

Глава 5

Функции для работы с памятью

Содержание главы 5

<u>5</u>	<u>Функции для работы с памятью</u>	4
<u>5.1</u>	<u>Катушки LAD</u>	4
<u>5.1.1</u>	<u>Одиночная катушка</u>	4
<u>5.1.2</u>	<u>Катушки установки и сброса</u>	5
<u>5.1.3</u>	<u>Блочный элемент памяти (триггер)</u>	7
<u>5.2</u>	<u>Блочные элементы FBD</u>	11
<u>5.2.1</u>	<u>Присваивание</u>	11
<u>5.2.2</u>	<u>Блочные элементы установки и сброса</u>	12
<u>5.2.3</u>	<u>Блочные элементы памяти</u>	14
<u>5.3</u>	<u>Коннекторы</u>	18
<u>5.3.1</u>	<u>Коннекторы в LAD</u>	18
<u>5.3.2</u>	<u>Коннекторы в FBD</u>	19
<u>5.4</u>	<u>Оценка фронта импульса</u>	21
<u>5.4.1</u>	<u>Как работает функция оценки фронта</u>	21
<u>5.4.2</u>	<u>Функция оценки фронта в LAD</u>	23
<u>5.4.3</u>	<u>Функция оценки фронта в FBD</u>	25
<u>5.5</u>	<u>Бинарный делитель частоты</u>	27
<u>5.5.1</u>	<u>Решение в LAD</u>	27
<u>5.5.2</u>	<u>Решение в FBD</u>	27
<u>5.6</u>	<u>Пример системы управления конвейером</u>	30

5 Функции для работы с памятью

5.1 Катушки LAD

В контактном плане (ladder diagram, LAD) функции для работы с памятью (или операции с памятью, memory functions) используются вместе с последовательными и параллельными схемами с целью воздействия на сигнальные состояния бинарных операндов с помощью результата логической операции (result of logic operation, RLO), генерируемого в CPU.

Доступны следующие функции для работы с памятью (функции памяти):

- Одиночная катушка (coil) как присваивание (назначение) RLO;
- Катушки R и S как индивидуально программируемые операции с памятью;
- Блочные элементы (boxes) RS и SR как функции, работающие с памятью;
- Коннекторы (midline outputs) как промежуточные буферы;
- Катушки P и N как элементы оценки (обнаружения) фронта импульса потока электроэнергии (электрического тока);
- Блочные элементы POS и NEG как элементы оценки фронта операндов.

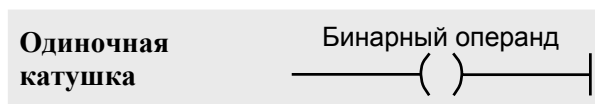
Коннекторы и элементы оценки фронта импульса подробно рассматриваются в последующих главах. Описываемые в данной главе функции для работы с памятью вы можете использовать совместно со всеми бинарными операндами. Ограничения возникают при использовании битов временных локальных данных в качестве меркеров фронта.

Примеры, приведенные в настоящей главе, также представлены на прилагаемой к книге дискете в функциональном блоке 105 программы «Basic functions» («Основные функции»), библиотека «LAD_Book».

Что касается пошагового программирования, то программные элементы для функций для работы с памятью вы найдете в каталоге программных элементов. Команда меню: View → Catalog (Вид → Каталог) [Ctrl + K] или Insert → Program Elements (Вставка → Программные элементы) в «Bit logic» («Битовая логика»).

5.1.1 Одиночная катушка

Одиночная катушка, как терминатор (завершающий элемент) цепи, назначает или направляет (assigns) поток электроэнергии (электрический ток) напрямую к операнду, расположенному при катушке. Функция одиночной катушки зависит от главного реле управления (Master Control Relay, MCR). Если MCR активировано, то бинарному операнду, расположенному над катушкой, назначается сигнальное состояние «0».



Если ток в катушке течет (достигает катушки), то операнд установлен; если тока нет, то операнд не установлен (сброшен) (рисунок 5.1, сеть 1, Network 1). С помощью NOT-контакта перед катушкой вы можете обратить функцию (сеть 2, Network 2).

Кроме того, вы можете направить ток в несколько катушек одновременно, параллельно скомпоновав их с помощью Т-ветви (сеть 3, Network 3). Все операнды, определенные над катушками, реагируют таким же образом. Параллельно можно составить до 16 катушек.

Последующие контакты вы можете скомпоновать в последовательные и параллельные схемы после Т-ветви и перед катушкой (сеть 4, Network 4).

Примеры с применением одиночных катушек приведены в параграфе 4.1 «Последовательные и параллельные схемы (LAD)».

5.1.2 Катушки установки и сброса

Катушки установки и сброса (set coil, reset coil) также могут завершать цепь. Эти катушки становятся активными, только когда через них протекает ток.



Если ток течет в катушке установки, то операнд над катушкой устанавливается в сигнальное состояние «1». Если ток течет в катушке сброса, то операнд над катушкой переустанавливается в сигнальное состояние «0» (сбрасывается). При отсутствии тока в катушке установки или сброса бинарный операнд остается без изменений (рисунок 5.1, сети 5 и 6, Network 5, 6).

Функция катушек установки и сброса зависит от главного реле управления (MCR). Если MCR активировано, то бинарный операнд над катушкой остается неизменным.

Обратите внимание, что операнд, используемый с катушкой установки или сброса, обычно сбрасывается при запуске (полный рестарт - complete restart). В особых случаях сигнальное состояние сохраняется. Это зависит от режима запуска (например, «теплый» рестарт – warm restart), используемого операнда (к примеру, статические локальные данные) и установок в CPU (например, характеристики по сохранению).

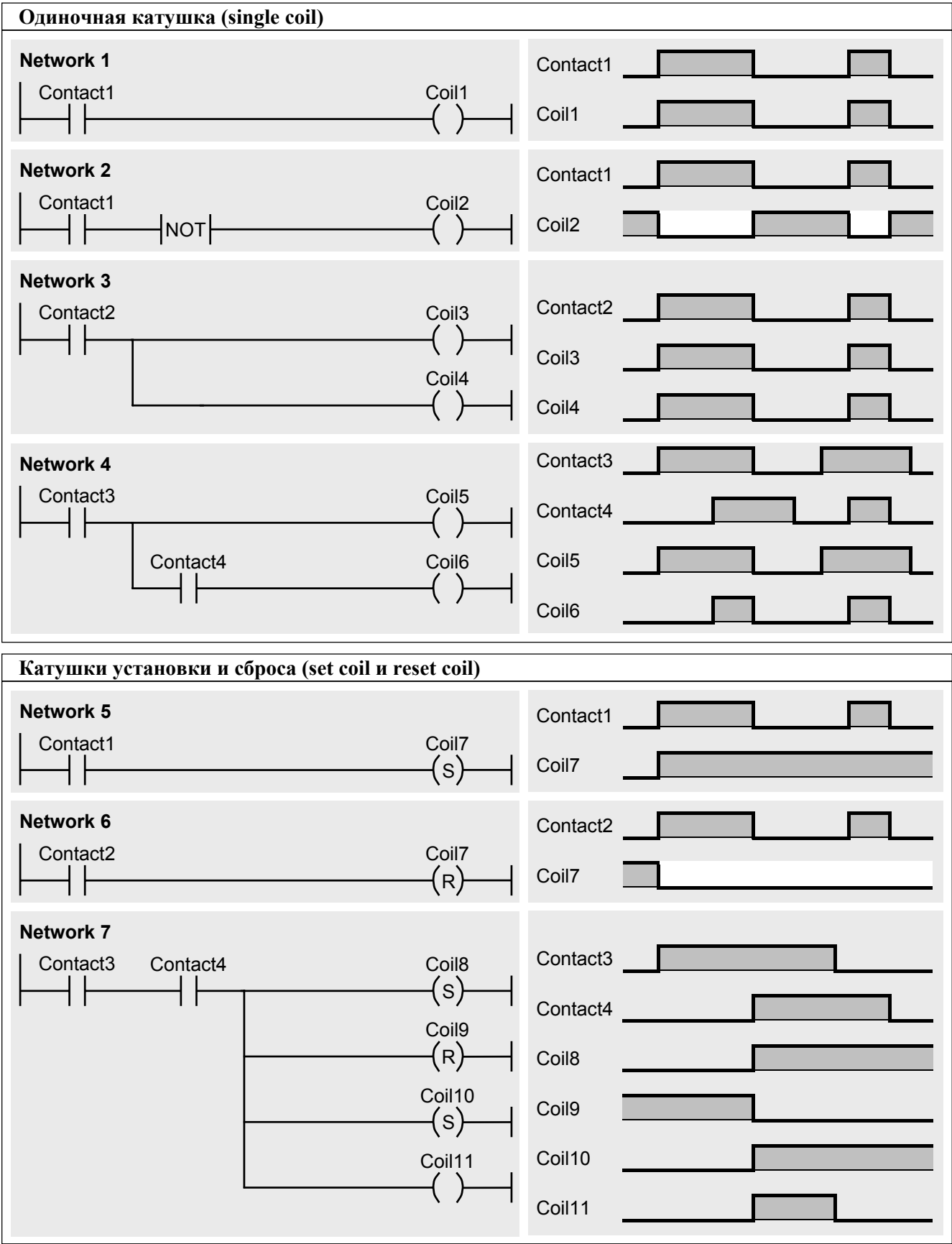


Рисунок 5.1 Одиночная катушка, катушки установки и сброса

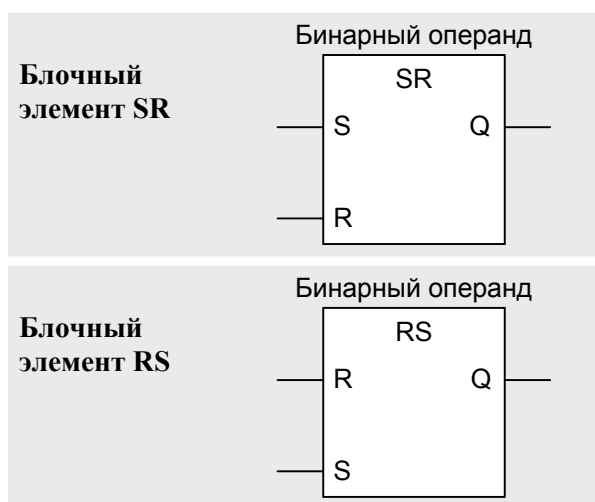
Вы можете сконфигурировать несколько катушек установки и сброса в любой комбинации и вместе с одиночными катушками в одной цепи (сеть 7, Network 7). Для достижения ясности в программировании вам рекомендуется группировать катушки установки и сброса, совместно воздействующие на операнд, попарно и использовать их в каждом случае только один раз. Кроме того, не допускайте дополнительного управления операндами одиночной катушкой.

Как в случае с одиночной катушкой, допустимо поместить контакты после других элементов ветви и перед катушками установки и сброса.

5.1.3 Блочный элемент памяти (триггер)

Функции катушек установки и сброса объединяются в блочном элементе функции для работы с памятью (memory box). Общий бинарный операнд располагается над блочным элементом. Вход S (set input) блочного элемента в данном случае соответствует катушке установки, вход R (reset input) – катушке сброса. Сигнальное состояние двоичного операнда, назначаемое функции для работы с памятью, находится на выходе Q функции памяти.

Имеются два варианта функции памяти: в виде блочных элементов SR (приоритет сброса) и RS (приоритет установки). Кроме различий в обозначении элементы отличаются также компоновкой входов S и R.



Функция памяти установлена (или, точнее, установлен бинарный операнд над блочным элементом памяти), если вход установки имеет сигнальное состояние «1», а вход сброса имеет сигнальное состояние «0». Функция памяти сброшена, если «1» находится на входе сброса, а «0» - на входе установки. Сигнальное состояние «0» на обоих входах не влияет на функцию для работы с памятью. Если оба входа одновременно равны «1», то две имеющиеся функции памяти реагируют по-разному: функция памяти SR сбрасывается, а функция памяти RS – устанавливается.

Поведение блочного элемента памяти зависит от главного реле управления. Если MCR активно, то бинарный операнд блочного элемента памяти сохраняется.

Заметьте, что операнд, используемый с функцией памяти, при запуске (полный рестарт) обычно сбрасывается. В особых случаях сигнальное состояние блочного элемента памяти сохраняется. Это зависит от режима старта (например, «теплый» рестарт), используемого операнда (к примеру, статические локальные данные) и установок в CPU (например, характеристики по сохранению).

Функция для работы с памятью SR (триггер с приоритетом сброса)

В блочном элементе памяти SR приоритет имеет вход сброса (reset input). Приоритет сброса означает, что функция памяти является или остается сброшенной, если ток течет «одновременно» во входе установки и входе сброса. Вход сброса имеет приоритет перед входом установки (рисунок 5.2а, сеть 8, Network 8).

Так как операторы выполняются последовательно, CPU изначально устанавливает операнд памяти (memory operand), потому что вход установки обрабатывается первым, но потом снова сбрасывает его, когда обрабатывает вход сброса. Пока оставшаяся программа обрабатывается, операнд памяти остается обнуленным.

Если операнд памяти является выходом, эта недолговременная установка имеет место только в выходной таблице образа процесса, а (внешний) выход в соответствующем модуле выхода остается неизменным. CPU не передает выходную таблицу образа процесса в модули выходов до конца программного цикла.

Функция памяти с приоритетом сброса является «нормальной» формой функции для работы с памятью, так как состояние сброса (сигнальное состояние «0») обычно безопаснее или менее рискованное состояние.

Функция для работы с памятью RS (триггер с приоритетом установки)

В блочном элементе памяти RS приоритет имеет вход установки (set input). Приоритет установки означает, что функция памяти является или остается установленной, если ток течет «одновременно» во входе установки и входе сброса. Поэтому вход установки имеет приоритет перед входом сброса (рисунок 5.2а, сеть 9, Network 9).

В соответствии с последовательным исполнением инструкций CPU сбрасывает операнд памяти с входом сброса, обрабатываемым первым, но затем вновь устанавливает его, когда обрабатывает вход установки. Операнд памяти остается установленным, пока обрабатывается оставшаяся программа.

Если операнд памяти является выходом, эта недолговременная установка имеет место только в выходной таблице образа процесса, а (внешний) выход в соответствующем модуле выхода остается неизменным. CPU не передает выходную таблицу образа процесса в модули выходов до конца программного цикла.

Приоритет установки при использовании функции для работы с памятью является исключением. Он используется, к примеру, в реализации буфера сообщения о сбое, если на входе установки постоянное (still) текущее сообщение об ошибке должно

продолжить установку функции памяти, несмотря на подтверждение на входе сброса.

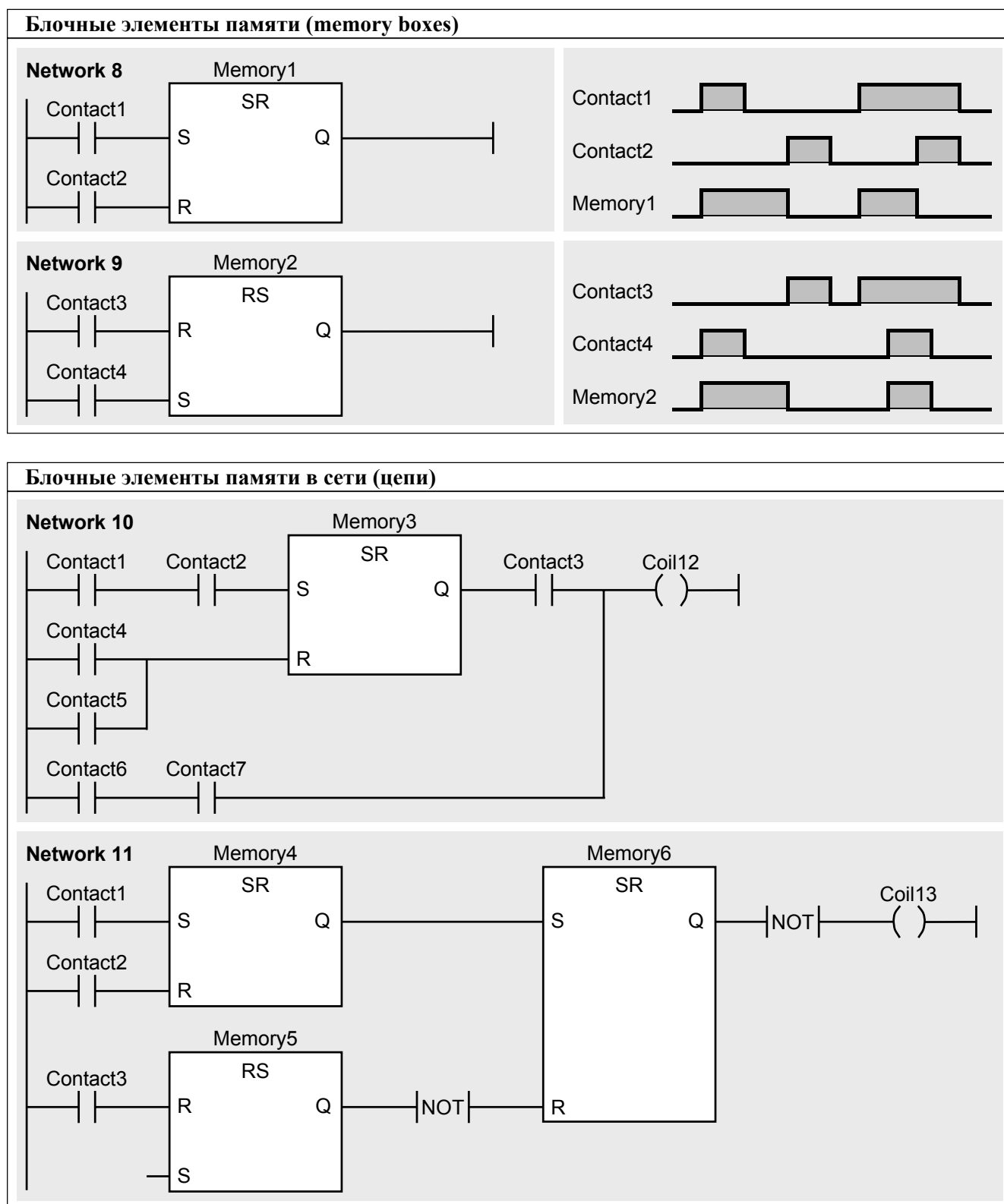


Рисунок 5.2а Функции памяти (LAD)

Функция памяти внутри цепи

Вы также можете поместить блочный элемент памяти внутри цепи. Контакты могут быть соединены последовательно и параллельно и при входах, и при выходах (рисунок 5.2а, сеть 10, Network 10). Второй вход блочного элемента памяти допускается оставлять неподключенным. В рамках одной цепи можно соединять несколько блочных элементов памяти между собой. Вы можете компоновать блочные элементы памяти последовательно и параллельно (сеть 11, Network 11). Расположить функцию памяти вы можете после Т-ветви или в звене, которое начинается от левой несущей (питающей шины).

Функция памяти с блокировкой

В релейной логической диаграмме функция памяти обычно реализуется с помощью блокировки или защелки (latching) управляемого выхода. Этот метод также может применяться при программировании контактного плана. Однако, он имеет недостаток: функция памяти не распознается непосредственно.

Сети 12 и 13 (Network 12, 13) на рисунке 5.2б приводят оба типа функции для работы с памятью, приоритет установки и приоритет сброса, используя блокировку. Принцип блокировки является простым. Бинарный операнд, управляемый катушкой, сканируется, и это сканирование («контакт катушки») соединяется параллельно с условием (состоянием) установки. Если *Contact1* (Контакт1) замыкается, *Coil14* (Катушка14) возбуждается и замыкает контакт, параллельный контакту *Contact1* (Контакт1). Если *Contact1* (Контакт1) снова размыкается, *Coil14* (Катушка14) остается под напряжением. Нагрузка с *Coil14* (Катушка14) снимается, если размыкается *Contact2* (Контакт2). Если сигнальное состояние «1» присутствует на *Contact1* (Контакт1) и *Contact2* (Контакт2), ток к катушке не течет (приоритет сброса). Эта ситуация выглядит по-другому на нижней схеме. Если сигнальное состояние «1» присутствует на *Contact3* (Контакт3) и *Contact4* (Контакт4), ток к катушке течет (приоритет установки).

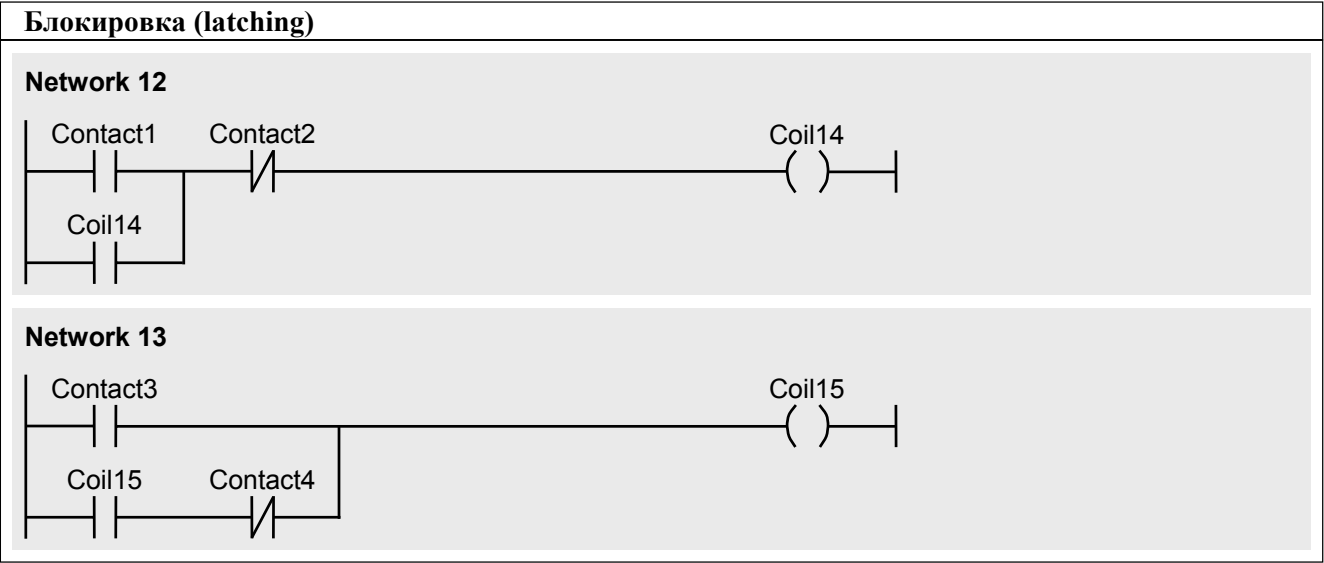


Рисунок 5.2б Функции памяти (LAD)

5.2 Блочные элементы FBD

В FBD блочные элементы памяти используются вместе с операциями бинарной логики с целью влияния на сигнальные состояния бинарных операндов с помощью результата логической операции (RLO), генерируемого CPU.

Доступны следующие функции для работы с памятью:

- Блочный элемент присваивания (assign box) для динамического управления;
- Блочные элементы установки (set) и сброса (reset) как индивидуально программируемые функции памяти;
- Блочные элементы RS и SR как вполне законченные функции памяти;
- Блочный элемент коннектора (midline output box) как промежуточный буфер;
- Блочные элементы P и N как элементы оценки (обнаружения) фронта результата логической операции;
- Блочные элементы POS и NEG как определители фронта операндов.

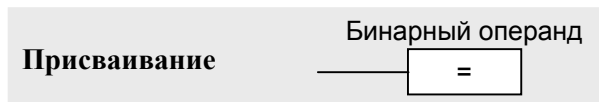
Блочные элементы коннектора и определители фронта подробно обсуждаются в следующих главах.

Вы можете использовать функции памяти, описываемые в данной главе, совместно со всеми бинарными операндами. Ограничения накладываются при использовании битов временных локальных данных в качестве меркеров фронта (edge memory bits).

Примеры, показанные в этой главе, также представлены на дискете, прилагаемой к книге, в функциональном блоке FB 105 раздела «Basic Functions» («Основные функции»), библиотека «FBD_Book». Программные элементы функций, работающих с памятью, для пошагового программирования вы найдете в каталоге программных элементов, пункт меню View → Catalog (Вид → Каталог) [Ctrl + K] или Insert → Program Elements (Вставка → Программные элементы) в разделе «Bit logic» («Битовая логика»).

5.2.1 Присваивание

Блочный элемент присваивания (assign) как терминатор (завершающий элемент) цепи присваивает (назначает) результат логической операции непосредственно операнду, смежному с блочным элементом. Если RLO равен «1» на входе блочного элемента присваивания, то бинарный операнд устанавливается; если RLO равен «0», то операнд сбрасывается. Действие блочного элемента присваивания зависит от главного реле управления (MCR). Если MCR активировано, то бинарному операнду над блочным элементом назначается сигнальное состояние «0».



Работу блочного элемента присваивания разъясняют несколько примеров на рисунке 5.3.

Сеть 1 (Network 1): операнд Output1 (Выход1) непосредственно принимает сигнальное состояние операнда Input1 (Вход1).

Сеть 2 (Network 2): вы можете применить инвертирование, чтобы обратить функцию блочного элемента присваивания.

Сеть 3 (Network 3): вы можете направить RLO нескольким блочным элементам одновременно путем вставки Т-ветви и компоновки блочных элементов с соответствующими операндами один под другим («множественный вывод или мультивывод»). Все операнды над блочными элементами реагируют таким же образом.

Сеть 4 (Network 4): вы можете вставить бинарные функции между Т-ветвью и завершающим блочным элементом, расширяя таким образом логическую операцию дополнительными функциональными блочными элементами.

Дополнительные примеры по блочному элементу присваивания вы найдете в параграфе 4.2 «Операции бинарной логики (FBD)».

5.2.2 Блочные элементы установки и сброса

Блочные элементы установки и сброса (set box и reset box) также могут завершать логическую операцию (функциональный план). Эти блочные элементы активируются только тогда, когда результат логической операции, направляющийся в блочный элемент, равен «1».



Если RLO, направляющийся в блочный элемент установки, равен «1», то операнд над элементом устанавливается в сигнальное состояние «1». Если RLO, идущий в блочный элемент сброса, равен «1», то операнд над элементом переходит в сигнальное состояние «0». Если RLO, передаваемый в блочный элемент установки или сброса, равен «0», бинарный операнд остается неизменным. Функция блочных элементов установки и сброса зависит от главного реле управления (MCR). Если MCR активировано, то бинарный операнд над блочным элементом воздействию не подвергается.

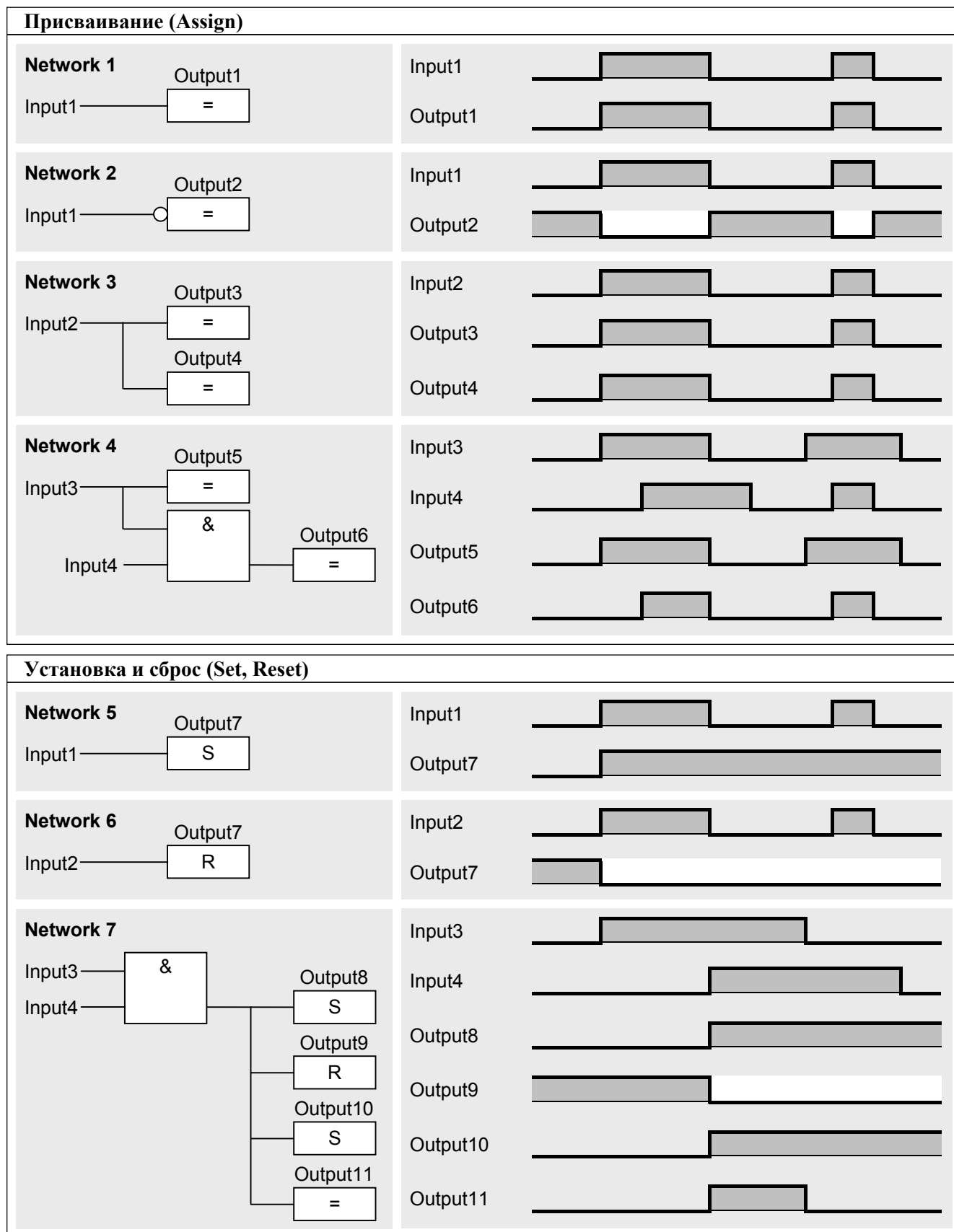


Рисунок 5.3 Присваивание, установка и сброс (FBD)

На рисунке 5.3 показаны несколько примеров работы блочных элементов установки и сброса.

Сеть 5 (Network 5): операнд *Output7 (Выход7)* устанавливается, когда операнд *Input1 (Вход1)* равен «1». Когда *Input1 (Вход1)* возвращается в сигнальное состояние «0», *Output7 (Выход7)* остается установленным.

Сеть 6 (Network 6): операнд *Output7 (Выход7)* сбрасывается, когда операнд *Input2 (Вход2)* равен «1». Когда *Input2 (Вход2)* возвращается в сигнальное состояние «0», *Output7 (Выход7)* остается сброшенным.

Сеть 7 (Network 7): вы можете сгруппировать несколько блочных элементов установки и сброса в любой комбинации и вместе с блочными элементами присваивания в одном плане после Т-ветви. Как и в случае с элементом присваивания, вы также можете запрограммировать бинарные функции после Т-ветви и перед элементами установки и сброса.

Для достижения прозрачности в программировании рекомендуется попарно группировать блочные элементы установки и сброса, воздействующие на операнд, и использовать их в каждом случае только один раз. Также следует избегать управления этими операндами со стороны блочного элемента присваивания.

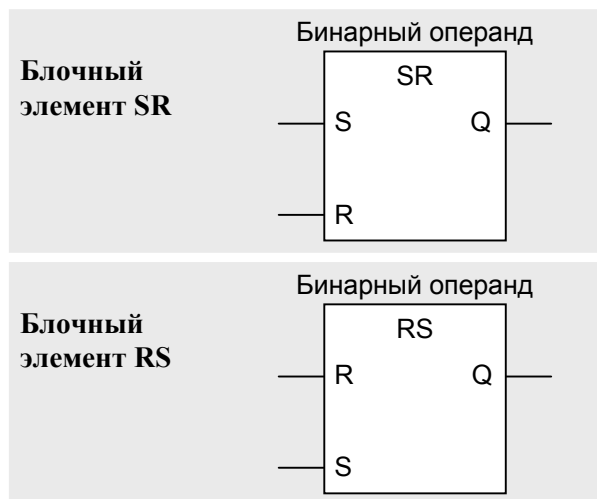
Обратите внимание на то, что операнд, используемый с блочными элементами установки и сброса, при запуске обычно сбрасывается (полный рестарт – complete restart). В особых случаях сигнальное состояние сохраняется. Это зависит от режима запуска (например, «теплый» рестарт – warm restart), задействованного операнда (например, статические локальные данные) и установок в CPU (таких как характеристики, отвечающие за сохранение).

5.2.3 Блочные элементы памяти

Функции блочных элементов установки и сброса объединены в блочном элементе функции памяти. Общий бинарный операнд расположен над блочным элементом. Вход S блочного элемента соответствует в данном случае блочному элементу установки (set box), а вход R – блочному элементу сброса (reset box). Сигнальное состояние двоичного операнда, назначаемое функции памяти, подается на выход Q функции памяти.

Предоставляются два варианта функции, работающих с памятью: в виде блочного элемента SR (приоритет сброса) и в виде блочного элемента RS (приоритет установки). Кроме обозначения элементы также отличаются друг от друга компоновкой входов S и R.

Функция памяти установлена (или точнее бинарный операнд над блочным элементом памяти установлен), когда вход установки (set input) равен «1» и вход сброса (reset input) равен «0». Функция памяти обнулена, когда вход сброса установлен в «1», а на входе установки – «0».



Сигнальное состояние «0» на обоих входах не оказывает влияния на функцию памяти. Если оба входа имеют сигнальное состояние «1» одновременно, рассматриваемые две функции памяти реагируют по-разному: функция памяти SR обнуляется, а функция памяти RS устанавливается. Функционирование блочного элемента памяти зависит от главного реле управления (MCR). Если MCR активировано, то бинарный операнд блочного элемента памяти остается без изменений.

Заметьте, что операнд, используемый с функцией памяти, при запуске обычно сброшен (полный рестарт). В особых случаях сигнальное состояние блочного элемента памяти сохраняется. Это зависит от режима запуска (например, теплый рестарт), используемого операнда (к примеру, статические локальные данные) и установок в CPU (таких как характеристик, отвечающих за сохранение).

Функция для работы с памятью SR

В блочном элементе памяти SR приоритет имеет вход сброса (reset input). Приоритет сброса означает, что функция памяти является или остается сброшенной, если RLO равен «1» на входах установки и сброса «одновременно». Таким образом, вход сброса имеет приоритет перед входом установки (рисунок 5.4, сеть 8, Network 8).

Так как операторы выполняются последовательно, CPU сначала устанавливает операнд памяти, потому что вход установки обрабатывается первым, но затем сбрасывает его снова, когда обрабатывает вход сброса. Пока остальная программа обрабатывается, операнд памяти остается обнуленным.

Если операнд памяти является выходом, эта кратковременная установка имеет место только в выходной таблице образа процесса, а (внешний) выход в соответствующем модуле выхода остается неизменным. CPU не передает выходную таблицу образа процесса в модули выходов до конца программного цикла.

Функция памяти с приоритетом сброса является «нормальной» формой функции для работы с памятью, так как состояние сброса (сигнальное состояние «0») является обычно безопаснее или менее рискованным.

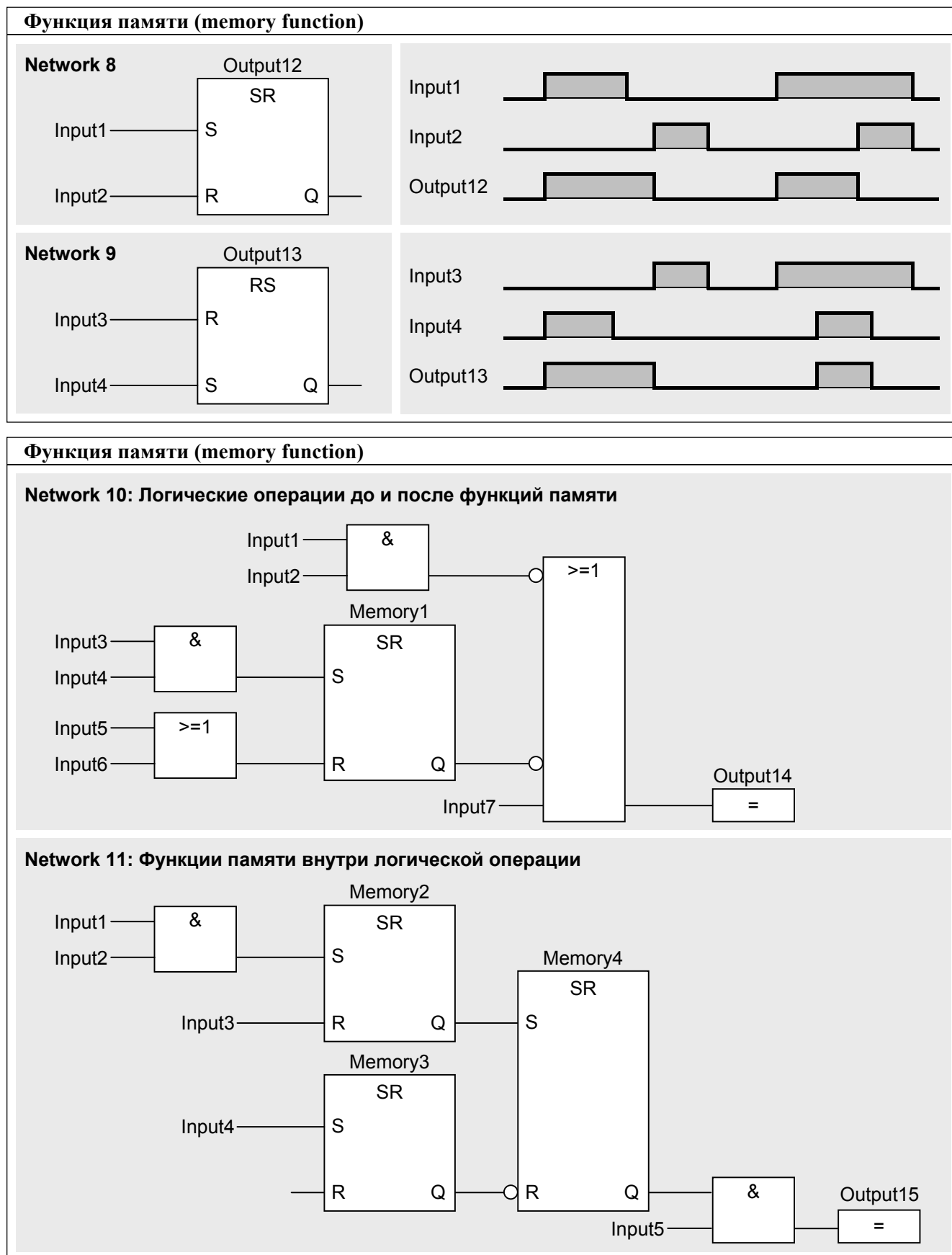


Рисунок 5.4 Функции памяти (FBD)

Функция для работы с памятью RS

В блочном элементе памяти RS приоритет имеет вход установки (set input). Приоритет установки означает, что функция памяти является или остается установленной, если RLO равен «1» на входах установки и сброса «одновременно». Поэтому вход установки имеет приоритет перед входом сброса (рисунок 5.4, сеть 9, Network 9).

Так как операторы выполняются последовательно, CPU сначала сбрасывает операнд памяти, потому что вход сброса обрабатываемым первым, затем вновь устанавливает его, когда обрабатывает вход установки. Операнд памяти остается установленным, пока обрабатывается остальная программа.

Если операнд памяти является выходом, эта недолговременная установка имеет место только в выходной таблице образа процесса, а (внешний) выход в соответствующем модуле выхода остается неизменным. CPU не передает выходную таблицу образа процесса в модули выходов до конца программного цикла.

Приоритет установки скорее является исключением, чем правилом. Он используется, к примеру, в реализации буфера сообщения о сбое, если на входе установки постоянное (still) текущее сообщение об ошибке должно продолжить установку функции памяти, несмотря на подтверждение на входе сброса.

Функция памяти внутри логической операции

Вы также можете поместить блочный элемент памяти внутри логической операции. Можно запрограммировать бинарные функции и на входах, и на выходе (рисунок 5.4, сеть 10, 11, Network 10, 11). Можно оставить второй вход блочного элемента памяти несоединенным. Внутри логической операции вы можете соединить несколько блочных элементов памяти. Блочные элементы памяти могут быть размещены один за другим или друг под другом после Т-ветви.

5.3 Коннекторы

Коннекторы (midline outputs) являются промежуточными буферами в контактном или функциональном планах. RLO, действительный для коннектора, хранится в операнде над этим коннектором. Этот операнд может быть снова опрошен в другой точке программы, что позволяет вам также постобрабатывать (post-process) RLO, действительный для коннектора, в другом месте программы.

Для промежуточного хранения двоичных результатов применимы следующие бинарные операнды:

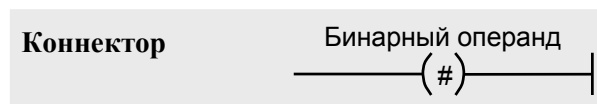
- Вы можете использовать биты временных локальных данных, если промежуточный результат вам требуется только внутри блока. Все кодовые блоки имеют временные локальные данные.
- Биты статических локальных данных доступны только в рамках функционального блока; они хранят сигнальное состояние до их повторного использования, даже за границами блока.
- Меркеры доступны глобально в определяемом моделью CPU фиксированном количестве; для достижения прозрачности программирования старайтесь избегать многократного использования меркеров (одних и тех же меркеров для различных задач).
- Биты данных в глобальных блоках данных также являются доступными на протяжении всей программы, но требуют, чтобы перед их использованием соответствующие блоки данных были открыты (даже если связаны через массовую адресацию).

Функционирование коннектора зависит от главного реле управления (MCR). Если MCR активировано, то бинарному операнду, соотнесенному с коннектором, присваивается бинарное состояние «0». Тогда и RLO вслед за коннектором будет равен «0» (то есть «поток электроэнергии» отсутствует).

Замечание: вы можете заменить сверхоперативную память (scratchpad memory) на временные локальные данные, доступные в каждом блоке.

5.3.1 Коннекторы в LAD

Коннектор является одиночной катушкой в цепи. RLO, действительный для этой точки (электрический ток, который течет в цепи, в данной точке), хранится в двоичном операнде над коннектором. Сам коннектор не оказывает влияния на электрический ток.



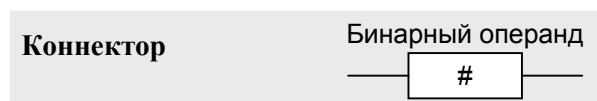
Вы можете сканировать бинарный операнд над коннектором в другой точке программы с помощью NO- и NC-контактов. В одной цепи может быть запрограммировано несколько коннекторов.

Коннектор можно поместить в ветви, которая начинается на левой несущей (шине питания). Он может также следовать за Т-ветвью, но не может завершать цепь; для этой цели применяется одиночная катушка.

Рисунок 5.5 иллюстрирует пример того, как промежуточный результат сохраняется в коннекторе. RLO из цепи, формируемый контактами *Contact1* (Контакт1), *Contact2* (Контакт2), *Contact4* (Контакт4) и *Contact5* (Контакт5), сохраняется в коннекторе *Midl_out1* (Коннектор1). Если условие логической операции выполняется (ток течет в коннекторе), и если *Contact3* (Контакт3) замкнут, то *Coil16* (Катушка16) возбуждается. Хранимый RLO используется в следующей сети (network) двумя способами. С одной стороны, производится проверка выполнения условия логической операции и битовой логической комбинации, осуществленной с *Contact6* (Контакт6), а с другой стороны, производится проверка невыполнения условия логической операции и битовой логической комбинации, осуществленной с *Contact7* (Контакт7).

5.3.2 Коннекторы в FBD

Коннектор – это блочный элемент присваивания внутри логической операции. RLO, действительный для этой точки, хранится в бинарном операнде над его блочным элементом.



Вы можете опросить бинарный операнд над коннектором в другой точке программы. В одном функциональном плане может быть запрограммировано несколько коннекторов. Блочный элемент коннектора не должен завершать логическую операцию; для этой цели служит блочный элемент присваивания.

Сети 12 и 13 (Network 12, 13) на рисунке 5.5 показывают, как промежуточный результат сохраняется в коннекторе. RLO из цепи, формируемый входами *Input1* (Вход1), *Input2* (Вход2), *Input3* (Вход3) и *Input4* (Вход4), сохраняется в коннекторе *Midl_out1* (Коннектор1). Если условие логической операции выполняется, и если сигнальное состояние входа *Input5* (Вход5) равно «1», то *Coil16* (Катушка6) активируется. Хранимый RLO используется в следующей сети (network) двумя способами. С одной стороны, производится проверка выполнения условия логической операции и битовой логической комбинации, осуществленной с *Input6* (Вход6), а с другой сто-

роны, производится проверка невыполнения условия логической операции и битовой логической комбинации, осуществленной с *Input7 (Вход7)*.

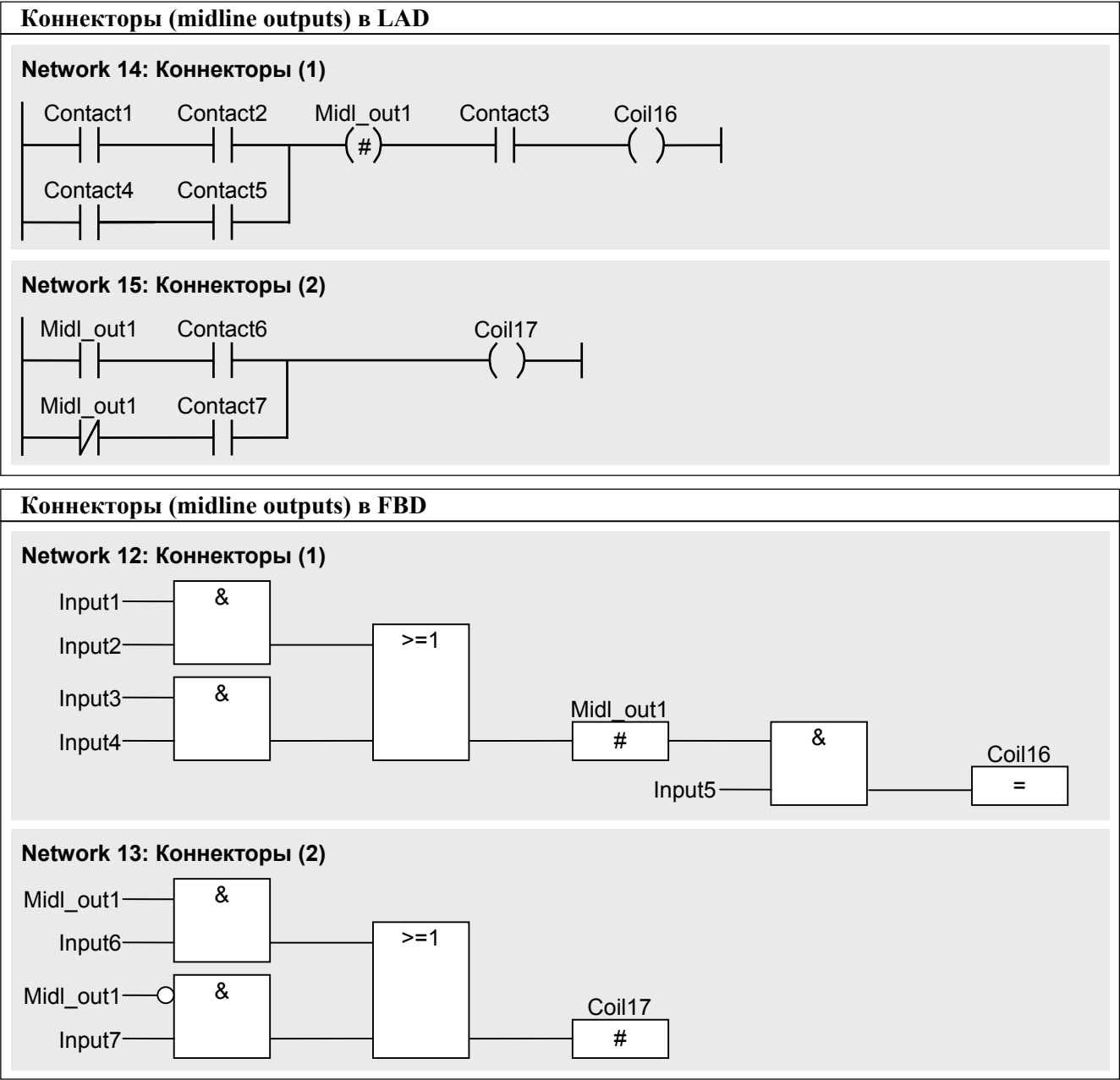


Рисунок 5.5 Коннекторы

5.4 Оценка фронта импульса

5.4.1 Как работает функция оценки фронта

С помощью функции оценки фронта (edge evaluation) вы можете обнаружить изменение в сигнальном состоянии, фронт сигнала. Фронт является положительным (возрастающим), когда сигнал меняется с «0» на «1». В противоположном случае фронт отрицателен (падает).

В цепи LAD и плане FBD эквивалентом оценки фронта является импульсный контактный элемент (pulse contact element). Если этот импульсный контактный элемент испускает импульс при включенном реле, это соответствует возрастающему фронту. Импульс из импульсного контактного элемента при выключении соответствует спадающему фронту.

Обнаружение фронта сигнала (изменение в сигнальном состоянии) реализуется в программе. CPU сравнивает текущий RLO (например, результат проверки входа) с сохраненным RLO. Если сигнальные состояния различны, то фронт сигнала присутствует.

Хранимый RLO расположен в «меркере фронта» («edge memory bit») (он не обязательно должен быть меркером). Это должен быть операнд, чье сигнальное состояние должно быть доступно, когда снова встречается оценка фронта (в следующем программном цикле), и который не используется в каком-либо другом месте программы. В качестве операндов могут применяться меркеры, биты данных в глобальных блоках данных и биты статических локальных данных в функциональных блоках.

Меркер фронта сохранит «старый» RLO, с которым CPU совершил последнюю обработку оценки фронта. Если фронт сигнала в настоящий момент присутствует, то есть если текущий RLO отличается от сигнального состояния меркера фронта, то CPU корректирует сигнальное состояние меркера фронта путем присваивания ему «значения» «нового» RLO. При следующей обработке оценки фронта (обычно в следующем программном цикле) сигнальное состояние меркера фронта то же, что и у текущего RLO (если оно между тем не изменилось), и CPU больше не производит обнаружение фронта.

Обнаруженный фронт индицируется результатом логической операции (RLO) после оценки фронта. Если CPU выявляет фронт сигнала, то он устанавливает RLO в «1» после оценки фронта (то есть ток течет). Если фронта сигнала не обнаружено, то RLO равен «0».

Следовательно, сигнальное состояние «1» после оценки фронта означает «фронт обнаружен» («edge detected»). Присутствие сигнального состояния «1» кратковременно, обычно в течение только одного программного цикла. Так как CPU не обнаруживает (не производит обнаружения) фронт в следующем цикле (если «входной RLO» оценки фронта не меняется), он устанавливает RLO обратно в «0» после оценки фронта.

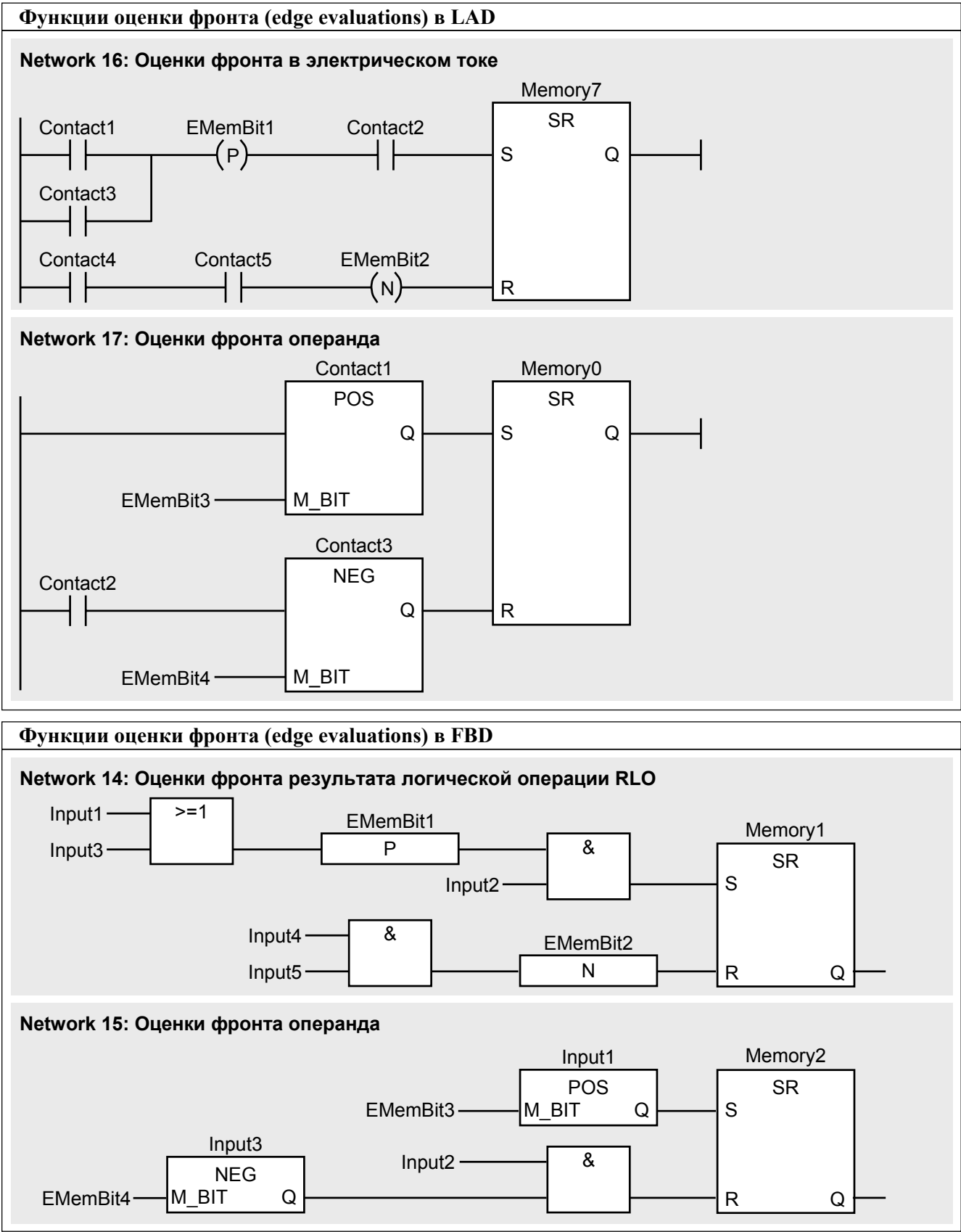
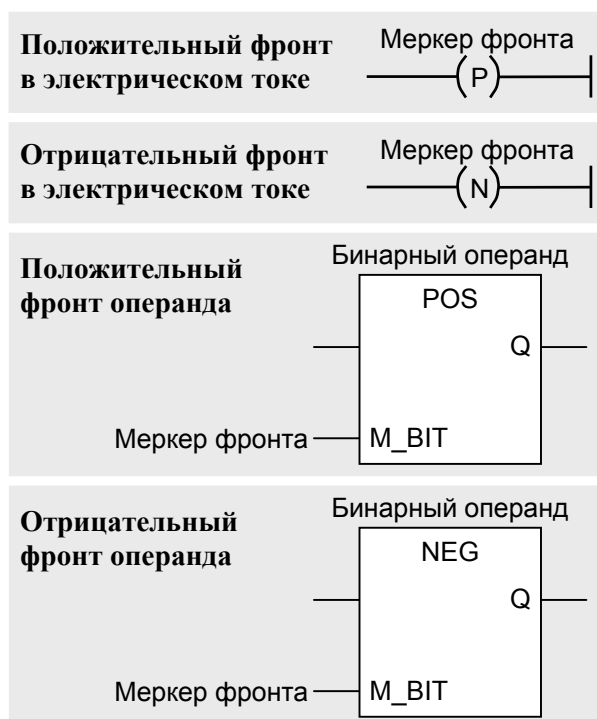


Рисунок 5.6 Функции оценки фронта

Обратите внимание на рабочие параметры оценки фронта, когда CPU запускается. Если фронт не должен быть обнаружен, RLO перед оценкой фронта должен совпадать с сигнальным состоянием меркера фронта при включенном CPU. При определенных обстоятельствах меркер фронта должен быть сброшен в процедуре запуска (в зависимости от требуемого режима работы и используемых операндов).

5.4.2 Функция оценки фронта в LAD

Язык программирования LAD для оценки фронта предоставляет четыре различных элемента:



Вы можете обрабатывать RLO сразу после оценки фронта, то есть, к примеру, сохранять его с помощью катушки установки, комбинировать его с использованием расположенных далее контактов или сохранять его в бинарном операнде (так называемом «импульсном меркере», «pulse memory bit»).

Импульсный меркер применяется, когда RLO из оценки фронта должен использоваться в другом месте программы; это, так сказать, промежуточный буфер для выявленного фронта (импульсный контактный элемент в диаграмме схемы). Операндами, применимыми в качестве импульсного меркера, являются меркеры, биты данных в глобальных блоках данных и биты временных и статических локальных данных.

Оценка фронта тока

Оценка фронта тока отображается катушкой, содержащей Р (для положительного, возрастающего фронта) или N (для отрицательного, спадающего фронта). Над катушкой располагается меркер фронта, бинарный операнд, в котором хранится «старый» RLO из предыдущей оценки фронта. Оценка фронта, как эта, обнаруживает изменение в потоке электроэнергии (электрическом токе) с «ток течет» на «ток не течет» и наоборот.

Пример на рисунке 5.6 в сети 16 (Network 16) показывает оценку положительного и отрицательного фронта. Если параллельная цепь, состоящая из *Contact1* (Контакт1) и *Contact3* (Контакт3), выполняется, оценка фронта испускает короткий импульс с помощью *EMemBit1* (МеркерФронта1). Если *Contact2* (Контакт2) замкнут в этот момент, то *Memory7* (Память7) устанавливается. *Memory7* (Память7) сбрасывается вновь импульсом от *EMemBit2* (МеркерФронта2), если последовательная цепь, состоящая из *Contact4* (Контакт4) и *Contact5* (Контакт5), прерывает электрический ток.

Вы можете запрограммировать оценку фронта с использованием катушки после Т-ветви или в цепи, которая начинается на левой несущей (питающей шине). Она не может быть помещена непосредственно у левой несущей (питающей шины).

Оценка фронта операнда

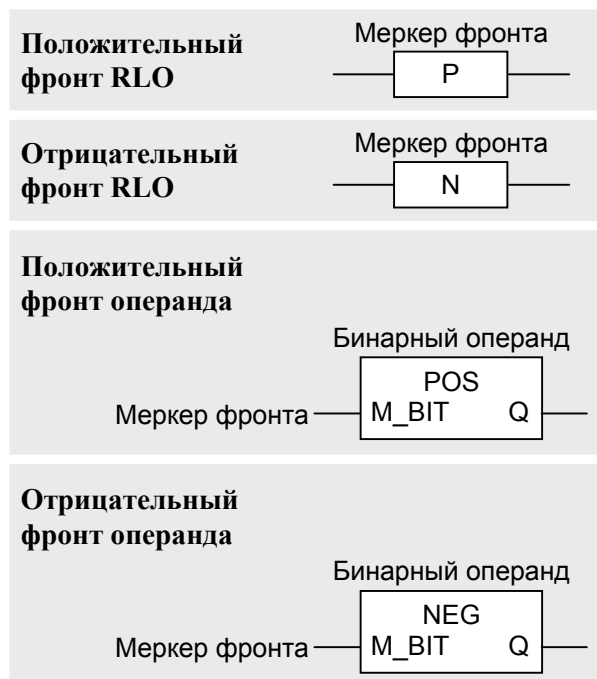
LAD представляет оценку фронта операнда с использованием блочного элемента (box). Над блочным элементом находится операнд, чье изменение сигнального состояния должно быть выявлено. Меркер фронта, который хранит «старое» сигнальное состояние из предыдущего программного цикла, расположен у входа M_BIT.

Элемент оценки фронта имеет немаркированный вход и выход Q и в таком виде «вставляется» в цепь вместо контакта. Если ток течет в немаркированном входе, выход Q при фронте испускает импульс; если в данном входе нет тока, выход Q остается обнуленным. Вы можете вставить эту оценку фронта на место любого контакта, даже в параллельную ветвь, которая не начинается на левой несущей (питающей шине).

Рисунок 5.6 демонстрирует использование оценки фронта операнда на примере сети 17 (Network 17). Оценка фронта в верхней ветви генерирует импульс, если операнд *Contact1* (Контакт1) изменяет свое сигнальное состояние с «0» на «1» (положительный фронт). Этот импульс устанавливает *Memory0* (Память0). Оценка фронта всегда активируется прямым соединением немаркированного входа с левой несущей (питающей шиной). Оценка фронта тока активируется контактом *Contact2* (Контакт2). Если оценка фронта активируется подачей «1» на немаркированный вход, то она испускает импульс при смене сигнального состояния бинарного операнда *Contact3* (Контакт3) с «1» на «0» (отрицательный фронт).

5.4.3 Функция оценки фронта в FBD

Язык программирования FBD для оценки фронта предоставляет четыре различных элемента:



RLO после оценки фронта может быть непосредственно обработан, к примеру, сохранен с помощью катушки установки, использован в комбинации с последующими двоичными функциями или назначен бинарному операнду (так называемому «импульсному меркеру»). Импульсный меркер используется, когда RLO из оценки фронта должен обрабатываться в другом месте программы; это, так сказать, промежуточный буфер для выявленного фронта. Операндами, пригодными для использования в качестве импульсного меркера, могут быть биты данных в глобальных блоках данных и биты временных и статистических локальных данных. После Т-ветви оценка фронта не может быть использована.

Оценка фронта RLO

Оценка фронта RLO отображается при помощи блочного элемента, содержащего P (для положительного, возрастающего фронта) или N (для отрицательного, спадающего фронта). Над блочным элементом указывается меркер фронта, бинарный операнд, содержащий «старый» RLO из предыдущего цикла. Оценка фронта как эта обнаруживает изменение RLO внутри схемы с RLO «1» на RLO «0» и наоборот.

Пример, приведенный на рисунке 5.6, сеть 14 (Network 14), показывает оценку положительного и отрицательного фронтов. Когда функция OR, состоящая из *Input1* (*Вход1*) и *Input3* (*Вход3*), выполняется, оценка фронта генерирует короткий импульс с помощью *EMemBit1* (*МеркерФронт1*). Если *Input2* (*Вход2*) равен «1» в данном случае, то *Memory1* (*Память1*) устанавливается. *Memory1* (*Память1*) вновь сбрасы-

вается импульсом из *EMemBit2* (*МеркерФронта2*), когда функция AND, включающая в себя *Input4* (*Вход4*) и *Input5* (*Вход5*), больше не выполняется.

Оценка фронта операнда

Оценка фронта операнда располагается в начале операции бинарной логики. Над блочным элементом указывается операнд, чье изменение сигнального состояния должно быть оценено. Меркер фронта, который хранит «старое» сигнальное состояние из последнего программного цикла, располагается у входа M_BIT. Выход Q равен «1», когда CPU обнаруживает изменение сигнального состояния операнда.

Сеть 15 (Network 15) на рисунке 5.6 демонстрирует оценку фронта операнда. Верхний элемент оценки фронта POS испускает импульс, когда *Input1* (*Вход1*) переходит из «0» в «1» (положительный фронт). Нижняя оценка фронта NEG генерирует импульс, если бинарный операнд, когда *Input3* (*Вход3*) переходит из «1» в «0» (отрицательный фронт). Когда операнд *Input2* (*Вход2*) равен «1», этот импульс обнуляет *Memory2* (*Память2*).

5.5 Бинарный делитель частоты

Бинарный делитель частоты (binary scaler), или счетчик, имеет один вход и один выход. Если сигнал на входе бинарного делителя частоты меняет свое состояние, например с «0» на «1», выход также меняет свое состояние (рисунок 5.7). Это «новое» сигнальное состояние сохраняется до следующего, в нашем случае положительного, изменения сигнального состояния. Только после этого происходит изменение сигнального состояния на выходе. Это означает, что на выходе бинарного делителя частоты входная частота уменьшается на половину.

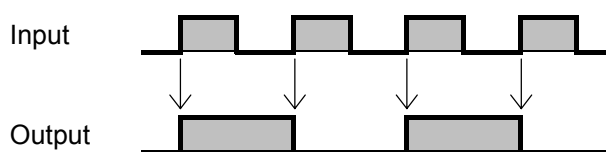


Рисунок 5.7 Импульсная диаграмма бинарного делителя частоты

5.5.1 Решение в LAD

Существует много различных способов решения этой задачи, два из которых представлены ниже.

Первое решение использует функции памяти (рисунок 5.8, сети 8 и 9, Network 8, 9). Если сигнальное состояние операнда *Input_1* (*Вход_1*) «1», то операнд *Output_1* (*Выход_1*) устанавливается; операнд *Memory_1* (*Память_1*) все еще обнулен. Если сигнальное состояние операнда *Input_1* (*Вход_1*) меняется на «0», *Memory_1* (*Память_1*) также устанавливается (*Output_1* (*Выход_1*) теперь «1»). Если снова *Input_1* (*Вход_1*) равен «1», *Output_1* (*Выход_1*) опять обнуляется (*Memory_1* (*Память_1*) теперь «1»). Если *Input_1* (*Вход_1*) еще раз будет равен «0», *Memory_1* (*Память_1*) сбрасывается (так как *Output_1* (*Выход_1*) опять обнулен). Таким образом, базовое состояние было снова получено после двух входных импульсов и одного выходного.

Второе решение применяет функцию блокировки, или защелки (сети 20 и 21, Network 20, 21) в цепных диаграммах. Принцип тот же, что и в первом решении, за исключением того, что условие сброса – как обычно при блокировке – «нулевая активность» («zero active»).

5.5.2 Решение в FBD

Данную задачу можно решить по-разному. Два способа решения приведены ниже.

Первое решение использует функции памяти (рисунок 5.9, сети 16 и 17, Network 16, 17). Если сигнальное состояние операнда *Input* (*Вход*) «1», то операнд *Output* (*Выход*) устанавливается (операнд *Memory* (*Память*) пока сброшен). Если сигнальное состояние операнда *Input* (*Вход*) меняется на «0», то *Memory* (*Память*) также устанавливается (*Output* (*Выход*) теперь «1»). Если снова *Input* (*Вход*) равен «1», то *Output*

(Выход) вновь сбрасывается (Memory (Память) теперь «1»). Если Input (Вход) еще раз будет равен «0», Memory (Память) сбрасывается (так как Output (Выход) теперь также обнулен). Таким образом, базовое состояние было снова получено после двух входных импульсов и одного выходного.

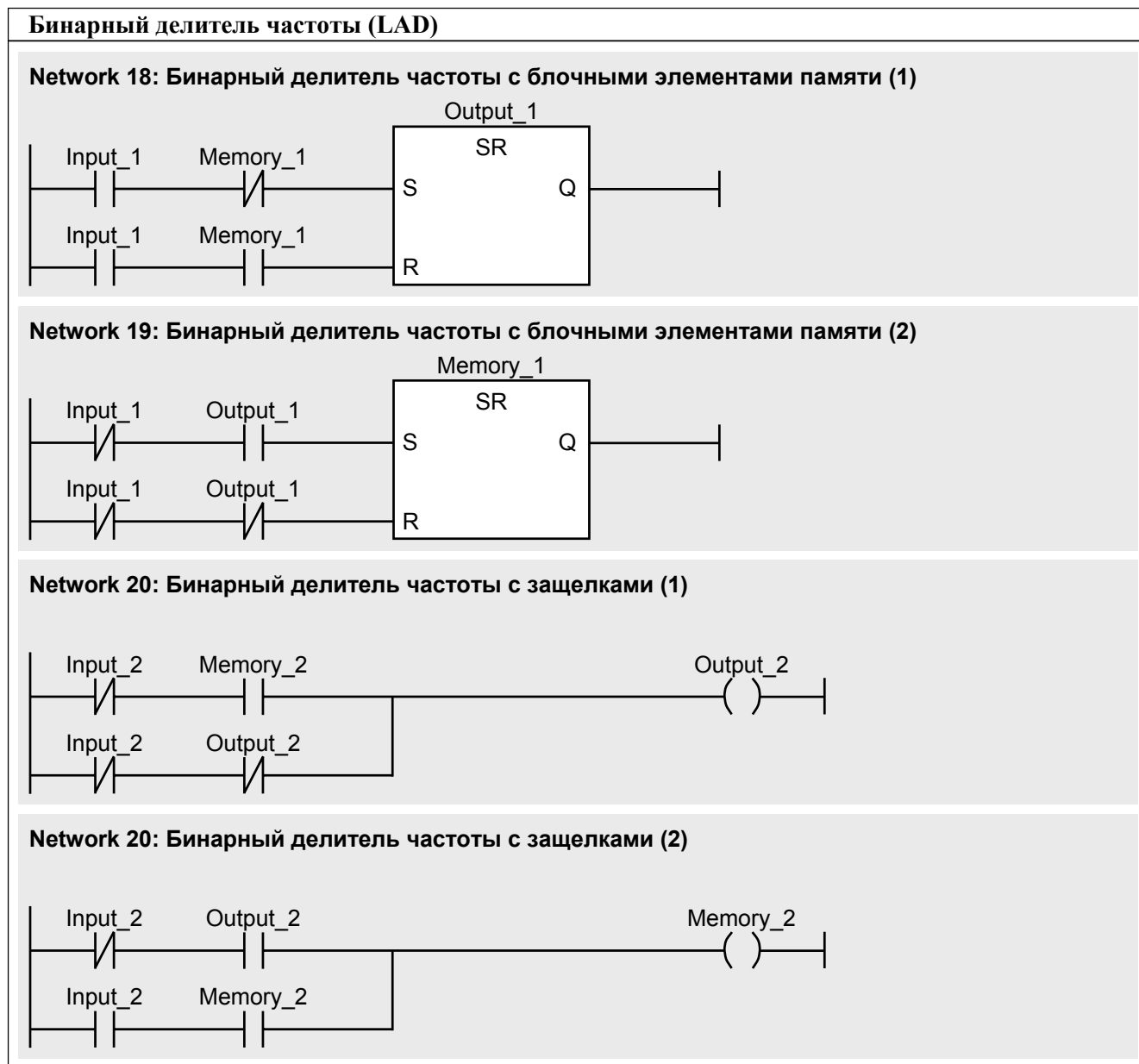


Рисунок 5.8 Примеры бинарного делителя частоты (LAD)

Второй способ решения использует оценку фронта операнда Input (Вход) (рисунок 5.9, сети 18 – 20, Network 18 – 20). Если у операнда Input (Вход) фронта не обнаружено, то RLO, следующий за оценкой уровня, равен «0», и выполняется инструкция перехода (jump) JCN (подробное описание операторов переходов вы найдете в главе 16 «Функции переходов»). В нашем примере метка перехода называется «bin» и присутствует в сети 20 (Network 20). Именно здесь возобновляется программное сканирование, если фронта не обнаружено. Фактический бинарный делитель частоты

ты имеется в сети 19 (Network 19): если *Output (Выход)* равен «0», то он установлен; если «1», то он обнулен. Эта сеть обрабатывается только тогда, когда обнаруживается фронт операнда *Input (Вход)*. По существу, каждый раз, когда выявляется фронт у *Input (Вход)*, *Output (Выход)* меняет свое сигнальное состояние.

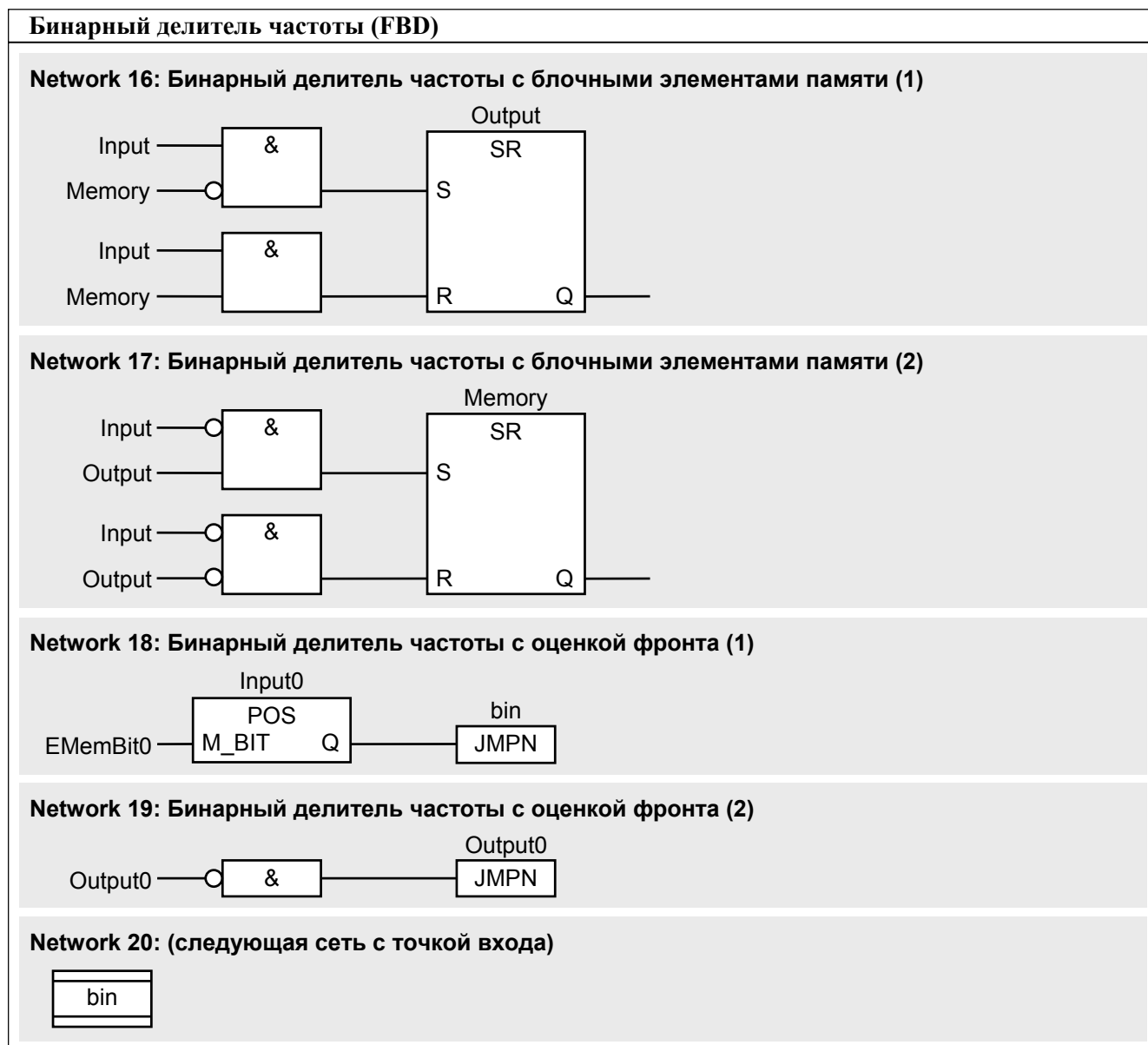


Рисунок 5.9 Примеры бинарных делителей частоты (FBD)

5.6 Пример системы управления конвейером

Следующий пример функционально простейшей системы управления транспортером конвейера иллюстрирует использование операций бинарной логики и функций памяти вместе с входами, выходами и меркерами.

Функциональное описание

- Когда транспортер пуст, контроллер запрашивает детали путем генерирования сигнала «readyload» (ready to load – готов к загрузке);
- Когда выдается сигнал «Start» («Старт»), транспортер запускается и перемещает детали;
- На конце транспортера конвейера датчик «end-of-belt» («конец транспортера»), например, световой барьер (фотоэлемент), обнаруживает детали, и в данной точке мотор транспортера выключается и включает сигнал «ready_remove» (ready to remove – готов к съему детали);
- Когда выдается сигнал «continue» («продолжить»), детали транспортируются дальше, пока датчики «end-belt» (end of belt – конец транспортера) не обнаружат их.

Пример программируется с входами, выходами и меркерами, и может быть запрограммирован в любом блоке в любом месте. В данном случае в качестве блока выбрана функция без значения функции.

Сигналы, символы

Дополняют функциональность системы управления транспортером конвейера несколько сигналов:

- Basic_st
Переводит контроллер в базовое состояние;
- Man_on
Включает транспортер вне зависимости от условий;
- /Stop
Останавливает конвейер на время присутствия «0» (NC-контакт в качестве датчика, «нулевая активность»);
- End_belt
Детали достигли конца транспортера;
- /Mfault
Сигнал ошибки (сбоя) от мотора транспортера (например, выключатель защиты

мотора); разработан как сигнал «нулевой активности», поэтому, к примеру, обрыв провода также генерирует сигнал ошибки.

Мы хотим использовать символическую адресацию, то есть операндам даны имена, которые будем применять при написании программы. Перед вводом программы мы создаем таблицу символов (таблица 5.1), содержащую входы, выходы, меркеры и блоки.

Таблица 5.1 Таблица символов (Symbol Table) для примера «Система управления транспортом конвейера»

Symbol (Символ)	Address (Адрес)	Data Type (Тип данных)	Comment (Комментарий)
Belt_control	FC 11	FC 11	Система управления конвейером
Basic_st	I 0.0	BOOL	Переводит контроллеры в базовое состояние
Man_on	I 0.1	BOOL	Включает мотор транспортера конвейера
/Stop	I 0.2	BOOL	Останавливает мотор транспортера конвейера (нулевая активность)
Start	I 0.3	BOOL	Запускает транспортер конвейера
Continue	I 0.4	BOOL	Уведомление об удалении деталей
Light_barrier1	I 1.0	BOOL	(Световой барьер) сигнал датчика «End of belt» («Конец транспортера») для транспортера 1 (belt 1)
/Mfault	I 2.0	BOOL	Выключатель защиты мотора транспортера 1, нулевая активность
Readyload	Q 4.0	BOOL	Загрузка новых деталей на транспортер (ready to load, готовность к загрузке)
Ready_rem	Q 4.1	BOOL	Удаление деталей с транспортера (ready to remove, готовность к удалению)
Belt_mot1_on	Q 5.0	BOOL	Включает мотор транспортера для транспортера 1
Load	M 2.0	BOOL	Команда загрузки деталей
Remove	M 2.1	BOOL	Команда удаления деталей
EM_Rem_N	M 2.2	BOOL	Меркер фронта для отрицательного фронта «remove»
EM_Rem_P	M 2.3	BOOL	Меркер фронта для положительного фронта «remove»
EM_Loa_N	M 2.4	BOOL	Меркер фронта для отрицательного фронта «load»
EM_Loa_P	M 2.5	BOOL	Меркер фронта для положительного фронта «load»

Программа для LAD

Пример располагается в функциональном блоке, который вы вызываете в организационном блоке OB 1 (выбран из каталога программных элементов «FC Blocks», «Блоки FC») для обработки в CPU.

Здесь будем программировать пример с блочными элементами памяти. В главе 19 «Параметры блока» показан тот же пример с использованием защелок (блокировок). Программа данной главы находится в функциональном блоке с параметрами блока, которые также могут быть вызваны так часто, как это необходимо (для нескольких транспортеров).

При программировании глобальные символы также могут использоваться без апострофов, что не допускает в их составе специальных литер. Если символ содержит специальную литеру (такую, как умлаут или пробел), то он должен быть заключен в апострофы. В скомпилированном блоке редактор помечает все глобальные символы, заключая их в апострофы.

На рисунке 5.10 показана программа для системы управления конвейером. На диске, поставляемой с книгой, вы можете найти эту программу в библиотеке «LAD_Book» в функциональном блоке FC 11 в разделе «Conveyor Example» («Пример конвейера»).

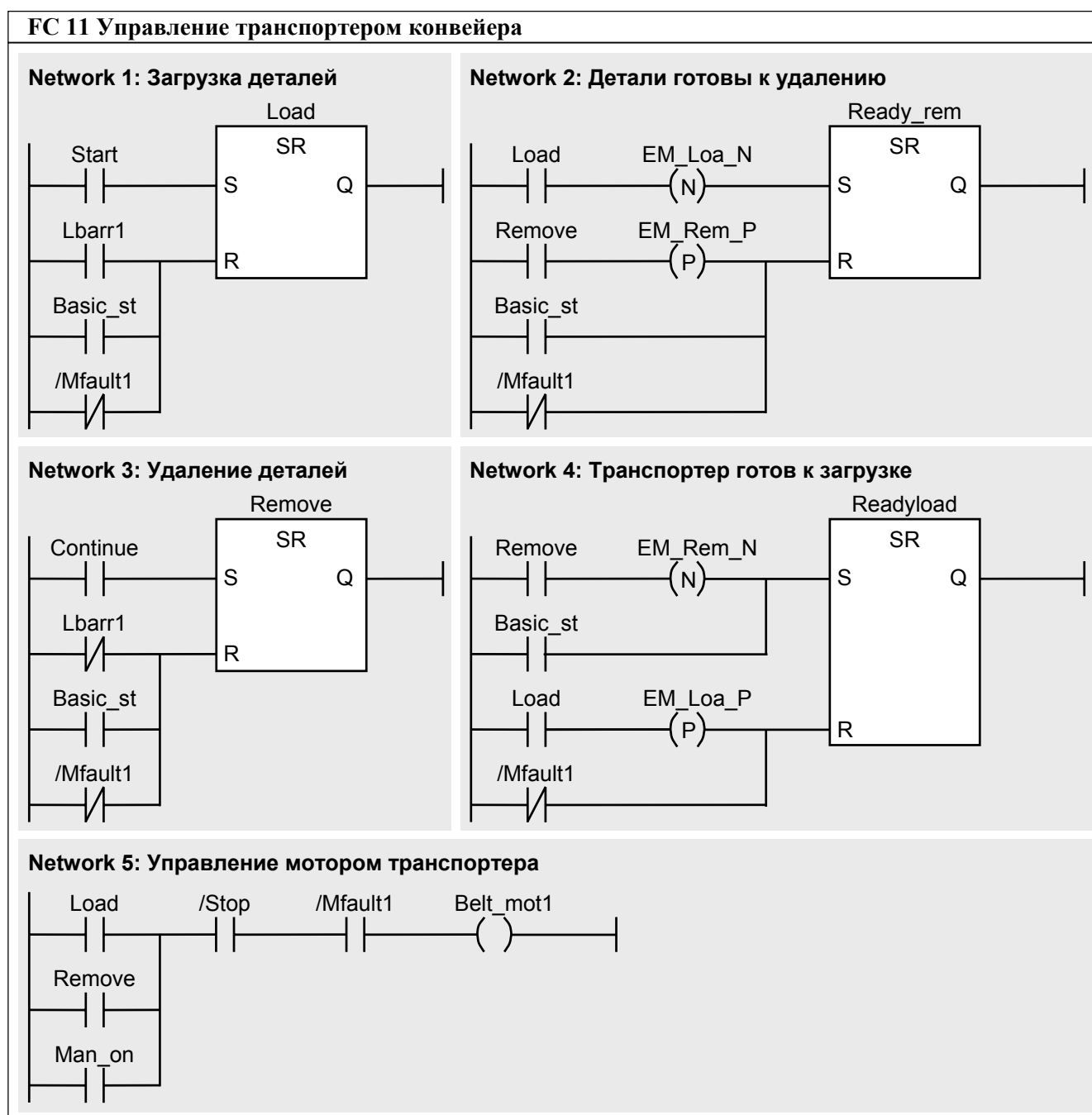


Рисунок 5.10 Образец системы управления конвейером (LAD)

Программа для FBD

Пример располагается в функции, которую вы вызываете в организационном блоке OB 1 (выбран из каталога программных элементов «FC Blocks», «Блоки FC») для обработки в CPU.

В главе 19 «Параметры блока» тот же пример приведен с использованием защелок (блокировок). Программа данной главы находится в функциональном блоке с параметрами блока, которые также могут быть вызваны так часто, как это требуется (для нескольких транспортеров).

При программировании глобальные символы также могут использоваться без апострофов, что не допускает в их составе специальных литер. Если символ содержит специальную литеру (такую, как умлаут или пробел), то он должен быть заключен в апострофы. В компилированном блоке редактор помечает все глобальные символы, заключая их в апострофы.

На рисунке 5.11 (а, б) показана программа для системы управления конвейером. На дискете, поставляемой с книгой, вы можете найти эту программу в библиотеке «FBD_Book» в функциональном блоке FC 11 в разделе «Conveyor Example» («Пример конвейера»).

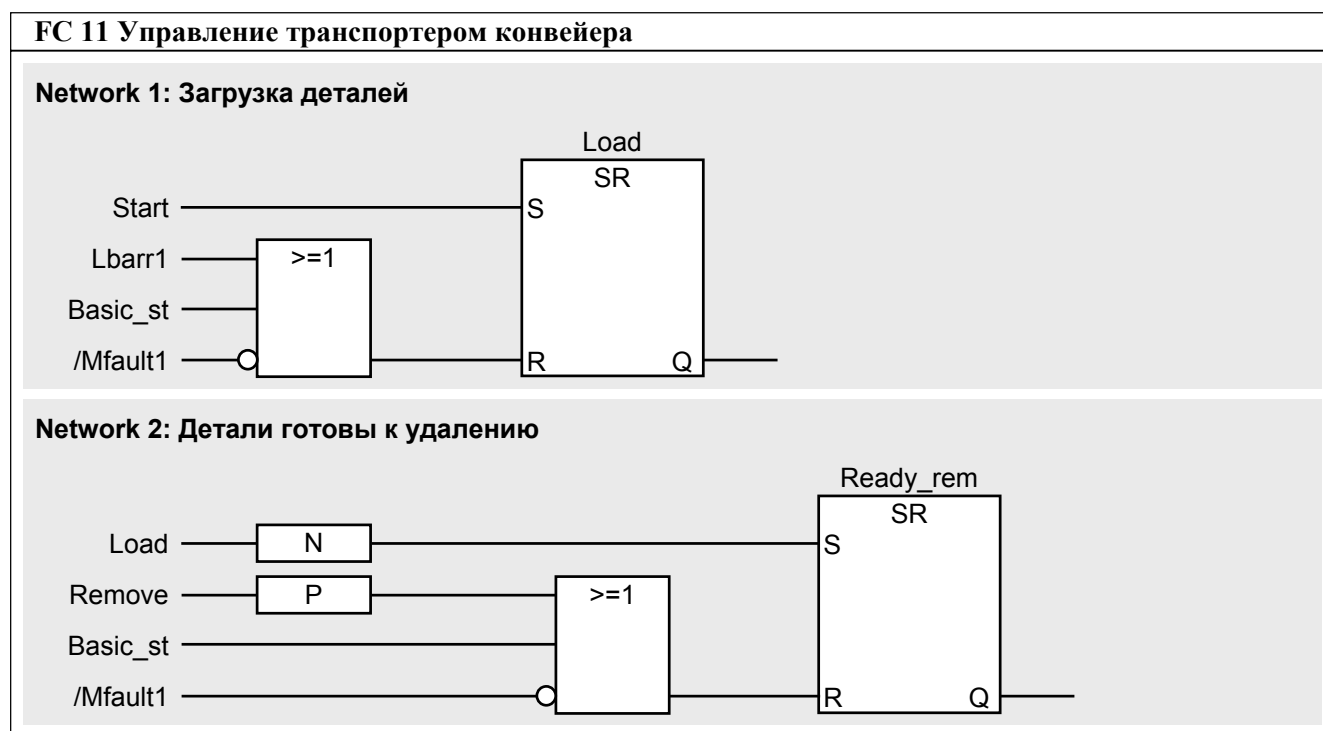


Рисунок 5.11а Пример системы управления конвейером (FBD)

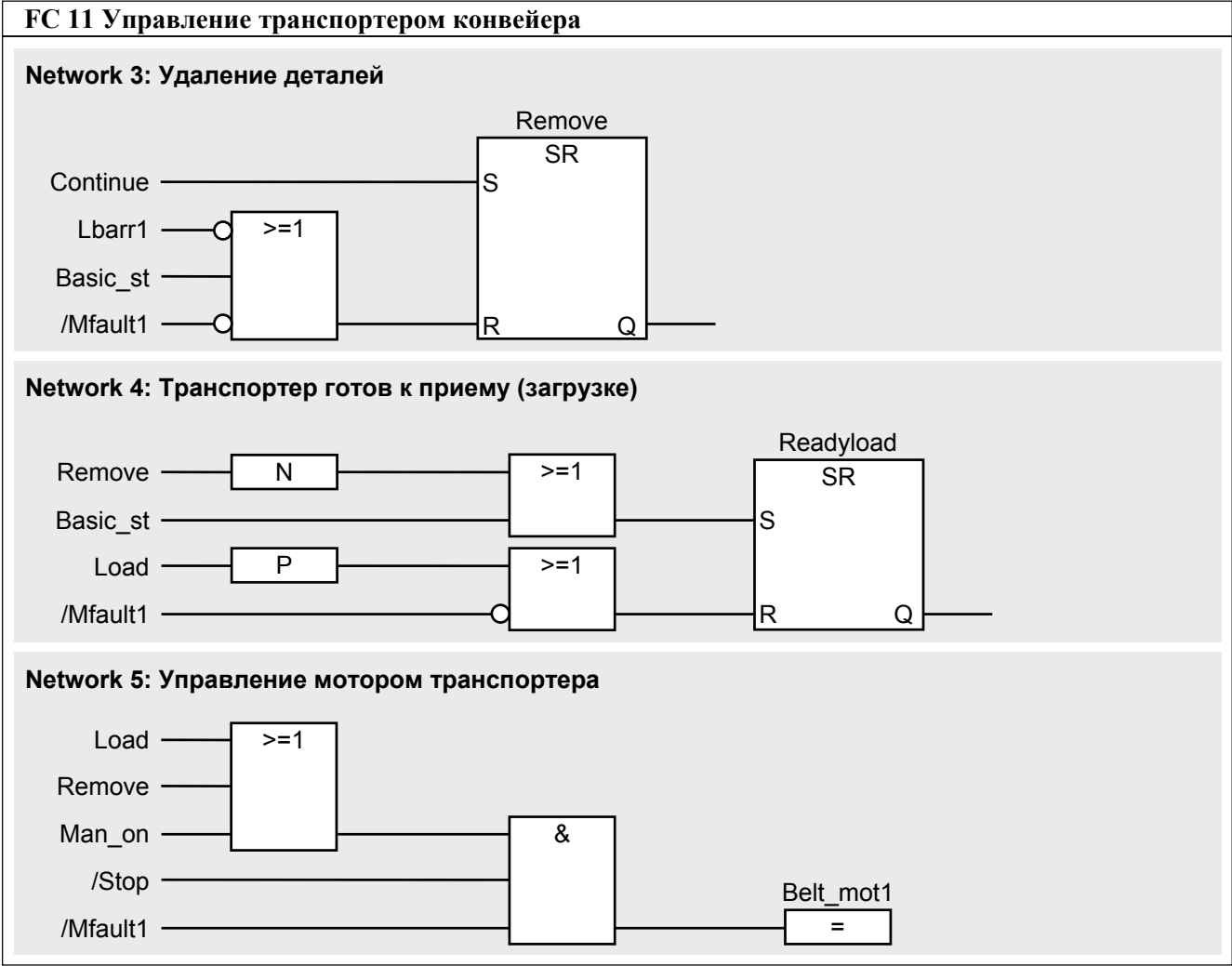


Рисунок 5.116 Пример системы управления конвейером (FBD)