Ожидание окончания переходных процессов в отдельных частях схемы

(на 30.09.17)

Начиная с версии 1.0.509 в RDS добавлен механизм для слежения за изменением значений, передаваемых группой связей, влияющих на вход заданного блока. Он нужен для обнаружения окончания переходных процессов в какой-либо части схемы и формирования сигнала готовности значения для блоков итеративного расчета (например, внешних максимизаторов). В состав RDS включен стандартный блок для формирования такого сигнала.

В данном тексте п. 1, 3 и 4 предназначены для пользователей, п. 2 – для программистов, создающих модели блоков, и “продвинутых” пользователей, п. 5 и 6 – для программистов.

1. Проблема выработки сигнала готовности и ее решение

В некоторых задачах, например, в задачах с внешним оптимизатором, есть часть схемы (“черный ящик”), включенная в обратную связь какого-либо управляющего блока (рис. 1).

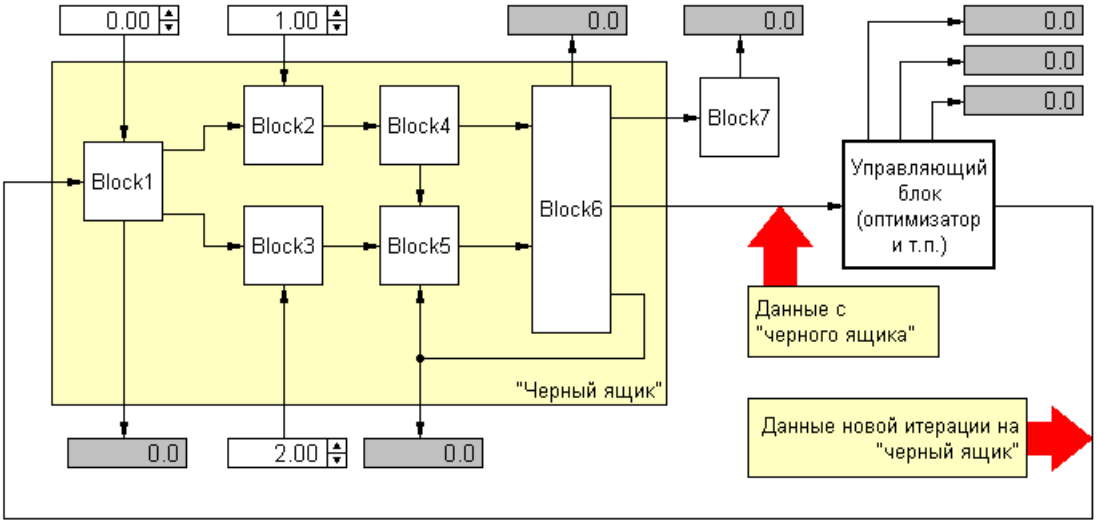

Рис. 1. “Черный ящик” в обратной связи управляющего блока

Расчет в этой части схемы выполняется не за один такт, поэтому управляющему блоку необходимо знать, когда на его входах, подключенных к “черному ящику”, установятся правильные значения. Пока все блоки в “черном ящике” не отработали, значения на его выходах нельзя считать соответствующими входным значениям. По мере срабатывания разных ветвей схемы выходы “черного ящика” будут изменяться, причем между такими изменениями может пройти довольно много тактов, если в схеме есть длинные цепочки. Управляющий блок должен дождаться прекращения ***всех*** изменений значений на входах ***всех*** блоков внутри “черного ящика”, чтобы можно было переходить к следующей итерации и выдавать на него новую порцию значений. Сейчас эта проблема обычно решается введением в управляющий блок задержки на заданное число тактов. Это полумера, потому что требуемое число тактов задержки зависит от структуры “черного ящика” и длины максимальной цепочки передачи в нем. Ситуация осложняется тем, что в “черном ящике” могут быть внутренние обратные связи, что не позволяет просто посчитать число внутренних

блоков и ввести это число (плюс один или плюс два для надежности) в качестве значения задержки – неизвестно, сколько тактов будут работать обратные связи, пока вычисления внутри “черного ящика” не завершатся.

Управляющие блоки обычно имеют входной сигнал готовности, поступление которого должно сигнализировать о завершении вычислений, но “вытащить” такой сигнал из “черного ящика” сложной структуры, как правило, не очень просто.

Теперь в RDS включен внутренний механизм обнаружения прекращения изменения передаваемых данных в заданном наборе связей. Для пользователя он реализован в виде блока (см. п. 3), который выделяет из схемы дерево связей и блоков, ведущих к своему входу (далее это дерево будет называться “подграфом влияния” или просто “подграфом”), и формирует выходной сигнал, когда данные, передаваемые по всем связям внутри этого дерева, перестанут изменяться (рис. 2). Для поддержки работы блока введено несколько сервисных функций и событий, описанных в п. 5.

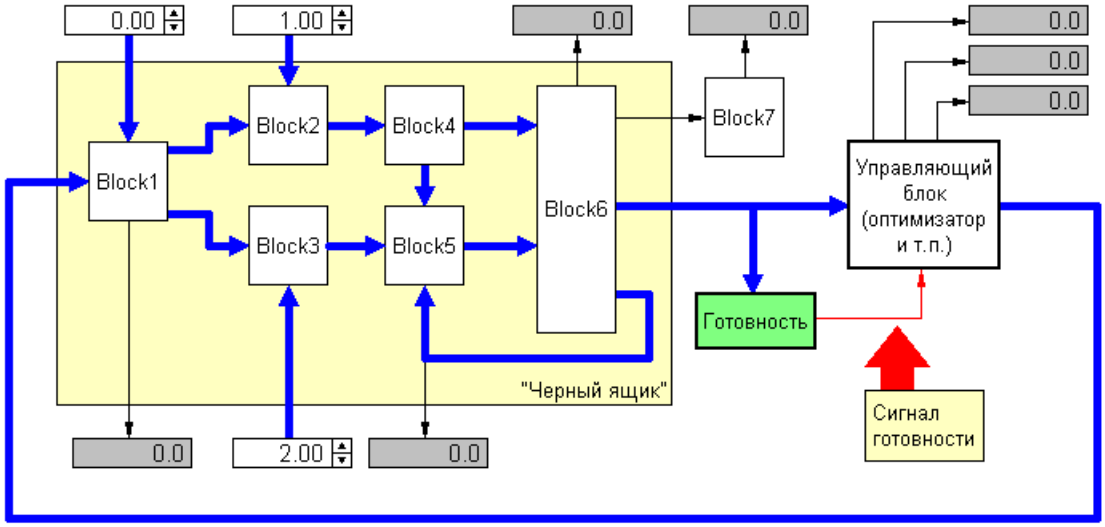

Рис. 2. Подключение формирователя сигнала готовности
(подграф, влияющий на готовность, выделен синим)

Технически выделение подграфа связей производится следующим образом: берется связь, идущая к входу блока готовности, затем все связи, идущие к входам блока, от которого идет эта связь, затем все связи, идущие к входам блоков, выдающих данные для этих связей и т. д. Таким образом, строится дерево связей, ведущих к заданному входу блока. На рис. 2 все связи подграфа, ведущие к входу блока готовности, выделены синим цветом.

После того, как множество связей, влияющих на вход блока готовности, составлено, при их срабатывании RDS будет сравнивать передаваемое с выхода блока значение с текущим значением входа, на который оно передается. Если они не совпадают с заданной точностью, весь подграф считается изменившимся. Только если ни одно срабатывание связи в подграфе не привело к изменению значения входа в течение заданного в настройках блока готовности числа тактов, переходный процесс в “черном ящике” считается завершенным.

Поскольку для анализируемых связей к простому копированию данных из выхода во вход добавляется еще и предварительное сравнение этих данных, включение этого механизма приведет к некоторому уменьшению скорости расчета. Оно будет примерно эквивалентно удвоению числа связей в анализируемом подграфе (на каждый вход – две операции вместо одной). Но включение в управляющем блоке задержки на число тактов, заведомо бОльшее числа тактов в самом длинном переходном процессе, используемое сейчас, замедляет расчет сильнее.

2. Поддержка нового механизма моделями блоков

Абсолютное большинство блоков, как стандартных, так и пользовательских, не требуют каких-либо модификаций для поддержки механизма обнаружения окончания переходного процесса. Проверка изменений передаваемых по связям значений делается на уровне RDS, без участия моделей самих блоков. Однако, некоторые сложные блоки могут выпасть из поля зрения нового механизма, поэтому их модели необходимо модифицировать. К таким блокам относятся:

- различные блоки-переключатели (на их выходы влияют не все входные значения, а только те, которые соответствуют текущему состоянию блока);
- блоки, передающие данные через динамические переменные (они не соединены связью и поэтому не обнаруживаются при анализе графа схемы);
- блоки, перезапускающие сами себя для выполнения длительного расчета (их выходы долго не изменяются, что может послужить ложным сигналом об окончании переходного процесса).

Во все стандартные модели подобных блоков ***уже внесены необходимые изменения***. Пользователи, которые не собираются создавать собственные модели блоков указанных типов и редко использующие в своих схемах динамические переменные и блоки-переключатели, могут полностью пропустить п. 2 и перейти к п. 3. Для разработчиков же здесь в общем виде описываются изменения, которые необходимо внести в модели таких блоков. Более подробно они описаны в п. 5 и 6.

2.1. Использование блоков-переключателей

Включение в RDS механизма слежения за переходными процессами в связях привело к усложнению моделей блоков-переключателей: выключателей связей, мультиплексоров и демультиплексоров. Для всех прочих блоков считается, что значения всех выходов блока зависят от значений всех входов, и, в большинстве случаев, это справедливо. Но у блоков-переключателей все не так. У простого выключателя связи выход зависит от входа только тогда, когда блок включен. У мультиплексора выход зависит только от выбранного в данный момент входа. У демультиплексора от входа зависит только один, выбранный в данный момент, выход (на остальные выходы данные не передаются).

В связи с этим стандартный способ выделения влияющего на готовность подграфа связей, описанный выше, для этих блоков не работает: нельзя считать, что на все выходы переключателя всегда влияют все его входы. В данный момент все стандартные блоки-переключатели в RDS модифицированы так, чтобы самостоятельно разбираться с переходными процессами на своих входах (принципы действия таких моделей описаны в п. 6.1, 6.2 и 6.3). Фактически, переключатель разрывает анализируемый подграф связей и сам запрашивает изменения в отдельных подграфах, подключенных к его входам, а затем, согласно логике своей работы, устанавливает признак изменения выходного подграфа. Пользователю, собирающему схемы и создающему обычные модели блоков, не обязательно знать, как устроена модель переключателя, и как она программно передает признаки изменения из входных подграфов в выходные. Для подавляющего большинства задач подходят стандартные блоки, в которых все необходимые механизмы уже встроены. Но для создания схем необходимо понимать, что в той части схемы, которая подключена к выходу переключателя, будут обнаруживаться только те переходные процессы, которые происходят в связях, подключенных к входу, соответствующему текущему состоянию блока.

В качестве примера рассмотрим условную схему на рис. 3. В ней к разным элементам входного массива “X” мультиплексора “MS dbl” подключены две ветви схемы: верхняя ветвь через блоки “Block2” и “Block4” принимает данные от внешнего управляющего блока, а нижняя ветвь (“Block3”, “Block5”) подключена к полю ввода внутри самого “черного ящика”.

Выбор входа мультиплексора, значение которого будет передаваться на его выход и поступать через блок “Block1” в управляющий блок, осуществляет двухпозиционный блок выбора варианта, подключенный к входу мультиплексора “N”.

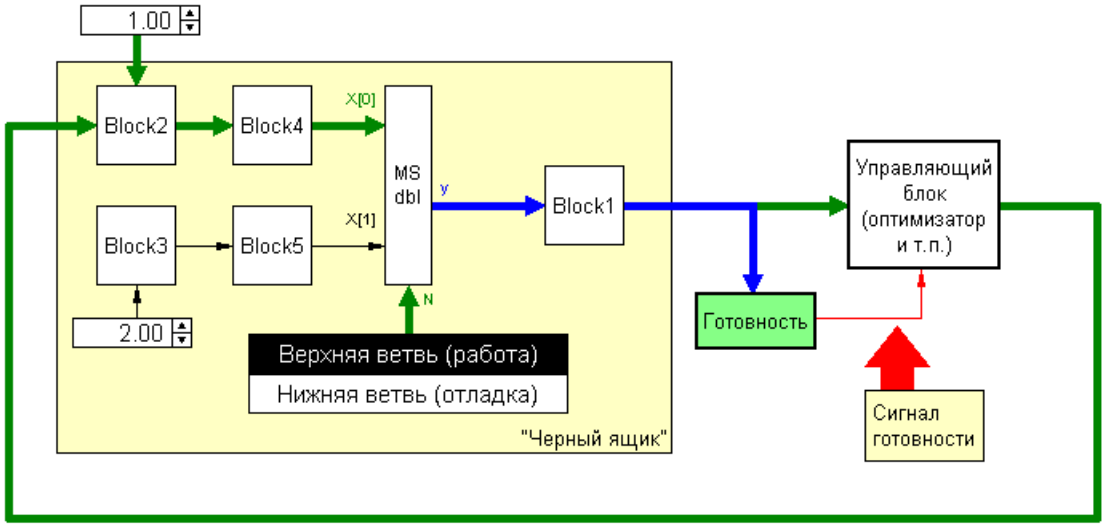

Рис. 3. Мультиплексор подключает к управляющему блоку только верхнюю ветвь схемы (синим выделен главный подграф, зеленым – подчиненные активные, тонкими черными линиями – подчиненный игнорируемый)

Такие конфигурации часто используются для отладки схемы. Если мультиплексор переключен на нижнюю ветвь, то значения, поступающие на вход блока “Block1”, можно задавать вручную и исследовать поведение этой части схемы. При переключении на верхнюю ветвь значения будут поступать с управляющего блока.

На рисунке мультиплексор переключен на прием данных с верхней ветви схемы. Это значит, что любые изменения в нижней ветви схемы будут игнорироваться блоком формирования сигнала готовности (“Готовность”). На сигнал готовности будет влиять только срабатывание связей, обозначенных жирными синими и зелеными линиями. Информацию об изменениях в синих связях получает непосредственно блок “Готовность”. Информацию об изменениях в зеленых и черных связях получает мультиплексор. При обнаружении любых изменений в подграфе связей, подключенном к входу “N”, мультиплексор немедленно взводит признак изменения в выходном подграфе: переключение ветвей схемы способно повлиять на выход мультиплексора и, таким образом, на ту часть схемы, которая к нему подключена. Любые изменения в черном подграфе, подключенном к “X[1]”, игнорируются: эта ветвь сейчас отключена и никак не может повлиять на выход блока. Обнаружив изменения в зеленом подграфе, подключенном “X[0]”, мультиплексор опять взводит признак изменения выходного подграфа: при текущем значении “N” этот вход передается на выход. Таким образом, изменения данных в зеленых связях обнаруживаются не блоком “Готовность”, а мультиплексором, который программно сообщает об изменениях далее по связям.

Конкретные примеры схем с выключателем связей, мультиплексором и демультиплексором приведены в п. 4.

2.2. Передача данных через динамические переменные

Если блок передает данные в другие блоки через динамические переменные, такая передача не может быть обнаружена при помощи анализа связей. Более того, обнаружить изменения при такой передаче средствами RDS затруднительно: в отличие от связи, в которой можно сравнить значение на выходе одного блока со значением на входе другого,

динамическая переменная существует в единственном экземпляре для всех подключенных к ней блоков, ее предыдущее значение нигде не хранится, и она может быть изменена в любой момент в любой реакции любого блока.

Чтобы при этом можно было все равно анализировать переходные процессы, блок теперь может установить в своих данных признак того, что изменение значений на его входах приводит к изменению значений динамических переменных, к которым он подключен. В стандартные блоки для передачи данных через динамические переменные этот признак уже добавлен. При обнаружении в анализируемом подграфе блока, подключенного к динамической переменной, автоматически просматриваются все остальные блоки, подключенные к той же переменной, и, если у кого-нибудь из них установлен упомянутый выше признак, весь подграф входных связей такого блока включаются в общий подграф (зеленые связи на рис. 4).

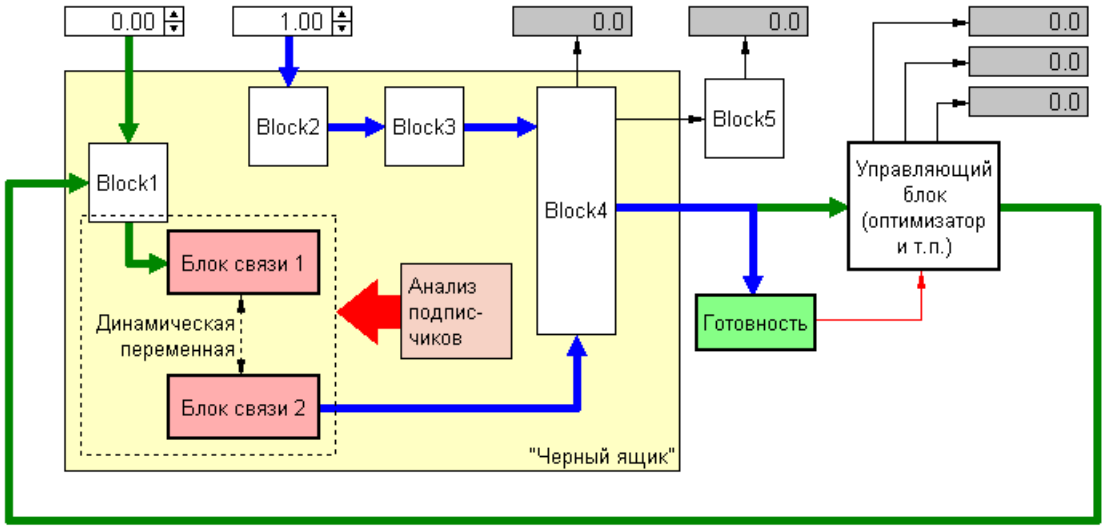

Рис. 4. Отслеживание передачи данных через динамическую переменную при анализе графа

Следует учитывать, что не всегда блок, подключенный к динамической переменной, изменяет ее при изменении своих входов. Например, блок, поставляющий всей схеме значение времени при динамическом расчете, меняет динамическую переменную “DynTime” по таймеру, а не по своим входам. Для динамических блоков понятие “завершение переходного процесса” определяется течением времени, а не числом выполненных тактов, и, поэтому, имеет другой смысл и должно анализироваться по-другому. Но у стандартных “блоков-туннелей”, включенных в RDS и подобных изображенным на рис. 4, значение динамической переменной жестко связано именно со значением входа, и для них работает описанный выше механизм. Если пользователь решит делать собственные аналогичные модели блоков, записывающих значения в динамические переменные при изменении входов, ему нужно будет программно сообщать об этом в RDS установкой соответствующего флага (см. п. 6.4).

Следует также иметь в виду, что анализ подграфа связей выполняется при переходе в режим моделирования из режима редактирования, поэтому при этом анализе будут учтены связи блоков только через динамические переменные, ***существующие на этот момент***. Если в процессе расчета блок подключится к другой динамической переменной, эта связь не будет влиять на проверку окончания переходного процесса. Сейчас все стандартные блоки RDS подписываются на динамические переменные и создают их в режиме редактирования, поэтому они будут работать правильно. Если же в какой-то экзотической задаче придется создавать динамические переменные “на лету” в процессе расчета, в такие блоки придется

закладывать логику работы, аналогичную логике блоков-переключателей (см. п. 2.1) – блок должен самостоятельно обнаруживать переходные процессы на своих входах и сообщать о них связанным блокам, чтобы они установили флаги изменений в выходных подграфах.

Пример схемы со связью блоков через динамические переменные, в которой определяется момент окончания переходного процесса, приведен в п. 4.7.

2.3. Блоки, перезапускающие сами себя

В некоторых случаях блок, получив на вход данные, должен выполнить достаточно сложный расчет, занимающий длительное время. Например, внутренний максимизатор многократно вызывает заложенную в него функцию с разными значениями аргументов, пока не найдет максимальное значение. Блок поиска равновесия тоже многократно вызывает несколько заложенных в него функций, пока их значения не придут в состояние равновесия (не перестанут изменяться). Если все эти вызовы выполнять в цикле в одном такте расчета, RDS не будет реагировать на действия пользователя, пока модель блока, закончив вычисления, не вернет управление главной программе. Чтобы избежать “зависания” при длительных вычислениях в рамках одного блока, модели таких блоков устроены следующим образом: модель выполняет только небольшую часть расчета, запоминает, где он остановился, взводит у своего блока сигнал запуска в следующем такте, запрещает передачу данных с выходов и завершается. В следующем такте модель снова запустится, выполнит еще одну часть расчета и снова перезапустится. Так будет продолжаться, пока весь расчет не будет выполнен – только тогда блок выдаст данные на выходы.

Такой режим работы может помешать правильному обнаружению завершения переходного процесса. Рассмотрим, например, схему на рис. 5. В ней в “черный ящик” включено несколько блоков с внутренним максимизатором на выходе.

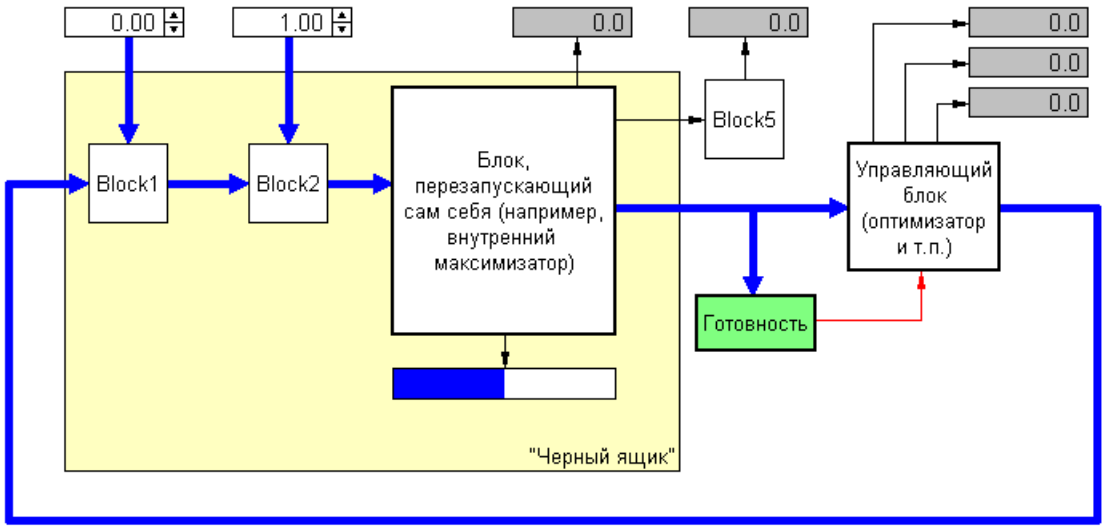

Рис. 5. Наблюдение за переходным процессом в схеме, в которой блок выполняет расчет за несколько тактов, перезапуская себя и не изменяя выходы

Связи, за переходным процессом в которых будет наблюдать блок “Готовность”, на рисунке обозначены синим цветом. При запуске расчета внешний максимизатор выдаст данные на блок “Block1”, он, в свою очередь, передаст данные блоку “Block2”, а тот – внутреннему максимизатору. Внутренний максимизатор начнет вычисления, но пока ничего не выдаст на выход – он перезапустится. В следующем такте он снова перезапустится, потом еще раз, и так будет продолжаться до тех пор, пока его расчет не будет завершен. Но на протяжении всех тактов, когда он перезапускается и продолжает расчет, по связям, обозначенным синим цветом, не будет передаваться новых значений. Блок “Готовность”

решит, что переходный процесс закончился, и выдаст внешнему максимизатору сигнал для начала новой итерации, что будет ошибкой.

Чтобы избежать этого, блоки могут установить у себя признак продолжающегося расчета. Если в анализируемом подграфе связей есть хотя бы один блок с таким признаком, переходный процесс в этом подграфе не будет считаться законченным. Внутренний максимизатор на рис. 5 будет устанавливать этот признак при каждом перезапуске, поэтому, хотя данные по связям и не будут передаваться, сигнал готовности не будет выдан раньше времени.

На данный момент установка этого признака включена во внутренний максимизатор, блок вычисления табличной функции и блок поиска равновесия (конкретный пример схемы с внутренним максимизатором приведен в п. 4.9). Если пользователь создает собственный блок, перезапускающийся для продолжения расчета, он должен программно устанавливать этот признак в модели блока при каждом перезапуске (см. п. 6.5).

3. Блок обнаружения переходного процесса

В набор стандартных блоков RDS введен блок, специально предназначенный для определения начала и окончания переходного процесса в части схемы, подключенной к его входу. На выходы этот блок выдает сигналы начала и окончания процесса, а также логический признак того, что переходный процесс в данный момент идет. Блок добавлен на вкладку “Управление” стандартной и альтернативной панелей блоков (рис. 6), в схеме он изображается прямоугольником с текстом “переходный процесс” (рис. 7).

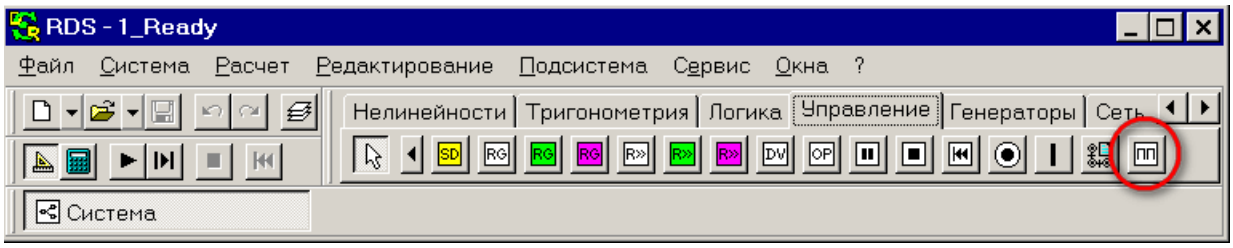

Рис. 6. Новый блок на панели блоков

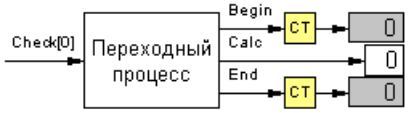

Рис. 7. Внешний вид блока в схеме

Блок имеет следующие входы и выходы (примеры подключения приведены в п. 4):

Check (вход, массив переменных произвольного типа) – вход для подключения связей, в подграфах которых отслеживаются переходные процессы. Этот вход – массив, поэтому связи нужно подключать к его отдельным элементам. Блок выдает сигнал об окончании переходных процессов только тогда, когда они завершатся во ***всех*** подключенных к этому входу подграфах (см. пример в п. 4.2).

Reset (вход, сигнал) – вход сброса блока, используемый в специальном режиме работы, задаваемом в настройках. Если в настройках включено использование этого сигнала, до его поступления блок будет находиться в режиме ожидания и не будет регистрировать переходные процессы. После поступления сигнала блок начинает слежение за связями, и снова переходит в режим ожидания по окончании переходных процессов. Такой режим работы может быть полезен, если блок стоит в обратной связи максимизатора и формирует для него сигнал очередной итерации, как на рис. 2. Нахождение в режиме ожидания до сообщения от максимизатора (или другого управляющего блока) о начале итерации позволяет

игнорировать переходные процессы, возникающие в схеме тогда, когда максимизатор выключен (например, при первом запуске расчета), и не выдавать на него “лишние” сигналы готовности. Пример такого подключения приведен в п. 4.8.

Begin (выход, сигнал) – выходной сигнал начала переходного процесса.

End (выход, сигнал) – выходной сигнал окончания переходного процесса (именно он обычно используется в качестве сигнала готовности для управляющего блока).

Calc (выход, логический) – признак наличия переходного процесса в данный момент.

Bad (выход, логический) – признак расходящегося процесса (если в настройках блока разрешено обнаружение таких процессов).

Error (выход, сигнал) – выходной сигнал обнаружения расходящегося процесса (см. пример в п. 4.11).

Out (выход, массив переменных произвольного типа) – выходной массив значений, поступивших на вход “Check” на момент окончания переходного процесса (см. пример в п. 4.10).

В окне настроек блока (рис. 8) задаются погрешности сравнения вещественных чисел при передаче по связям и включаются специальные режимы работы.

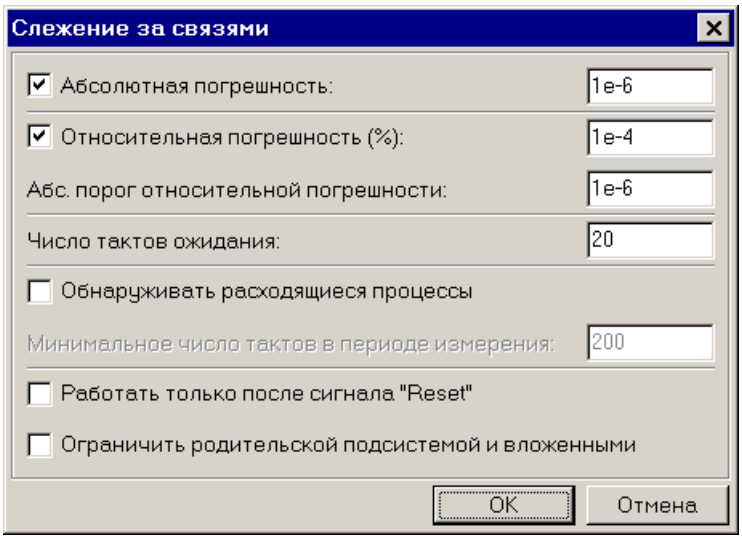

Рис. 8. Окно настроек блока

Окно содержит следующие настройки:

“**Абсолютная погрешность**” – если флажок включен, то при передаче вещественного значения по связи оно будет считаться не изменившимся, если его изменение по модулю не превышает значения в поле ввода справа (здесь значение, как и в других полях погрешностей, можно вводить в стандартном формате “*N*e*M*”, обозначающем $N \cdot 10^M$).

“**Относительная погрешность (%)**” – если флажок включен, то при передаче вещественного значения по связи оно будет считаться не изменившимся, если его изменение по модулю не превышает вычисленного по введенной относительной погрешности значения. Абсолютное значение для этой относительной погрешности вычисляется следующим образом: из старого (до передачи) и нового (после передачи) значения входа блока берется максимальное по модулю, умножается на введенную относительную погрешность и делится на 100. Если изменение входа блока по модулю не превышает это вычисленное значение, вход считается не изменившимся.

“**Абс. порог относительной погрешности**” – порог изменения, ниже которого проверка относительной погрешности не производится. Введение этого параметра связано с тем, что при переходных процессах, плавно сходящихся к нулевому значению на входе блока, вычисляемое значение относительной погрешности стремится к нулю вместе с самим значением, и проверка попадания в эту погрешность никогда не выполнится. Поэтому при

достижении погрешностью, вычисленной по введенному значению в процентах, заданного порога, проверка отключается.

Если включены оба флажка погрешностей, значение будет считаться не изменившимся только если его изменение будет не будет превышать наименьшей из двух погрешностей. Если оба флажка выключены, для отсутствия изменений значение до передачи должно в точности (до бита) совпадать со значением после нее.

Важное замечание про погрешности. Следует учитывать, что введенные значения погрешностей относятся не только к значению, передаваемому связью, подключенной к входу блока “Check[…]”, но и ко всем значениям всех связей в той части схемы, которая влияет на вход блока. Будет ошибкой задавать погрешности, исходя из требуемой точности вычисления конечного значения. Например, если требуемая точность значения выхода “черного ящика”, подключенного к блоку обнаружения переходного процесса, составляет одну десятую, будет ошибкой задать эту одну десятую в качестве абсолютной погрешности, не учитывая структуру схемы. Если, например, перед выходом схемы находится блок умножения на десять, то изменению в одну десятую на выходе будет соответствовать изменение в одну сотую на входе этого умножителя. И это изменение в одну сотую блок обнаружения переходного процесса не должен пропускать, потому что через несколько тактов оно доберется до выхода и существенно повлияет на него. Поэтому на практике либо следует задавать в качестве погрешностей очень маленькие числа, либо вообще отключать оба флажка погрешностей. В последнем случае переходные процессы будут считаться законченными, только если в схеме не будет вообще никаких изменений значений. Это может несколько затянуть расчет, особенно схема устроена так, что значения в ней подходят к установившимся, например, по экспоненте. Но, зато, блок выдаст сигнал готовности только тогда, когда изменения в схеме полностью прекратятся.

Кроме того, следует учитывать, что задаваемые погрешности описывают изменение значений в схеме не за какой-то период времени, а за один такт. В отличие от динамических расчетов, где можно оперировать понятием “скорость изменения значения”, здесь речь идет не о скорости, а о разности между соседними членами числовой последовательности, которую формирует схема в процесс расчета. Оценить допустимый порог этой разности может быть достаточно сложно.

“**Число тактов ожидания**” – число тактов расчета, в течение которого изменения во всех связях исследуемого подграфа схемы должны укладываться в допустимые погрешности, прежде чем блок выдаст сигнал окончания переходного процесса.

“**Обнаруживать расходящиеся процессы**” – флажок, включение которого разрешает блоку проверять, не расходится ли переходный процесс в наблюдаемом подграфе связей. Проверка работает следующим образом: на протяжении заданного в поле ввода ниже числа тактов интервала измерения блок определяет максимальное изменение значений входов блока в подграфе. Если в двух соседних интервалах измерения это значение не уменьшается, процесс считается расходящимся, и вместо сигнала “End” блок выдаст сигнал “Error”. Если флажок выключен, проверка не производится, и при расходящемся переходном процессе блок будет все равно ждать его окончания (сигнал “Error” при этом не выдается).

“**Минимальное число тактов в периоде измерения**” – число тактов в интервале измерения для определения расходящегося переходного процесса (блок может автоматически скорректировать это значение в большую сторону, если в подграфе много связей и блоков). Следует учитывать, что слишком маленькое число в этом поле ввода может вызвать ложные срабатывания проверки на расходящийся процесс.

“**Работать только после сигнала "Reset"**” – флажок включает режим работы, при котором блок переходит в режим ожидания при первом запуске расчета и после обнаружения окончания переходного процесса. В режиме ожидания блок не будет реагировать на новые

переходные процессы до прихода на вход сигнала “Reset”, описанного выше. Пример использования этого режима приведен в п. 4.8.

“**Ограничить родительской подсистемой и вложенными**” – флажок включает режим работы, при котором будут игнорироваться все изменения на входах блоков, находящихся снаружи подсистемы, в которой находится данный блок, даже если выходы этих блоков влияют на внутренние блоки подсистемы. В настоящее время этот режим не используется, но может понадобиться в будущем.

4. Примеры схем с использованием блока

Большинство реальных переходных процессов в схемах происходят слишком быстро, чтобы их можно было наблюдать. Достаточно типичным примером схемы с естественным переходным процессом может служить сумматор с коэффициентом, меньшим единицы в обратной связи (рис. 9, именно эта схема использовалась для отладки при разработке блока).

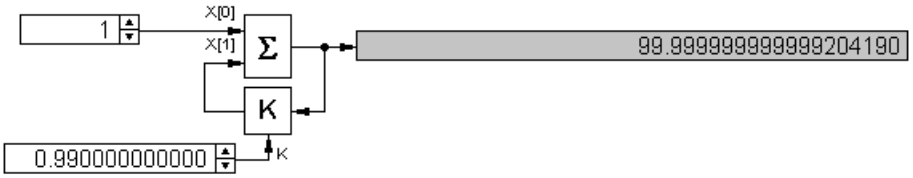

Рис. 9. Сумматор с обратной связью

Выход сумматора в такой схеме всегда сходится к какому-либо значению. Число тактов, за которое переходный процесс прекратится, зависит от коэффициента в обратной связи, а реальное время переходного процесса – от быстродействия конкретной машины. Такая схема плохо подходит для демонстрации работы блока, поскольку в ней приходится подбирать коэффициент индивидуально для каждой машины, чтобы переходный процесс можно было увидеть в реальном времени.

Из-за этого во всех описываемых ниже примерах схем в качестве тестового переходного процесса используется работа специально созданного блока (рис. 10, блок “Счет”), который сглаживает изменения своего входного значения, дополнительно в каждом такте модулируя его шумом с амплитудой, уменьшающейся до нуля в течение заданного времени после приближения выходного значения ко входному.

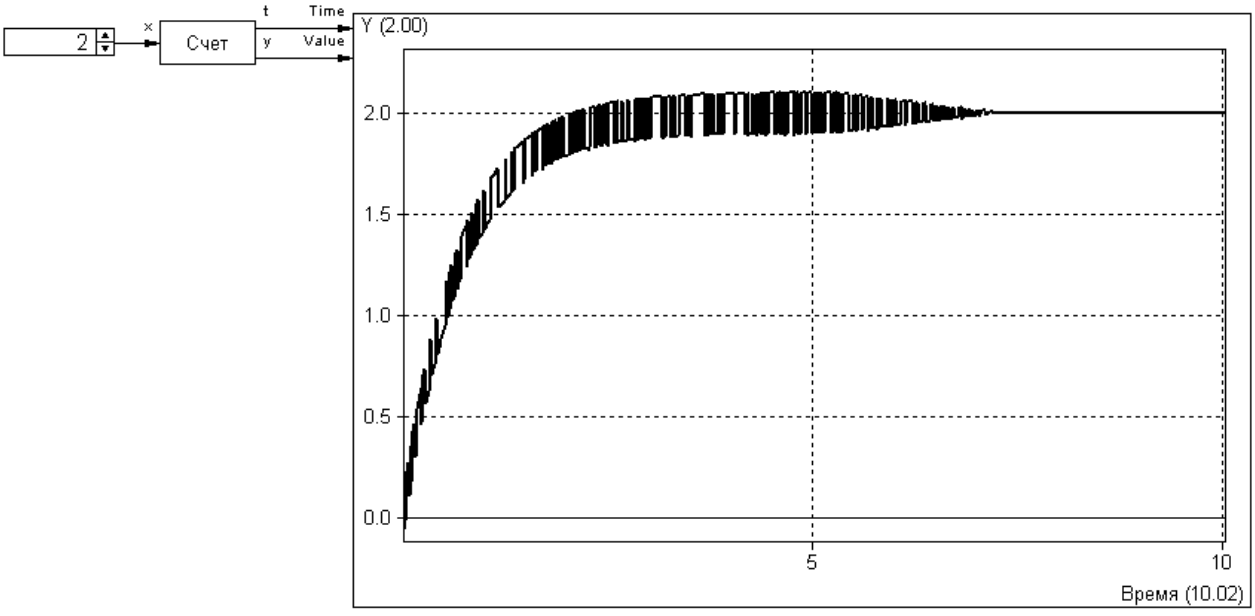

Рис. 10. Тестовый блок расчета

Таким образом, если значение на выходе этого блока равно входному, ничего не происходит. Если входное значение изменится, выходное начнет “дергаться” то вверх, то вниз в каждом такте, постепенно приближаясь к входному (внутрь блока заложен аналог апериодического звена первого порядка, работающего по реальному времени системы). Когда незашумленное расчетное значение выхода приблизится к входному, шум начнет линейно уменьшаться до нуля и, в конце концов, выходное значение снова станет равным входному, и переходный процесс прекратится. Поскольку блок работает в реальном времени, его переходный процесс будет продолжаться примерно одинаковое время на машине с любым быстродействием. Кроме того, его можно вывести на график, как на рис. 10, что удобно для наблюдений. Пример схемы с этим блоком (пока без блока обнаружения переходного процесса), а также с сумматором с обратной связью, находится в файле “0_TestProcess.rds”.

В описанных ниже примерах схем к выходам блока обнаружения переходного процесса обычно подключены два счетчика и индикатор-семафор, как на рис. 7:

- к сигнальному выходу “Begin” подключен счетчик, выход которого будет увеличиваться на единицу при каждом поступлении сигнала о начале переходного процесса;
- к логическому выходу “Calc” подключен индикатор-семафор, зажигающийся зеленым, если в исследуемом подграфе схемы нет переходного процесса, и красным, если процесс идет;
- к сигнальному выходу “End” подключен счетчик, выход которого будет увеличиваться на единицу при каждом поступлении сигнала об окончании переходного процесса.

4.1. Схема “1_Ready.rds” – наблюдение за процессом

Схема “1_Ready.rds” содержит три ветви, к каждой из которых подключен блок обнаружения переходного процесса (рис. 11).

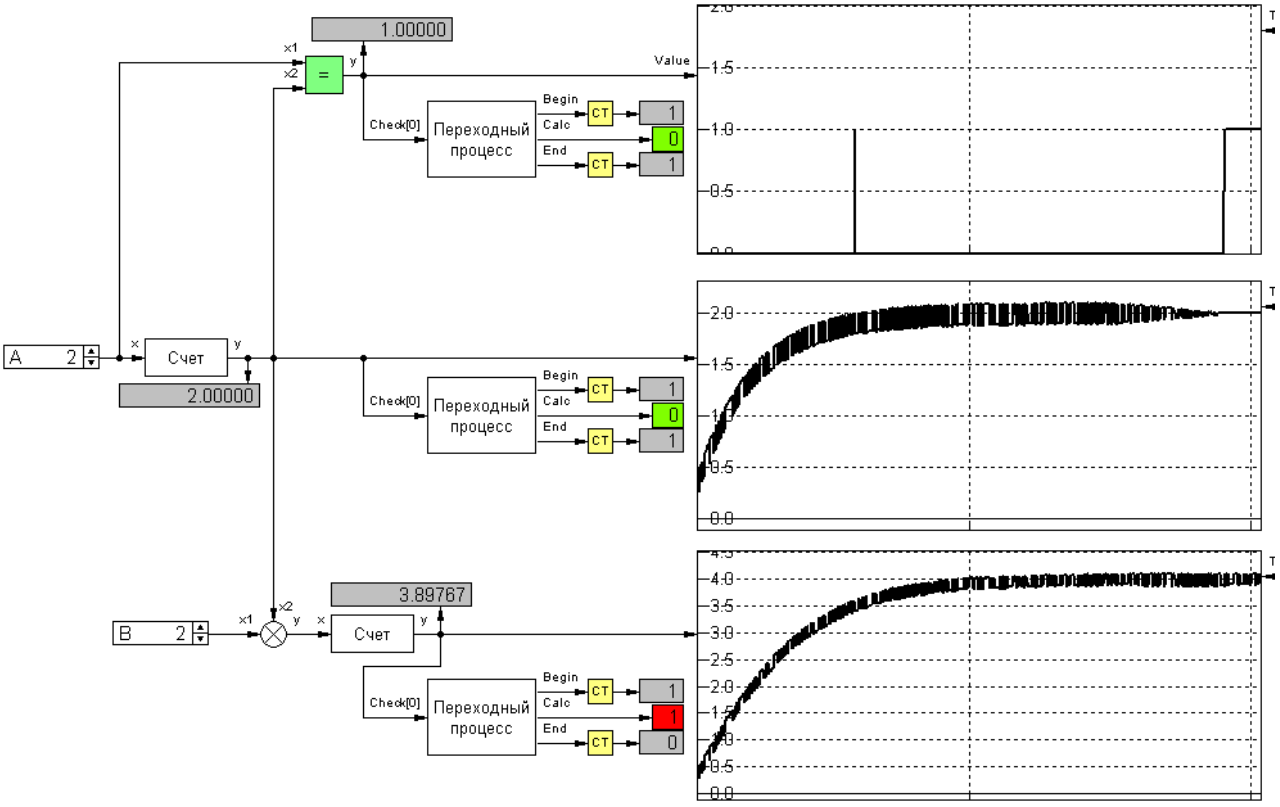

Рис. 11. Три переходных процесса в трех ветвях схемы

В средней ветви к тестовому блоку “Счет” подключено поле ввода “A”, и наблюдение за переходным процессом ведется относительно выхода тестового блока. В нижней ветви выход блока “Счет” из средней ветви складывается со значением из поля ввода “B” и подается на второй тестовый блок, на выходе которого ведется наблюдение за переходным процессом. Наконец, в верхней ветви схемы находится блок сравнения значения из поля ввода “A” и выхода первого тестового блока, и ведется наблюдение за переходным процессом на логическом выходе блока сравнения.

Если запустить расчет, начнутся все три переходных процесса: все три семафора окрасятся в красный цвет, и на выходах счетчиков, подключенных к сигналам “Begin”, появятся единицы. На среднем и нижнем графиках можно будет наблюдать изменения выходов блоков “Счет” в процессе их работы. Верхний график, подключенный к логическому блоку сравнения, будет показывать горизонтальную линию на уровне нуля с возможными одиночными выбросами до единицы: пока блок “Счет” в средней ветви не закончит расчет и шум на его выходе не прекратится, выход этого блока не будет равен его входу, за исключением редких моментов из-за шума. Поэтому логическое значение на выходе блока сравнения будет держаться на нуле. Но, несмотря на это, блок обнаружения переходного процесса, подключенный к этому выходу, будет регистрировать процесс, поскольку в другой части схемы, от которой зависит наблюдаемый выход, значения меняются.

Когда блок “Счет” в средней ветви схемы закончит работу, практически одновременно закончатся переходные процессы в средней и зависящей от нее верхней ветвях: семафоры окрасятся в зеленый, и на выходах счетчиков, подключенных к сигналам “End”, увеличатся значения. Переходный процесс в нижней ветви схемы будет идти еще некоторое время, пока не завершит работу второй блок “Счет”. Теперь изменение значения в поле ввода “A” будет вызывать переходные процессы во всех трех ветвях, поскольку нижняя и верхняя зависят от средней. Изменение значения в поле ввода “B” будет вызывать переходный процесс только в нижней ветви. Это будет наглядно видно по семафорам и счетчикам сигналов, подключенным к блокам обнаружения переходных процессов.

4.2. Схема “2_Ready.rds” – один блок на два процесса

Схема “2_Ready.rds” содержит две независимых ветви с общим блоком обнаружения переходного процесса (рис. 12).

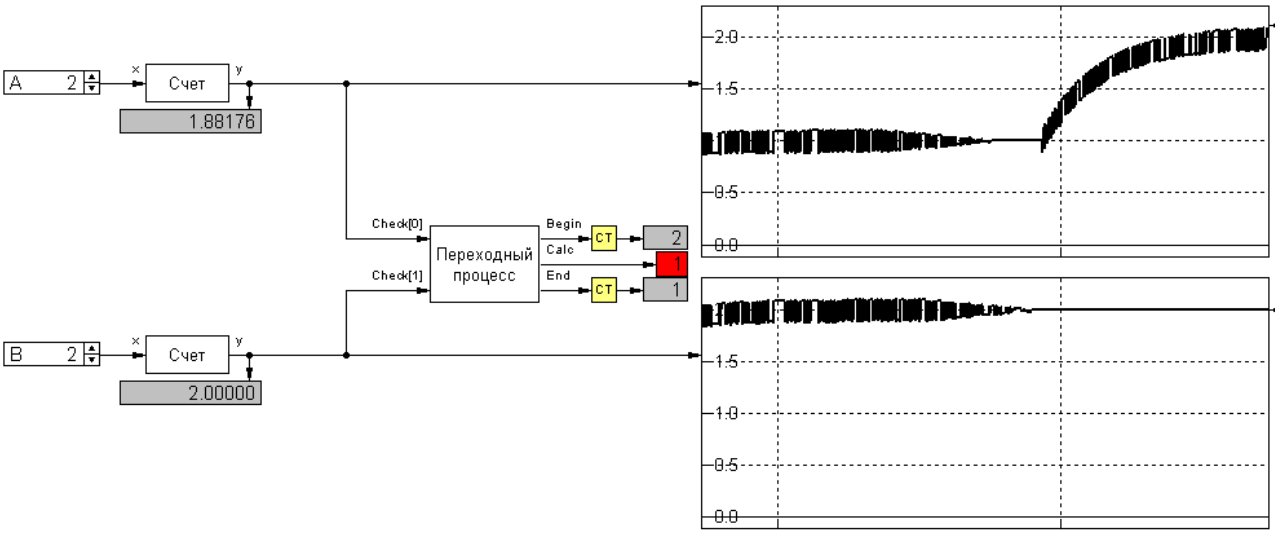

Рис. 12. Общий блок наблюдения для двух независимых процессов

Выход верхней ветви схемы присоединен к входу блока “Check[0]”, выход нижней – к “Check[1]”. Это значит, что блок сообщит об окончании переходного процесса только тогда, когда прекратятся изменения данных в обеих частях схемы. Это видно на рисунке: в нижней

ветви процесс уже закончился (график показывает ровную линию), но в верхней он еще идет, поэтому семафор окрашен в красный цвет, а выход счетчика сигналов “Begin” на единицу больше выхода счетчика сигналов “End” (соответствующий сигнал об окончании процесса еще не поступил).

4.3. Схема “3_Spoiler.rds” – обнуляющий выход блок

Схема “3_Spoiler.rds” содержит две ветви (рис. 13). В верхней ветви поле ввода “A” подключено к входу тестового блока “Счет”, на выходе которого наблюдается переходный процесс. К этому же выходу подключен специальный блок “Ноль”, который, независимо от того, что поступает на его вход, выдает ноль на свой выход. Этот выход в нижней ветви суммируется со значением поля ввода “B”, и на выходе сумматора наблюдается второй переходный процесс.

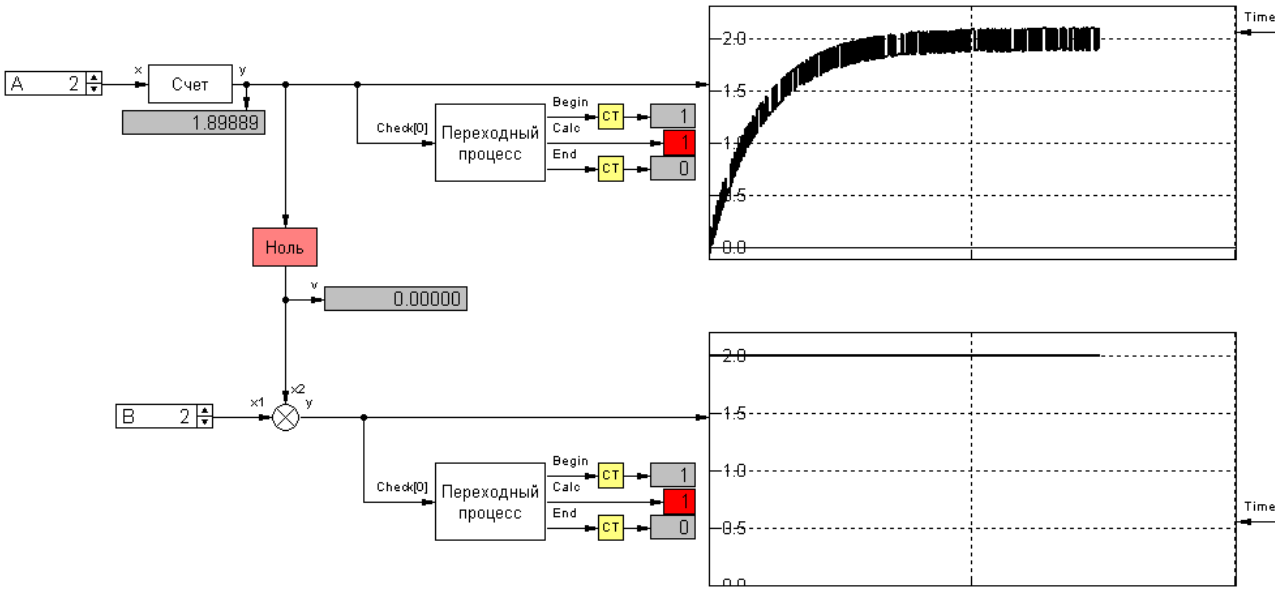

Рис. 13. Обнуляющий выход блок не мешает обнаружению переходного процесса

Можно видеть что, несмотря на то, что на выходе сумматора в нижней ветви значение остается постоянным и равным значению поля ввода “B”, блок показывает там наличие переходного процесса. Это происходит потому, что блок обнаружения переходного процесса в нижней ветви видит, что на входе блока “Ноль”, который формально может влиять на нижнюю ветвь, изменяется значение. Механизм обнаружения переходных процессов, естественно, не знает о том, что модель блока “Ноль” написана так, что его вход игнорируется и не влияет на выход. Это было полезно в схеме из примера “1_Ready.rds” (п. 4.1), где переходный процесс обнаруживался на выходе блока сравнения, хотя само значение на его выходе не менялось. Но, в данном случае, если бы блок “Ноль” был выключателем, переведенным в выключенное состояние пользователем, его вход нужно было бы игнорировать. Именно поэтому, как уже упоминалось в п. 2.1, модели блоков-переключателей должны быть написаны специальным образом, чтобы правильно передавать на выход информацию о переходных процессах на своих входах. Общие правила написания таких моделей приведены в п. 6.

4.4. Схема “4_Switcher.rds” – влияние выключателей

Схема “4_Switcher.rds” (рис. 14) состоит из трех ветвей. В верхней ветви к полю ввода “A” подключен блок “Счет”, выход которого через стандартные блоки-выключатели подключается к сумматорам в средней и нижней ветви, причем в нижней ветви этот выход

подается на сумматор через обнуляющий выход блок “Ноль” из предыдущего примера “3_Spoiler.rds” (п. 4.3).

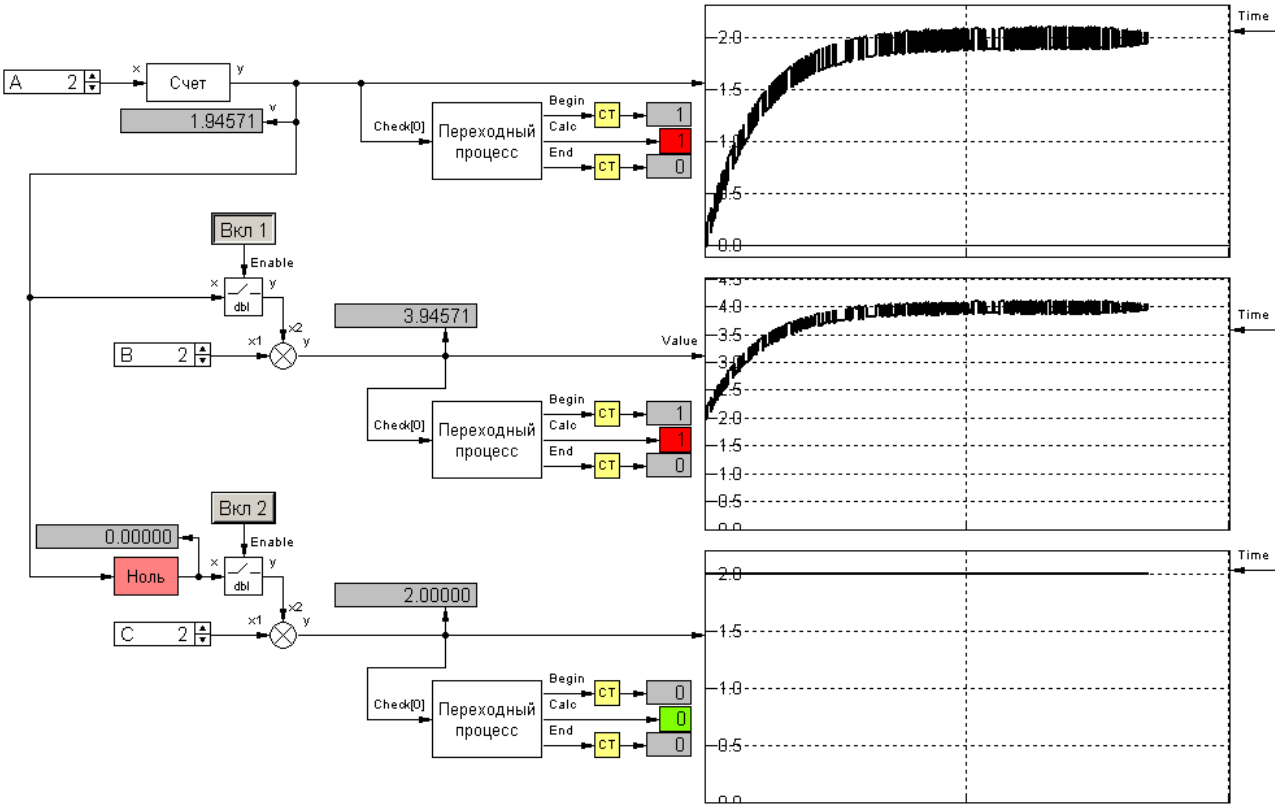

Рис. 14. Блоки-выключатели отсекают информацию о переходных процессах на входах в выключенном состоянии

Модели блоков-выключателей сделаны так, что в выключенном состоянии информация о переходном процессе на входе блока не доходит до выхода. На рисунке видно, что в верхней ветви схемы идет переходный процесс. В средней ветви схемы выключатель включен (кнопка “Вкл 1” нажата), поэтому переходный процесс там тоже наблюдается. В нижней же ветви выключатель разрывает соединение, и переходный процесс там не обнаруживается – семафор окрашен в зеленый цвет.

Если нажать кнопку “Вкл 2”, подключив тем самым нижнюю ветвь схемы к верхней, переходный процесс, идущий в верхней ветви, будет обнаруживаться и в нижней, несмотря на то, что реальные данные с выхода блока “Счет” на нижнюю ветвь не проходят из-за блока “Ноль”, и значение на графике не будет изменяться.

С точки зрения обнаружения переходных процессов блоки-выключатели работают следующим образом:

- переходный процесс на входе включения “Enable” всегда обнаруживается на выходе блока;
- переходный процесс на входе данных “x” обнаруживается на выходе блока только тогда, когда блок включен, т. е. на входе “Enable” не нулевое значение.

4.5. Схема “5_MS.rds” – мультиплексор

В схеме “5_MS.rds” (рис. 15) мультиплексор переключает два входа на один выход.

Левая часть схемы состоит из двух независимых ветвей: верхняя подключена к входу мультиплексора “X[0]”, нижняя – к “X[1]”. Выход мультиплексора через сумматор подключен к третьей (правой) ветви схемы.

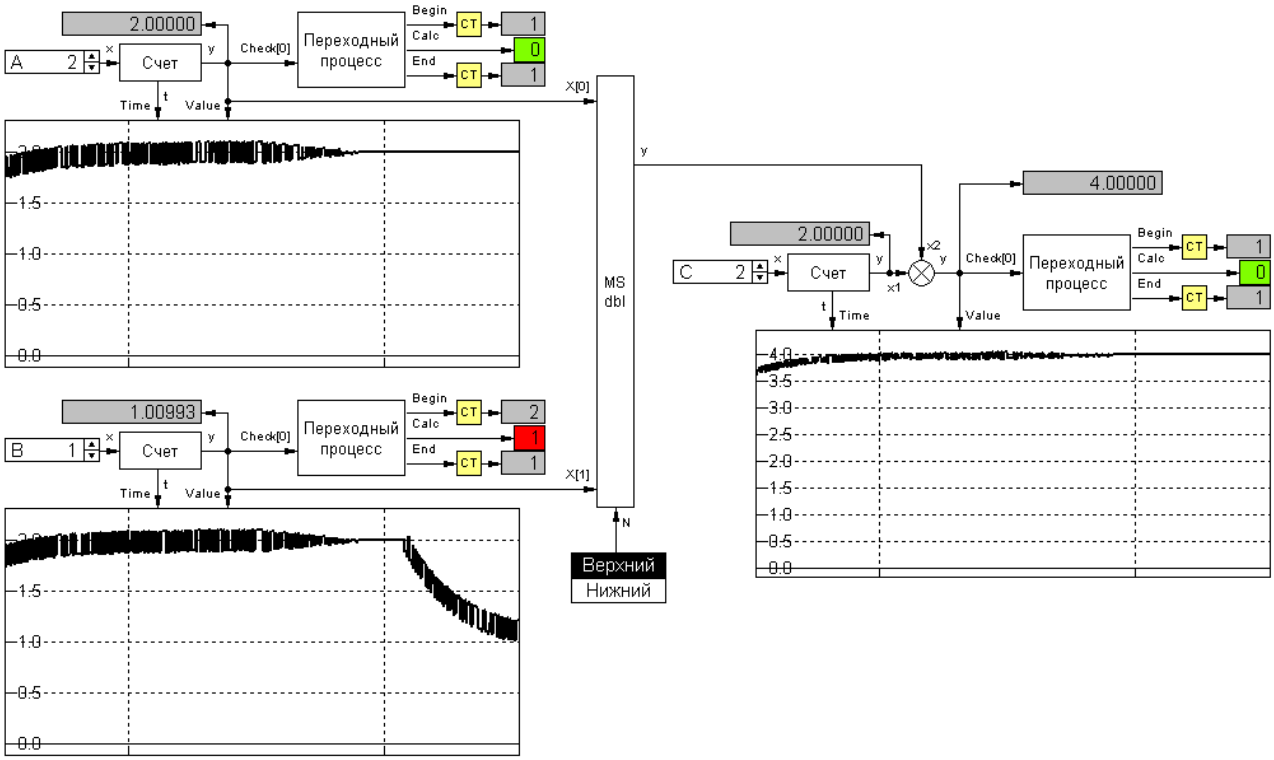

Рис. 15. Мультиплексор “MS dbl” передает на выход только один из входов

Мультиплексор передает на свой выход значение только с того элемента входного массива “X”, индекс которого поступает на вход “N”. Информацию о переходных процессах он передает таким же образом. На рисунке видно, что в левой нижней ветви схемы идет переходный процесс. Но в правой ветви он не обнаруживается: блок выбора варианта, подключенный к входу “N” мультиплексора, разрешает передачу на выход только данных с входа “X[0]”, а в подключенной к нему ветви переходного процесса нет. Если щелкнуть мышью по надписи “Нижний”, к выходу мультиплексора подключится вход “X[1]”, и переходный процесс на выходе сразу же будет обнаружен.

С точки зрения обнаружения переходных процессов мультиплексоры работают следующим образом:

- переходный процесс на входе номера “N” всегда обнаруживается на выходе блока;
- из всех переходных процессов в частях схемы, подключенных к *отдельным элементам* входного массива “X”, на выходе будет обнаружен только тот, который подключен к элементу с индексом, равным значению входа “N”;
- если к входу “X” подключена матрица (этот вариант практически не используется), переходный процесс на таком входе будет всегда обнаруживаться на выходе.

4.6. Схема “6_SW.rds” – демультиплексор

В схеме “6_SW.rds” (рис. 16) демультиплексор переключает один вход на два выхода.

В левой части схемы находится тестовый блок “Счет”, выход которого подключен к входу “x” демультиплексора “SW dbl”. В правой части схемы – две независимых ветви: верхняя подключена к выходу демультиплексора “Y[0]”, правая – к “Y[1]”. К входу “N” подключен блок выбора варианта, с помощью которого можно выбирать выход, получающий значение входа.

На рисунке видно, что в левой части схемы идет переходный процесс. Блок выбора варианта разрешает передачу данных оттуда только в правую верхнюю ветвь, поэтому переходный процесс обнаруживается и там (семафор окрашен в красный цвет). Данные в

нижнюю правую ветвь не передаются, поэтому там не виден переходный процесс (семафор зеленый), хотя формально на входе одного из блоков во влияющем подграфе есть изменение данных.

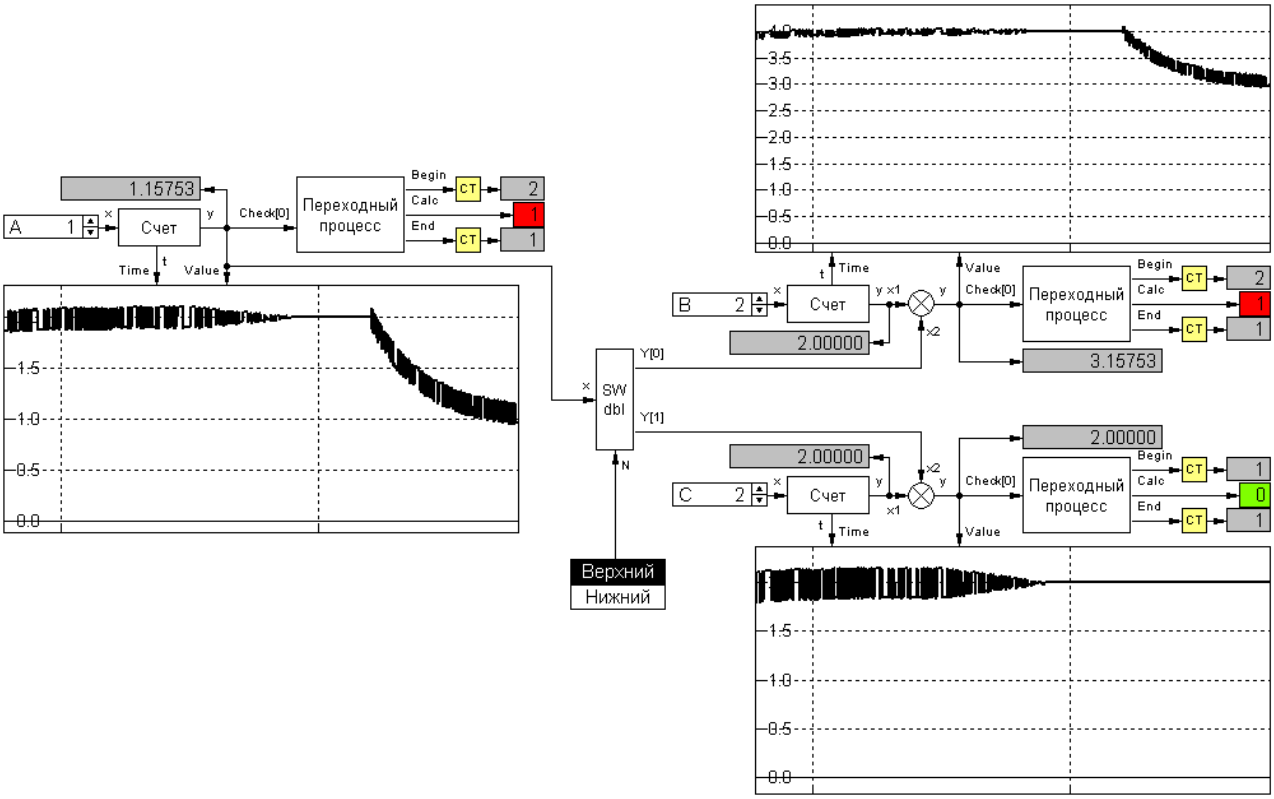

Рис. 16. Демультиплексор “SW dbl” передает вход только на один из выходов

С точки зрения обнаружения переходных процессов демультиплексоры работают следующим образом:

- переходный процесс на входе номера “N” всегда обнаруживается на *всех* выходах блока;
- переходный процесс на входе данных обнаруживается только в той части схемы, которая подключена к элементу выходного массива данных “Y[…]”, индекс которого равен текущему значению входа “N”.

4.7. Схема “7_DynVar.rds” – передача данных через динамическую переменную

Схема “7_DynVar.rds” (рис. 17) представляет собой схему примера “1_Ready.rds” (п. 4.1, рис. 11), в которой линии связи, соединяющие ветви схемы, заменены на передачу данных через динамические переменные.

На рисунке можно видеть, что, несмотря на то, что между блоком “Счет”, присоединенным к полю ввода “A”, и остальными частями схемы нет линий связи, переходный процесс, вызванный его работой, обнаруживается во всех трех ветвях. Причем в верхней ветви схемы, получающей логическое значение через динамическую переменную “Equal”, данные не изменяются, то есть обнаружение переходного процесса (красный семафор) вызвано не изменением значений в самой ветви и ее подграфе связей, а изменениями значений “по ту сторону” динамической переменной.

Такое обнаружение переходного процесса возможно из-за изменений, внесенных в стандартный блок работы с динамическими переменными. Если пользователь создает

собственные (например, автокомпилируемые) блоки, записывающие данные в динамические переменные, ему нужно уведомить RDS об этом (см. п. 2.2 и 6.4).

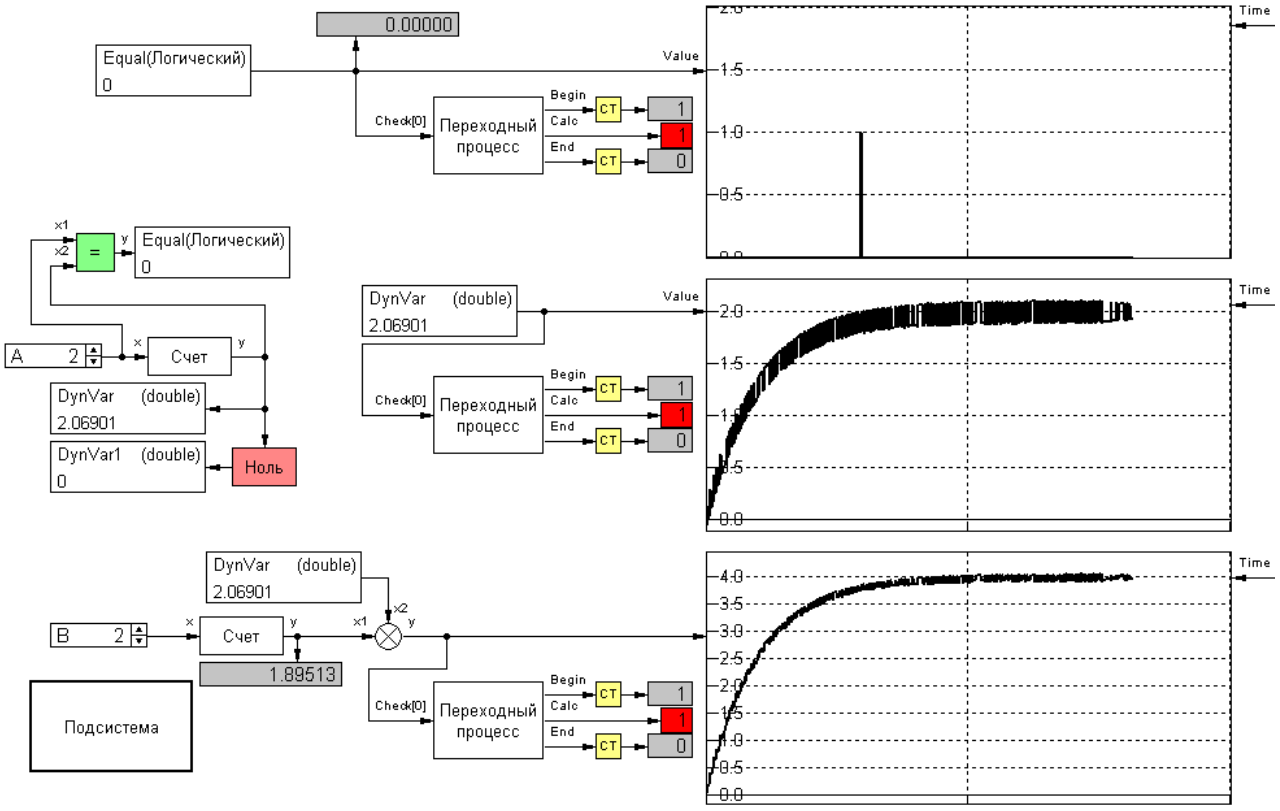

Рис. 17. Несмотря на отсутствие линий связи, переходные процессы обнаруживаются в частях схемы, получающих данные через динамические переменные

В левой нижней части схемы находится подсистема (ее содержимое на рисунке не приводится), внутри которой переходный процесс в данных, передающихся через динамическую переменную “DynVar1”, тоже обнаруживается, несмотря на полное отсутствие связей, входящих в эту подсистему.

4.8. Схема “8_Max.rds” – “черный ящик” с внешним максимизатором

В схеме “8_Max.rds” (рис. 18) блок наблюдения за переходным процессом используется для формирования сигнала готовности данных очередной итерации в обратной связи внешнего максимизатора.

В верхней части схемы из отдельных блоков собран “черный ящик”, вычисляющий формулу

$$-(x_1-x_{1o})^2-(x_2-x_{2o})^2.$$

Чтобы переходные процессы в этой части схемы можно было увидеть, в нее включены два тестовых блока “Счет”, задерживающих установление конечных значений. В нижней части схемы находится внешний максимизатор, настроенный на перебор двух параметров: x_1 (“P[0]”) и x_2 (“P[1]”) в диапазоне [–5,5] с шагом 1. В качестве значений x_{1o} и x_{2o} заданы 1 и 2 соответственно. Таким образом, очевидно, что максимизируемая функция представляет собой перевернутый параболоид вращения с максимумом в точке (1,2).

Для того, чтобы максимизатор не начинал следующую итерацию раньше времени, к выходу схемы вычисления максимизируемой функции (“черного ящика”) подключен вход “Check[0]” блока наблюдения за переходным процессом, выходной сигнал “End” (окончание процесса) которого подается на вход “CritReady” (готовность значения критерия)

максимизатора. Сигнал начала итерации максимизатора “Go” одновременно подается на входы “Check[1]” и “Reset” блока наблюдения, причем блок настроен так, чтобы не реагировать на новый переходный процесс до сигнала “Reset” (см. п. 3). На вход “Crit” максимизатора подается вычисленное значение функции.

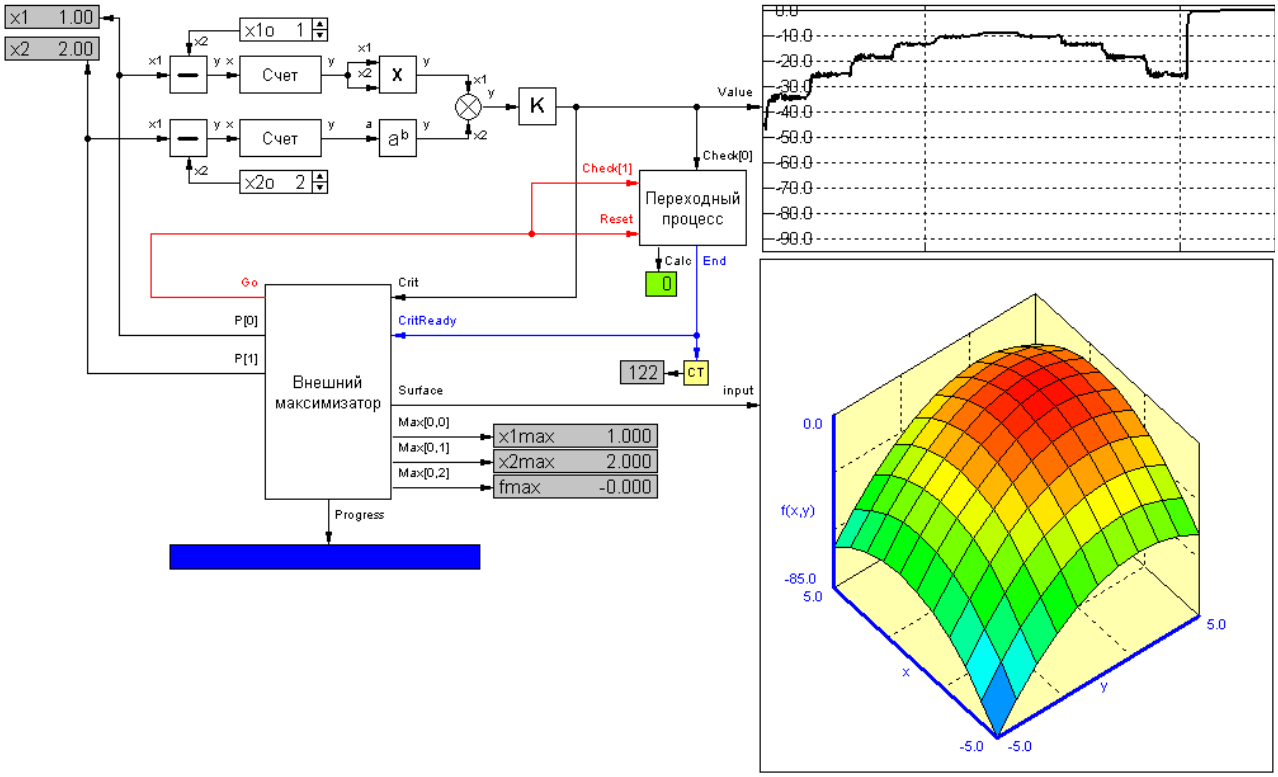

Рис. 18. Блок наблюдения за переходным процессом, формирующий сигнал готовности данных для внешнего максимизатора

Когда максимизатор начинает очередную итерацию, он устанавливает на своих выходах “P[…]” очередную пару значений параметров, выдает сигнал “Go” и начинает ждать сигнала на входе “CritReady”. Сигнал “Go” попадает на вход “Reset” блока наблюдения и выводит его из режима ожидания. Одновременно он попадает на вход “Check[1]” этого же блока, из-за чего блок немедленно регистрирует начало переходного процесса. В схеме “черного ящика” при этом начинается изменение данных в процессе вычисления значения функции для новых значений параметров. Только когда эти изменения закончатся и выходное значение установится, блок наблюдения выдаст сигнал “End”, который, поступив в максимизатор, заставит его взять вычисленное значение функции с входа “Crit” и перейти к следующей итерации.

Если запустить расчет, можно увидеть, что, несмотря на сильные колебания значений на входе максимизатора “Crit”, трехмерный график отобразит правильный вид исследованной функции – это значит, что значения функции считывались в правильные моменты. Счетчик сигналов “End” при этом покажет значение “122”: значения функции были вычислены в сетке 11x11 точек, и последнюю итерацию максимизатор выполнил с найденными оптимальными значениями параметров.

В данной схеме можно было не настраивать блок наблюдения на переход в режим ожидания по окончании переходного процесса и не подключать сигнал к его входу “Reset” – схема работала бы и так. Но в более сложных схемах, где в некоторых режимах работы “черный ящик” может использоваться не только для максимизации, лучше сделать эти настройки и подключения, чтобы переходные процессы, не связанные с максимизацией, игнорировались и не подавали сигналов на максимизатор.

Подача сигнала “Go” с максимизатора на вход “Check[1]” блока наблюдения в данной схеме необходима в любом случае. Дело в том, что вычисляемая “черным ящиком” функция может оказаться такой, что ее максимум будет найден в самой последней итерации перебора. Если это произойдет, после окончания перебора для последнего прохода с максимальными значениями максимизатор снова установит на выходе *те же самые* значения параметров, и будет ждать сигнала о готовности данных функции. Но, поскольку значения на входе “черного ящика” не изменились, переходного процесса в схеме не будет, и блок наблюдения никогда на подаст сигнала “End” на основании наблюдения только за самим “черным ящиком” через вход “Check[0]”. Подача сигнала “Go” на “Check[1]” гарантирует, что блок зарегистрирует начало переходного процесса в каждой, даже “лишней”, итерации, какие бы значения максимизатор не подал на “черный ящик”.

4.9. Схема “9_Restarter.rds” – перезапускающийся внутренний максимизатор

Схема “9_Restarter.rds” (рис. 19) иллюстрирует обнаружение переходного процесса в схеме, блоки которой разбивают свой внутренний расчет на несколько тактов, перезапуская сами себя. Обычно это делается для того, чтобы не блокировать полностью работу RDS до завершения длительного расчета: пока модель блока не вернет управление, RDS не будет реагировать на действия пользователя и другие события. В данной схеме в качестве такого перезапускающегося блока используется внутренний максимизатор, внутрь которого заложена функция

$$-x^2 - k\,y^2.$$

Для того, чтобы переходный процесс в схеме не протекал слишком быстро, и его можно было увидеть, к оператору вычисления функции добавлена задержка при помощи вызова “`Sleep(5)`”.

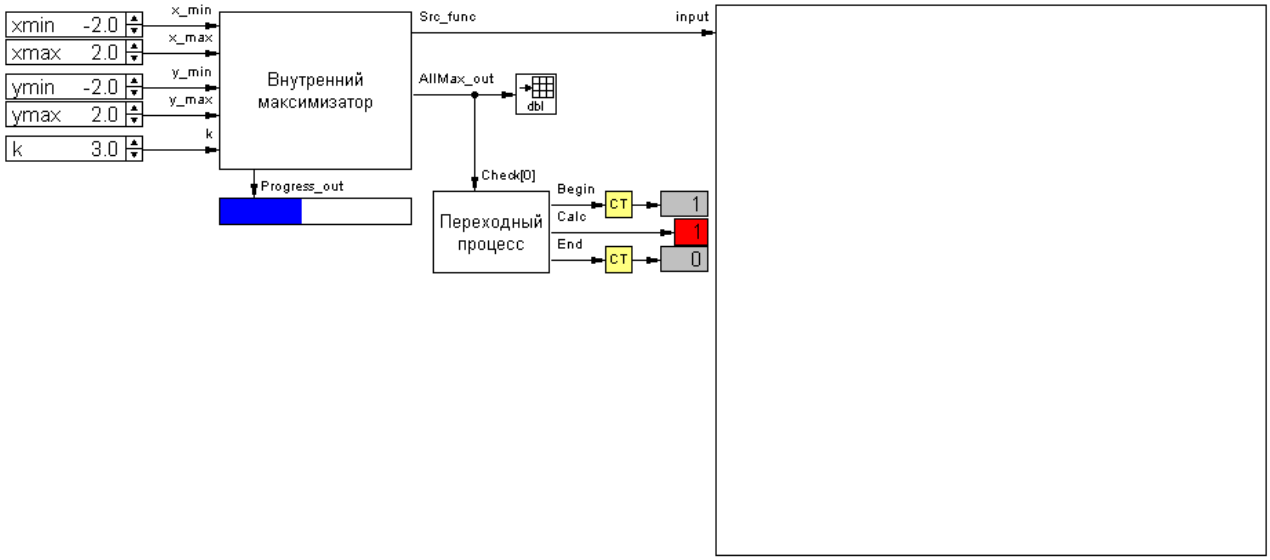

Рис. 19. Схема с перезапускающимся блоком – переходный процесс идет, хотя изменений в наблюдаемом подграфе нет

На рис. 19 схема изображена в середине переходного процесса – максимизатор продолжает вычислять функцию и перезапускаться, это видно по не заполненному до конца индикатору, подключенному к выходу “Progress_out”. Блок наблюдения за переходным процессом, подключенный к выходу “AllMax_out” (матрица всех найденных максимумов) показывает наличие переходного процесса, несмотря на то, что никакие данные в подграфе, влияющем на этот выход (в него входит связь от “AllMax_out” и все входные связи

максимизатора) не передаются уже в течение многих тактов, и значения на входах не меняются. Это достигается за счет того, что максимизатор, перезапускаясь, устанавливает специальный признак незавершенного расчета.

Когда расчет в макисимизаторе закончится, он уже не будет устанавливать этот признак, и блок наблюдения получит информацию о завершении переходного процесса (рис. 20).

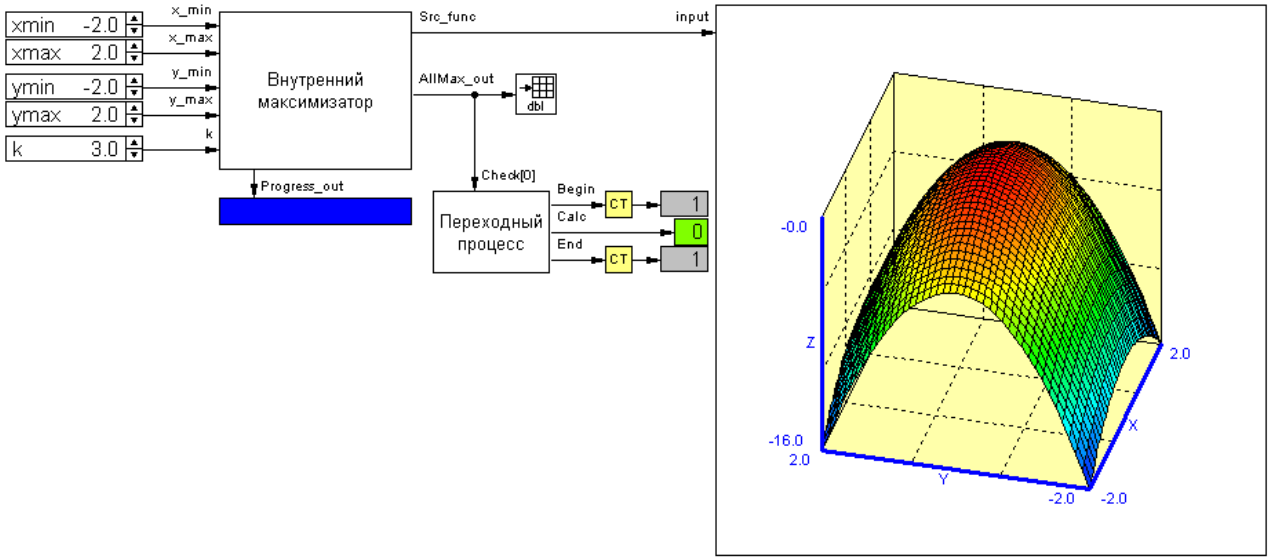

Рис. 20. Схема с перезапускающимся блоком – переходный процесс завершен

Модели всех стандартных перезапускающихся блоков уже изменены таким образом, чтобы сообщать в RDS о переходных процессах. Если пользователь будет создавать собственные модели подобных блоков, ему нужно будет ввести в них установку признака незавершенного расчета (см. п. 6.5).

4.10. Схема “10_Lock.rds” – запоминание входных значений

Схема “10_Lock.rds” (рис. 21) иллюстрирует использование блока обнаружения переходного процесса в качестве регистра-защелки для входа “Check”.

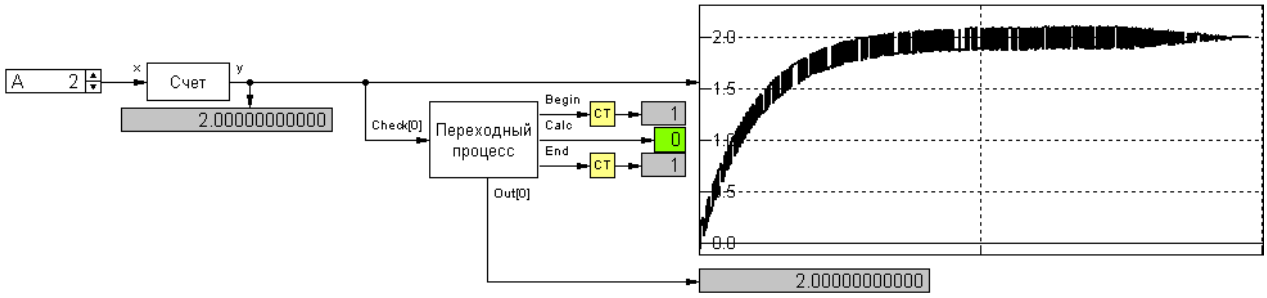

Рис. 21. Запоминание входных значений блока

У блока есть выходной массив “Out”, на который выдаются значения, оказавшиеся во входном массиве “Check” на момент окончания переходного процесса. Значение, поданное на “Check[0]”, будет передано на “Out[0]”, “Check[1]” – на “Out[1]” и т. д. Значения выдаются только тогда, когда переходный процесс закончится, они не меняются, пока процесс идет. Хотя значение на входе “Check[0]” изменяется постоянно, значение на индикаторе под графиком на рисунке изменится только по окончании переходного процесса. Если изменить значение в поле ввода “A”, значение на индикаторе будет оставаться неизменным до

окончания нового переходного процесса в схеме, и только потом скачком изменится на новое, установившееся, значение.

4.11. Схема “11_Bad.rds” – расходящийся процесс

Схема “11_Bad.rds” (рис. 19) содержит специальный демонстрационный блок “Шум”, выдающий на выход постоянно изменяющиеся значения. Выход этого блока пропускается через обнуляющий блок и через выключатель суммируется со схемой, выдающей сходящийся переходный процесс (см. рис. 9). К выходу сумматора подключен блок обнаружения переходного процесса, в настройках которого разрешена проверка на расхождение (см. п. 3).

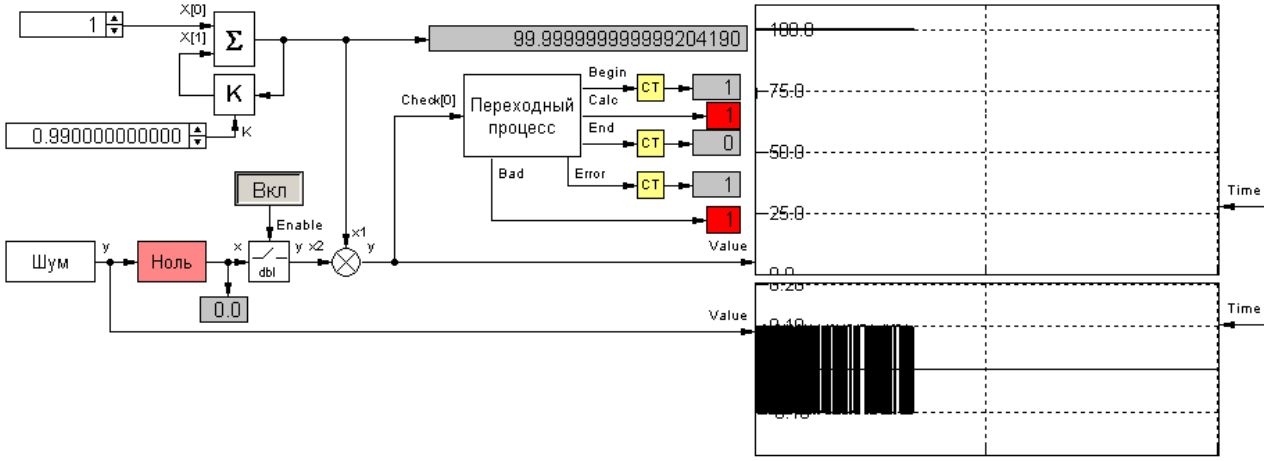

Рис. 22. Обнаружение расходящегося переходного процесса

В этой схеме для иллюстрации расходящегося процесса нельзя использовать блок “Счет”, создававший тестовый переходный процесс во всех предыдущих примерах. Дело в том, что блок “Счет” заканчивает свой процесс по времени, чтобы его хорошо было видно на графиках. Из-за этого процесс может занимать очень много тактов, причем число тактов зависит от быстродействия машины, на которой работает схема. А обнаружение расходящегося процесса работает именно по числу тактов, поэтому, в большинстве случаев, выход блока “Счет” будет считаться расходящимся процессом – по числу тактов он слишком длинный. Здесь для генерации сходящегося процесса используется сумматор с меньшим единицы коэффициентом в обратной связи, его выход плавно увеличивается до установившегося значения с постоянно уменьшающимся приращением – это типичный сходящийся процесс. Скорость схождения можно регулировать, увеличивая коэффициент в обратной связи: чем ближе он к единице, тем дольше будет идти процесс (при коэффициенте, равном единице или большем, процесс расходится).

Если выключатель включен, в анализируемый подграф будет входить выход блока “Шум”, и блок обнаружения переходного процесса вместо сигнала “End” выдаст сигнал “Error”.

5. Новые описания, функции и события для наблюдения за переходными процессами

В п. 5 описываются дополнения к программному интерфейсу RDS, предназначенные для создания моделей блоков с наблюдением за переходными процессами. Общие принципы использования этих дополнений для различных типовых моделей приведены в п. 6.

5.1. Дополнительные флаги блока

Для того, чтобы модели блоков-переключателей и перезапускающих себя блоков (см. п. 2) могли участвовать в обнаружении переходных процессов, в поле `Flags` структуры данных блока `RDS_BLOCKDATA` (см. приложение А.2.3) добавлено несколько новых флагов:

`RDS_CONNCHANGECTRL`

Если этот флаг взведен, блок "разрывает" анализируемый подграф связей. При этом он должен самостоятельно определять, как переходные процессы на его входах влияют на переходные процессы на выходах. Обычно этот флаг взводят модели блоков-переключателей, мультиплексоров и т. п. (см. п. 2.1). После установки этого флага блок начинает реагировать на дополнительные события (см. п. 5.5).

`RDS_DYNVARBYINPUT`

Если этот флаг взведен, входы блока влияют на значения всех динамических переменных, на которые он подписан. Этот факт используется при анализе подграфа связей, влияющего на какое-либо выходное значение (см. п. 2.2). При этом считается, что любой вход влияет на все динамические переменные, как-либо разделить их невозможно.

`RDS_MULTICALC`

Взведением этого флага блок сообщает о том, что он может перезапускать сам себя для выполнения длительных расчетов (см. п. 2.3). Для таких блоков будет проверяться состояние флага `RDS_MULTICALCON`, описанного ниже, и переходный процесс не будет считаться завершенным, пока блок не прекратит перезапускаться.

`RDS_MULTICALCON`

Если блок перезапустил сам себя для продолжения длительного расчета, который он не успел выполнить за прошедший такт, этот флаг необходимо установить. Флаг автоматически сбрасывается перед каждым вызовом реакции блока на такт расчета, поэтому его необходимо устанавливать при каждом перезапуске. Флаг проверяется только у блоков, у которых установлен еще и `RDS_MULTICALC`.

5.2. `RDS_CWCREATEDATA` – структура описания наблюдения за процессом

Структура `RDS_CWCREATEDATA` используется при создании вспомогательного объекта для наблюдения за переходным процессом, в ней указываются значения погрешностей и другие параметры наблюдения. Поля структуры соответствуют настройкам стандартного блока обнаружения переходного процесса, описанного в п. 3.

```
typedef struct
{ DWORD servSize;     // Размер этой структуры в байтах
  int Ticks;          // Число последовательных тактов
                      // без изменений
  double RelDelta;    // Относительная погрешность в долях
                      // значения, или -1 для отключения
  double MinDeltaRel; // Абсолютный порог относительной погрешности
  double AbsDelta;    // Абсолютная погрешность или -1
                      // для отключения
  BOOL CheckBadProcess;    // Проверка на расходящийся процесс
  int MinBadProcessTicks;  // Интервал измерения (в тактах) для
                           // расходящегося процесса
} RDS_CWCREATEDATA;
typedef RDS_CWCREATEDATA *RDS_PCWCREATEDATA;
```

Поля структуры:

`servSize`
: Размер этой структуры в байтах. Перед вызовом любой сервисной функции, работающей с этой структурой, полю `servSize` необходимо присвоить значение `sizeof(RDS_CWCREATEDATA)`.

`Ticks`
: Число последовательных тактов расчета, в течение которых изменения во всех связях исследуемого подграфа схемы должны укладываться в допустимые погрешности, прежде чем наступит событие окончания переходного процесса.

`RelDelta`
: Относительная погрешность изменения входа блока при срабатывании связи, ниже которой вход будет считаться не изменившимся. Погрешность задается в долях значения входа (не в процентах, как в настройках стандартного блока), задание “–1” отключает проверку относительной погрешности.

`MinDeltaRel`
: Порог абсолютного изменения входа блока при срабатывании связи, ниже которого проверка относительной погрешности не производится. Без этого порога при переходных процессах, плавно сходящихся к нулевому значению на входе блока, вычисляемое значение относительной погрешности стремится к нулю вместе с самим значением, и проверка попадания в эту погрешность никогда не выполнится.

`AbsDelta`
: Абсолютная погрешность изменения входа блока при срабатывании связи, ниже которой вход будет считаться не изменившимся. Задание “–1” отключает проверку абсолютной погрешности.

`CheckBadProcess`
: Разрешение (`TRUE`) или запрет (`FALSE`) определения расходящихся переходных процессов.

`MinBadProcessTicks`
: Минимально допустимое число тактов в интервале проверки уменьшения изменений на входах блоков для определения расходящегося переходного процесса (блок может автоматически скорректировать это значение в большую сторону, если в подграфе много связей и блоков).

Если не заданы ни относительная, ни абсолютная погрешности, отсутствием изменений при передаче значения по связи будет считаться точное (до бита) совпадение значений.

5.3. Функции и макросы

5.3.1. Макрос `RDS_CWCREATEDATA_RESET` – инициализация структуры описания

Макрос `RDS_CWCREATEDATA_RESET` предназначен для начальной установки всех полей в структуре `RDS_CWCREATEDATA` (см. п. 5.2) таким образом, чтобы все погрешности и все дополнительные проверки были отключены.

```
RDS_CWCREATEDATA_RESET(
      data       // Структура RDS_CWCREATEDATA
)
```

Определение:

```
#define RDS_CWCREATEDATA_RESET(data) \
      {data.servSize=sizeof(data); \
       data.Ticks=data.MinBadProcessTicks=0; \
       data.RelDelta=data.MinDeltaRel=data.AbsDelta=-1.0; \
       data.CheckBadProcess=FALSE;}
```

Параметр:

`data`

Очищаемая структура `RDS_CWCREATEDATA` (сам объект, не указатель).

Примечания:

Этим макросом желательно очищать структуру `RDS_CWCREATEDATA` перед использованием и вызовом функций, принимающих в качестве параметра указатель на нее. Если в будущем в структуру будут добавлены новые поля, использование этого макроса гарантирует, что при компиляции старых моделей блоков эти новые поля структуры будут инициализированы так, что соответствующие им проверки будут отключены.

5.3.2. `rdsCWCreate` – функция создания объекта-наблюдателя

Функция `rdsCWCreate` создает вспомогательный объект RDS, который наблюдает за переходным процессом и сообщает указанному блоку о его начале, окончании или расхождении.

```
RDS_HOBJECT RDSCALL rdsCWCreate(
      RDS_BHANDLE Block,    // Блок
      LPSTR VarName,        // Имя входа или NULL
      int VarNum,           // Номер входа
      int Id,               // Произвольное число для реакции блока
      RDS_PCWCREATEDATA Data,// Параметры наблюдения
      DWORD Flags           // Флаги (RDS_CWF_*)
);
```

Тип указателя на эту функцию:

```
RDS_HoBhSIICwDw
```

Параметры:

`Block`

Идентификатор блока, на входах которого требуется наблюдение за переходным процессом, или `NULL`, если объект создается для блока, вызвавшего функцию. Именно у этого блока будут вызываться реакции на события, связанные с переходными процессами.

`VarName`

Имя входа блока, на котором будет определяться переходный процесс, или `NULL`, если имя не задается (имена и номера входов можно добавить позднее установкой параметров объекта `RDS_CWPARAM_ADDVARNAME` и `RDS_CWPARAM_ADDVARNUM`, описанной ниже в п. 5.3.5 и 5.3.6). Имя может содержать индексы массивов, поля структур и т. д.

`VarNum`

Номер входа блока, на котором будет определяться переходный процесс, или –1, если номер не задается (имена и номера входов можно добавить позднее). Если к входу с

данным номером подключено несколько связей, например, к разным элементам массива, будет обнаруживаться переходный процесс в любой из них.

`Id`

Произвольное целое число, передающееся в реакции блока `RDS_BFM_CONNCHANGEON` (п. 5.4.1), `RDS_BFM_CONNCHANGEOFF` (п. 5.4.2) и `RDS_BFM_CONNCHANGEERROR` (п. 5.4.3). В наблюдении за процессом не участвует.

`Data`

Указатель на структуру `RDS_CWCREATEDATA` (п. 5.2), которая должна быть заполнена параметрами наблюдения.

`Flags`

Объединенные битовым ИЛИ флаги, управляющие наблюдением:

`RDS_CWF_NOREACTION`	Не вызывать стандартные реакции блока при начале/окончании/расхождении переходного процесса (используется для организации “подчиненных” наблюдателей в блоках-переключателях, см. п. 6).
`RDS_CWF_EXACTNAMEMATCH`	При указании имени наблюдаемого входа вместо номера имя в присоединенной связи должно в точности совпадать с указанным, иначе проверяется только начало имени (например, при указании “x” будет найдена и “x[0]”).
`RDS_CWF_PARENTONLY`	Ограничить отслеживание переходного процесса только родительской подсистемой блока и вложенными в нее.

Возвращаемое значение:

Уникальный идентификатор созданного объекта.

Примечания:

Эта функция создает вспомогательный объект для наблюдения за переходным процессом в заданном блоке. Созданный объект будет вызывать у блока реакции на начало, окончание и расхождение (если разрешено в параметрах) переходного процесса. Для уничтожения объекта и прекращения наблюдения используется стандартная функция `rdsDeleteObject` (см. приложение А.5.22.4).

5.3.3. `rdsCWCreateCopy` – функция создания копии объекта-наблюдателя

Функция `rdsCWCreateCopy` создает вспомогательный объект RDS, который наблюдает за переходным процессом, считывая параметры наблюдения из другого, ранее созданного, объекта (чаще всего привязанного к какому-то другому блоку). Обычно она применяется в блоках-переключателях для создания “подчиненных” наблюдателей (см. п. 6).

```
RDS_HOBJECT RDSCALL rdsCWCreateCopy(
      RDS_BHANDLE Block,    // Блок
      LPSTR VarName,        // Имя входа или NULL
      int VarNum,           // Номер входа
      int Id,               // Произвольное число для реакции блока
      DWORD Flags,          // Флаги (RDS_CWF_*)
      RDS_HOBJECT Copy,     // Объект для копирования
);
```

Тип указателя на эту функцию:

```
RDS_HoBhSIIDwHo
```

Параметры:

`Block`

Идентификатор блока, на входах которого требуется наблюдение за переходным процессом, или `NULL`, если объект создается для блока, вызвавшего функцию. Именно у этого блока будут вызываться реакции на события, связанные с переходными процессами.

`VarName`

Имя входа блока, на котором будет определяться переходный процесс, или `NULL`, если имя не задается (имена и номера входов можно добавить позднее установкой параметров объекта `RDS_CWPARAM_ADDVARNAME` и `RDS_CWPARAM_ADDVARNUM`, описанной ниже в п. 5.3.5 и 5.3.6). Имя может содержать индексы массивов, поля структур и т. д.

`VarNum`

Номер входа блока, на котором будет определяться переходный процесс, или –1, если номер не задается (имена и номера входов можно добавить позднее). Если к входу с данным номером подключено несколько связей, например, к разным элементам массива, будет обнаруживаться переходный процесс в любой из них.

`Id`

Произвольное целое число, передающееся в реакции блока `RDS_BFM_CONNCHANGEON` (п. 5.4.1), `RDS_BFM_CONNCHANGEOFF` (п. 5.4.2) и `RDS_BFM_CONNCHANGEERROR` (п. 5.4.3). В наблюдении за процессом не участвует.

`Flags`

Объединенные битовым ИЛИ флаги, управляющие наблюдением (совпадают с флагами функции `rdsCWCreate`):

`RDS_CWF_NOREACTION`	Не вызывать стандартные реакции блока при начале/окончании/расхождении переходного процесса (используется для организации “подчиненных” наблюдателей в блоках-переключателях, см. п. 6).
`RDS_CWF_EXACTNAMEMATCH`	При указании имени наблюдаемого входа вместо номера имя в присоединенной связи должно в точности совпадать с указанным, иначе проверяется только начало имени (например, при указании “x” будет найдена и “x[0]”).
`RDS_CWF_PARENTONLY`	Ограничить отслеживание переходного процесса только родительской подсистемой блока и вложенными в нее.

`Copy`

Идентификатор уже существующего объекта-наблюдателя, параметры которого копируются.

Возвращаемое значение:

Уникальный идентификатор созданного объекта.

Примечания:

Эта функция создает вспомогательный объект для наблюдения за переходным процессом в заданном блоке по данным другого такого же объекта. Созданный объект будет вызывать у блока реакции на начало, окончание и расхождение (если разрешено в параметрах) переходного процесса. Для уничтожения объекта и прекращения наблюдения используется стандартная функция `rdsDeleteObject` (см. приложение А.5.22.4).

5.3.4. `rdsCWGetParameters` – функция чтения параметров объекта-наблюдателя

Функция `rdsCWGetParameters` считывает параметры вспомогательного объекта-наблюдателя в структуру `RDS_CWCREATEDATA` (см. п. 5.2).

```
BOOL RDSCALL rdsCWGetParameters(
     RDS_HOBJECT Watcher,  // Объект-наблюдатель
     RDS_PCWCREATEDATA Data// Параметры наблюдения
);
```

Тип указателя на эту функцию:

```
RDS_BHoCw
```

Параметры:

`Watcher`

Идентификатор объекта-наблюдателя, параметры которого считываются.

`Data`

Указатель на структуру `RDS_CWCREATEDATA`, в которую записываются параметры наблюдения из объекта `Watcher`. Поле `servSize` у этой структуры должно быть заполнено до вызова.

Возвращаемое значение:

Успешность операции.

5.3.5. Команда `RDS_CWPARAM_ADDVARNAME` – добавление имени входа

Команда `RDS_CWPARAM_ADDVARNAME` добавляет новое имя входа к набору входов блока, за переходными процессами в которых наблюдает объект.

Вызов команды:

```
char *strVarName=... // Имя входа блока
rdsSetObjectStr(Watcher,RDS_CWPARAM_ADDVARNAME,0,strVarName);
```

Параметры:

`Watcher`

Идентификатор объекта-наблюдателя.

`strVarName`

Указатель на строку (`char*`), в которой записано имя добавляемого входа блока.

5.3.6. Команда RDS_CWPARAM_ADDVARNUM – добавление номера входа

Команда RDS_CWPARAM_ADDVARNUM добавляет новый номер входа к набору входов блока, за переходными процессами в которых наблюдает объект.

Вызов команды:

```
int iVarNum=... // Номер входа блока
rdsSetObjectInt(Watcher,RDS_CWPARAM_ADDVARNUM,0,iVarNum);
```

Параметры:

Watcher

Идентификатор объекта-наблюдателя.

iVarNum

Номер добавляемого входа блока.

5.3.7. rdsCWTranslateChanges – функция передачи признака изменения из одного объекта в другой

Функция rdsCWTranslateChanges передает признаки изменения в подграфе связей, за которым наблюдает один объект, в другой объект. Используется при создании блоков-переключателей для передачи информации о переходном процессе из “подчиненного” подграфа связей в основной (см. п. 6).

```
BOOL RDSCALL rdsCWTranslateChanges(
     RDS_HOBJECT FromWatcher,   // Исходный объект
     RDS_HOBJECT ToWatcher      // Объект-получатель
);
```

Тип указателя на эту функцию:

```
RDS_BHoHo
```

Параметры:

FromWatcher

Идентификатор объекта-наблюдателя, признаки изменения которого считываются для передачи в другой объект.

ToWatcher

Идентификатор объекта-наблюдателя, получающего считанные признаки изменения.

Возвращаемое значение:

Наличие изменений в подграфе, за которым следит объект FromWatcher.

Примечания:

Эта функция используется в сложных блоках-переключателях, которые “разрывают” анализируемый подграф. Например, модель мультиплексора (см. пример в п. 4.5) должна передавать наблюдателю за выходным подграфом признаки изменения только в той входной ветви, которая сейчас подключена к выходу. Это делается при помощи rdsCWTranslateChanges.

5.4. Основные реакции блоков

5.4.1. RDS_BFM_CONNCHANGEON – начало переходного процесса

Поток, в котором вызывается функция модели:

Поток расчета.

Первый параметр функции модели (`int CallMode`):

Константа RDS_BFM_CONNCHANGEON.

Третий параметр функции модели (`void *ExtParam`):

Указатель на структуру RDS_CONNCHANGEDATA, в которой содержится идентификатор объекта-наблюдателя и описание события.

Возвращаемое функцией модели значение:

Не используется, можно возвращать любое значение.

Событие RDS_BFM_CONNCHANGEON возникает в момент начала переходного процесса в наблюдаемом подграфе связей у блока, если реакции при создании объекта-наблюдателя не были явно запрещены (флаг RDS_CWF_NOREACTION).

При реакции на это событие в параметре ExtParam передается указатель на структуру RDS_CONNCHANGEDATA:

```
typedef struct {
  RDS_HOBJECT Object; // Объект-наблюдатель
  int Id;             // Идентификатор
  BOOL Changes;       // Начало или конец процесса
  BOOL Error;         // Процесс расходится
  RDS_BHANDLE ObjBlock;// Запросивший блок
} RDS_CONNCHANGEDATA;
typedef RDS_CONNCHANGEDATA *RDS_PCONNCHANGEDATA;
```

Поля структуры:

`Object`

Идентификатор объекта-наблюдателя, вызвавшего реакцию блока, созданный функциями `rdsCWCreate` (п. 5.3.2) или `rdsCWCreateCopy` (п. 5.3.3).

`Id`

Идентификатор, переданный при создании объекта. Может использоваться в модели для более удобного определения, какой именно объект сигнализирует о переходном процессе, если блок создал несколько таких объектов.

`Changes`

Признак начала (TRUE) или окончания (FALSE) переходного процесса. Структура RDS_CONNCHANGEDATA используется в нескольких реакциях блока, это поле дублирует вызвавшее реакцию событие.

`Error`

Признак расходящегося переходного процесса (содержит TRUE только если структура передается как параметр в событии RDS_BFM_CONNCHANGEERROR, см. п. 5.4.3).

`ObjBlock`

Идентификатор блока, за входным подграфом которого наблюдает объект.

Примечания:

При реакции на событие `RDS_BFM_CONNCHANGEON` поле `Changes` структуры параметров всегда содержит `TRUE`, поле `Error` – `FALSE`.

5.4.2. `RDS_BFM_CONNCHANGEOFF` – окончание переходного процесса

Поток, в котором вызывается функция модели:

Поток расчета.

Первый параметр функции модели (`int CallMode`):

Константа `RDS_BFM_CONNCHANGEOFF`.

Третий параметр функции модели (`void *ExtParam`):

Указатель на структуру `RDS_CONNCHANGEDATA`, в которой содержится идентификатор объекта-наблюдателя.

Возвращаемое функцией модели значение:

Не используется, можно возвращать любое значение.

Событие `RDS_BFM_CONNCHANGEOFF` возникает в момент окончания переходного процесса в наблюдаемом подграфе связей у блока, если реакции при создании объекта-наблюдателя не были явно запрещены (флаг `RDS_CWF_NOREACTION`). Если в параметрах объекта была разрешена проверка на расходящиеся процессы, и процесс разошелся, это событие не будет вызвано, даже если позже переходный процесс окончится.

При реакции на это событие в параметре `ExtParam` передается указатель на структуру `RDS_CONNCHANGEDATA` (см. п. 5.4.1). Поля `Changes` и `Error` переданной структуры будут содержать содержать значение `FALSE`.

5.4.3. `RDS_BFM_CONNCHANGEERROR` – переходный процесс расходится

Поток, в котором вызывается функция модели:

Поток расчета.

Первый параметр функции модели (`int CallMode`):

Константа `RDS_BFM_CONNCHANGEERROR`.

Третий параметр функции модели (`void *ExtParam`):

Указатель на структуру `RDS_CONNCHANGEDATA`, в которой содержится идентификатор объекта-наблюдателя.

Возвращаемое функцией модели значение:

Не используется, можно возвращать любое значение.

Событие `RDS_BFM_CONNCHANGEERROR` возникает при срабатывании проверки расхождения переходного процесса в наблюдаемом подграфе связей у блока, если при создании объекта-наблюдателя эта проверка была включена, а реакции не были явно

запрещены (флаг RDS_CWF_NOREACTION). После этого событие окончания переходного процесса не возникает, даже если процесс позже сойдется.

При реакции на RDS_BFM_CONNCHANGEERROR в параметре ExtParam передается указатель на структуру RDS_CONNCHANGEDATA (см. п. 5.4.1). Поля Error и Changes этой структуры будут содержать содержать значение TRUE.

5.5. Дополнительные реакции блоков-переключателей

У блоков-переключателей, установивших флаг RDS_CONNCHANGECTRL (см. п. 5.1), есть дополнительный набор событий, необходимых для работы сложной логики влияния процессов на разных входах на процессы на разных выходах (см. п. 6). Такие блоки при переходе в режим моделирования обычно создают объекты-наблюдатели для разных по назначению входов, а затем, уже в режиме расчета, передают информацию о наличии изменений в отдельных входных подграфах в другие объекты, наблюдающие за их выходами.

В момент перехода в режим моделирования все блоки-переключатели (т. е. блоки с флагом RDS_CONNCHANGECTRL) вызываются для реакции на событие RDS_BFM_CONNCHGCTRLSTART (п. 5.5.1). В этой реакции они могут, если это необходимо, инициализировать какие-либо внутренние структуры данных, которые понадобятся им для дальнейшей работы.

Затем RDS выполняет составление подграфов связей для уже созданных объектов наблюдателей. В процессе этого анализа для каждой уже добавленной в подграф связи рассматриваются блоки, к выходам которых эти связи подключены. Если у такого блока не взведен флаг RDS_CONNCHANGECTRL, т. е. блок не является переключателем, в подграф, как обычно, добавляются все его входные связи, и процесс повторяется. Если же флаг взведен, вместо автоматического добавления входных связей сам блок вызывается для реакции на событие RDS_BFM_CONNCHGCTRLASK (п. 5.5.2), при этом ему сообщается, какой именно объект-наблюдатель запрашивает информацию о переходном процессе для какого именно выхода. Реагируя на это событие, блок должен самостоятельно создать новые объекты-наблюдатели для своих входов, влияющих на данный выход, и запомнить их – они будут опрашиваться им в процессе расчета. Появление новых наблюдателей приводит к новому анализу подграфов уже для них, и процедура продолжается до тех пор, пока все необходимые дополнительные наблюдатели не будут созданы и их подграфы не будут проанализированы.

Когда анализ подграфов будет полностью завершен, все блоки-переключатели будут вызваны для реакции на событие RDS_BFM_CONNCHGCTRLEND (п. 5.5.3). При этом блоки могут очистить вспомогательные структуры данных, если они были созданы в реакции на RDS_BFM_CONNCHGCTRLSTART.

Далее в режиме расчета у блоков-переключателей будет возникать событие RDS_BFM_CONNCHGCTRLPROCESS (п. 5.5.4). Реагируя на него, блок должен при помощи функции rdsCWTranslateChanges (п. 5.3.7) передать наличие изменений из входных подграфов в выходные согласно своей логике работы. Например, мультиплексор передает на свой выход изменения только в тех входных подграфах, которые подключены к активному в данный момент входу данных или к входу выбора.

5.5.1. RDS_BFM_CONNCHGCTRLSTART – предварительный вызов блоков-переключателей

Поток, в котором вызывается функция модели:

Главный поток RDS.

Первый параметр функции модели (`int CallMode`):

Константа `RDS_BFM_CONNCHGCTRLSTART`.

Третий параметр функции модели (`void *ExtParam`):

Не используется (`NULL`).

Возвращаемое функцией модели значение:

Не используется, можно возвращать любое значение.

Событие `RDS_BFM_CONNCHGCTRLSTART` возникает у блоков, установивших флаг `RDS_CONNCHANGECTRL` (см. п. 5.1), при переходе в режим моделирования. После него блок может быть вызван один или несколько раз для реакции на событие `RDS_BFM_CONNCHGCTRLASK` (п. 5.5.2), чтобы он мог создать вспомогательные объекты-наблюдатели для отдельных входов.

5.5.2. `RDS_BFM_CONNCHGCTRLASK` – запрос изменений для выхода блока

Поток, в котором вызывается функция модели:

Главный поток RDS.

Первый параметр функции модели (`int CallMode`):

Константа `RDS_BFM_CONNCHGCTRLASK`.

Третий параметр функции модели (`void *ExtParam`):

Указатель на структуру `RDS_BFM_CONNCHGASKDATA`.

Возвращаемое функцией модели значение:

Не используется, можно возвращать любое значение.

Событие `RDS_BFM_CONNCHGCTRLASK` возникает у блоков, установивших флаг `RDS_CONNCHANGECTRL` (см. п. 5.1), если какому-то объекту-наблюдателю нужна информация о переходном процессе на выходе данного блока. В реакции на него блок обычно создает свои объекты-наблюдатели для входов, чтобы передавать их изменения на выходы в реакции на событие `RDS_BFM_CONNCHGCTRLPROCESS` (п. 5.5.4).

При реакции на событие `RDS_BFM_CONNCHGCTRLASK` в параметре `ExtParam` передается указатель на структуру `RDS_BFM_CONNCHGASKDATA`:

```
typedef struct {
  RDS_HOBJECT Object; // Объект-наблюдатель за выходом блока
  LPSTR VarName;      // Имя переменной-выхода
  RDS_CHANDLE Owner;  // Связь-владелец точки выхода
  int VarNum;         // Номер переменной-выхода в блоке
} RDS_BFM_CONNCHGASKDATA;
typedef RDS_BFM_CONNCHGASKDATA *RDS_BFM_PCONNCHGASKDATA;
```

Поля структуры:

`Object`

Идентификатор объекта-наблюдателя за данным выходом блока, которому требуется информация о переходных процессах на входах.

`VarName`

Указатель на строку с именем переменной-выхода, к которой подключена связь, входящая в подграф для объекта `Object`.

`Owner`

Идентификатор связи, подключенной к выходу.

`VarNum`

Номер переменной-выхода.

Примечания:

Для сложных переменных (например, структур и массивов) в поле `VarName` содержится полное имя со всеми индексами элементов или полями структур. В поле `VarNum` при этом будет содержаться только номер самого выхода. Например, если выходной массив “Y” имеет в блоке номер 3, и связи подключены к его элементам 0 и 1, при вызове реакции для нулевого элемента в `VarName` будет находиться “Y[0]”, а в `VarNum` – “3”. При вызове для первого элемента в `VarName` будет находиться “Y[1]”, а в `VarNum` – снова “3”.

5.5.3. `RDS_BFM_CONNCHGCTRLEND` – завершающий вызов блоков-переключателей

Поток, в котором вызывается функция модели:

Главный поток RDS.

Первый параметр функции модели (`int CallMode`):

Константа `RDS_BFM_CONNCHGCTRLEND`.

Третий параметр функции модели (`void *ExtParam`):

Не используется (`NULL`).

Возвращаемое функцией модели значение:

Не используется, можно возвращать любое значение.

Событие `RDS_BFM_CONNCHGCTRLEND` возникает у блоков, установивших флаг `RDS_CONNCHANGECTRL` (см. п. 5.1), при переходе в режим моделирования после того, как анализ всех подграфов связей для всех объектов-наблюдателей закончен. Реагируя на него, блоки могут очистить вспомогательные структуры данных, если они были созданы в реакции на `RDS_BFM_CONNCHGCTRLSTART`. (см. п. 5.5.1).

5.5.4. `RDS_BFM_CONNCHGCTRLPROCESS` – передача изменений на выход

Поток, в котором вызывается функция модели:

Поток расчета.

Первый параметр функции модели (`int CallMode`):

Константа `RDS_BFM_CONNCHGCTRLPROCESS`.

Третий параметр функции модели (`void *ExtParam`):

Не используется (`NULL`).

Возвращаемое функцией модели значение:

Не используется, можно возвращать любое значение.

Событие RDS_BFM_CONNCHGCTRLPROCESS возникает у блоков, установивших флаг RDS_CONNCHANGECTRL (см. п. 5.1), в каждом такте расчета. Реагируя на это событие, блок должен передать наличие изменений из своих входных подграфов в выходные при помощи функции rdsCWTranslateChanges (п. 5.3.7). Объекты-наблюдатели за входными подграфами должны быть созданы в реакции на событие RDS_BFM_CONNCHGCTRLASK (п. 5.5.2) для каждого выходного подграфа, объект наблюдения за которым запросил данные. Основные принципы реакции на это событие в разных типах блоков приведены ниже в п. 6.

6. Общие правила построения некоторых типовых моделей блоков

6.1. Блоки-выключатели связей

Блоки-выключатели имеют один или несколько входов данных и соответствующие им выходы данных. Кроме того, они имеют логический вход разрешения, при единице на котором данные со входов передаются на выходы, а при нуле – не передаются (см. пример в п. 4.4).

Чтобы такой выключатель мог “отсекать” переходные процессы на входах в выключенном состоянии, его модель должна вести список объектов-наблюдателей за выходами. Каждый элемент списка должен содержать идентификатор выходного наблюдателя, сообщаемый блоку, а также идентификаторы наблюдателя за входом разрешения и наблюдателя за входами данных, которые блок создает сам.

Модель блока должна быть построена следующим образом.

В реакции на RDS_BFM_CONNCHGCTRLSTART (п. 5.5.1) список выходных наблюдателей очищается, все дополнительные наблюдатели за входами, созданные самим блоком, уничтожаются при помощи вызовов rdsDeleteObject (см. приложение А.5.22.4).

В реакции на RDS_BFM_CONNCHGCTRLASK (п. 5.5.2) блок выполняет следующие действия:

- объект-наблюдатель выхода, вызвавший реакцию, добавляется в список выходных наблюдателей (назовем этот выходной объект-наблюдатель “Y*i*”, где *i* – номер добавленного элемента списка);
- в этом же *i*-м элементе списка при помощи вызова rdsCWCreateCopy (п. 5.3.3) создается и запоминается объект-наблюдатель за входом разрешения (назовем его объектом “A*i*”), в него копируются параметры объекта, вызвавшего реакцию (т. е. выходного наблюдателя “Y*i*”);
- в этом же элементе списка таким же образом создается и запоминается объект-наблюдатель за входом данных (объект “X*i*”), причем если входов несколько, то дополнительные входы добавляются в объект при помощи команд RDS_CWPARAM_ADDVARNAME (п. 5.3.5) и RDS_CWPARAM_ADDVARNUM (п. 5.3.6).

В реакции на RDS_BFM_CONNCHGCTRLPROCESS (п. 5.5.4) ***для каждого элемента*** списка выходных наблюдателей (для всех *i*) выполняются следующие действия:

- изменения на входе разрешения (объект “A*i*”) при помощи rdsCWTranslateChanges (п. 5.3.7) передаются выходному наблюдателю (объекту “Y*i*”);

- если на входе разрешения идет переходный процесс (`rdsCWTranslateChanges` вернула `TRUE`), другим вызовом `rdsCWTranslateChanges` изменения с входов данных (объект “X*i*”) передаются выходному наблюдателю (объекту “Y*i*”);
- если на входе разрешения нет переходного процесса, и при этом значение этого входа не равно нулю (работа разрешена), изменения с входов данных (объект “X*i*”) передаются выходному наблюдателю (объекту “Y*i*”).

Таким образом, переходный процесс на входе разрешения будет наблюдаться на выходах всегда. Переходный процесс на входах данных будет наблюдаться на выходах либо если на входе разрешения тоже есть переходный процесс (еще не известно, будет ли работа блока разрешена), либо если на входе разрешения не ноль (данные с входов будут передаваться на выходы).

Список выходных наблюдателей нужен потому, что за выходами блока могут наблюдать несколько объектов с разными параметрами (погрешностями, разрешением определения расходящихся процессов), и информацию о процессе на входах нужно передавать для них независимо.

6.2. Мультиплексоры (много входов, один выход)

Мультиплексор имеет несколько одинаковых входов данных или входной массив, к отдельным элементам которого подключаются связи, и один выход данных. Кроме того, у него есть целочисленный вход управления, на который подается номер того входа данных (элемента массива), значение которого передается на выход (см. пример в п. 4.5).

Чтобы мультиплексор блокировал переходные процессы на неактивных входах, его модель должна вести список объектов-наблюдателей за выходами. Каждый элемент списка должен содержать:

- идентификатор выходного наблюдателя;
- идентификатор наблюдателя за входом управления для данного выходного;
- массив идентификаторов наблюдателей за отдельными входами данных (элементами массива) для данного выходного – по одному наблюдателю на вход.

Модель блока должна быть построена следующим образом.

В реакции на `RDS_BFM_CONNCHGCTRLSTART` (п. 5.5.1) список выходных наблюдателей очищается, все наблюдатели за входами, созданные самим блоком, уничтожаются при помощи вызовов `rdsDeleteObject` (см. приложение А.5.22.4).

В реакции на `RDS_BFM_CONNCHGCTRLASK` (п. 5.5.2) блок выполняет следующие действия:

- объект-наблюдатель выхода, вызвавший реакцию, добавляется в список выходных наблюдателей (назовем этот выходной объект-наблюдатель “Y*i*”, где *i* – номер добавленного элемента списка);
- в этом же элементе списка при помощи вызова `rdsCWCreateCopy` (п. 5.3.3) создается и запоминается объект-наблюдатель за входом управления (назовем его объектом “A*i*”), в него копируются параметры объекта, вызвавшего реакцию (выходного наблюдателя “Y*i*”);
- в этом же элементе списка создается массив идентификаторов входных наблюдателей (по одному объекту на каждый вход данных мультиплексора, назовем их объектами “X*i*[*j*]”, где *j* – номер входа), в них копируются параметры выходного наблюдателя.

В реакции на `RDS_BFM_CONNCHGCTRLPROCESS` (п. 5.5.4) ***для каждого элемента*** списка выходных наблюдателей (для всех *i*) выполняются следующие действия:

- изменения на входе управления (объект “A*i*”) при помощи `rdsCWTranslateChanges` (п. 5.3.7) передаются выходному наблюдателю (объекту “Y*i*”) ;

- если на входе управления идет переходный процесс (`rdsCWTranslateChanges` вернула `TRUE`), данные о переходном процессе ***в каждом*** объекте-наблюдателе входа данных из созданного массива (т. е. по очереди все объекты "X*i*[*j*]" для всех *j*) передаются выходному наблюдателю (объекту "Y*i*");
- если на входе управления нет переходного процесса, в выходной наблюдатель (объект "Y*i*") передается информация о переходном процессе только из объекта, индекс которого в массиве входных наблюдателей равен текущему значению входа управления (т. е. только из одного объекта "X*i*[*N*]", где *N* равно числу на входе управления).

Таким образом, переходный процесс на входе управления будет наблюдаться на выходе всегда. Переходный процесс на входе данных с номером *N* будет наблюдаться на выходе либо если на входе управления есть переходный процесс, либо если этот вход сейчас активен (на входе управления – значение *N*).

Здесь, как и в выключателе, список выходных наблюдателей нужен потому, что за выходами блока могут наблюдать несколько объектов с разными параметрами.

6.3. Демультиплексоры (один вход, много выходов)

Демультиплексор имеет один вход данных, с которого он передает данные на единственный выход (элемент выходного массива), номер которого подается на вход управления (см. пример в п. 4.6).

Чтобы переходный процесс на входе демультиплексора наблюдался только на активном в данный момент выходе, его модель должна вести независимые списки объектов-наблюдателей для каждого отдельного выхода. Проще всего организовать их в виде массива списков, где номер элемента массива будет определять номер выхода, для которого данный список (элемент массива) построен. Каждый элемент любого из этих списков должен содержать:

- идентификатор выходного наблюдателя;
- идентификатор наблюдателя за входом управления для данного выходного;
- идентификатор наблюдателя за входом данных для данного выходного.

Модель блока должна быть построена следующим образом.

В реакции на `RDS_BFM_CONNCHGCTRLSTART` (п. 5.5.1) массив списков наблюдателей полностью очищается, все наблюдатели за входами, созданные самим блоком, уничтожаются при помощи вызовов `rdsDeleteObject` (см. приложение А.5.22.4).

В реакции на `RDS_BFM_CONNCHGCTRLASK` (п. 5.5.2) блок выполняет следующие действия:

- по полям `VarName` и `VarNum` структуры `RDS_BFM_CONNCHGASKDATA` определяется номер выхода *n*, для которого запрашивается наблюдение за переходным процессом, и для дальнейшей работы из массива списков наблюдателей берется список-элемент с этим номером (если такого элемента нет, он создается);
- объект-наблюдатель, вызвавший реакцию, добавляется в рабочий, т. е. определенный на прошлом шаге, список (назовем этот объект "Y*n*[*i*]", где *n* – номер выхода, *i* – номер добавленного элемента списка для этого выхода);
- в этом же (*i*-м) элементе рабочего (*n*-го) списка при помощи вызова `rdsCWCreateCopy` (п. 5.3.3) создается и запоминается объект-наблюдатель за входом управления (назовем его "A*n*[*i*]"), в него копируются параметры объекта, вызвавшего реакцию (выходного наблюдателя "Y*n*[*i*]");
- в этом же элементе рабочего списка таким же образом создается объект-наблюдатель за входом данных (объект "X*n*[*i*]").

В реакции на RDS_BFM_CONNCHGCTRLPROCESS (п. 5.5.4) ***для каждого элемента списка наблюдателей в каждом элементе созданного массива*** (т. е., фактически, для каждого выходного наблюдателя, перебор по всем *n* и *i*) выполняются следующие действия:

- изменения на входе управления (объект "A*n*[*i*]", где *n* – индекс в массиве списков для отдельных выходов, *i* – индекс в массиве наблюдателей *n*-го списка) при помощи rdsCWTranslateChanges (п. 5.3.7) передаются выходному наблюдателю (объекту "Y*n*[*i*]");
- если на входе управления идет переходный процесс (rdsCWTranslateChanges вернула TRUE), данные о переходном процессе на входе данных (объект "X*n*[*i*]") передаются выходному наблюдателю (объекту "Y*n*[*i*]");
- если на входе управления нет переходного процесса, данные о переходном процессе на входе данных (объект "X*n*[*i*]") передаются выходному наблюдателю (объекту "Y*n*[*i*]") ***только в том случае***, если номер списка в массиве (*n*) равен текущему значению входа управления.

Таким образом, на выходе данных с номером *N* переходный процесс будет наблюдаться либо если есть переходный процесс на входе управления, либо если на вход управления подано значение *N*, и при этом есть переходный процесс на входе данных.

Для каждого выхода ведется список наблюдателей из-за того, что за этим выходом может наблюдать одновременно несколько объектов с разными параметрами.

6.4. Блок, вычисляющий динамические переменные по значениям входов

Блок, который вычисляет значения динамических переменных по данным, поступающим на его входы, должен предпринять специальные меры, чтобы переходные процессы в подграфах его входов обнаруживались на выходах других блоков, подписанных на эти же динамические переменные. Для этого достаточно в любой реакции блока ***до перехода в режим моделирования*** установить в поле Flags структуры данных блока RDS_BLOCKDATA (см. приложение А.2.3) битовый флаг RDS_DYNVARBYINPUT (см. п. 5.1). Проще всего сделать это в реакции блока на событие инициализации RDS_BFM_INIT (см. приложение А.2.4.7).

В функцию модели блока указатель на структуру данных блока передается во втором параметре. Таким образом, установка этого флага будет выглядеть примерно так:

```
extern "C" __declspec(dllexport)
     int RDSCALL имя_модели(
           int CallMode,
           RDS_PBLOCKDATA BlockData,
           LPVOID ExtParam)
{
  switch(CallMode)
    { case RDS_BFM_INIT:
        BlockData->Flags|=RDS_DYNVARBYINPUT;
      …
```

В автокомпилируемых блоках структура данных доступна через переменную rdsbcppBlockData, т. е. установка флага будет выглядеть так:

```
rdsbcppBlockData->Flags|=RDS_DYNVARBYINPUT;
```

На данный момент блок не может указывать, переходные процессы на каких входах на какие динамические переменные влияют. Считается, что любое изменение на любом входе влияет на все переменные сразу.

6.5. Блок, перезапускающийся при длительном расчете

Если блок разбивает какой-либо длительный внутренний расчет на несколько тактов, возвращая в промежутках управление в RDS для того, чтобы интерфейс пользователя не “зависал”, этот блок должен сообщить о таком перезапуске и продолжающемся расчете, чтобы отсутствие изменений на его выходах не было принято за окончание переходного процесса. Для этого блок должен сделать две вещи:

- заранее, ***до перехода в режим моделирования*** (например, при инициализации блока), установить в поле `Flags` структуры данных блока `RDS_BLOCKDATA` (см. приложение А.2.3) битовый флаг `RDS_MULTICALC` (см. п. 5.1);
- при каждом перезапуске в реакции на такт расчета (`RDS_BFM_MODEL`, см. приложение А.2.4.9) устанавливать в этом же поле битовый флаг `RDS_MULTICALCON`.

Установка флага `RDS_MULTICALC` нужна для сообщения RDS о том, что данный блок способен перезапускаться. Блоки без этого флага не будут проверяться на перезапуск для ускорения расчета.

Установка флага `RDS_MULTICALCON` сообщает о том, что блок перезапустился в данном такте и продолжает что-то вычислять, а, значит, переходные процессы в подграфах связей, присоединенных к его выходам, нельзя считать завершившимися. Флаг автоматически очищается перед каждым запуском модели, поэтому блоку не нужно его сбрасывать.

В функцию модели блока указатель на структуру данных блока передается во втором параметре. Для флага `RDS_MULTICALC` установка будет выглядеть примерно так:

```
extern "C" __declspec(dllexport)
      int RDSCALL имя_модели(
            int CallMode,
            RDS_PBLOCKDATA BlockData,
            LPVOID ExtParam)
{
      …
      BlockData->Flags|=RDS_MULTICALC;
      …
```

В автокомпилируемых блоках структура данных доступна через переменную `rdsbcppBlockData`, т. е. установка флага будет выглядеть так:

```
rdsbcppBlockData->Flags|=RDS_MULTICALC;
```