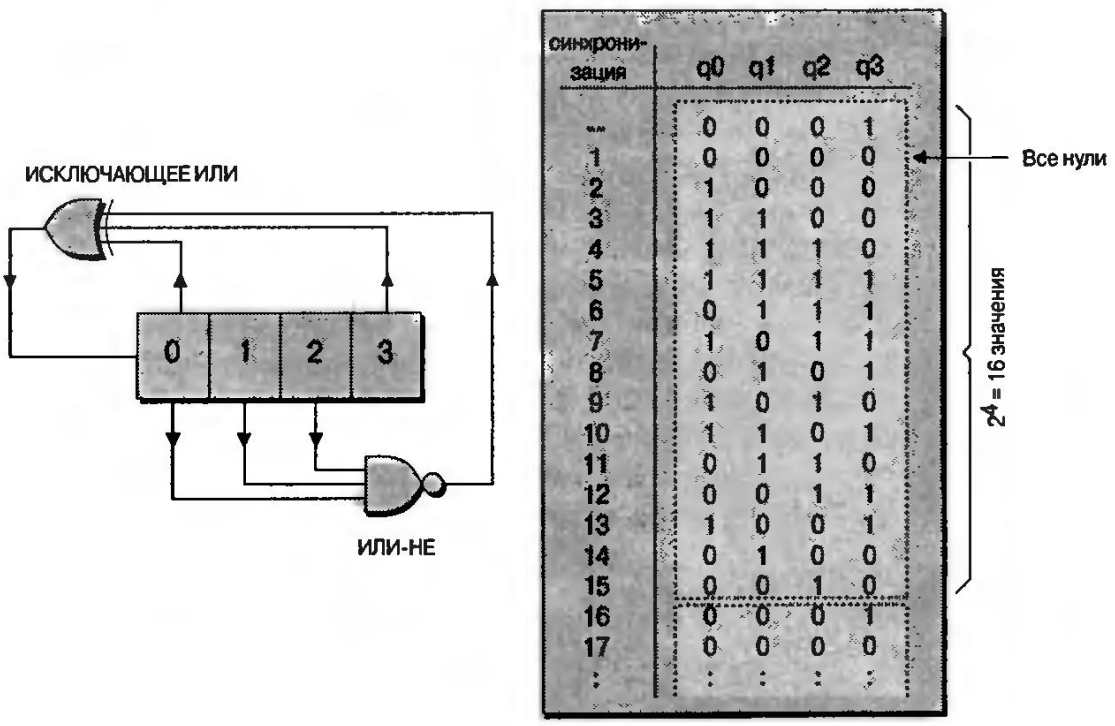


Рис. В.9. Модифицированный LFSR-регистр с длиной последовательности 2^n

ром старший бит[1] установлен в 1, а в остальных ячейках должны быть записаны нули. В обычной версии LFSR-регистра следующий синхроимпульс приведёт к установке в единицу младшего бита и сбросу в 0 всех остальных. Но в модифицированном LFSR-регистре, изображённом на **Рис. В.9**, всё будет по другому. В этом устройстве на выходе логического элемента ИЛИ-НЕ всегда будет находиться значение логического 0, за исключением двух случаев — когда все биты будут установлены в 0, а также в предшествующем случае, то есть когда старший бит установлен в 1, а остальные находятся в 0. Эти две ситуации приводят к появлению на выходе элемента ИЛИ-НЕ логической 1, которая инвертирует выходное значение элемента ИСКЛЮЧАЮЩЕЕ ИЛИ. Это, в свою очередь, вызывает появление в формируемой последовательности значения, состоящего из всех 0, после чего регистр возвращается к обычной работе. (При использовании в цепи обратной связи логического элемента ИСКЛЮЧАЮЩЕЕ ИЛИ-НЕ элемент ИЛИ-НЕ может быть заменён на функцию И, которая вызывает появление в последовательности значения, состоящего из всех единиц.)

Доступ к предыдущему значению

В некоторых приложениях иногда необходимо использовать предыдущие значения регистров. Например, в некоторых очередях FIFO состояние полного заполнения обнаруживается, когда указатель записи указывает на слово, предшествующее указателю чтения[2]. В этом случае в схеме предполагается наличие компаратора, который должен сравнивать текущее значение указателя записи с предыдущим значением указателя чтения. Аналогично, состояние опустошения очереди может быть обнаружено, когда указатель чтения указывает на ячейку памяти, предшествующую указателю записи. В этом случае подразумевается на-

---

[1] Очень часто в сдвиговых регистрах старший бит обозначают с правой стороны, а младший — с левой. В других устройствах обозначение старших и младших бит обычно указывается в противоположном порядке.

[2] А теперь произнесите это быстро!

личие второго компаратора, который должен сравнивать текущее значение указателя чтения с предыдущим значением указателя записи.

При использовании двоичных счётчиков (допустим, что по некоторым причинам мы решили применить их для реализации очереди FIFO) получить предыдущее значение можно двумя способами. Во-первых, можно установить дополнительные *теневые регистры*. Каждый раз при увеличении значения счётчика его текущее значение предварительно копируется в теневой регистр. Также для получения предыдущего значения из текущих данных можно воспользоваться блоком комбинационной логики. К сожалению, оба этих метода приводят к значительным издержкам в виде дополнительных логических схем. В отличие от них, LFSR-регистры в действительности могут запоминать своё предыдущее состояние. Для этого всего лишь необходимо к старшему биту регистра добавить ещё один бит **(Рис. В.10)**.

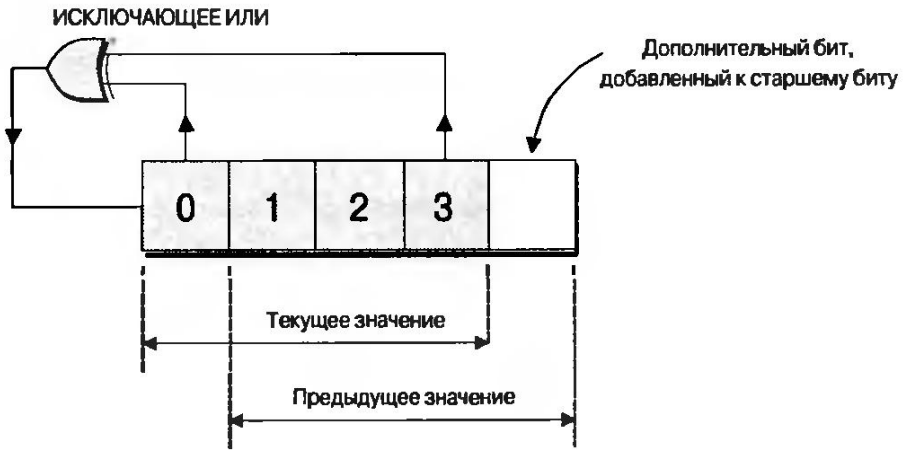


Рис. В.10. Доступ к предыдущему значению в LFSR-регистре

Шифрование и дешифровка

Необычные последовательности значений, генерируемые LFSR-регистрами, могут использоваться для шифрования (кодирования) и дешифровки (декодирования) данных. Битовый поток данных может быть зашифрован с помощью выполнения над ним операции ИСКЛЮЧАЮЩЕЕ ИЛИ с выходным сигналом LFSR-регистра **(Рис. В.11)**.

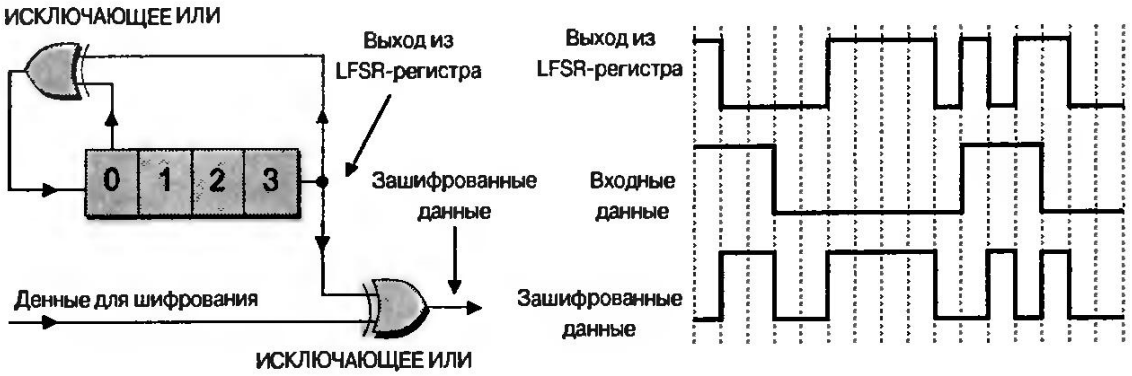


Рис. В.11. Шифрование данных с помощью LFSR-регистра

Поступающий на приёмник поток данных может быть дешифрован при помощи выполнения операции ИСКЛЮЧАЮЩЕЕ ИЛИ с последовательностью данных, генерируемых таким же LFSR-регистром. Безусловно, это довольно простая форма шифрования, которая не отличается большой защищенностью, но не требует больших затрат на реализацию и подходит для некоторых приложений.

Контроль с помощью циклических избыточных кодов

Традиционной сферой применения LFSR-регистров является вычисление *циклических избыточных кодов (CRC — cyclic redundancy code)*, которые используются для обнаружения ошибок в канале связи. В этом случае поток передаваемых данных используется для модификации значений формируемой LFSR-регистром последовательности при помощи подключения его к цепи обратной связи (**Рис. В.12**).

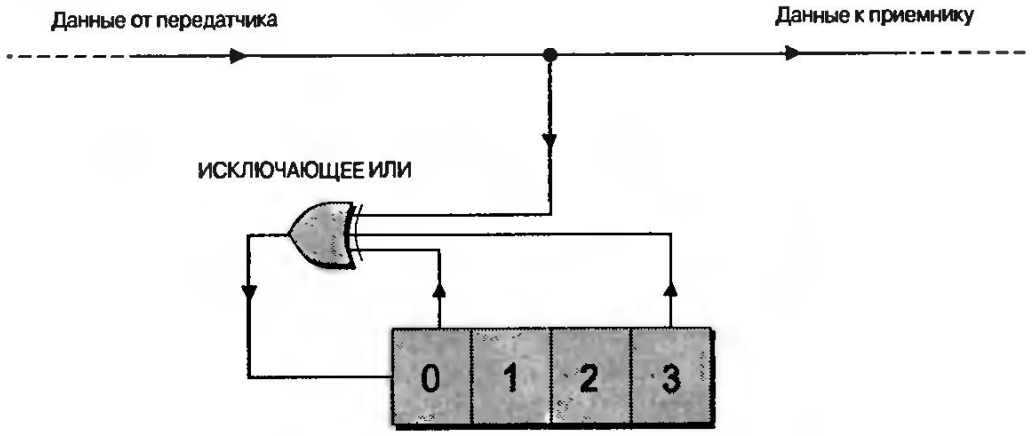


Рис. В.12. Вычисление CRC

Итоговое CRC-значение, сохранённое в LFSR-регистре, называется *контрольной суммой*, и зависит от каждого передаваемого бита. После того, как все данные будут переданы, передатчик посылает приёмнику значение контрольной суммы. Приёмник также содержит аналогичный CRC-блок, и с приёмом данных осуществляет формирование своего собственного значения контрольной суммы. После приёма всех данных приёмник сравнивает внутреннее значение контрольной суммы с тем, которое было сформировано передатчиком, и на основании этого сравнения делает выводы о том, были ли искажения данных в процессе их передачи.

Этот вид обнаружения ошибок очень эффективен с точки зрения малого количества информации, который необходимо передавать вместе с основным блоком данных. Однако недостатком этого метода служит то, что при этом вы не узнаете о наличии ошибки до тех пор, пока не закончится передача данных, и в случае ошибки вам придётся передавать всю информацию заново.

В реальном мире 4-битные блоки подсчёта контрольных сумм не обеспечивают требуемый уровень достоверности о целостности передаваемых данных, так как могут формировать только $(24 - 1) = 15$ уникальных значений. Это может привести к проблеме, называемой *совмещением* (или *aliasing*), при которой итоговое значение контрольной суммы совпадает с ожидаемым, но на самом деле вызвано действием многочисленных ошибок, которые накладываются друг на друга и в итоге приводят к требуемому CRC-значению. При увеличении количества бит в блоке CRC увеличивается и количество генерируемых им уникальных значений, вследствие чего уменьшается вероятность того, что многочисленные ошибки спровоцируют появление такого значения контрольной суммы, которое совпадёт с ожидаемым. Поэтому на практике обычно используют как минимум 16-битные CRC-блоки, которые могут генерировать 65535 уникальных значений.

Различные протоколы связи используют разное количество бит для вычисления значения контрольной суммы и разные отводы. Отво-

ды выбираются таким образом, чтобы ошибка в одном отдельном бите приводила к максимально возможному изменению итогового значения контрольной суммы. Таким образом, кроме *максимальной длины* LFSR-регистры могут также классифицироваться по *максимальному смещению*.

В дополнение к проверке целостности данных в системах связи, CRC-блоки находят широкое применение и в других приложениях, например, их можно использовать для обнаружения компьютерных вирусов. В этом приложении под компьютерными вирусами мы будем подразумевать самовоспроизводящиеся программы, реализованные в компьютерной системе для различных целей. Эти цели могут простираться от мелких пакостей, таких как отображение смешных или надоедливых сообщений, до более гнусных дел, в виде разрушения данных или операционной системы.

Один из механизмов, с помощью которых компьютерные вирусы могут скрываться и распространяться, заключается в том, что они присоединяют себя к существующим программам. Когда программа начинает выполняться, первым делом инициируется тело вируса, то есть поверхностная проверка системы покажет наличие в системе только ожидаемых файлов. Для того чтобы бороться с такой формой вирусной атаки, можно для каждой программы в системе сгенерировать свою контрольную сумму, значение которой будет основано на двоичном коде, формирующем соответствующую программу. Спустя некоторое время, антивирусная программа может пересчитать контрольные суммы для каждой программы и сравнить их с начальными данными. Различие данных может служить сигналом к тому, что к коду программы прикрепился вирус[1)].

Сжатие данных

Описанные выше блоки вычисления контрольных сумм также могут применяться и для сжатия данных. Одно такое приложение входит в механизм тестирования печатной платы и называется *функциональным тестом*. Печатная плата, на которой могут находиться тысячи компонентов и проводников, подключается к функциональному тестеру с помощью торцевого разъема, который может содержать сотни контактов.

Тестер подаёт образцовые сигналы на входы печатной платы, в течение некоторого времени, необходимого для прохождения сигналов по плате, ожидает ответной реакции и затем сравнивает сигналы на выходе с сохранёнными ожидаемыми значениями. Этот процесс затем повторяется для последовательности образцовых входных сигналов, число которых может достигать десятков и сотен тысяч.

Если в ходе предварительного тестирования в работе печатной платы была обнаружена ошибка, то для выявления причины сбоев может использоваться более сложная форма анализа, которая называется *управляемым пробником*. В этом случае тестер указывает оператору на необходимость расположить пробник на определённом участке печатной

[1)] К сожалению, разработчики компьютерных вирусов прибегают к всё более сложным формам построения своих детищ, и некоторые их виды могут формировать собственное значение контрольной суммы. Когда вирус этого типа прикрепляется к файлу, он может подставлять в своё тело определённые двоичные значения, которые действуют таким образом, что во время проверки контрольной суммы антивирусная программа получает значение, совпадающее с начальными данными.

платы, и после этого начинает прогон всей тестовой последовательности сначала. В ходе проверки тестер сравнивает действительные данные, поступающие с пробника, с ожидаемыми значениями, сохранёнными в системе. Этот процесс (размещение пробника и перезапуск тестовой последовательности) повторяется до тех пор, пока тестер не выявит неисправный компонент или дорожку.

Особое внимание при поддержке стратегии управляемого пробника уделяется количеству ожидаемых данных, подлежащих хранению. Давайте рассмотрим тестовую последовательность, содержащую 10000 тестовых воздействий для печатной платы, состоящей из 10000 проводников. При отсутствии механизма сжатия данных в системе необходимо хранить 10000 бит ожидаемых данных для каждого проводника, то есть 100 миллионов бит для всей системы. Кроме того, для каждого положения управляемого пробника тестер должен сравнивать 10000 бит наблюдаемых данных с таким же количеством данных, сохранённых в системе. Таким образом, использование данных без применения механизма сжатия будет неэффективно с точки зрения требований к хранению и обработки информации.

Одним из решений этой проблемы может послужить использование CRC-блока для вычисления контрольных сумм на основе LFSR-регистра. Последовательность ожидаемых значений для каждого проводника может быть пропущена через программно-реализованный 16-битный CRC-блок. Аналогично, последовательность действительных значений, наблюдаемых управляемым пробником, может быть пропущена через идентичный CRC-блок, но уже реализованный аппаратно. В этом случае полученное значение контрольной суммы будет называться *сигнатурой*, а метод управляемого пробника — *сигнатурным анализом*. При этом независимо от количества используемых тестовых воздействий в системе будет храниться только два байта данных для каждого проводника. К тому же для каждого положения управляемого пробника тестер будет сравнивать только два байта данных, снятых пробником, с двумя байтами, хранимыми в системе. Таким образом, сжатие данных позволяет сократить объем хранимых данных и время, необходимое для сравнения результатов, что позволяет увеличить скорость проверки печатной платы.

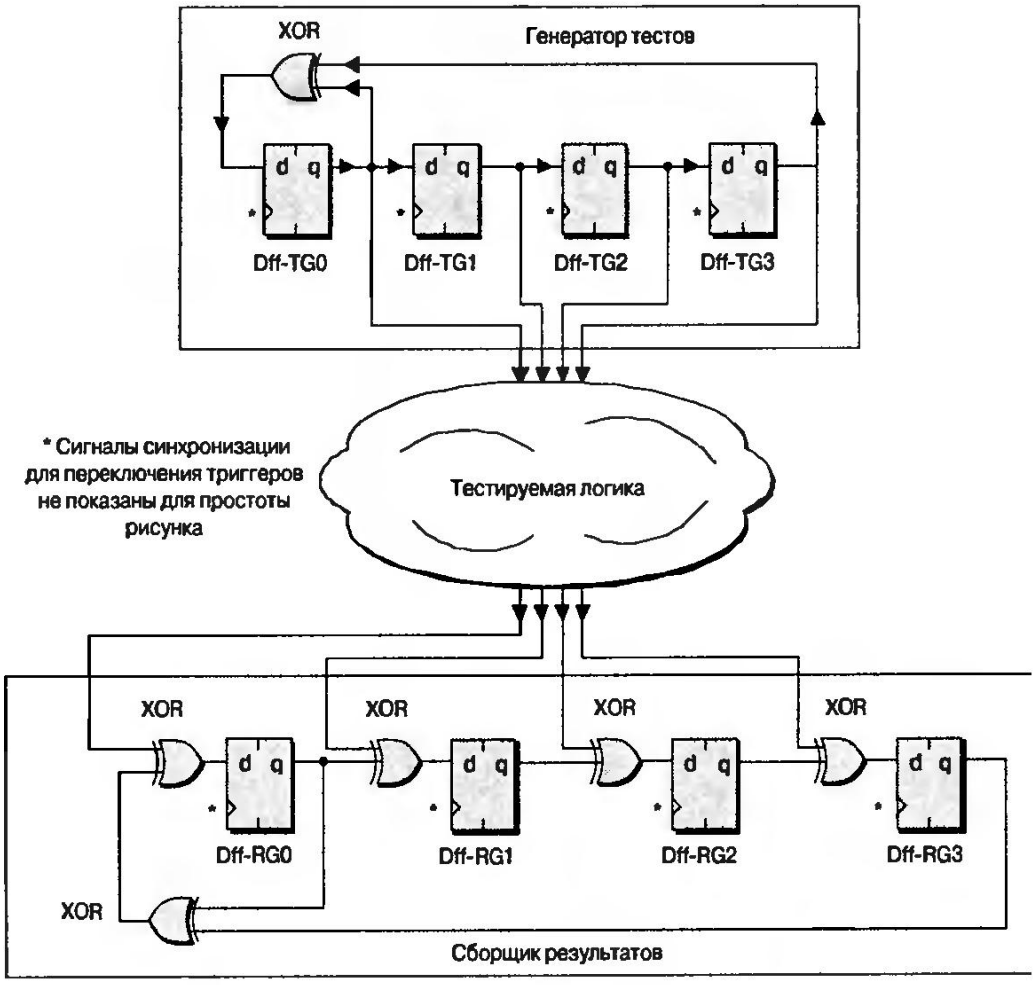
Встроенное самотестирование

Одна из стратегий тестирования, применяемых в сложных интегральных микросхемах, заключается в использовании *встроенных средств самотестирования*. Устройства, использующие такие средства, содержат специальные схемы генерирования тестов и сбора результатов, которые могут быть реализованы на основе LFSR-регистров (**Рис. В.13**).

LFSR-регистр, формирующий генератор тестов, используется для создания последовательности тестовых воздействий, LFSR-регистр в блоке сбора результатов применяется для наблюдения за результатами тестирования. Следует заметить, что регистр в блоке сбора результатов модифицирован для приёма данных в параллельном виде.

Также может понадобиться дополнительная логика для загрузки новых начальных данных в генератор тестов и для доступа к итоговым значениям в блоке сбора результатов. Эти схемы для упрощения рисунка не показаны.

Заметим, что LFSR-регистры могут быть различной длины, так как у тестируемого блока количество входов, на которые подаются воздействия, и выходов, с которых снимаются результаты, могут различаться.



Примечание. XOR — ИСКЛЮЧАЮЩЕЕ ИЛИ

Рис. В.13. Встроенные средства самотестирования

Ещё следует заметить, что все триггеры в генераторе тестов должны работать с одним синхросигналом. Также по одному синхроимпульсу должны срабатывать и триггеры блока сбора результатов. Эти два синхросигнала могут быть общими для двух блоков, но могут и различаться (в этом случае должна быть предусмотрена схема их синхронизации). Синхросигналы на **Рис. В.13** не показаны с целью его упрощения.

Генераторы псевдослучайных последовательностей

Многие компьютерные программы используют в своей работе элементы случайности. Компьютерные игры, такие как Space Invaders, используют случайные события для ещё большего вовлечения игроков. Графические программы могут использовать случайные числа для создания сложных и замысловатых моделей. Все формы компьютерного моделирования могут использовать случайности для формирования более реального представления действительности. Например, цифровые системы моделирования (смотрите также гл. 19) могут получить преимущество от применения случайных воздействий в виде внешних прерываний. Случайные воздействия могут дать более реалистичные результаты проверки устройства и позволят обнаружить проблемы, не доступные для более структурированных тестов.

Генераторы случайных чисел могут быть выполнены как аппаратно, так и программно. Большинство этих генераторов на самом деле не являются случайными, но создают видимость случайности и поэтому называются *псевдослучайными*. На практике псевдослучайные числа

имеют определённые преимущества над действительно случайными последовательностями, так как большинство компьютерных приложений обычно требуют повторяемости. Например, разработчик системы, повторяя цифровое моделирование, ожидает идентичного ответа, которое было получено из системы при предыдущем проходе. Однако ему также необходима возможность модифицировать начальное значение генератора псевдослучайных чисел с тем, чтобы при необходимости порождать другие последовательности.

Существуют различные методы генерирования псевдослучайных чисел, один из которых заключается в использовании LFSR-регистра с изменяемым отводом значений для обеспечения достаточно хорошего источника псевдослучайного сигнала.

И последнее, но не менее важное

LFSR-регистры довольно просты в проектировании и находят широкое применение в различных сферах, но необходимо заметить, что выбор оптимального полинома (который в конечном счёте сводится к выбору точки отвода) для конкретного приложения представляет собой задачу, которая подвластна только мастерам мистических искусств, не говоря уже о том, что математика этого процесса может быть настолько сложной, что может заставить даже взрослого человека потерять самообладание и расплакаться (я даже не буду говорить о круговых полиномах[1]), которые являются основой выбора необходимого отвода).

[1] Главным образом потому, что я понятия не имею о том, что собой представляют круговые полиномы!