

Основные функции

Этот раздел книги посвящен описанию функций языков программирования LAD и FBD, которые представляют в определенном смысле «основную функциональность». Эти функции позволят вам программировать PLC на базе контакторов или реле управления.

В контактном плане (ladder diagram, LAD) компоновка контактов в последовательные и параллельные схемы определяет комбинирование бинарных сигнальных состояний. В функциональном плане (function block diagram, FBD) блочные элементы, аналогичные электронным переключающим системам, представляют булевские (логические) функции AND (И) и OR (ИЛИ).

Функции для работы с памятью находят свое отражение в RLO (Result of logic operation, результат логической операции), поэтому, например, возможно сканирование и передача данных дальше, в другие части программы.

Функции перемещения (move) задействованы в обмене значениями отдельных операндов и переменных или в копировании целых областей данных.

Синхронизирующие реле в контакторных управляющих системах являются таймерами в программируемых контроллерах. Таймеры, встроенные в CPU, позволяют вам, к примеру, запрограммировать время ожидания и наблюдения.

Наконец, счетчики могут вести отсчет по возрастанию и по убыванию в интервале значений 0...999.

Настоящий раздел книги рассказывает о функциях, работающих с областями операндов для входов, выходов и памяти меркеров. Входы и выходы представляют собой связующие элементы в процессе или модели предприятия. Память меркеров соответствует дополнительным контакторам, которые хранят двоичные состояния. Следующие разделы книги приводят описание остальных областей операндов, которые вы можете обрабатывать с использованием бинарной логики. По существу, они являются битами данных в блоках глобальных данных, а также битами временных и статических локальных данных.

В главе 5 «Функции по работе с памятью» вы найдете примеры программирования с операциями бинарной логики и функциями по работе с памятью. Глава 8 «Счетчики» содержит пример, демонстрирующий таймеры и счетчики. В обоих случаях примеры представлены в виде функций FC без параметров блоков. Те же примеры в функциональных блоках (FB) с параметрами блоков вы можете найти в главе 19 «Параметры блоков».

4 Бинарные логические операции (binary logic)

Последовательные и параллельные схемы (LAD), функции AND, OR и исключающее OR (FBD); отрицание (инвертирование); (принимая во внимание тип датчика)

5 Функции для работы с памятью (memory functions)

Катушки LAD; блочные элементы FBD; выводы средней линии (коннекторы); оценка фронта (проверка наличия фронта сигнала); пример с ленточным транспортером конвейера

6 Функции перемещения (move functions)

Блочный элемент MOVE, системные функции для передачи данных

7 Таймеры (timers)

Пять различных видов запуска таймера, сброс и сканирование таймера; IEC-таймеры

8 Счетчики (counters)

Установка счетчика; счет по возрастанию и по убыванию (прямой и обратный счет); сброс и сканирование счетчика; IEC-счетчики; пример подачи питания

Глава 4

Операции бинарной логики

Содержание главы 4

4	<u>Операции бинарной логики</u>	4
4.1	<u>Последовательные и параллельные схемы (LAD)</u>	4
4.1.1	<u>НО-контакт и NC-контакт</u>	4
4.1.2	<u>Последовательные схемы</u>	7
4.1.3	<u>Параллельные схемы</u>	7
4.1.4	<u>Комбинации операций бинарной логики</u>	8
4.1.5	<u>Инвертирование результата логической операции</u>	10
4.2	<u>Операции бинарной логики (FBD)</u>	11
4.2.1	<u>Элементарные операции бинарной логики</u>	11
4.2.2	<u>Комбинации операций бинарной логики</u>	17
4.2.3	<u>Инвертирование результата логической операции</u>	18
4.3	<u>Действия в зависимости от типа датчика</u>	19

4 Операции бинарной логики

4.1 Последовательные и параллельные схемы (LAD)

Бинарные сигнальные состояния группируются в LAD (контактные планы) посредством последовательных (series) и параллельных (parallel) соединений контактов. Последовательное соединение соответствует функции AND (И), а параллельное соединение – функции OR (ИЛИ). Вы будете использовать контакты для проверки сигнальных состояний следующих двоичных операндов:

- Входные и выходные биты, память меркеров;
- Таймеры и счетчики;
- Биты глобальных данных;
- Биты временных локальных данных;
- Биты слова состояния (оценка результатов вычислений).

Вы можете обратиться к операнду через контакт, используя либо абсолютный, либо символический адрес. LAD применяет только контакты NO (сканируют с ожиданием сигнального состояния «1») и контакты NC (сканируют сигнальное состояние «0»).

Цепь, или звено (rung), может состоять из единственного контакта или же большого числа соединенных контактов. Звено всегда должно быть завершено, к примеру, катушкой (coil). Катушка управляет двоичным операндом с помощью RLO («power flow», «протекание тока») цепи.

Примеры, показанные в этой главе, также имеются на дискете, поставляемой с книгой, в функциональном блоке FB 104 программы «Basic Functions» («Основные функции»), библиотека «LAD_Book». Что касается пошагового программирования, то элементы для двоичных логических операций вы можете найти в каталоге программных элементов (Program Element Catalog) – с помощью команды меню View → Catalog (Вид → Каталог) [Ctrl + K] или Insert → Program Elements (Вставка → Программные элементы) – в разделе «Bit Logic» («Битовая логика»).

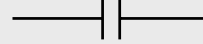
4.1.1 NO-контакт и NC-контакт

Чтобы объяснить комбинации битовой логики в контактном плане, ниже мы будем использовать графические представления и термины «contact closed» («контакт замкнут»), «power flowing» («протекание тока») и «coil energized» («катушка под напряжением», «через/в катушке течет ток» или «катушка возбуждена»). Если «ток» («power») течет в точке контактного плана, то это означает, что в этой точке битовая логика выполняется. Результатом логической операции (RLO) является «1». Если «ток» протекает в одиночной катушке, то она находится под напряжением. Соответствующий двоичный операнд имеет сигнальное состояние «1».

LAD использует два вида контактов для сканирования битовых операндов: NO-контакт и NC-контакт.

**Нормально разомкнутый контакт
(Normally-open, NO)**

Бинарный операнд



**Нормально замкнутый контакт
(Normally-closed, NC)**

Бинарный операнд



Нормально разомкнутый контакт (NO-контакт)

Нормально разомкнутый контакт соответствует сканированию с ожиданием сигнального состояния «1». Если сканированный бинарный операнд имеет сигнальное состояние «1», то NO-контакт активирован, замыкается и «ток течет».

На рисунке 4.1 (верхняя схема) показан пример, где датчик (sensor) S1 подключен к входу I 1.0 и сканируется NO-контактом. Если датчик S1 разомкнут, то вход I 1.0 содержит «0», и ток через NO-контакт не проходит. Контакт (contactor) K1, управляемый выходом Q 4.0, не включен.

Если датчик S1 активирован, то вход I 1.0 имеет сигнальное состояние «1». Ток от левой несущей (питающей шины) течет через NO-контакт в катушку, и контактор K1, подключенный к выходу Q 4.0, активируется. NO-контакт сканирует вход с ожиданием сигнального состояния «1» и затем замыкается, независимо от того, каким контактом, NO или NC, при входе является датчик.

Нормально замкнутый контакт (NC-контакт)

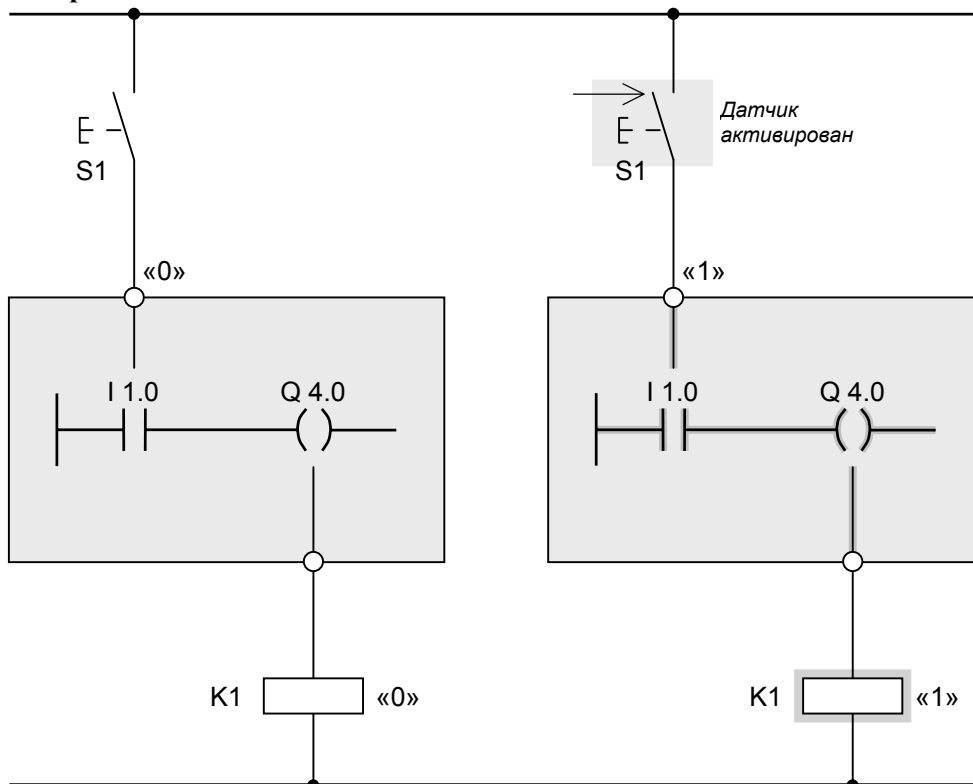
Ток протекает через NC-контакт, если бинарный операнд имеет сигнальное состояние «0». Если сигнальное состояние «1», то NC-контакт «размыкается», и протекание тока прерывается.

В примере на рисунке 4.1 (нижняя схема) ток течет через NC-контакт, если датчик S2 незамкнут (вход I 1.1 имеет сигнальное состояние «0»). Ток также течет в катушке и возбуждает контактор K2 на выходе Q 4.1.

Если датчик S2 активирован, то вход I 1.1 имеет сигнальное состояние «1», и NC-контакт размыкается. Протекание тока прерывается, и с контактора K2 снимается нагрузка.

NC-контакт проверяет вход на наличие нулевого сигнального состояния и пребывает затем в замкнутом состоянии, несмотря на то, каким контактом на входе является датчик, NO- или NC-контактом (обратитесь также к параграфу 4.3 «Действия в зависимости от типа датчика»).

Как работает NO-контакт



Как работает NC-контакт

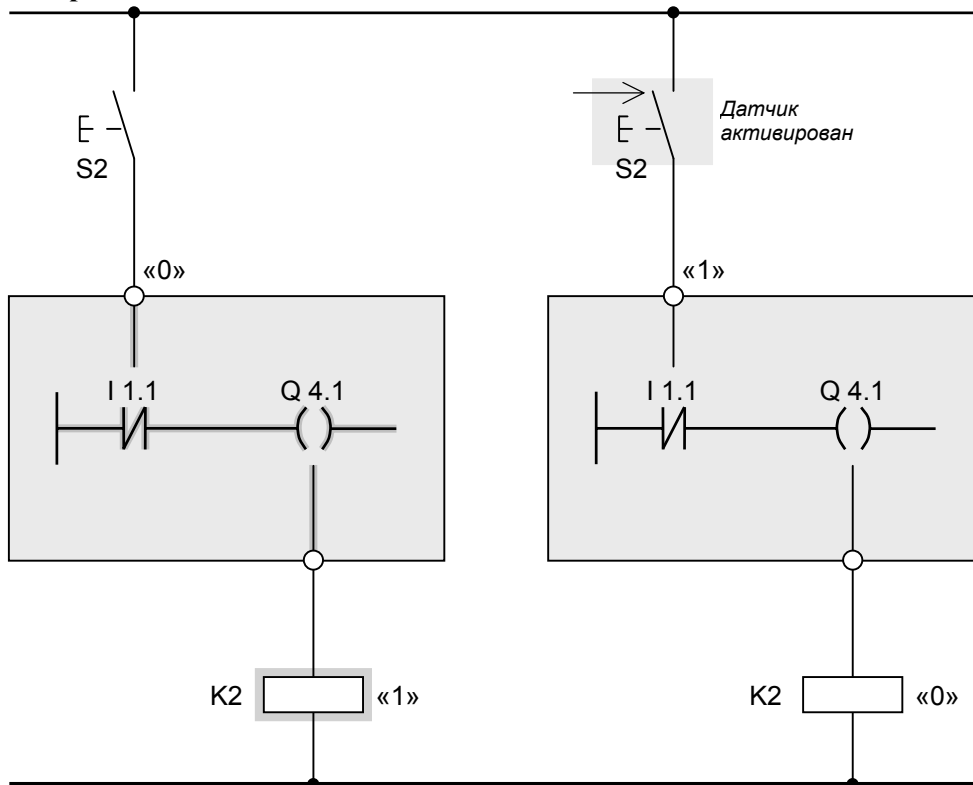


Рисунок 4.1 NO-контакты и NC-контакты

4.1.2 Последовательные схемы

В последовательных схемах два или более контактов соединены последовательно. Ток в последовательной схеме течет, когда все контакты замкнуты.

На рисунке 4.2 показаны типичные последовательные схемы. В сети 1 (network 1) последовательная схема содержит три контакта; любые бинарные операнды могут сканироваться. Все контакты являются НО-контактами. Если все соответствующие операнды имеют сигнальное состояние «1» (то есть если НО-контакты активированы), то в цепи ток течет к катушке. Операнд, управляемый катушкой, устанавливается в «1». Во всех других случаях ток не течет, и операнд *Coil1* (*Катушка1*) сбрасывается в «0».

Сеть 2 (Network 2) демонстрирует последовательный план с одним НС-контактом. Ток течет через НС-контакт, если соответствующий операнд находится в сигнальном состоянии «0» (то есть НС-контакт не активирован). Таким образом, ток протекает через последовательную схему в случае, если операнд *Contact4* (*Контакт4*) имеет сигнальное состояние «1», а операнд *Contact5* (*Контакт5*) имеет сигнальное состояние «0».

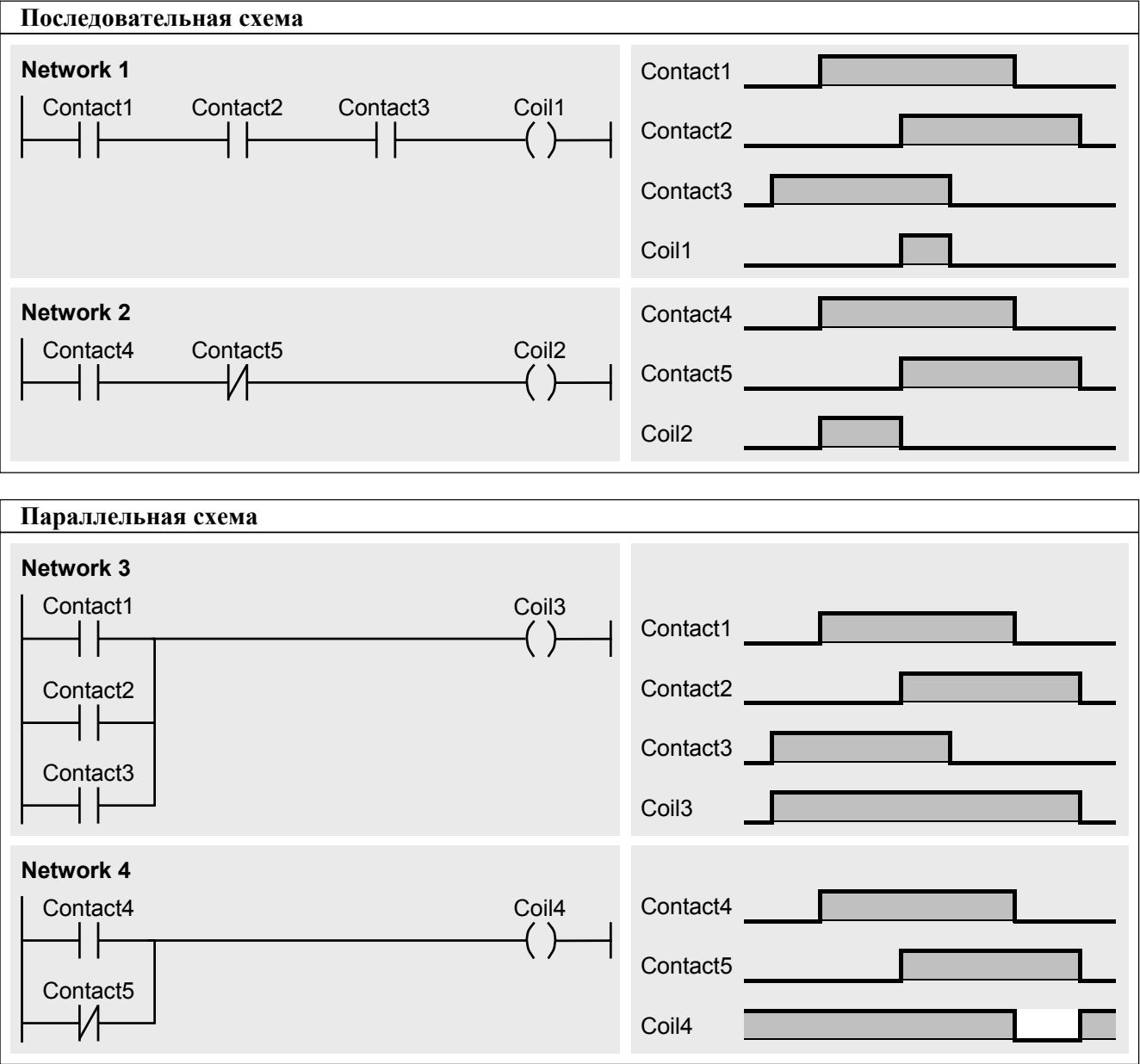
4.1.3 Параллельные схемы

Когда два или более контактов скомпонованы один под другим, мы называем это параллельной схемой. Ток протекает через параллельную схему, если один из контактов замкнут.

На рисунке 4.2 показаны типичные параллельные схемы. В сети 3 (Network 3) параллельный план состоит из трех контактов; сканироваться может любой операнд. Все контакты являются НО-контактами. Если один из операндов имеет сигнальное состояние «1», то по схеме к катушке течет ток. Операнд, управляемый катушкой, устанавливается в «1». Если все сканируемые операнды находятся в сигнальном состоянии «0», то ток к катушке не проходит, и операнд *Coil3* (*Катушка3*) сбрасывается в «0».

Сеть 4 (Network 4) демонстрирует параллельный план с одним НС-контактом. Ток течет через НС-контакт, если соответствующий операнд – «0», то есть ток протекает через параллельную схему в случае, если операнд *Contact4* (*Контакт4*) имеет сигнальное состояние «1», или операнд *Contact5* (*Контакт5*) имеет сигнальное состояние «0».

В LAD вы также можете запрограммировать ветвь в середине цепи, как это изображено в примере на рисунке 4.3, сеть 8 (Network 8). У вас может быть, таким образом, параллельная ветвь, которая не начинается на левой несущей (шине питания). Использование программных элементов LAD в такой ветви ограничено; этот случай рассмотрен в соответствующих главах. «Открытая» («разомкнутая», «орен») параллельная цепь называется «Т-ветвь».



Параллельная схема

Network 3

Timing diagram for Network 3:

- Contact1: High from t1 to t2.
- Contact2: High from t3 to t4.
- Contact3: High from t1 to t2.
- Coil3: Pulse at t1.

Network 4

Timing diagram for Network 4:

- Contact4: High from t1 to t2.
- Contact5: High from t3 to t4.
- Coil4: Pulse at t1.

Рисунок 4.2 Последовательные и параллельные схемы

4.1.4 Комбинации операций бинарной логики

Вы можете объединять последовательные и параллельные схемы, например, путем группировки нескольких последовательных схем в параллельные планы или нескольких параллельных схем в последовательные планы. Вы можете комбинировать последовательные и параллельные схемы, даже когда оба типа являются сложными изначально (рисунок 4.3).

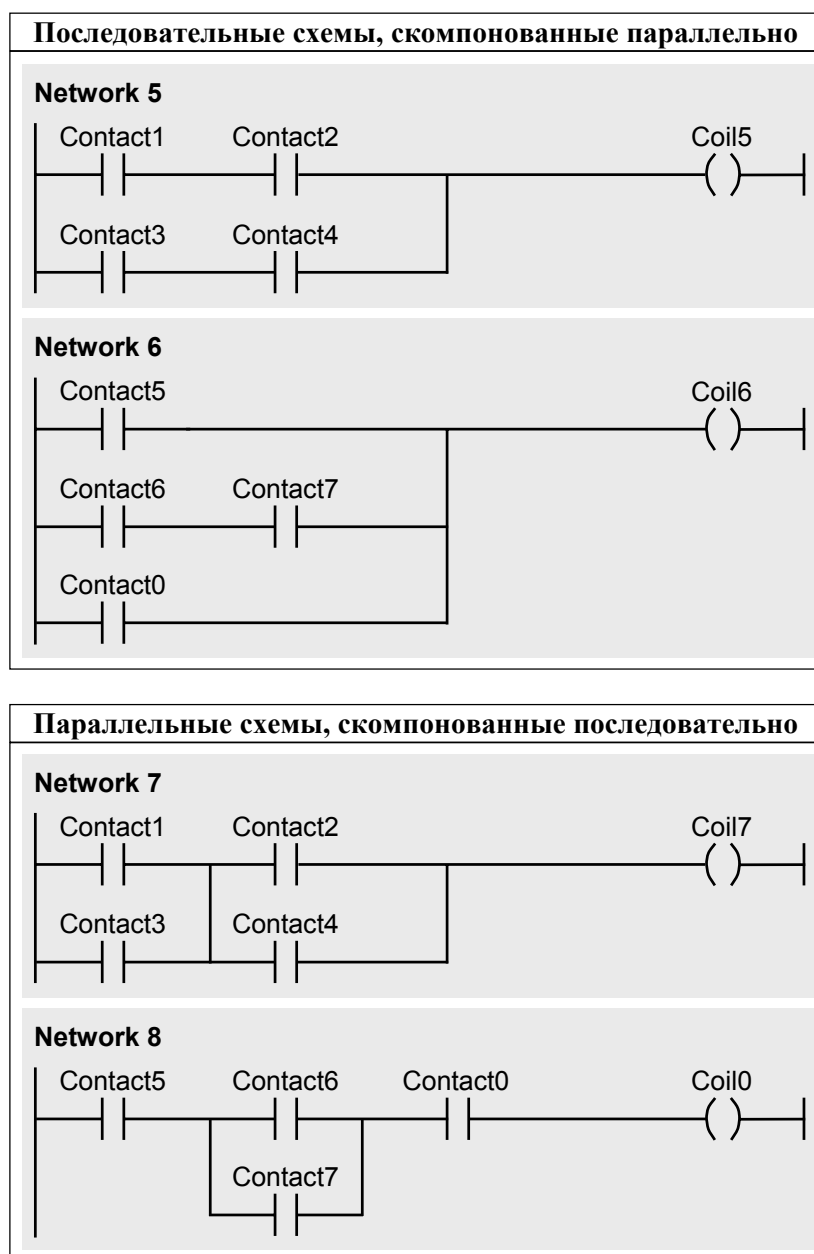


Рисунок 4.3 Последовательные и параллельные схемы в комбинации

Параллельное соединение последовательных схем

Наряду с контактами вы можете также скомпоновать последовательные схемы одна под другой. На рисунке 4.3 приведены два примера. В сети 5 (Network 5) ток течет в катушке (достигает ее), если *Contact1* (Контакт1) и *Contact2* (Контакт2) замкнуты или *Contact3* (Контакт3) и *Contact4* (Контакт4) находятся в замкнутом состоянии. В нижней цепи (сеть 6, Network 6) ток течет, если замкнуты *Contact5* (Контакт5) или *Contact6* (Контакт6) и *Contact7* (Контакт7) или *Contact0* (Контакт0).

Последовательное соединение параллельных схем

Как и в случае контактов, вы также можете группировать параллельные схемы в последовательности. Рисунок 4.3 содержит два подобных примера. В сети 7 (Network 7) ток течет в катушке (достигает ее), если либо *Contact1* (Контакт1) или *Contact3* (Контакт3) и либо *Contact2* (Контакт2) или *Contact4* (Контакт4) замкнуты. Чтобы позволить току течь в нижнем примере (сеть 8, Network 8), *Contact5* (Контакт5), *Contact0* (Контакт0) и либо *Contact6* (Контакт6), либо *Contact7* (Контакт7) должны быть замкнуты.

4.1.5 Инвертирование результата логической операции

NOT-контакт инвертирует результат логической операции (RLO). Можно использовать этот контакт, чтобы, к примеру, передать в катушку инвертированное значение последовательной цепи (рисунок 4.4, сеть 9, Network 9). Ток будет течь в катушке только тогда, когда нет тока в контакте NOT, то есть если разомкнут *Contact1* (Контакт1) или *Contact2* (Контакт2) (обратите внимание в рисунке на смежную диаграмму импульсов).

То же самое применимо по аналогии для сети 10 (Network 10), в которой NOT-контакт вставлен после параллельной цепи. Здесь *Coil10* (Катушка10) устанавливается, если ни один из контактов не замкнут.

Вы можете вставить NOT вместо другого контакта в цепь, которая начинается на левой несущей (питающей шине). Добавление NOT-контакта в параллельную ветвь, начинающуюся в середине цепи, не допускается.

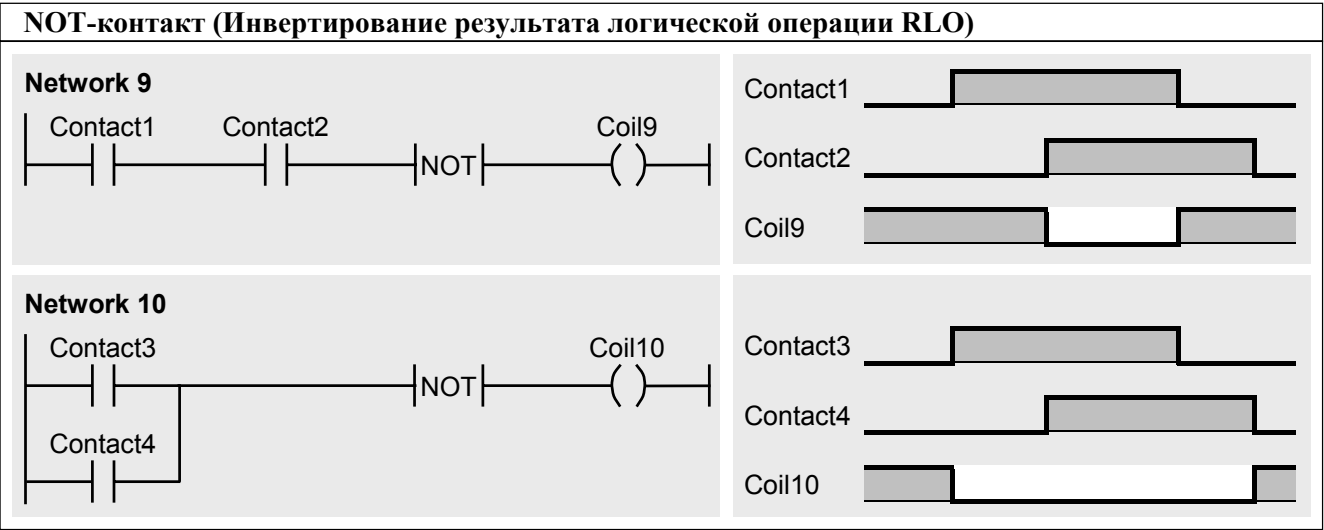


Рисунок 4.4 Примеры включений NOT-контакта

4.2 Операции бинарной логики (FBD)

В FBD логические операции, выполняемые над двоичными сигнальными состояниями, принимают форму функций AND (И), OR (ИЛИ) и Exclusive OR (Исключающее ИЛИ). Операнды, чьи сигнальные состояния вы хотите сканировать и комбинировать, подаются (записываются) на входы этих функций. Вы можете сканировать следующие операнды:

- Входные и выходные биты, меркеры (рассматриваются в данном разделе);
- Таймеры и счетчики;
- Биты глобальных данных;
- Биты временных локальных данных;
- Биты статических локальных данных;
- Биты слова состояния (результаты оценки и вычислений).

Каждый бинарный операнд может быть адресован абсолютно или символически. При сканировании бинарного операнда, или внутри схемы бинарной логики (функционального плана), вы можете инвертировать результат логической операции с помощью символа инвертирования (отрицания). Этот символ представляет собой кружок.

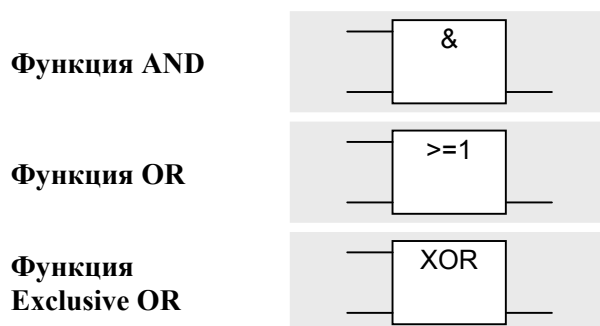
В FBD для каждого сегмента (сети, схемы) программируется одна схема бинарной логики (binary logic circuit). Логическая схема может состоять из одного или очень большого числа соединенных между собой функциональных элементов. Логическая схема, или логическая операция (logic operation), всегда должна быть завершена, например, оператором присваивания. Присваивание воздействует на бинарный операнд с помощью результата логической операции.

Примеры, представленные в этой главе, также имеются на прилагаемой дискете, в функциональном блоке FB 104 программы «Basic Functions» («Основные функции»), в библиотеке «FBD_Book». Для пошагового программирования элементы, составляющие операции бинарной логики, вы можете найти в каталоге программных элементов (Program Element Catalog) – с помощью команды меню View → Catalog (Вид → Каталог) [Ctrl + K] или Insert → Program Elements (Вставка → Программные элементы) – в разделе «Bit Logic» («Битовая логика»).

4.2.1 Элементарные операции бинарной логики

FBD использует бинарные функции AND (И), OR (ИЛИ) и Exclusive OR (Исключающее ИЛИ). Все функции могут иметь (теоретически) любое количество функциональных входов (входов функции). Если вход ведет напрямую к функциональному элементу, то сигнальное состояние сканируемого операнда непосредственно используется в логической операции; если вход снабжен знаком отрицания (кружок),

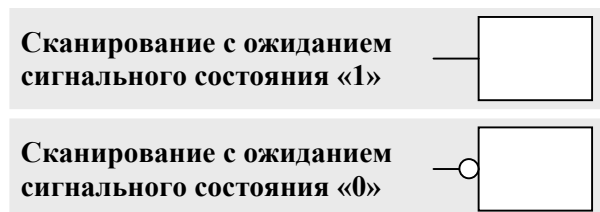
то сигнальное состояние сканируемого операнда инвертируется перед выполнением логической операции (см. ниже).



Теоретически количество бинарных функций и область действия бинарной функции неограничены. Однако, на практике границы регулируются размерами блока или объемом главной памяти CPU.

Сканирование и назначение сигнальных состояний

Перед тем как бинарные функции выполняют логические операции над сигнальными состояниями, они сканируют (считывают) бинарные операнды на своих входах. Операнд может сканироваться с ожиданием «1» или «0». Если сканируется с ожиданием «1», то вход функции направлен непосредственно в блочный элемент. Сканирование с ожиданием «0» распознается по символу отрицания на входе функции.

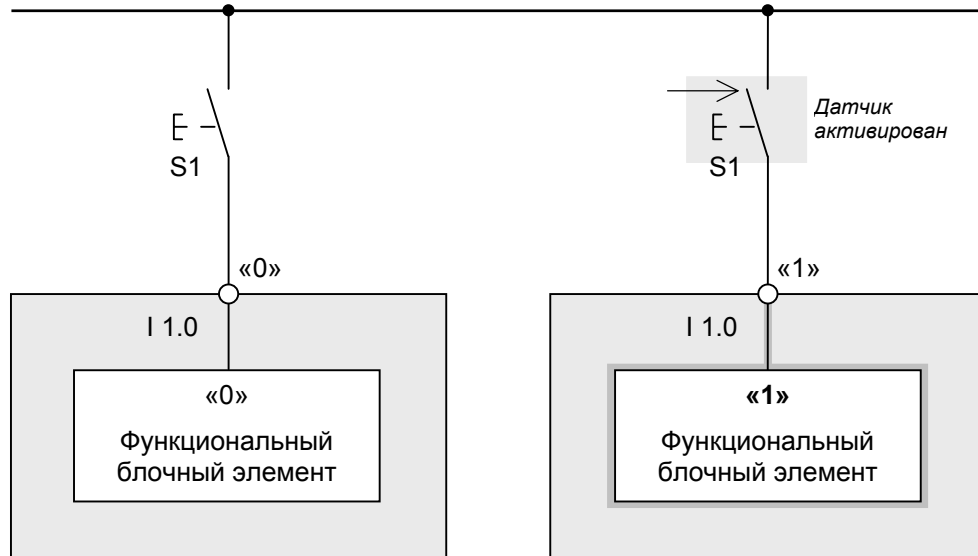


Сканирование с ожиданием «1» генерирует результат сканирования (scan result) «1», когда сигнальное состояние сканированного двоичного операнда «1». Сканирование выводит результат сканирования «0», когда сигнальное состояние бинарного операнда «0». Сканирование с ожиданием сигнального состояния «0» инвертирует результат сканирования, то есть результат сканирования равен «1», когда статус сканированного бинарного операнда «0». Бинарные функции комбинируют *результат сканирования*, поданный «непосредственно» на блочный элемент. С точки зрения функциональности эти два метода сканирования бинарных операндов позволяют вам использовать NO-контакты и NC-контакты идентично.

Приведем пример: «0» подается на входной модуль неактивированного NO-контакта (рисунок 4.5). Сканирование с ожиданием сигнального состояния «1» передает это состояние в функциональный блочный элемент (блочный элемент функции). Для выполнения того же для NC-контакта вы должны сканировать вход с помощью NC-

контакт с ожиданием сигнального состояния «0» (должен иметься кружок для инвертирования). Сигнальное состояние «1», подаваемое на входной модуль неактивированного NC-контакта, затем конвертируется в сигнальное состояние «0» в функциональном блочном элементе.

Сканирование с ожиданием сигнального состояния «1»



Сканирование с ожиданием сигнального состояния «0»

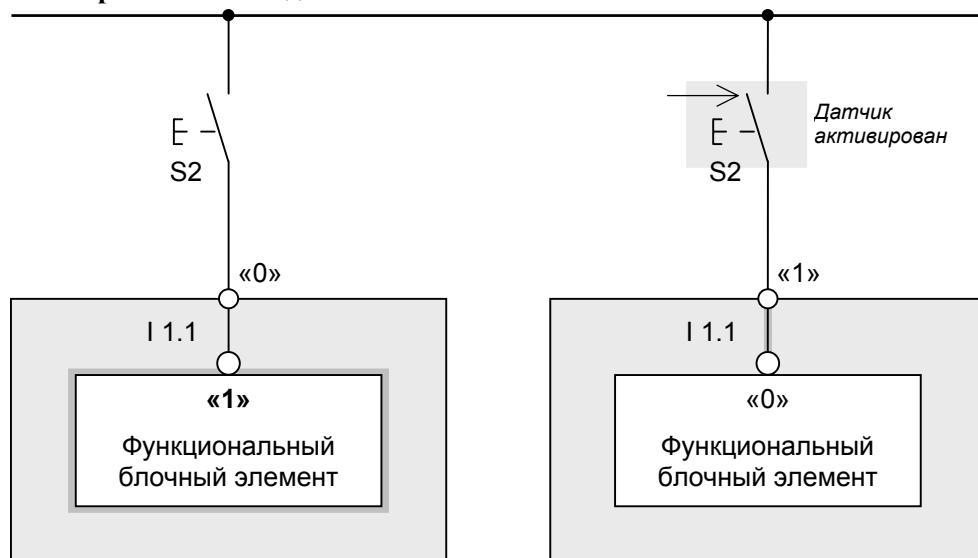


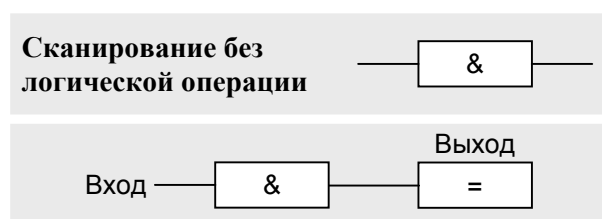
Рисунок 4.5 Сканирование с ожиданием сигнального состояния «1» и «0»

Если теперь активировать оба контакта, и NC, и NO, функциональный блочный элемент в обоих случаях покажет сигнальное состояние «1». Дополнительную информацию вы можете найти в параграфе 4.3 «Действия в зависимости от типа датчика».

Вы всегда должны соединять (с каким-либо элементом) выход бинарной функции; в простейшем случае можно просто соединить выход с блочным элементом Assign

(Присваивание) (обратитесь также к главе 5 «Функции по работе с памятью»). С помощью данного результата логической операции вы также можете запустить таймер, выполнить цифровую операцию, вызвать блок и так далее. В следующей главе содержится вся необходимая информация.

Чтобы присвоить сигнальное состояние бинарного операнда непосредственно другому бинарному операнду, не выполняя никаких дополнительных логических операций, например, подсоединить вход напрямую к выходу, обычно используется функция AND (И), хотя допустимо было бы применить OR (ИЛИ) или Exclusive OR (Исключающее ИЛИ).



Просто выберите функцию AND (И), подключив только один вход функции и удалив другой.

Функция AND (И)

Функция AND комбинирует два бинарных состояния друг с другом и генерирует результат логической операции (RLO) «1», если оба состояния (оба сканированных результата) равны «1». Если функция AND имеет несколько входов, результаты сканирования всех входов должны быть равны «1», чтобы совокупный RLO был «1». В остальных случаях функция AND на своем функциональном выходе выдает RLO «0».

Рисунок 4.6 содержит пример функции AND. В сети 1 (Network 1) функция AND имеет три входа, каждый из которых может быть подключен к любому бинарному операнду. Все операнды сканируются с ожиданием сигнального состояния «1», поэтому сигнальные состояния операндов непосредственно (напрямую) объединяются по AND. Если все сканированные операнды имеют сигнальное состояние «1», то функция AND установит операнд *Output1* (*Выход1*) в «1» посредством блочного элемента Assign (Присваивание) (обратитесь к следующей главе). В остальных случаях условие AND не выполняется, и операнд *Output1* (*Выход1*) сбрасывается в «0».

Сеть 2 (Network 2) показывает функцию AND с инвертированным входом. Инвертирование (отрицание) входа обозначается кружком. Результат сканирования для инвертированного операнда равен «1», если этот операнд равен «0», то есть в примере условие AND выполняется, когда операнд *Input4* (*Вход4*) равен «1» и операнд *Input5* (*Вход5*) равен «0».

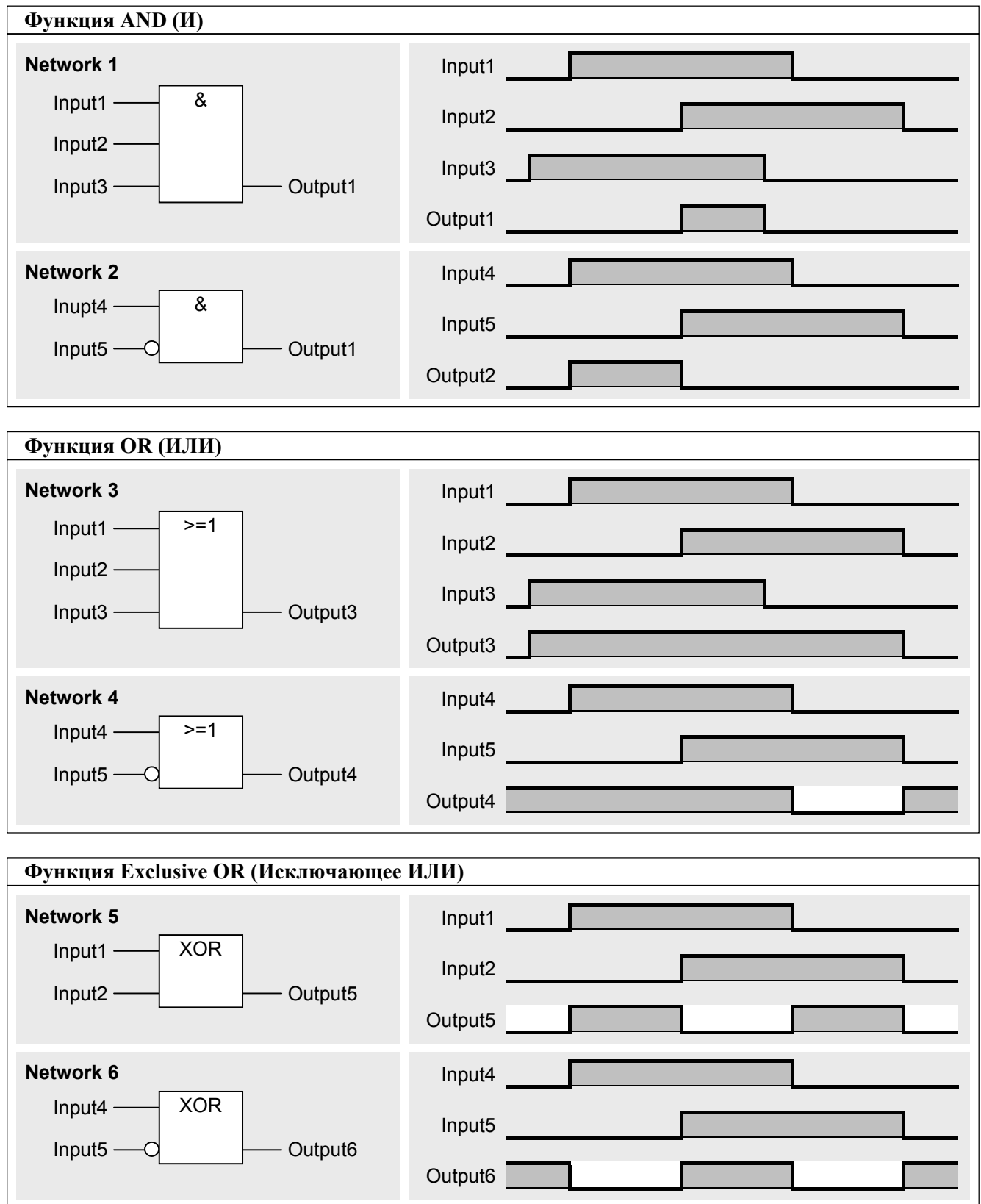


Рисунок 4.6 Примеры бинарных функций

Функция OR (ИЛИ)

Функция OR комбинирует два бинарных сигнальных состояния и возвращает RLO «1», когда одно из этих состояний (один из результатов сканирования) равно «1». Если функция OR имеет несколько входов, то для того, чтобы результат логической операции (RLO) был «1», требуется только один вход, результат сканирования которого равен «1». Функция OR возвращает RLO «0», когда результаты сканирования всех входов равны «0».

Рисунок 4.6 иллюстрирует пример функции OR. На сети 3 (Network 3) функция OR оснащена тремя входами; каждый из этих входов может быть соединен с любым бинарным операндом. Все операнды сканируются с ожиданием сигнального состояния «1», поэтому сигнальное состояние операндов напрямую объединяются по OR. Если один или более сканированных операндов имеют сигнальное состояние «1», то следующий оператор установит операнд *Output3 (Выход3)* в «1». Если все сканированные операнды имеют сигнальное состояние «0», то условие OR не удовлетворяется, и операнд *Output1 (Выход1)* сбрасывается в «0».

Сеть 4 (Network 4) демонстрирует функцию OR с инвертированным входом. Инвертирование (отрицание) представлено кружком. Результат сканирования инвертированного операнда равен «1», если данный операнд равен «0», то есть в примере условие OR выполняется, если операнд *Input4 (Вход4)* имеет сигнальное состояние «1» или операнд *Input5 (Вход5)* имеет сигнальное состояние «0».

Функция Exclusive OR (Исключающее ИЛИ)

Функция Exclusive OR комбинирует друг с другом два бинарных состояния и возвращает RLO «1», когда два состояния (результаты сканирования) не являются одинаковыми, и RLO «0», если два состояния (результаты сканирования) идентичны.

На рисунке 4.6 приведен пример функции исключающего OR. На сети 5 (Network 5) два входа, оба сканируемые с ожиданием сигнального состояния «1», ведут в функцию Exclusive OR. Если только один из сканированных операндов равен «1», то условие исключающего OR выполняется, и операнд *Output5 (Выход5)* устанавливается в «1». Если оба операнда «1» или «0», то операнд *Output5 (Выход5)* равен «0».

Схема 6 изображает функцию исключающего OR с инвертированным входом. Инвертирование (отрицание) представляется кружком. Результат сканирования инвертированного операнда равен «1», когда операнд сброшен в «0», то есть условие исключающего OR в примере выполняется, если оба входных операнда имеют одинаковые сигнальные состояния.

Вы также можете запрограммировать функцию Exclusive OR с количеством входов более двух. В таком случае условие исключающего OR удовлетворяется (в случае непосредственного сканирования), если нечетное число сканированных операндов имеют результат сканирования «1».

4.2.2 Комбинации операций бинарной логики

Вы можете свободно комбинировать бинарные функции одна с другой. Например, можно объединить несколько функций AND по функции OR или две функции OR по функции исключающего OR. Количество функций в логической операции (схеме или сети) теоретически неограничено.

Использование «Т-ветви» в логической операции (контактном плане) предоставляет дополнительные возможности, позволяющие программировать для одной схемы более одного выхода (обратитесь к параграфу 5.2 «Блочные элементы FBD»).

Вы можете соединить выход одной бинарной функции с входом другой с целью выполнения сложной бинарной логической операции. На рисунке 4.7 представлено несколько примеров.

Сеть 9 (Network 9): вы осуществляете наблюдение концевых выключателей (limit switches) на окончаниях осей X и Y. Эти концевые выключатели не могут быть активированы попарно; иначе возникнет сообщение об ошибке концевых выключателей.

Схема 10 (Network 10): вы можете соединить произвольные входы функции с бинарными функциями, например, можно поместить функцию исключающего OR перед вторым входом функции AND.

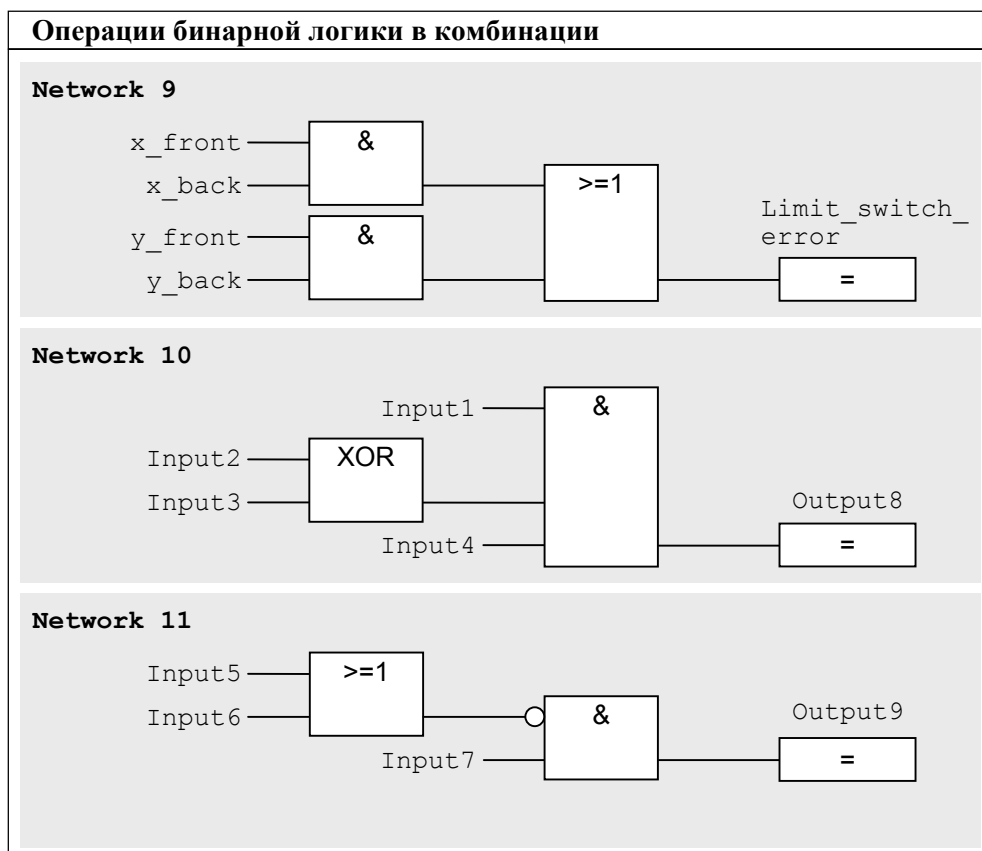


Рисунок 4.7 Примеры операций бинарной логики в комбинации

Сеть 11 (Network 11): используя символ отрицания (инвертирования), вы инвертируете RLO, даже между бинарными функциями, например, вы можете инвертировать RLO функции OR и использовать его в качестве входа в функцию AND.

4.2.3 Инвертирование результата логической операции

Кружок на входе или выходе символа функции инвертирует результат логической операции. Вы можете использовать инвертирование (отрицание)

- для сканирования бинарного операнда, что эквивалентно сканированию с ожиданием сигнального состояния «0» (см. выше),
- между двумя бинарными функциями (что эквивалентно инвертированию результата логической операции) или
- на выходе бинарной функции (к примеру, если вы хотите установить или сбросить бинарный операнд, когда условие не выполнено, то есть когда RLO = «0»).

Инвертирование не может следовать сразу за Т-ветвью.

Рисунок 4.8 показывает функцию NAND (функцию AND с инвертированным выходом) и функцию NOR (функцию OR с инвертированным выходом). RLO функции NAND равен «0» только тогда, когда все входы имеют сигнальное состояние «1». Функция NOR возвращает RLO «1» только тогда, когда ни один из входов не обладает сигнальным состоянием «1».

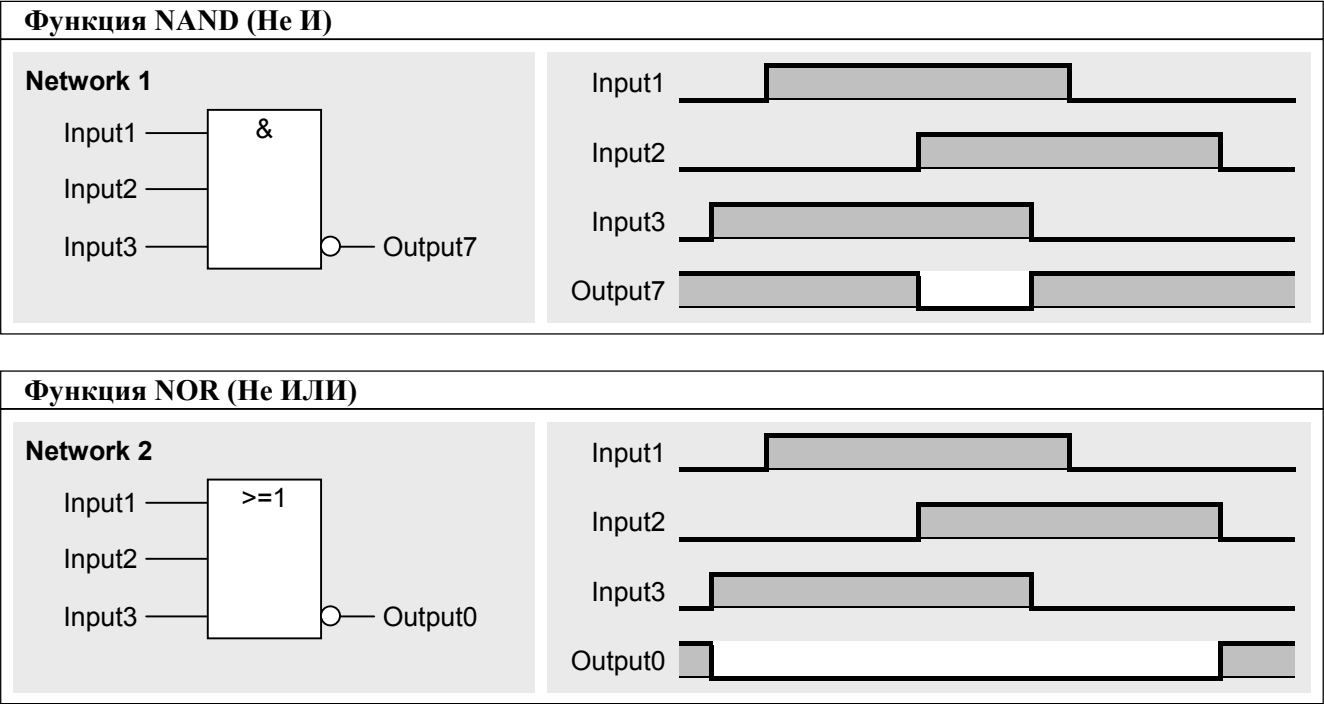


Рисунок 4.8 Функции NAND и NOR

4.3 Действия в зависимости от типа датчика

При сканировании датчика (sensor) в пользовательской программе вы должны уделять внимание тому, каким контактом он является – NC или NO. Когда датчик активирован, в зависимости от его типа на соответствующем входе имеется различное сигнальное состояние: «1» в случае NO-контакта и «0», если это NC-контакт. CPU не располагает средствами определения, занят вход NO-контактом или NC-контактом. Центральный процессор может лишь распознать сигнальное состояние «1» или сигнальное состояние «0».

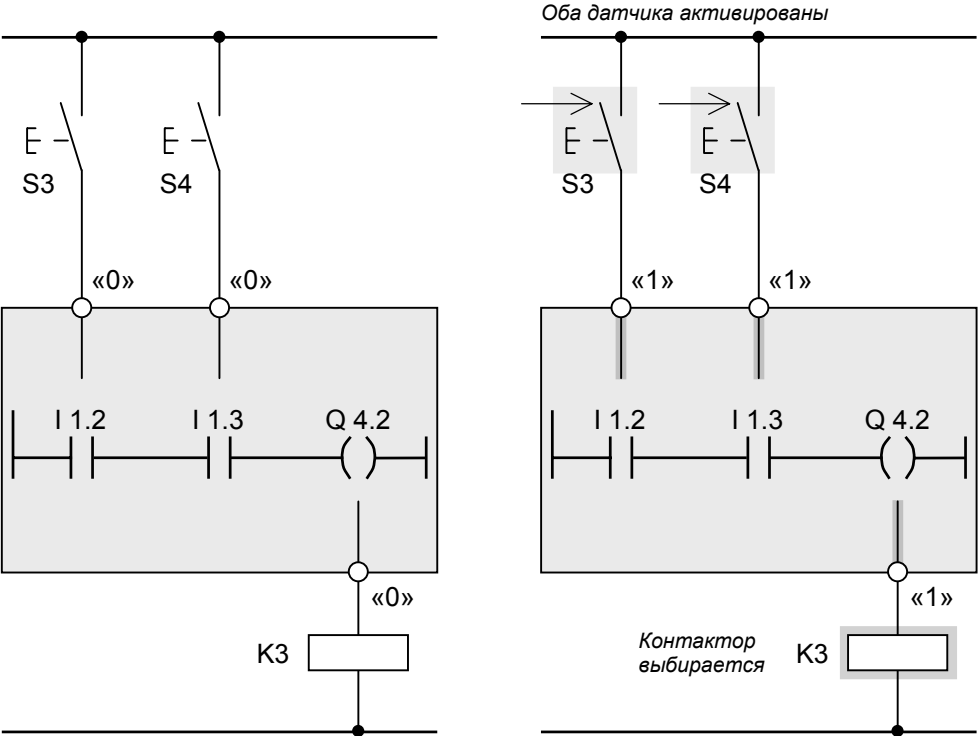
LAD-программирование

Если вы создаете структуру программы таким способом, что вы хотите получить результат сканирования «1» при активировании датчика для того, чтобы в последствии этот результат комбинировать, то для различных видов датчиков вы должны сканировать вход по-разному. Для этой цели вам доступны NO-контакты и NC-контакты. NO-контакт возвращает «1», если сканированный вход также равен «1». NC-контакт возвращает «1», если сканированный вход равен «0». Таким образом, вы можете также непосредственно (напрямую) сканировать входы, которые должны вызывать действия, находясь в состоянии «0» (входы с нулевой активностью), и в дальнейшем перепропустить (re-gate) результат сканирования.

Пример на рисунке 4.9 демонстрирует приемы программирования в зависимости от типа датчика. В первом случае два NO-контакта подключены к программируемому контроллеру, во втором случае – один NO-контакт и один NC-контакт. В обоих вариантах контактор, соединенный с выходом, должен быть выбран, если оба датчика активированы. Если NO-контакт активирован, сигнальное состояние на входе «1», и оно сканируется NO-контактом, поэтому ток может протекать, когда датчик активирован. Если оба NO-контакта активированы, ток течет по цепи к катушке, и контактор выбирается.

Если активирован NC-контакт, сигнальное состояние на входе «0». Чтобы в этом случае ток протекал при активированном датчике, результат должен сканироваться NC-контактом. Следовательно, во втором случае NO-контакт и NC-контакт должны быть подключены последовательно с целью выбора контактора при активировании обоих датчиков.

Случай 1: оба датчика являются NO-контактами



Случай 2: один NO- и один NC-контакт

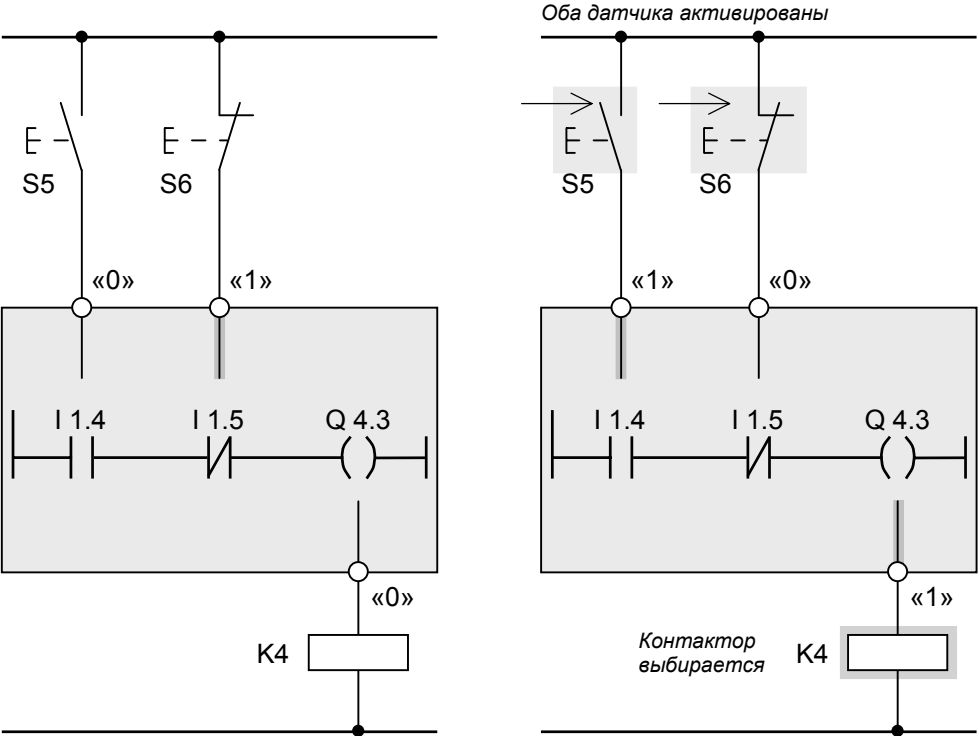


Рисунок 4.9 Действия в зависимости от типа датчика (LAD)

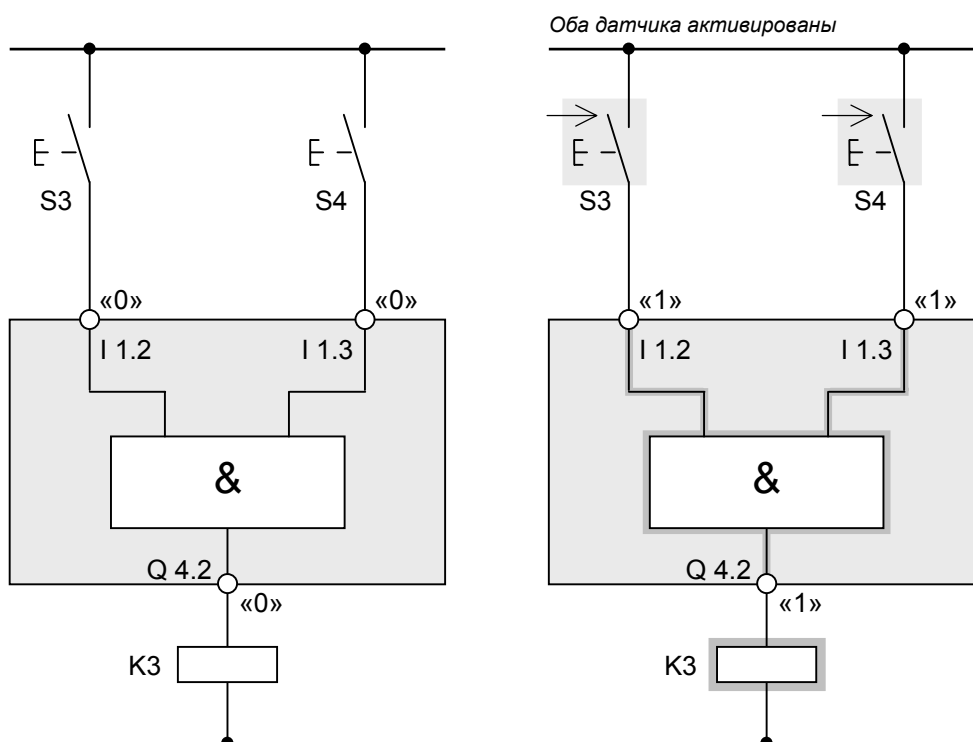
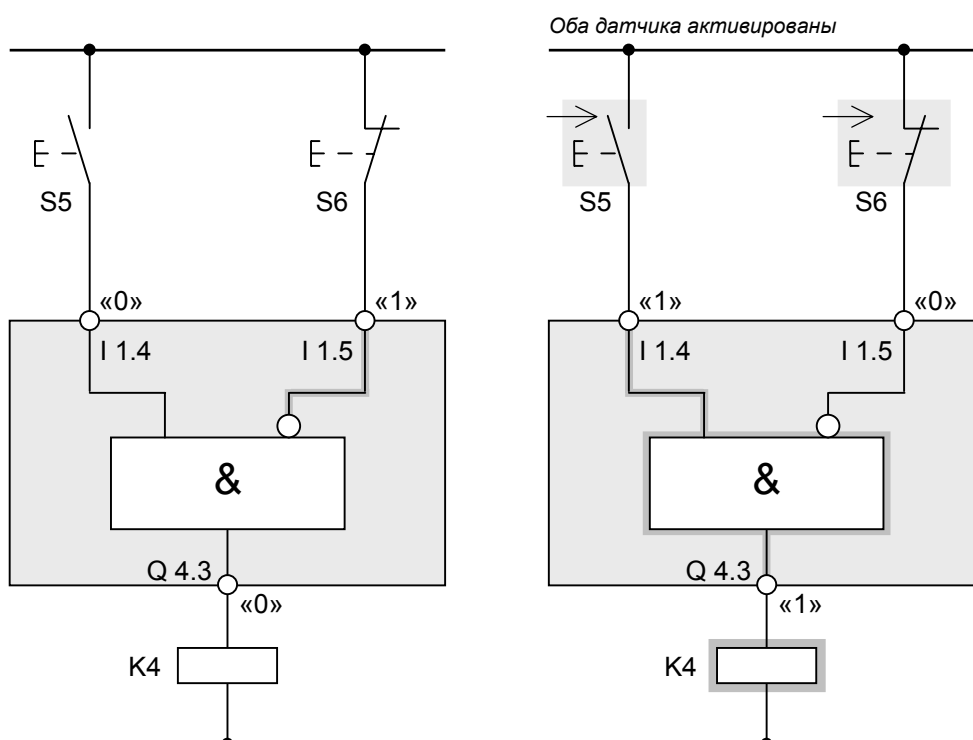
FBD-программирование

Если вы создаете структуру программы таким способом, что вы хотите получить результат сканирования «1» при активировании датчика для того, чтобы в последствии этот результат комбинировать, то для различных видов датчиков вы должны сканировать вход по-разному. NO-контакт при активировании генерирует сигнальное состояние «1» и сканируется непосредственно, когда активация датчика должна вызвать генерирование результата сканирования «1». NC-контакт возвращает при активации сигнальное состояние «0»; если вы хотите получить результат сканирования «1» при активации NC-контакта, то он должен быть инвертирован и затем сканирован.

Поэтому вы также можете сканировать входы, которые должны вызывать действия, даже если они имеют сигнальное состояние «0» (входы с нулевой активностью), и в дальнейшем комбинировать результат сканирования.

Пример на рисунке 4.10 показывает программирование, зависимое от типа датчика. В первом случае два NO-контакта подключены к программируемому контроллеру, во втором случае – один NO-контакт и один NC-контакт. В обоих вариантах контактор, соединенный с выходом, выбирается, если оба датчика активированы. Если активирован NO-контакт, сигнальное состояние на входе «1», и оно сканируется непосредственно (напрямую), поэтому результат сканирования равен «1», когда датчик активирован. Если активированы оба NO-контакта, условие AND выполняется, и контактор выбирается.

Если NC-контакт активирован, сигнальное состояние на входе «0». Для получения результата сканирования «1» вход при сканировании должен быть инвертирован. Во втором случае вам потребуется функция AND с прямым и инвертированным входами, чтобы выбрать контактор при активировании обоих датчиков.

Случай 1: оба датчика являются NO-контактами**Случай 2: один NO- и один NC-контакт****Рисунок 4.10** Действия в зависимости от типа датчика (FBD)