9. Библиотека блоков Simulink

9.1. Sources источники сигналов

9.1.1. Источник постоянного сигнала Constant

Назначение:
Задает постоянный по уровню сигнал.

Параметры:
1. **Constant value** Постоянная величина.
2. **Interpret vector parameters as 1D –** Интерпретировать вектор параметров как одномерный (при установленном флажке). Данный параметр встречается у большинства блоков библиотеки **Simulink**. В дальнейшем он рассматриваться не будет.

Значение константы может быть действительным или комплексным числом, вычисляемым выражением, вектором или матрицей.

Рис. 9.1.1 иллюстрирует применение этого источника и измерение его выходного сигнала с помощью цифрового индикатора **Display**.

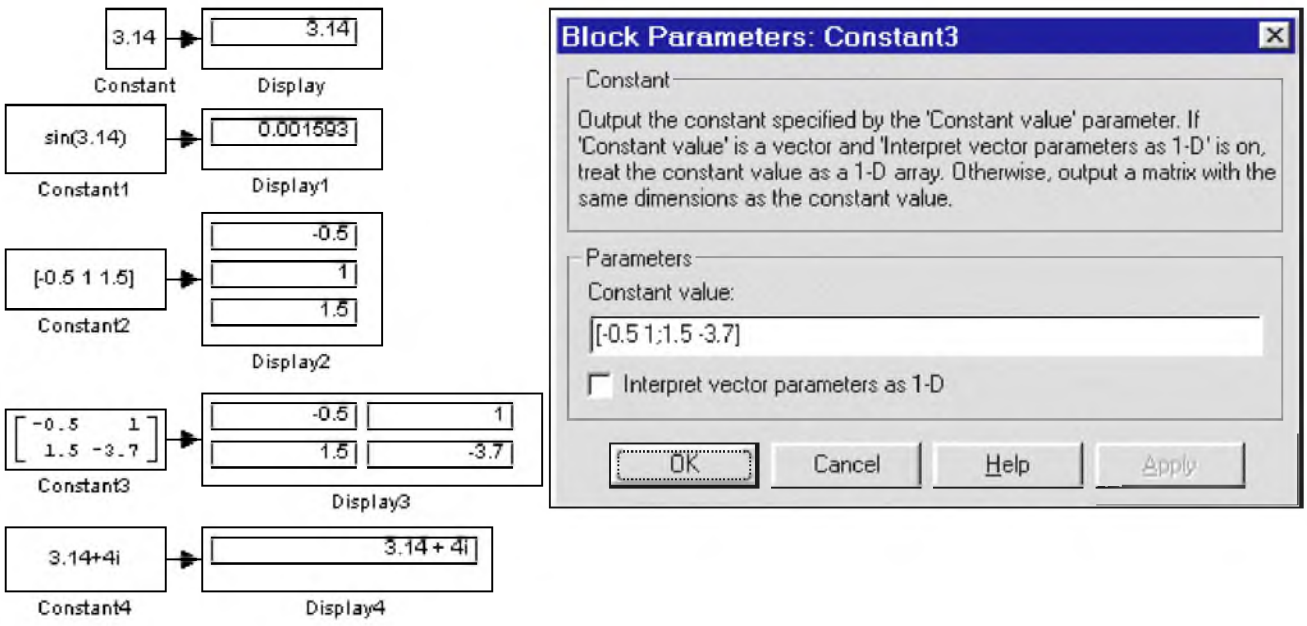


Рис. 9.1.1. Источник постоянного воздействия **Constant**

9.1.2. Источник синусоидального сигнала Sine Wave

Назначение:
Формирует синусоидальный сигнал с заданной частотой, амплитудой, фазой и смещением.

Для формирования выходного сигнала блоком могут использоваться два алгоритма. Вид алгоритма определяется параметром **Sine Type** (способ формирования сигнала):

- **Timebased** – По текущему времени.

- **Samplebased** – По величине шага модельного времени.

9.1.2.1. Формирование выходного сигнала по текущему значению времени для непрерывных систем

Выходной сигнал источника в этом режиме соответствует выражению:

y = Amplitude sin(frequency* time + phase) + bias.*

Параметры:

1. **Amplitude** Амплитуда.
2. **Bias** – Постоянная составляющая сигнала.
3. **Frequency (rads/sec)** Частота (рад/с).
4. **Phase (rads)** Начальная фаза (рад).
5. **Sample time** – Шаг модельного времени. Используется для согласования работы источника и других компонентов модели во времени. Параметр может принимать следующие значения:
   0 (по умолчанию) – Используется при моделировании непрерывных систем.
   > 0 (положительное значение) – Задается при моделировании дискретных систем. В этом случае шаг модельного времени можно интерпретировать как шаг квантования по времени выходного сигнала.
   1 – Шаг модельного времени устанавливается таким же, как и в предшествующем блоке, т.е. блоке, откуда приходит сигнал в данный блок.
   Этот параметр может задаваться для большинства блоков библиотеки **Simulink**. В дальнейшем он рассматриваться не будет.

При расчетах для очень больших значений времени точность расчета выходных значений сигнала падает вследствие значительной ошибки округления.

9.1.2.2. Формирование выходного сигнала по текущему значению времени для дискретных систем

Алгоритм определения значения выходного сигнала источника для каждого последующего шага расчета определяется выражением (в матричной форме):

$$\begin{bmatrix}\sin(t+\Delta t)\\ \cos(t+\Delta t)\end{bmatrix} = \begin{bmatrix}\cos(\Delta t) & \sin(\Delta t)\\ -\sin(\Delta t) & \cos(\Delta t)\end{bmatrix}\begin{bmatrix}\sin(t)\\ \cos(t)\end{bmatrix},$$

где Δ t – постоянная величина, равная значению **Sample time**.

В данном режиме ошибка округления для больших значений времени также уменьшает точность расчета.

9.1.2.3. Формирование выходного сигнала по величине модельного времени и количеству расчетных шагов на один период

Выходной сигнал источника в этом режиме соответствует выражению:

*y= Amplitude** sin[(*k + Number of offset samples) / Samples per* period] + *bias* ,

где ***k*** – номер текущего шага расчета.

Параметры:

1. **Amplitude** Амплитуда.
2. **Bias** – Постоянная составляющая сигнала.
3. **Samples per period** – Количество расчетных шагов на один период синусоидального сигнала:
   Samples per period = 2π / (*frequency* Sample time)*
4. **Number of offset samples** – Начальная фаза сигнала. Задается количеством шагов модельного времени:
   Number of offset samples = Phase Samples per period/* (2π).
5. **Sample time** – Шаг модельного времени.

В данном режиме ошибка округления не накапливается, поскольку **Simulink** начинает отсчет номера текущего шага с нуля для каждого периода.

На Рис. 9.1.2 показано применение блока с разными значениями шага модельного времени (**Sample time** = 0 для блока **Sine Wave 1**и **Sample time** = 0.1 для блока **Sine Wave 2**). Для отображения графиков выходных сигналов в модели использован виртуальный осциллограф (**Scope**).

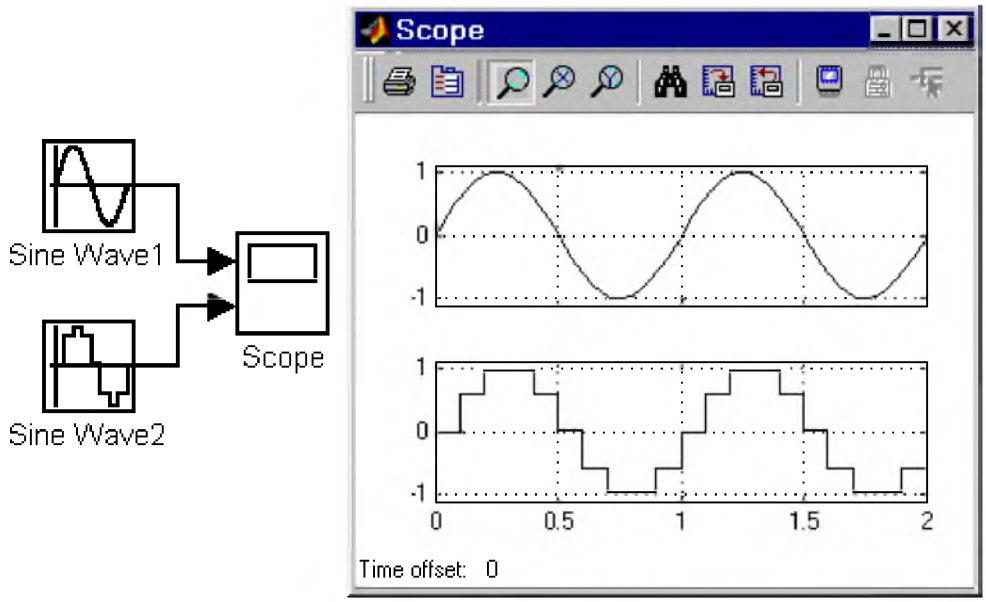


Рис. 9.1.2. Блок **Sine Wave**

9.1.3. Источник линейно изменяющегося воздействия Ramp

Назначение:
Формирует линейный сигнал вида **y** = ***Slope* time + Initial value.***

Параметры:

1. **Slope** — Скорость изменения выходного сигнала.

2. **Start time** — Время начала формирования сигнала.

3. **Initial value** — Начальный уровень сигнала на выходе блока.

На Рис. 9.1.3. показано использование данного блока.

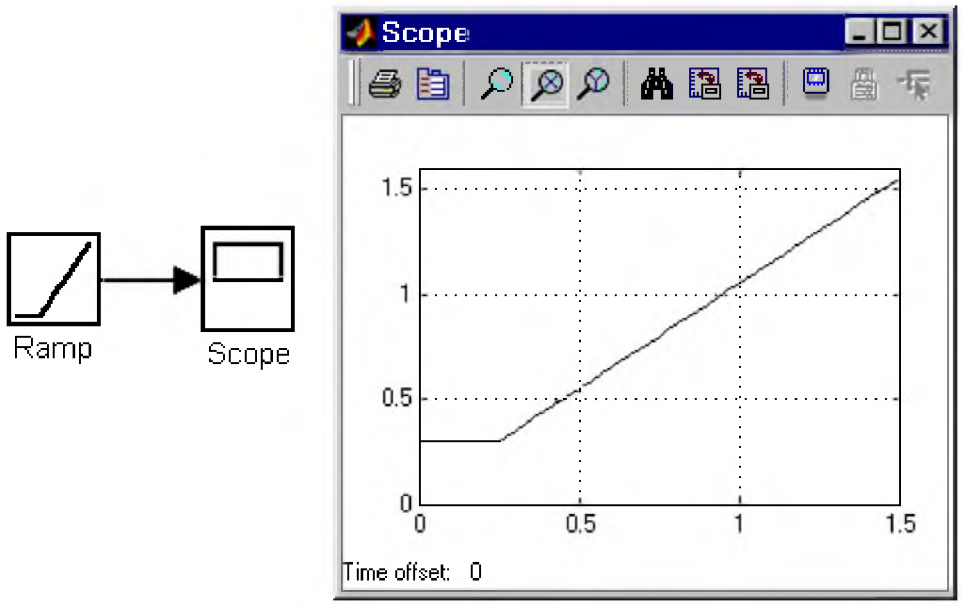


Рис. 9.1.3. Блок **Ramp**

9.1.4. Генератор ступенчатого сигнала Step

Назначение:
Формирует ступенчатый сигнал.

Параметры:

1. **Step time** Время наступления перепада сигнала (с).
2. **Initial value** Начальное значение сигнала.
3. **Final value** Конечное значение сигнала.

Перепад может быть как в большую сторону (конечное значение больше чем начальное), так и в меньшую (конечное значение меньше чем начальное). Значения начального и конечного уровней могут быть не только положительными, но и отрицательными (например, изменение сигнала с уровня –5 до уровня –3).

На Рис. 9.1.4. показано использование генератора ступенчатого сигнала.

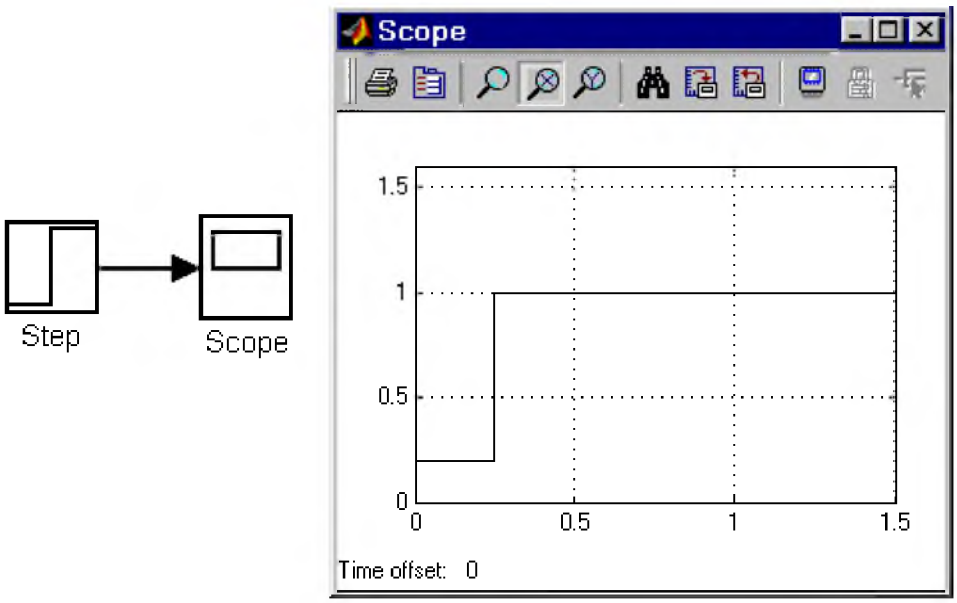


Рис. 9.1.4. Блок **Step**

9.1.5. Генератор сигналов Signal Generator

Назначение:
Формирует один из четырех видов периодических сигналов:

1. **sine** — Синусоидальный сигнал.
2. **square** — Прямоугольный сигнал.
3. **sawtooth** — Пилообразный сигнал.
4. **random** — Случайный сигнал.

Параметры:

1. **Wave form** – Вид сигнала.
2. **Amplitude** – Амплитуда сигнала.
3. **Frequency** Частота (рад/с).
4. **Units** – Единицы измерения частоты. Может принимать два значения:
   Hertz Гц.
   rad/sec – рад/с.

На Рис. 9.1.5. показано применение этого источника при моделировании прямоугольного сигнала.

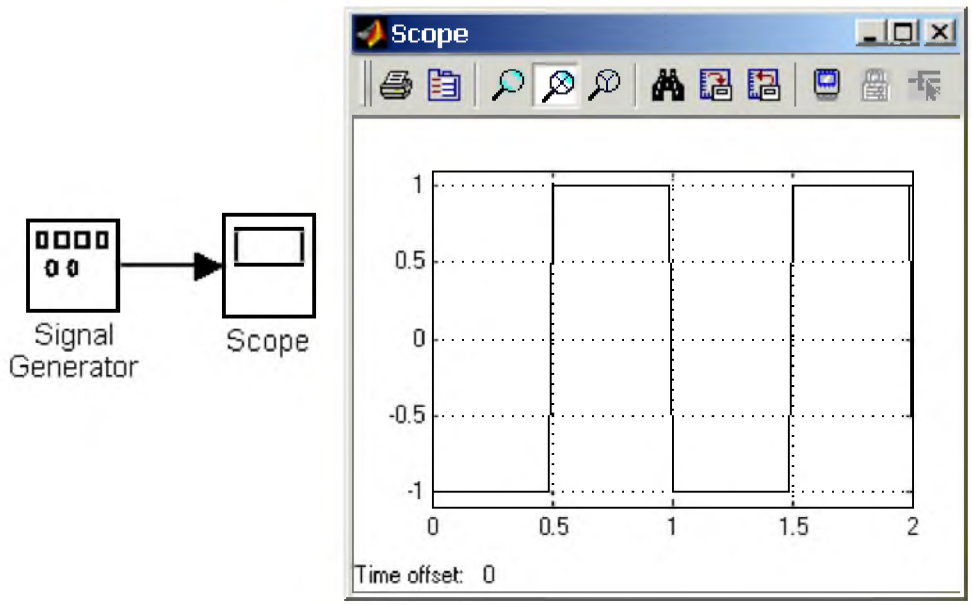


Рис. 9.1.5. Блок генератора сигналов

9.1.6. Источник случайного сигнала с равномерным распределением Uniform Random Number

Назначение:
Формирование случайного сигнала с равномерным распределением.

Параметры :

1. **Minimum** – Минимальный уровень сигнала.
2. **Maximum** – Максимальный уровень сигнала.
3. **Initial seed** – Начальное значение.

Пример использования блока и график его выходного сигнала представлен на Рис. 9.1.6.

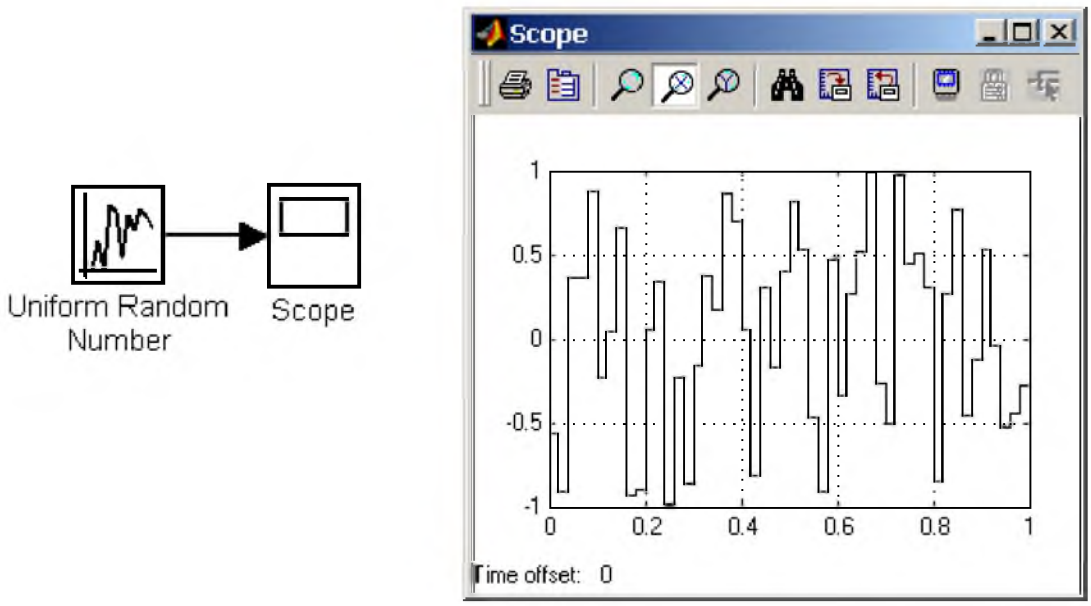


Рис. 9.1.6. Источник случайного сигнала с равномерным распределением

9.1.7. Источник случайного сигнала с нормальным распределением Random Number

Назначение:
Формирование случайного сигнала с нормальным распределением уровня сигнала.

Параметры:

1. **Mean** Среднее значение сигнала
2. **Variance** Дисперсия (среднеквадратическое отклонение).
3. **Initial seed** – Начальное значение.

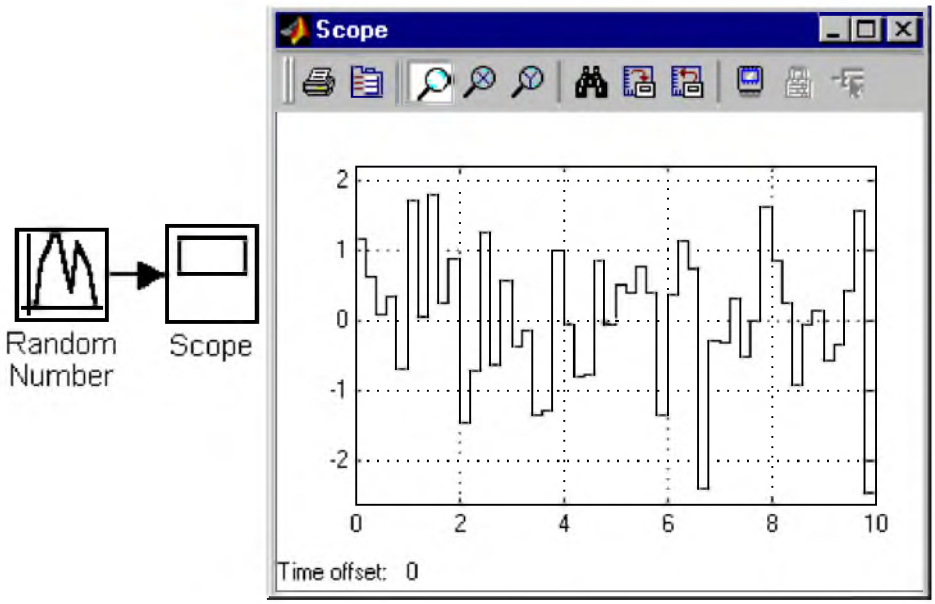


Рис. 9.1.7. Источник случайного сигнала с нормальным распределением

9.1.8. Источник импульсного сигнала Pulse Generator

Назначение:
Формирование прямоугольных импульсов.

Параметры:

1. **Pulse Type** – Способ формирования сигнала. Может принимать два значения:
   Timebased – По текущему времени.
   Samplebased – По величине модельного времени и количеству расчетных шагов.
2. **Amplitude** — Амплитуда.
3. **Period** — Период. Задается в секундах для **Timebased Pulse Type** или в шагах модельного времени для **Samplebased Pulse Type**.
4. **Pulse width** — Ширина импульсов. Задается в в **%** по отношению к периоду для **Timebased Pulse Type** или в шагах модельного времени для **Samplebased Pulse Type**.
5. **Phase delay** — Фазовая задержка. Задается в секундах для **Timebased Pulse Type** или в шагах модельного времени для **Samplebased Pulse Type.**
6. **Sample time** — Шаг модельного времени. Задается для **Samplebased Pulse Type**.

Пример использования **Pulse Generator** показан на рис. 9.1.8.

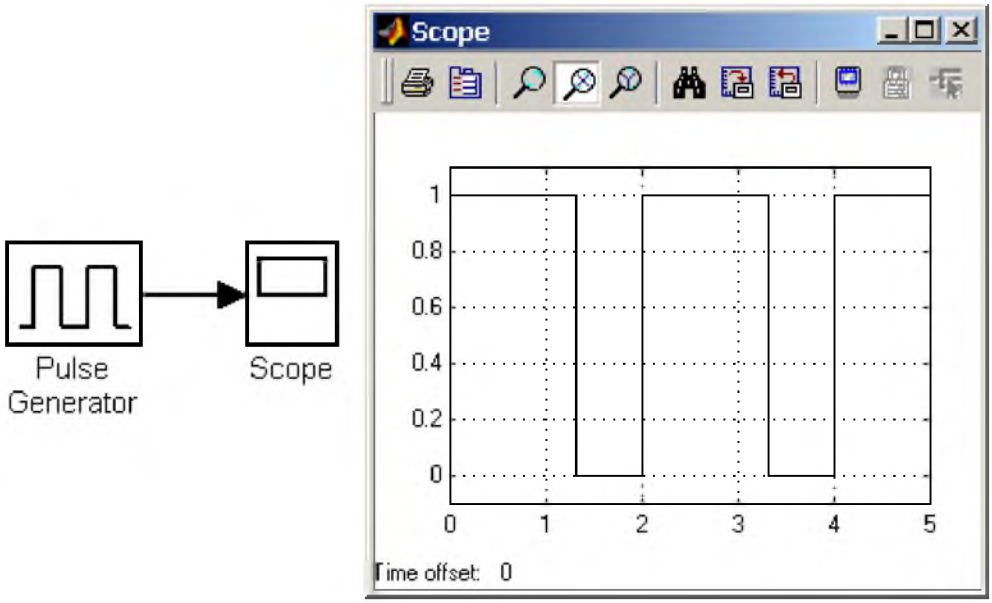


Рис. 9.1.8. Источник прямоугольных импульсов

9.1.9. Генератор линейноизменяющейся частоты Chirp Generator

Назначение:
Формирование синусоидальных колебаний, частота которых линейно изменяется.

Параметры:

1. **Initial frequence** — Начальная частота (Гц);
2. **Target time** — Время изменения частоты (с);
3. **Frequence at target time** — Конечное значение частоты (Гц).

Пример использования блока показан на Рис. 9.1.9.

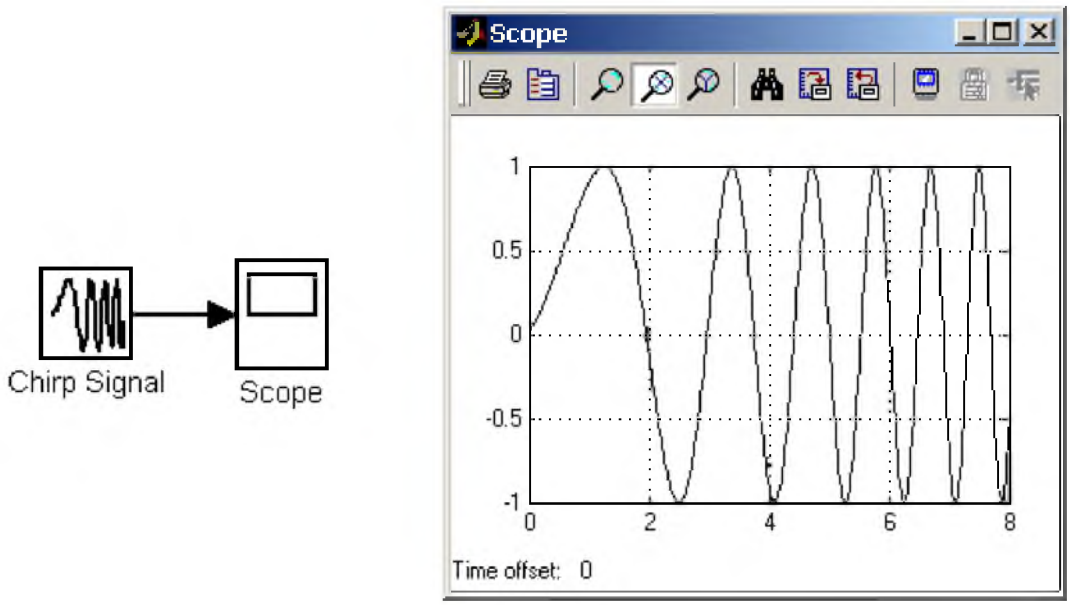


Рис. 9.1.9. Генератор линейноизменяющейся частоты

9.1.10. Генератор белого шума BandLimited White Noice

Назначение:
Создает сигнал заданной мощности, равномерно распределенной по частоте.

Параметры:

1. **Noice Power** – Мощность шума.
2. **Sample Time** – Модельное время.
3. **Seed** Число, необходимое для инициализации генератора случайных чисел.

Рис. 9.1.10 показывает работу этого генератора.

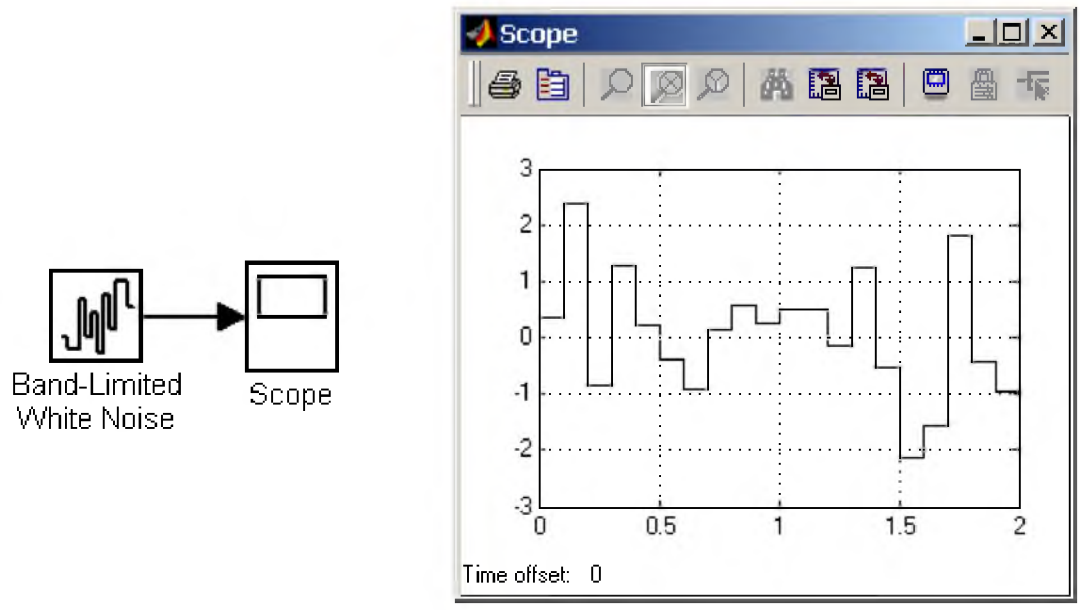


Рис. 9.1.10. Генератор белого шума

9.1.11. Источник временного сигнала Clock

Назначение:
Формирует сигнал, величина которого на каждом шаге расчета равна текущему времени моделирования.

Параметры

1. **Decimation** Шаг, с которым обновляются показания времени на изображении источника (в том случае, если установлен флажок параметра **Display time**). Параметр задается как количество шагов расчета. Например, если шаг расчета модели в окне диалога **Simulation parameters** установлен равным **0.01** с, а параметр **Decimation** блока **Clock** задан равным **1000**, то обновление показаний времени будет производиться каждые **10** с модельного времени.
2. **Display time** Отображение значения времени в блоке источника.

На рис. 9.1.11 показан пример работы данного источника.

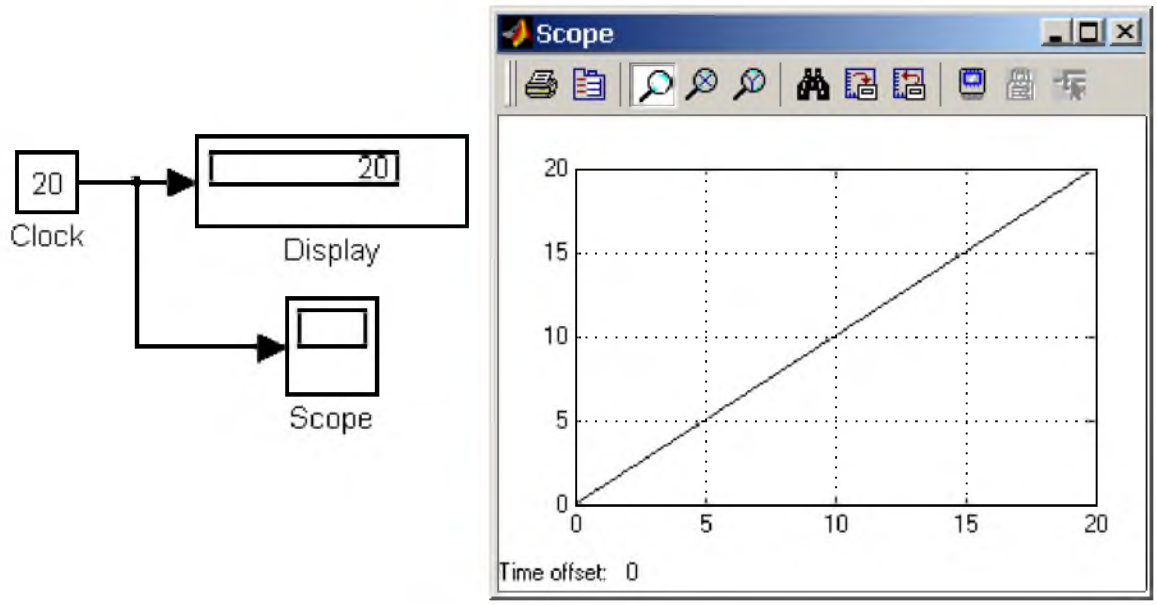


Рис. 9.1.11. Источник временного сигнала

9.1.12. Цифровой источник времени Digital Clock

Назначение:
Формирует дискретный временной сигнал.

Параметр:

Sample time – Шаг модельного времени (с).

На Рис. 9.1.12 показана работа источника **Digital Clock**.

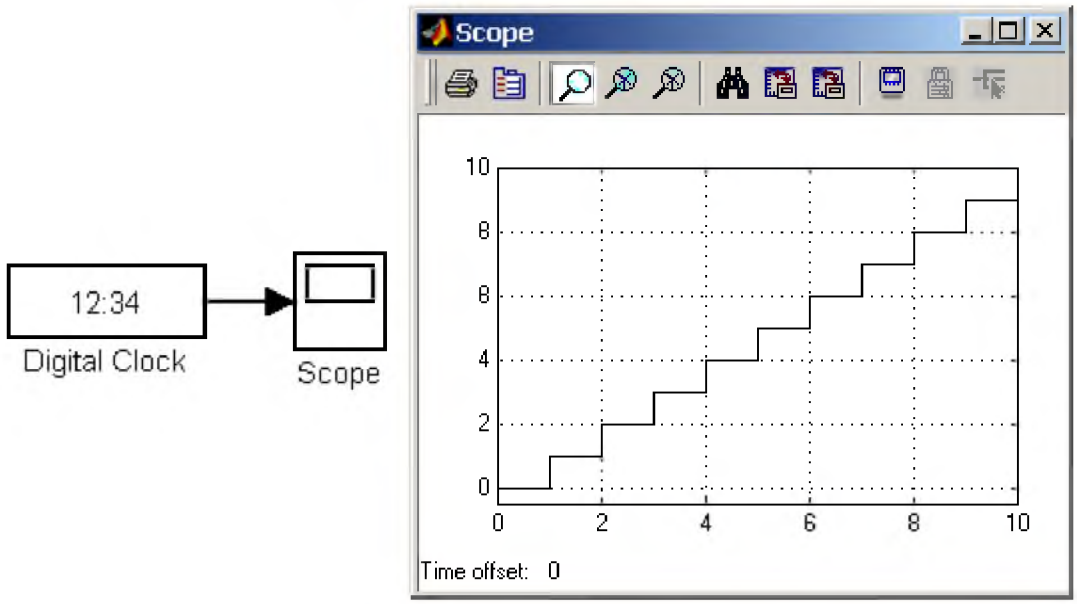


Рис. 9.1.12. Цифровой источник временного сигнала

9.1.13. Блок считывания данных из файла From File

Назначение:
Получение данных из внешнего файла.

Параметры:

1. **File Name** Имя файла с данными.
2. **Sample time** Шаг изменения выходного сигнала блока.

Данные в файле должны быть представлены в виде матрицы:

$$\begin{bmatrix} t_1 & t_2 & \dots t_{final} \\ u1_1 & u1_2 & \dots u1_{final} \\ \dots & & \\ un_1 & un_2 & \dots un_{final} \end{bmatrix}.$$

Матрица должна состоять, как минимум, из двух строк. Значения времени записаны в первой строке матрицы, а в остальных строках находятся значения сигналов, соответствующие данным моментам времени. Значения времени должны быть записаны в возрастающем порядке. Выходной сигнал блока содержит только значения сигналов, а значения времени в нем отсутствуют. Если шаг расчета текущей модели не совпадает с отсчетами времени в файле данных, то **Simulink** выполняет линейную интерполяцию данных.

Файл данных (**mat**файл), из которого считываются значения, не является текстовым. Структура файла подробно описана в справочной системе **MATLAB**. Пользователям **Simulink** удобнее всего создавать **mat**файл с помощью блока **To File** (библиотека **Sinks**). На рис. 9.1.13 показан пример использования данного блока. Из файла **data.mat** считываются значения синусоидального сигнала.

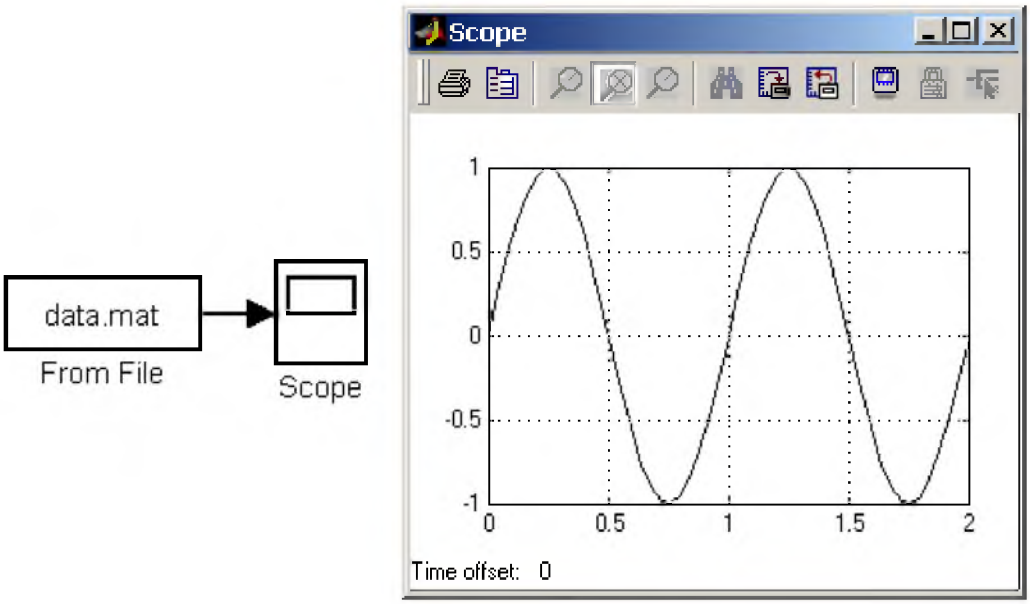


Рис. 9.1.13. Блок **From File**

9.1.14. Блок считывания данных из рабочего пространства From Workspace

Назначение:
Получение данных из рабочего пространства **MATLAB.**

Параметры:

1. **Data** – Имя переменной (матрицы или структуры) содержащей данные.
2. **Sample time** Шаг изменения выходного сигнала блока.
3. **Interpolate data** — Интерполяция данных для значений модельного времени не совпадающих со значениями в переменной **Data**.
4. **Form output after final data value by –** Вид выходного сигнала по окончании значений времени в переменной **Data**:
   Extrapolate – Линейная экстраполяция сигналов.
   SettingToZero – Нулевые значения сигналов.
   HoldingFinalValue – Выходные значения сигналов равны последним значениям.
   CyclicRepetition – Циклическое повторение значений сигналов. Данный вариант может использоваться, только если переменная **Data** имеет формат **Structure without time**.

На рис. 9.1.14 показан пример использования данного блока. Данные в переменную **simin** рабочей области **MATLAB** загружаются из файла с помощью блока **Read data**.

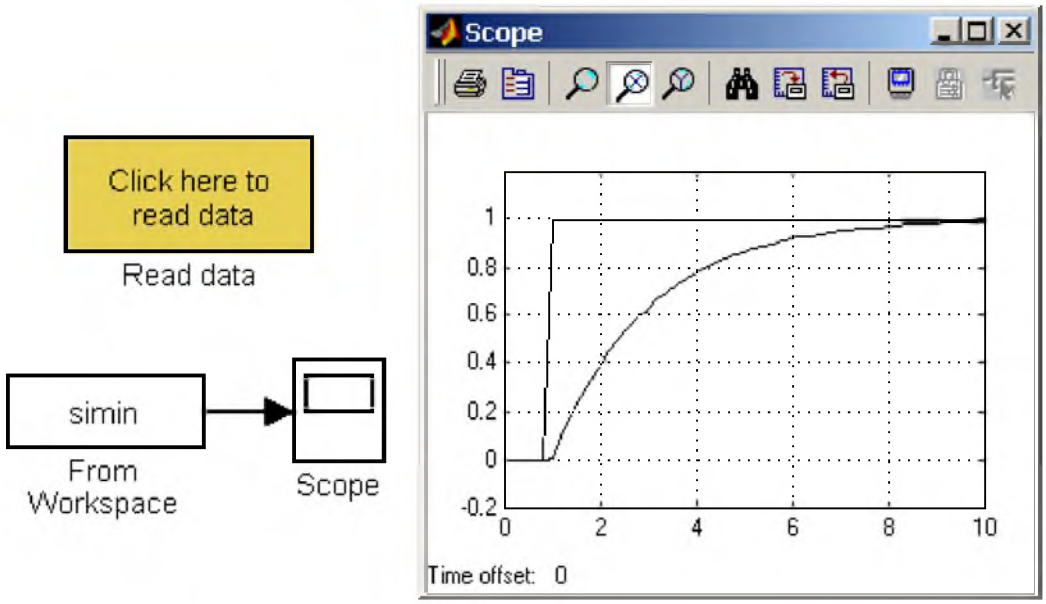


Рис. 9.1.14. Блок **From File**

9.1.15. Блок сигнала нулевого уровня Ground

Назначение:
Формирование сигнала нулевого уровня.

Параметры:

Нет.

Если какойлибо вход блока в модели не подсоединен, то при выполнении моделирования в главном окне **MATLAB** появляется предупреждающее сообщение. Для устранения этого на неподключенный вход блока можно подать сигнал с блока **Ground**.

На рис. 9.1.15 даны примеры использования блока. В первом случае сигнал с блока **Ground** поступает на один из входов сумматора, а во втором на один из входов блока умножения. Показания блоков **Display** подтверждают, что вырабатываемый блоком **Ground** сигнал имеет нулевое значение. Из рисунка также видно, что тип выходного сигнала блока устанавливается автоматически, в соответствии с типами сигналов, подаваемых на другие входы блоков (в данном случае – на входы блоков **Sum** и **Product**).

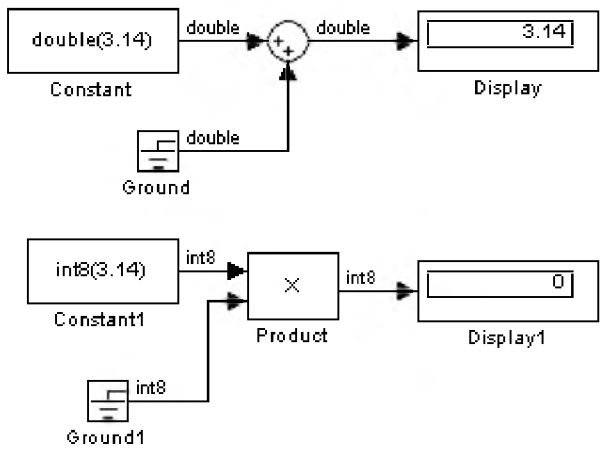


Рис. 9.1.15. Применение блока **Ground**

9.1.16. Блок периодического сигнала Repeating Sequence

Назначение:
Формирование периодического сигнала.

Параметры:

1. **Time values** – Вектор значений модельного времени.
2. **Output values –** Вектор значений сигнала для моментов времени заданных вектором **Time values**.

Блок выполняет линейную интерполяцию выходного сигнала для моментов времени не совпадающих со значениями заданными вектором **Time values**. На рис. 9.1.16 показан пример использования блока для формирования пилообразного сигнала. Значения модельного времени заданы вектором **[0 3]**, а значения выходного сигнала вектором **[0 2]**.

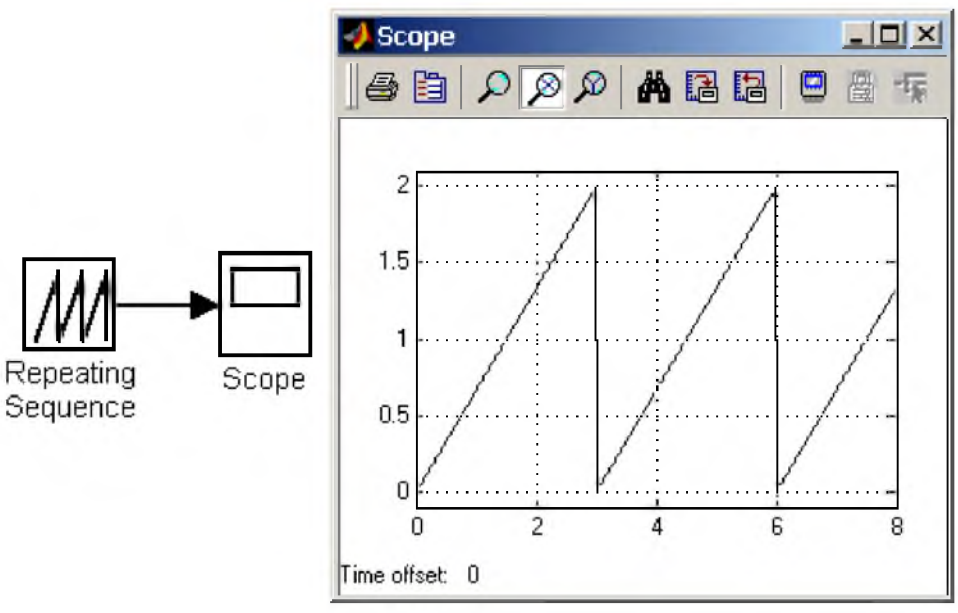


Рис. 9.1.16. Использование блока **Repeating Sequence**

9.1.17. Блок входного порта Inport

Назначение:

Создает входной порт для подсистемы или модели верхнего уровня иерархии.

Параметры:

- **Port number** – Номер порта.
- **Port dimensions** – Размерность входного сигнала. Если этот параметр равен **–1**, то размерность входного сигнала будет определяться автоматически.

- **Sample time** –Шаг модельного времени.
- **Data type** – Тип данных входного сигнала: **auto, double, single, int8, uint8, int16, uint16, int32, uint32** или **boolean**.

- **Signal type** – Тип входного сигнала:

1. **auto** – Автоматическое определение типа.
2. **real** –Действительный сигнал.
3. **complex** Комплексный сигнал.

- **Interpolate data** (флажок) – Интерполировать входной сигнал. В случае, если временные отсчеты входного сигнала считываемого из рабочей области **MATLAB** не совпадают с модельным временем, то блок будет выполнять интерполяцию входного сигнала. При использовании блока **Inport** в подсистеме данный параметр не доступен.

9.1.17.1. Использование блока Inport в подсистемах

Блоки **Inport** подсистемы являются ее входами. Сигнал, подаваемый на входной порт подсистемы через блок **Inport,** передается внутрь подсистемы. Название входного порта будет показано на изображении подсистемы как метка порта.

При создании подсистем и добавлении блока **Inport** в подсистему **Simulink** использует следующие правила:

1. При создании подсистемы с помощью команды **Edit/Create subsystem** входные порты создаются и нумеруются автоматически начиная с **1**.
2. Если в подсистему добавляется новый блок **Inport**, то ему присваивается следующий по порядку номер.
3. Если какой либо блок **Inport** удаляется, то остальные порты переименовываются таким образом, чтобы последовательность номеров портов была непрерывной.
4. Если в последовательности номеров портов имеется разрыв, то при выполнении моделирования **Simulink** выдаст сообщение об ошибке и остановит расчет. В этом случае необходимо вручную переименовать порты таким образом, чтобы последовательность номеров портов не нарушалась.

На рис. 9.1.17 показана модель, использующая подсистему и схема этой подсистемы.

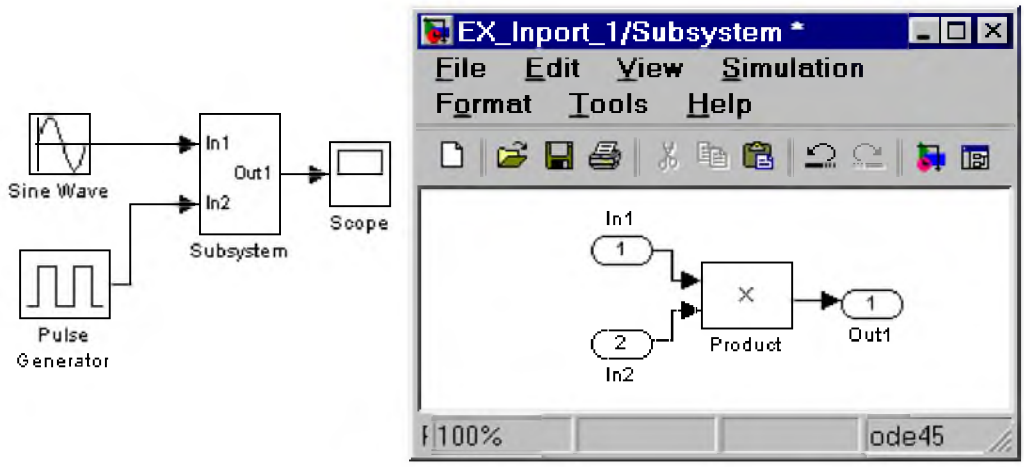


Рис. 9.1.17. Использование блока **Inport** в подсистеме

9.1.17.2. Использование блока Inport в модели верхнего уровня

Входной порт в системе верхнего уровня используется для передачи сигнала из рабочей области **MATLAB** в модель.

Для передачи сигнала из рабочего пространство **MATLAB** требуется не только установить в модели входной порт, но и выполнить установку параметров ввода на вкладке **Workspace I/O** окна диалога **Simulation parameters…** (должен быть установлен флажок для параметра **Input** и задано имя переменной, которая содержит входные данные). Тип вводимых данных: **Array** (массив), **Structure** (структура) или **Structure with time** (структура с полем "*время*") задается на этой же вкладке.

На рис. 9.1.18 показана модель, считывающая входной сигнал из рабочего пространства **MATLAB**. Подсистема **Load Data** обеспечивает ввод данных из файла в рабочую область **MATLAB**.

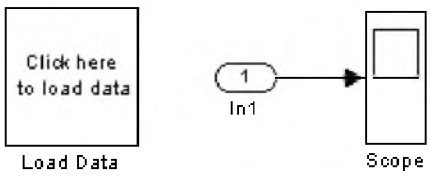


Рис. 9.1.18. Модель, считывающая входной сигнал из рабочего пространства **MATLAB** с помощью блока **Input.**

9. Библиотека блоков Simulink

9.2. Sinks приемники сигналов

9.2.1. Осциллограф Scope

Назначение:

Строит графики исследуемых сигналов в функции времени. Позволяет наблюдать за изменениями сигналов в процессе моделирования.

Изображение блока и окно для просмотра графиков показаны на рис. 9.2.1.

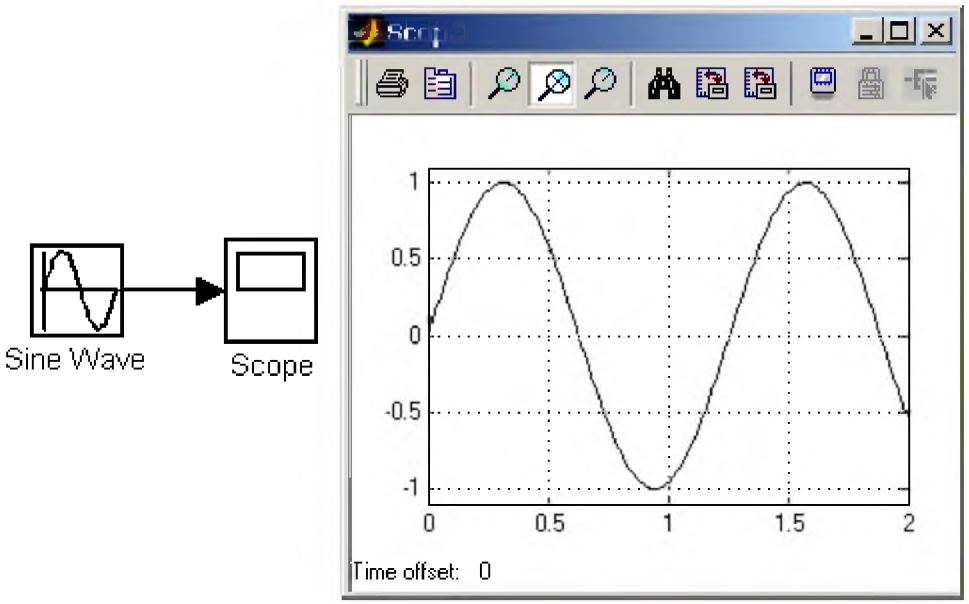


Рис. 9.2.1. Осциллограф **Scope**

Для того, чтобы открыть окно просмотра сигналов необходимо выполнить двойной щелчок левой клавишей “мыши” на изображении блока. Это можно сделать на любом этапе расчета (как до начала расчета, так и после него, а также во время расчета). В том случае, если на вход блока поступает векторный сигнал, то кривая для каждого элемента вектора строится отдельным цветом.

Настройка окна осциллографа выполняется с помощью панелей инструментов (рис.9.2.2).

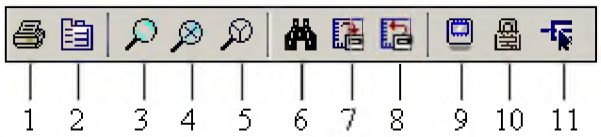


Рис. 9.2.2. Панель инструментов блока **Scope**

Панель инструментов содержит 11 кнопок:

1. **Print** – печать содержимого окна осциллографа.
2. **Parameters** – доступ к окну настройки параметров.
3. **Zoom** – увеличение масштаба по обеим осям.
4. **Zoom Xaxis** – увеличение масштаба по горизонтальной оси.
5. **Zoom Yaxis** – увеличение масштаба по вертикальной оси.
6. **Autoscale** – автоматическая установка масштабов по обеим осям.
7. **Save current axes settings** – сохранение текущих настроек окна.
8. **Restore saved axes settings** – установка ранее сохраненных настроек окна.
9. **Floating scope** – перевод осциллографа в “*свободный*” режим.
10. **Lock/Unlock axes selection** – закрепить/разорвать связь между текущей координатной системой окна и отображаемым сигналом. Инструмент доступен, если включен режим **Floating scope**.

11. **Signal selection** – выбор сигналов для отображения. Инструмент доступен, если включен режим **Floating scope**.

Изменение масштабов отображаемых графиков можно выполнять несколькими способами:

1. Нажать соответствующую кнопку (, или) и щелкнуть один раз левой клавишей “мыши” в нужном месте графика. Произойдет 2,5 кратное увеличение масштаба.

2. Нажать соответствующую кнопку (, или) и, нажав левую клавишу “мыши”, с помощью динамической рамки или отрезка указать область графика для увеличенного изображения. Рис. 9.2.3 поясняет этот процесс.

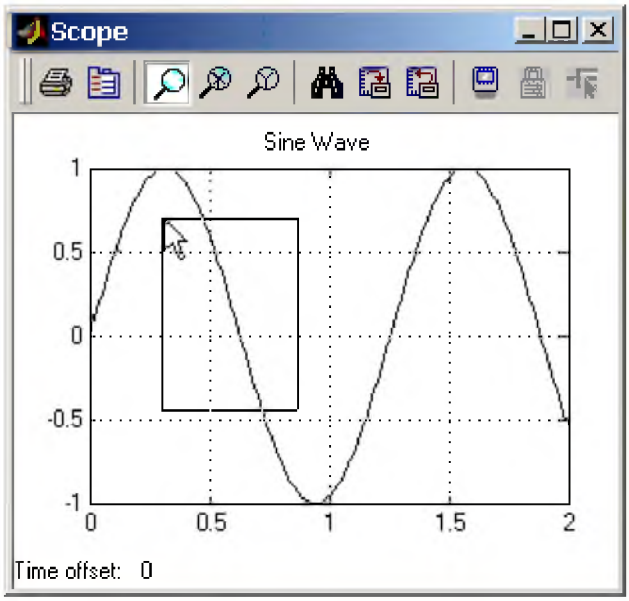


Рис. 9.2.3. Увеличение масштаба графика.

3. Щелкнуть правой клавишей “мыши” в окне графиков и, выбрать команду **Axes properties…** в контекстном меню. Откроется окно свойств графика, в котором с помощью параметров **Ymin** и **Ymax** можно указать предельные значения вертикальной оси. В этом же окне можно указать заголовок графика (**Title**), заменив выражение **%<SignalLabel>** в строке ввода. Окно свойств показано на рис. 9.2.4.

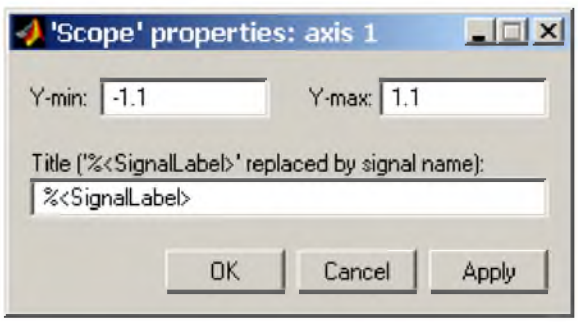


Рис. 9.2.4. Окно свойств графика.

Параметры:

Параметры блока устанавливаются в окне **'Scope' parameters**, которое открывается с помощью инструмента (**Parameters**) панели инструментов. Окно параметров имеет две вкладки:

General – общие параметры.

Data history – параметры сохранения сигналов в рабочей области **MATLAB**.

Вкладка общих параметров показана на рис. рис. 9.2.5.

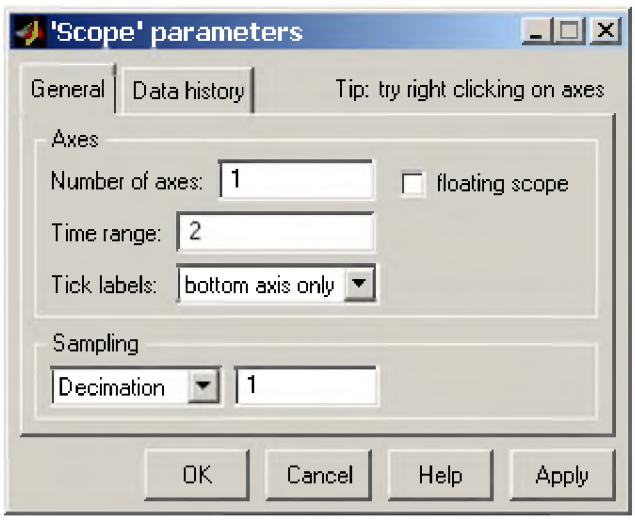


Рис. 9.2.5. Вкладка общих параметров **General**.

На вкладке **General** задаются следующие параметры:

1. **Number of axes** — число входов (систем координат) осциллографа. При изменении этого параметра на изображении блока появляются дополнительные входные порты.

2. **Time range** — величина временного интервала для которого отображаются графики. Если время расчета модели превышает заданное параметром **Time range**, то вывод графика производится порциями, при этом интервал отображения каждой порции графика равен заданному значению **Time range**.

3. **Tick labels** — вывод/скрытие осей и меток осей. Может принимать три значения (выбираются из списка):

- **all** – подписи для всех осей,
- **none** – отсутствие всех осей и подписей к ним,
- **bottom axis only** – подписи горизонтальной оси только для нижнего графика.

4. **Sampling** — установка параметров вывода графиков в окне. Задает режим вывода расчетных точек на экран. При выборе **Decimation** кратность вывода устанавливается числом, задающим шаг выводимых расчетных точек. На рис. 9.2.6 и 9.2.7. показаны графики синусоидальных сигналов рассчитанных с фиксированным шагом **0.1** с. На рис. 9.2.6 в окне блока **Scope** выводится каждая расчетная точка (параметр **Decimation** равен **1**). На рис. 9.2.7 показан вывод каждого второго значения (параметр **Decimation** равен **2**). Маркерами на графиках отмечены расчетные точки.

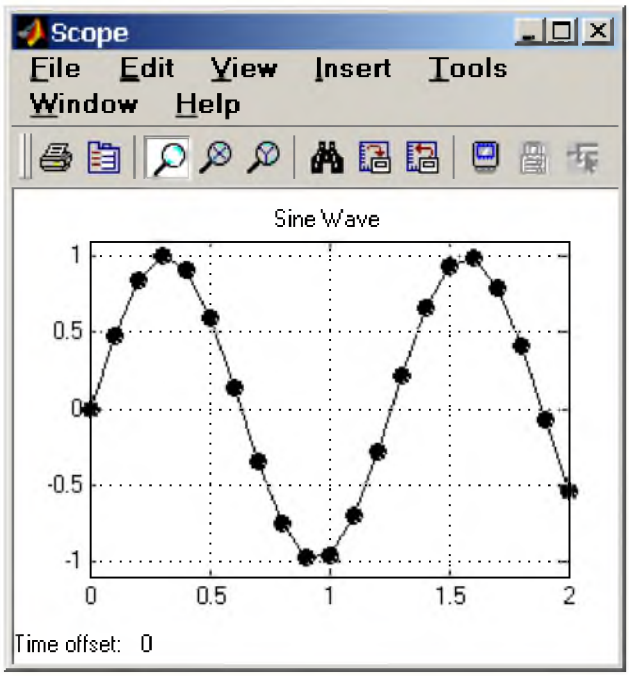


Рис. 9.2.6. Отображение синусоидального сигнала (**Decimation** = **1**).

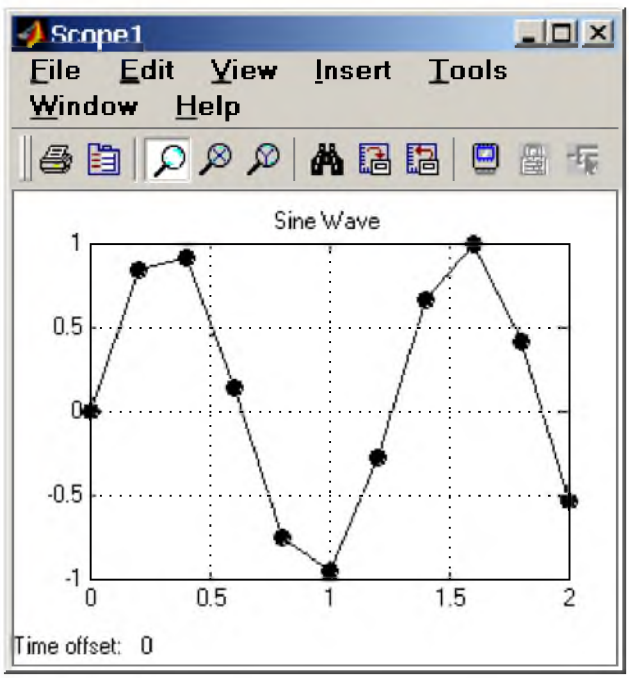


Рис. 9.2.7. Отображение синусоидального сигнала (**Decimation** = **2**).

В том случае, если режим вывода расчетных точек задается как **Sample time**, то его числовое значение определяет интервал квантования при отображении сигнала. На рис. 9.2.8 показан график синусоидального сигнала, для случая, когда значение параметра **Sample time** равно **0.1** .

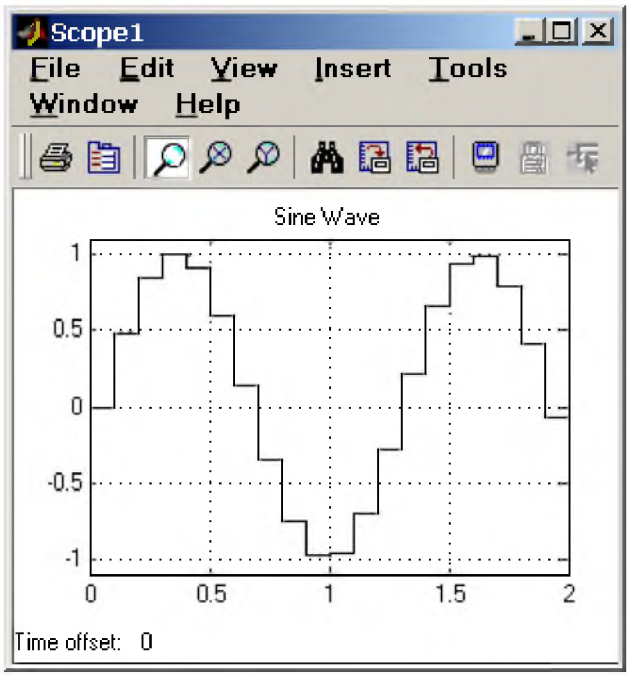


Рис. 9.2.8. Отображение синусоидального сигнала (**Sample time** = **0.1**).

5. **floating scope** – перевод осциллографа в "*свободный*" режим (при установленном флажке).

На вкладке **Data history** (рис. 9.2.9)задаются следующие параметры:

1. **Limit data points to last** – максимальное количество отображаемых расчетных точек графика. При превышении этого числа начальная часть графика обрезается. В том случае, если флажок параметра **Limit data points to last** не установлен, то **Simulink** автоматически увеличит значение этого параметра для отображения всех расчетных точек.

2. **Save data to workspace** – сохранение значений сигналов в рабочей области **MATLAB**.

3. **Variable name –** имя переменной для сохранения сигналов в рабочей области **MATLAB**.

4. **Format –** формат данных при сохранении в рабочей области **MATLAB**. Может принимать значения:

- **Array** – массив,
- **Structure** – структура,
- **Structure with time** – структура с дополнительным полем "время".

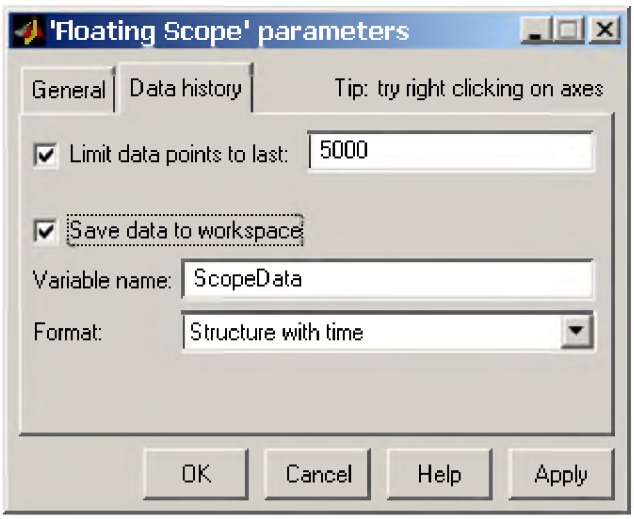


Рис. 9.2.9. Вкладка **Data history**.

9.2.2. Осциллограф Floating Scope

Осциллограф **Floating Scope**, по сути, есть обычный осциллограф **Scope**, переведенный в *"свободный"* режим. Пример модели с осциллографом **Floating Scope** показан на рис. 9.2.10.

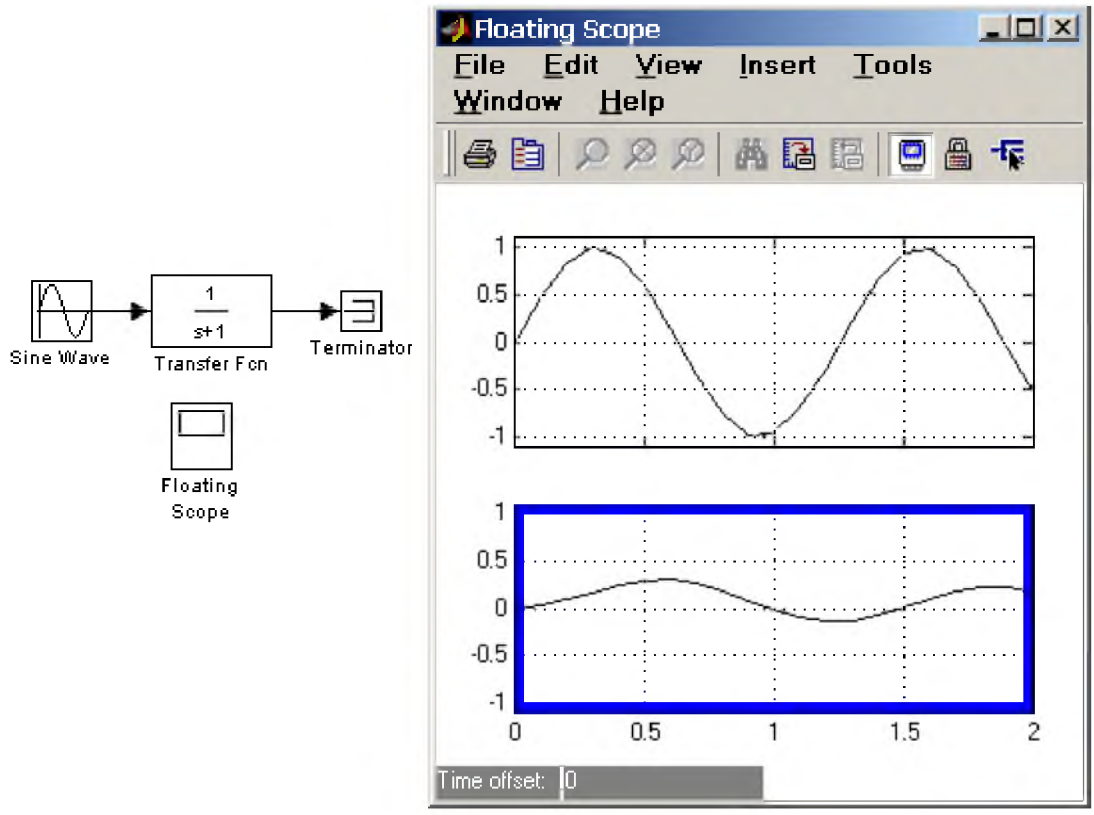


Рис. 9.2.10. Пример модели с осциллографом **Floating Scope**.

В этом режиме блок осциллографа не имеет входов, а выбор отображаемого сигнала осуществляется с помощью инструмента (**Signal selection**) панели инструментов. Для выбора сигналов необходимо выполнить следующие действия:

1. Выделить систему координат, в которой будет отображаться график. Это достигается с помощью одиночного щелчка левой клавишей “мыши” внутри нужной системы. Выбранная система координат будет подсвечена по периметру синим цветом.

2. С помощью инструмента  открыть окно диалога **Signal Selector** (рис. 9.2.11).

3. Отметить флажком имена блоков, сигналы с выхода которых требуется исследовать.

После выполнения расчета в окне блока **Floating Scope** будут отображены выбранные сигналы.

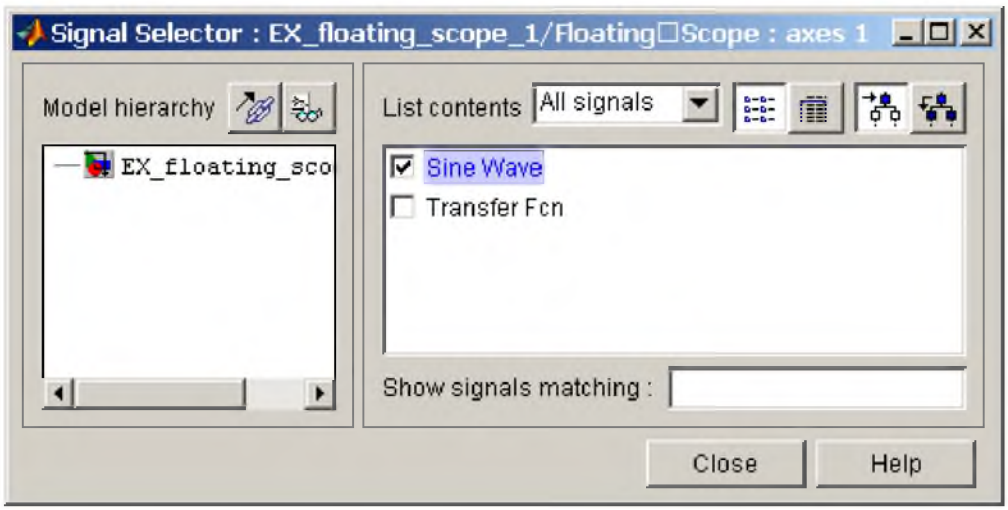


Рис. 9.2.11. Окно диалога **Signal Selector**

9.2.3. Графопостроитель XY Graph

Назначение:

Строит график одного сигнала в функции другого (график вида **Y(X)**).

Параметры:

xmin – Минимальное значение сигнала по оси **X**.

xmax – Максимальное значение сигнала по оси **X**

ymin – Минимальное значение сигнала по оси **Y**.

ymax – Максимальное значение сигнала по оси **Y**

Sample time – шаг модельного времени.

Блок имеет два входа. Верхний вход предназначен для подачи сигнала, который является аргументом (**X**), нижний – для подачи значений функции (**Y**).

На рис.9.2.12, в качестве примера использования графопостроителя, показано построение фазовой траектории колебательного звена.

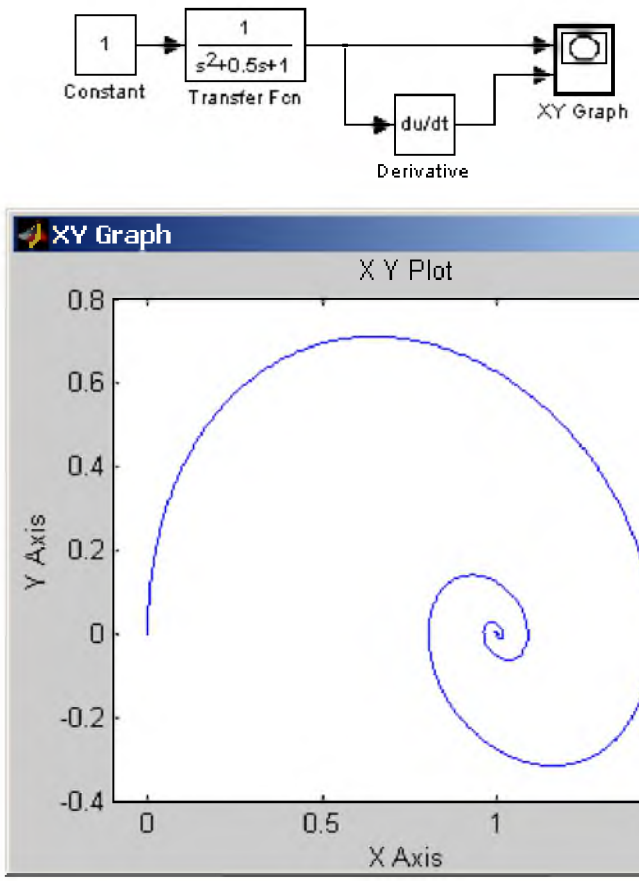


Рис. 9.2.12. Пример использования графопостроителя **XY Graph**.

Графопостроитель можно использовать и для построения временных зависимостей. Для этого на первый вход следует подать временной сигнал с выхода блока **Clock**. Пример такого использования графопостроителя показан на рис. 9.2.13.

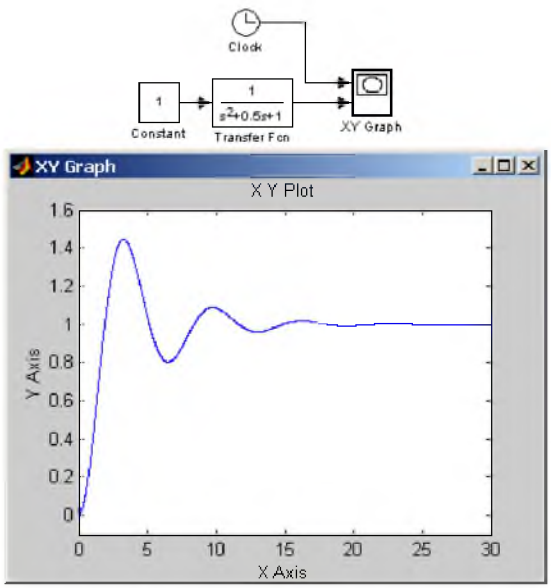


Рис. 9.2.13. Пример использования блока **XY Graph** для отображения временных зависимостей.

9.2.4. Цифровой дисплей Display

Назначение:

Отображает значение сигнала в виде числа.

Параметры:

- **Format** – формат отображения данных. Параметр **Format** может принимать следующие значения:
    1. **short** – 5 значащих десятичных цифр.
    2. **long** – 15 значащих десятичных цифр.
    3. **short_e** –5 значащих десятичных цифр и 3 символа степени десяти.
    4. **long_e** – 15 значащих десятичных цифр и 3 символа степени десяти.
    5. **bank** – "*денежный*" формат. Формат с фиксированной точкой и двумя десятичными цифрами в дробной части числа.
- **Decimation** – кратность отображения входного сигнала. При **Decimation = 1** отображается каждое значение входного сигнала, при **Decimation = 2** отображается каждое второе значение, при **Decimation = 3** – каждое третье значение и т.д.
- **Sample time** – шаг модельного времени. Определяет дискретность отображения данных.
- **Floating display** (флажок)– перевод блока в "*свободный*" режим. В данном режиме входной порт блока отсутствует, а выбор сигнала для отображения выполняется щелчком левой клавиши "мыши" на соответствующей лини связи. В этом режиме для параметра расчета **Signal storage reuse** должно быть установлено значение **off** (вкладка **Advanced** в окне диалога **Simulation parameters…**).

На рис. 2.9.14 показано применение блока **Display** с использованием различных вариантов параметра **Format**.

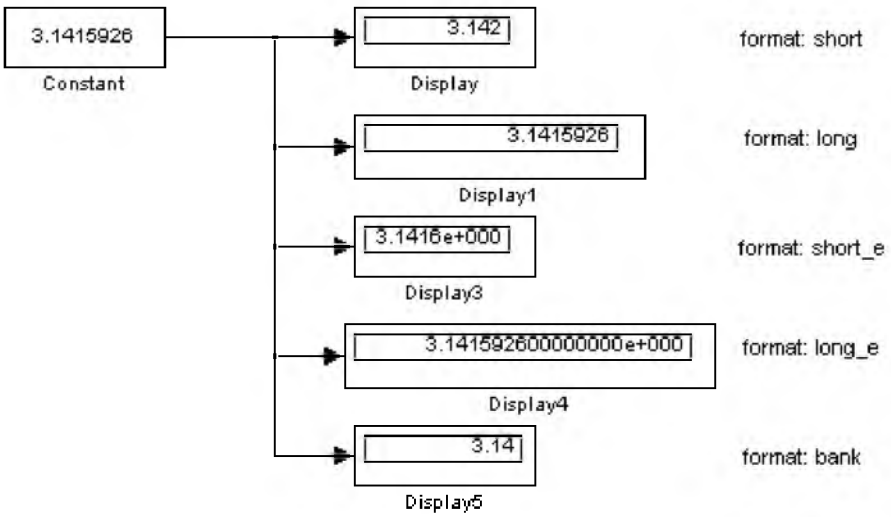


Рис. 9.2.14. Применение блока **Display** с использованием различных вариантов параметра **Format.**

Блок **Display** может использоваться для отображения не только скалярных сигналов, но также векторных, матричных и комплексных. Рис. 2.9.15 иллюстрирует это. Если все отображаемые значения не могут поместиться в окне блока, в правом нижнем углу блока появляется символ ⇲, указывающий на необходимость увеличить размеры блока (см. блок **Display4** на рис. 2.9.15).

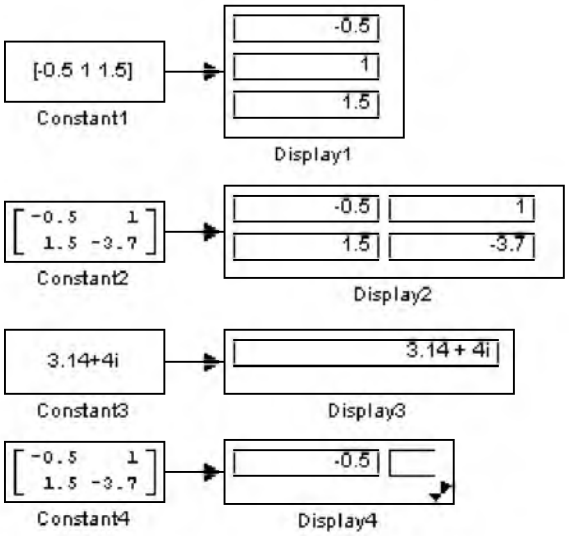


Рис. 9.2.15 Применение блока **Display** для отображения векторных, матричных и комплексных сигналов.

9.2.5. Блок остановки моделирования Stop Simulation

Назначение:

Обеспечивает завершение расчета, если входной сигнал блока становится не равным нулю.

Параметры:

Нет.

При подаче на вход блока ненулевого сигнала **Simulink** выполняет текущий шаг расчета, а затем останавливает моделирование. Если на вход блока подан векторный сигнал, то для остановки расчета достаточно, чтобы один элемент вектора стал ненулевым. На рис. 2.9.16 показан пример использования данного блока. В примере остановка расчета происходит, если выходной сигнал блока **Transfer Function** становится большим или равным **0.99**.

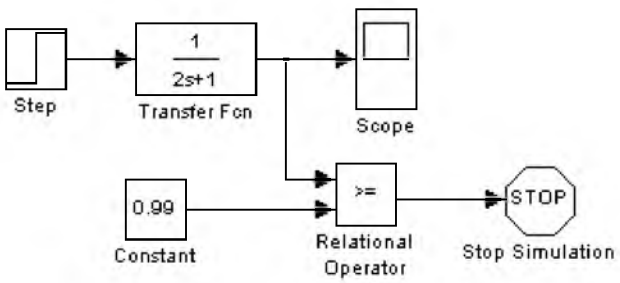


Рис. 9.2.16. Применение блока **Stop Simulation**

9.2.6. Блок сохранения данных в файле To File

Назначение:

Блок записывает данные, поступающие на его вход, в файл.

Параметры:

- **Filename –** имя файла для записи. По умолчанию файл имеет имя **untitled.mat**. Если не указан полный путь файла, то файл сохраняется в текущей рабочей папке.
- **Variable name –** имя переменной, содержащей записываемые данные.
- **Decimation** – кратность записи в файл входного сигнала. При **Decimation = 1** записывается каждое значение входного сигнала, при **Decimation = 2** записывается каждое второе значение, при **Decimation = 3** – каждое третье значение и т.д.
- **Sample time** – шаг модельного времени. Определяет дискретность записи данных.

Данные в файле сохраняются в виде матрицы:

$$\begin{bmatrix} t_1 & t_2 & \dots t_{final} \\ u1_1 & u1_2 & \dots u1_{final} \\ \dots & & \\ un_1 & un_2 & \dots un_{final} \end{bmatrix}.$$

Значения времени записываются в первой строке матрицы, а в остальных строках будут находиться значения сигналов, соответствующих данным моментам времени.

Файл данных (**mat**файл), в который записываются данные, не является текстовым. Структура файла подробно описана в справочной системе **MATLAB**. Пользователям **Simulink** удобнее всего считывать данные из **mat**файла с помощью блока **From File** (библиотека **Sources**).

На рис. 9.2.17 показан пример использования данного блока. Результаты расчета сохраняются в файле **result.mat**.

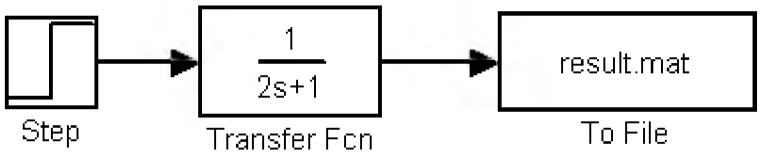


Рис. 9.2.17. Применение блока **To File**

9.2.7. Блок сохранения данных в рабочей области To Workspace

Назначение:

Блок записывает данные, поступающие на его вход, в рабочую область **MATLAB**.

Параметры:

- **Variable name –** имя переменной, содержащей записываемые данные.
- **Limit data points to last** – максимальное количество сохраняемых расчетных точек по времени (отсчет ведется от момента завершения моделирования). В том случае, если значение параметра **Limit data points to last** задано как **inf**, то в рабочей области будут сохранены все данные.
- **Decimation** – кратность записи данных в рабочую область.

- **Sample time** – шаг модельного времени. Определяет дискретность записи данных.
- **Save format** – формат сохранения данных. Может принимать значения:
    1. **Matrix** – матрица. Данные сохраняются как массив, в котором число строк определяется числом расчетных точек по времени, а число столбцов – размерностью вектора подаваемого на вход блока. Если на вход подается скалярный сигнал, то матрица будет содержать лишь один столбец.
    2. **Structure** – структура. Данные сохраняются в виде структуры, имеющей три поля: **time** – время, **signals** – сохраняемые значения сигналов, **blockName** – имя модели и блока **To Workspace**. Поле **time** для данного формата остается не заполненным.
    3. **Structure with Time** – структура с дополнительным полем (время). Для данного формата, в отличие от предыдущего, поле **time** заполняется значениями времени.

На рис. 9.2.18 показан пример использования данного блока. Результаты расчета сохраняются в переменной **simout**.

Для считывания данных сохраненных в рабочей области **MATLAB** можно использовать блок **From Workspace** (библиотека **Sources**).

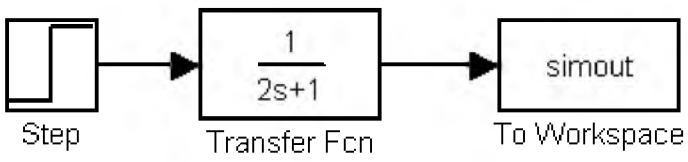


Рис. 9.2.18. Применение блока **To Workspace**

9.2.8. Концевой приемник Terminator

Назначение:

Блок используется для подачи сигнала с неиспользуемого выхода другого блока.

Параметры:

Нет.

В том случае, если выход блока оказывается не подключенным ко входу другого блока, **Simulink** выдает предупреждающее сообщение в командном окне **MATLAB**. Для исключения этого необходимо использовать блок **Terminator**. На рис. 9.2.19 показан пример использования концевого приемника. Извлекаемый, с помощью блока **Demux,** из матрицы второй элемент не никак не используется, поэтому он подается на вход блока **Terminator**.

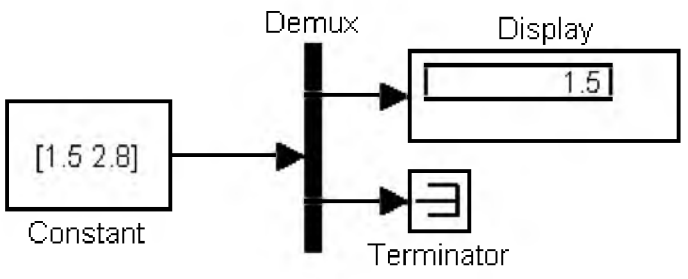


Рис. 9.2.19. Применение блока **Terminator**

9.2.9. Блок выходного порта Outport

Назначение:

Создает выходной порт для подсистемы или для модели верхнего уровня иерархии.

Параметры:

- **Port number** – номер порта.
- **Output when disabled** – вид сигнала на выходе подсистемы, в случае если подсистема выключена. Используется для управляемых подсистем. Может принимать значения (выбираются из списка):
    1. **held** – выходной сигнал подсистемы равен последнему рассчитанному значению.
    2. **reset** – – выходной сигнал подсистемы равен значению задаваемому параметром **Initial output**.
- **Initial output** значение сигнала на выходе подсистемы до начала ее работы и в случае, если подсистема выключена. Используется для управляемых подсистем.

9.2.9.1. Использование блока Outport в подсистемах

Блоки **Outport** подсистемы являются ее выходами. Сигнал, подаваемый в блок **Outport** внутри подсистемы, передается в модель (или подсистему) верхнего уровня. Название выходного порта будет показано на изображении подсистемы как метка порта.

При создании подсистем и добавлении блока **Outport** в подсистему **Simulink** использует следующие правила:

1. При создании подсистемы с помощью команды **Edit/Create subsystem** выходные порты создаются и нумеруются автоматически начиная с **1**.
2. Если в подсистему добавляется новый блок **Outport**, то ему присваивается следующий по порядку номер.
3. Если какой либо блок **Outport** удаляется, то остальные порты переименовываются таким образом, чтобы последовательность номеров портов была непрерывной.
4. Если в последовательности номеров портов имеется разрыв, то при выполнении моделирования **Simulink** выдаст сообщение об ошибке и остановит расчет. В этом случае необходимо вручную переименовать порты таким образом, чтобы последовательность номеров портов не нарушалась.

На рис. 9.2.19 показана модель, использующая подсистему и схема этой подсистемы.

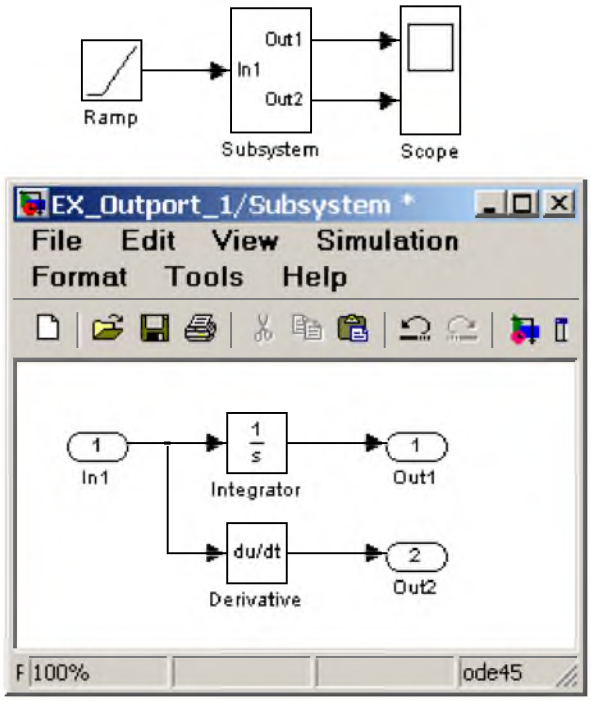


Рис. 9.2.19. Использование блока **Outport** в подсистеме

В том случае, если подсистема является управляемой, то для ее выходных портов можно задать вид выходного сигнала для тех временных интервалов, когда подсистема заблокирована. На рис. 9.2.20 показана модель, использующая управляемую подсистему (схема подсистемы такая же, как и в предыдущем примере). Для первого выходного порта подсистемы параметр **Output when disabled** задан как **held**, а для второго – как **reset**, причем величина начального значения задана равной нулю. Графики сигналов показывают, что когда подсистема заблокирована, сигнал первого выходного порта остается неизменным, а сигнал второго становится равным заданному начальному значению (нулю).

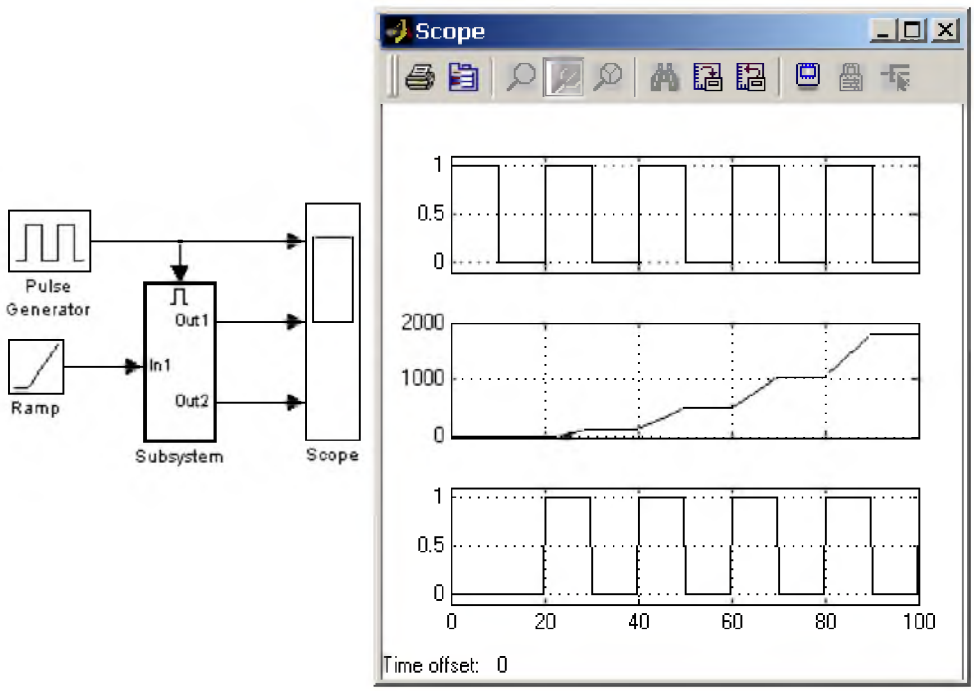


Рис. 9.2.20. Управляемая подсистема с различными настройками выходных портов.

9.2.9.2. Использование блока Outport в модели верхнего уровня

Выходной порт в системе верхнего уровня используется в двух случаях:

1. Для передачи сигнала в рабочее пространство **MATLAB**.
2. Для обеспечения связи функций анализа с выходами модели.

Для передачи сигнала в рабочее пространство **MATLAB** требуется не только установить в модели выходные порты, но и выполнить установку параметров вывода на вкладке **Workspace I/O** окна диалога **Simulation parameters…** (должен быть установлен флажок для параметра **Output** и задано имя переменной для сохранения данных). Тип сохраняемых данных **Array** массив, **Structure** (структура) или **Structure with time** (структура с полем "*время*") задается на этой же вкладке.

На рис. 9.2.21 показана модель, передающая сигналы в рабочее пространство **MATLAB**.

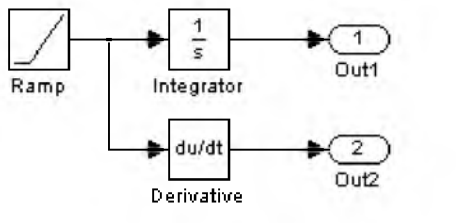


Рис. 9.2.21. Модель, передающая сигналы в рабочее пространство **MATLAB** с помощью блоков **Outport**.

Блок **Outport** может использоваться также для связи модели с функциями анализа, например: **linmod** или **trim**.

9. Библиотека блоков Simulink

9.3. Continuous – аналоговые блоки

9.3.1. Блок вычисления производной Derivative

Назначение:

Выполняет численное дифференцирование входного сигнала.

Параметры:

Нет.

Для вычисления производной используется приближенная формула Эйлера:

$$\frac{du}{dt} = \frac{\Delta u}{\Delta t},$$

где Δu – величина изменения входного сигнала за время Δt,

Δt – текущее значение шага модельного времени.

Значение входного сигнала блока до начала расчета считается равным нулю. Начальное значение выходного сигнала также полагается равным нулю.

Точность вычисления производной существенно зависит от величины установленного шага расчета. Выбор меньшего шага расчета улучшает точность вычисления производной.

На рис. 9.3.1 показан пример использования дифференцирующего блока для вычисления производной прямоугольного сигнала. В рассматриваемом примере, для повышения наглядности, шаг расчета выбран достаточно большим.

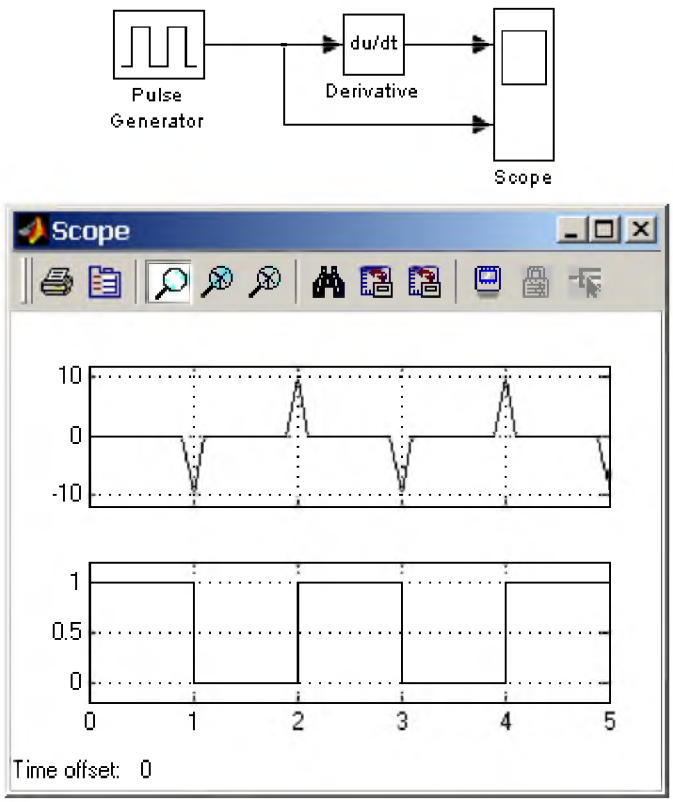


Рис.9.3.1. Использование блока **Derivative** для дифференцирования сигнала.

Данный блок используется для дифференцирования аналоговых сигналов. При дифференцировании дискретного сигнала с помощью блока **Derivative** его выходной сигнал будет представлять собой последовательность импульсов соответствующих моментам времени скачкообразного изменения дискретного сигнала.

9.3.2. Интегрирующий блок Integrator

Назначение:

Выполняет интегрирование входного сигнала.

Параметры:

- **External reset** – Внешний сброс. Тип внешнего управляющего сигнала, обеспечивающего сброс интегратора к начальному состоянию. Выбирается из списка:

1. **none** – нет (сброс не выполняется),
2. **rising** нарастающий сигнал (передний фронт сигнала),
3. **falling** спадающий сигнал (задний фронт сигнала),
4. **either** – нарастающий либо спадающий сигнал,
5. **level** – не нулевой сигнал (сброс выполняется если сигнал на управляющем входе становится не равным нулю);

В том случае, если выбран какойлибо (но не **none**), тип управляющего сигнала, то на изображении блока появляется дополнительный управляющий вход. Рядом с дополнительным входом будет показано условное обозначение управляющего сигнала.

- **Initial condition source** — Источник начального значения выходного сигнала. Выбирается из списка:
    1. **internal** – внутренний
    2. **external** – внешний. В этом случае на изображении блока появляется дополнительный вход, обозначенный x_0, на который необходимо подать сигнал задающий начальное значение выходного сигнала интегратора.
- **Initial condition** — Начальное условие. Установка начального значения выходного сигнала интегратора. Параметр доступен, если выбран внутренний источник начального значения выходного сигнала.
- **Limit output** (флажок) — Использование ограничения выходного сигнала.
- **Upper saturation limit** — Верхний уровень ограничения выходного сигнала. Может быть задан как числом, так и символьной последовательностью **inf**, то есть $+\infty$.
- **Lower saturation limit** — Нижний уровень ограничения выходного сигнала. Может быть задан как числом, так и символьной последовательностью **inf**, то есть $-\infty$.
- **Show saturation port** — управляет отображением порта, выводящего сигнал, свидетельствующий о выходе интегратора на ограничение. Выходной сигнал данного порта может принимать следующие значения:

    1. **Ноль**, если интегратор не находится на ограничении.
    2. **+1**, если выходной сигнал интегратора достиг верхнего ограничивающего предела.
    3. **1**, если выходной сигнал интегратора достиг нижнего ограничивающего предела.

- **Show state port** (флажок) — Отобразить/скрыть порт состояния блока. Данный порт используется в том случае, если выходной сигнал интегратора требуется подать в качестве сигнала обратной связи этого же интегратора. На пример, при установке начальных условий через внешний порт или при сбросе интегратора через порт сброса. Выходной сигнал с этого порта может использоваться также для организации взаимодействия с управляемой подсистемой.
- **Absolute tolerance** — Абсолютная погрешность.

На рис. 9.3.2 показан пример работы интегратора при подаче на его вход ступенчатого сигнала. Начальное условие принято равным нулю.

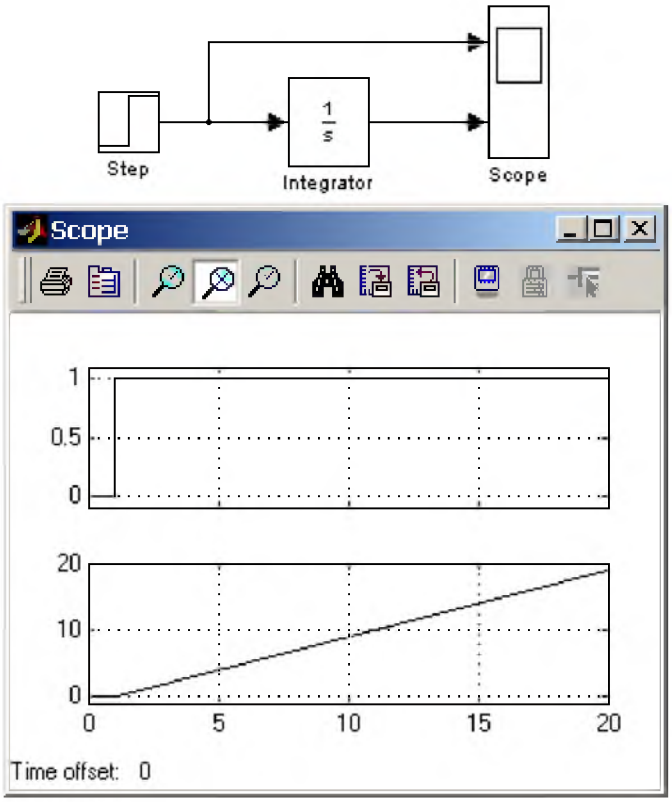


Рис. 9.3.2. Интегрирование ступенчатого сигнала.

Пример на рис. 9.3.3 отличается от предыдущего подачей начального значения через внешний порт. Начальное значение выходного сигнала в данном примере задано равным **–10**.

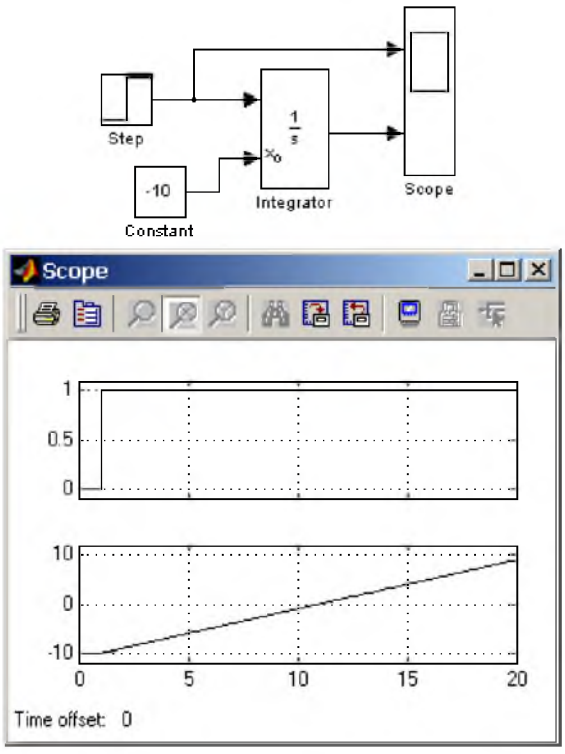


Рис. 9.3.3. Интегрирование ступенчатого сигнала с установкой начального значения выходного сигнала.

Пример на рис. 9.3.4 демонстрирует использование входного порта для сброса выходного сигнала и порта состояния интегратора с целью организации обратной связи. Схема работает следующим образом: входной постоянный сигнал преобразуется интегратором в линейноизменяющийся, по достижении выходным сигналом значения равного **1** блок **Relational Operator** вырабатывает логический сигнал, по переднему фронту которого происходит сброс выходного сигнала интегратора до начального значения равного нулю. В результате на выходе интегратора формируется пилообразный сигнал, изменяющийся от **0** до **+1**.

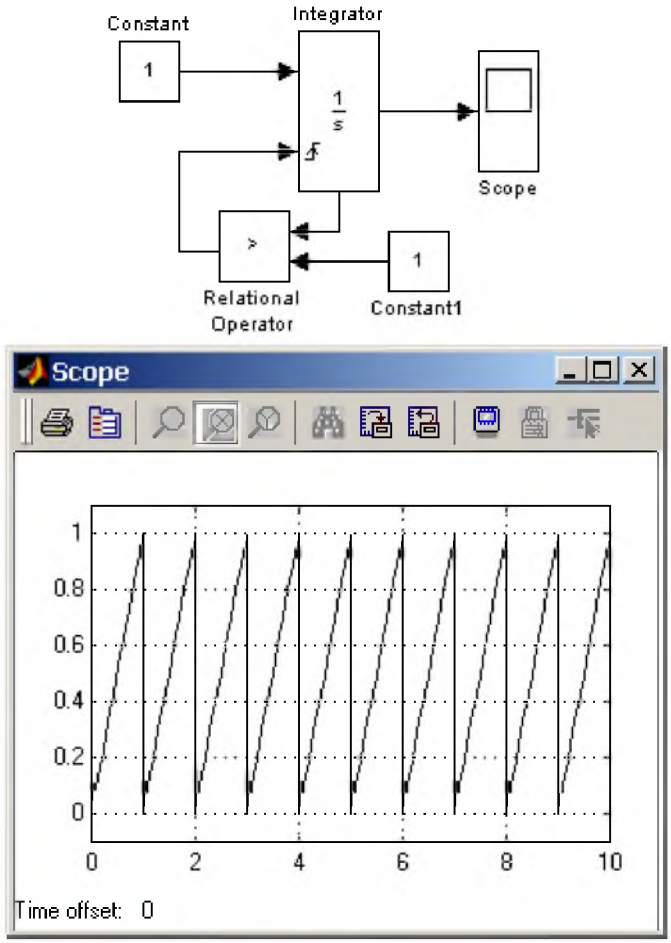


Рис. 9.3.4. Генератор пилообразного сигнала на основе интегратора.

Следующая схема (рис. 9.3.5) использует установку начального значения интегратора с помощью его выходного сигнала. В первый момент времени начальное значение выходного сигнала интегратора с помощью блока **IC** (**Initial Condition**) устанавливается равным нулю. По достижении выходным сигналом значения равного **1** блок **Relational Operator** подает сигнал сброса выходного сигнала интегратора на начальный уровень, при этом сигналом, задающим начальный уровень, оказывается инвертированный выходной сигнал интегратора (т.е. **1**). Далее цикл работы схемы повторяется. В отличие от предыдущей схемы выходным сигналом генератора является двуполярный сигнал.

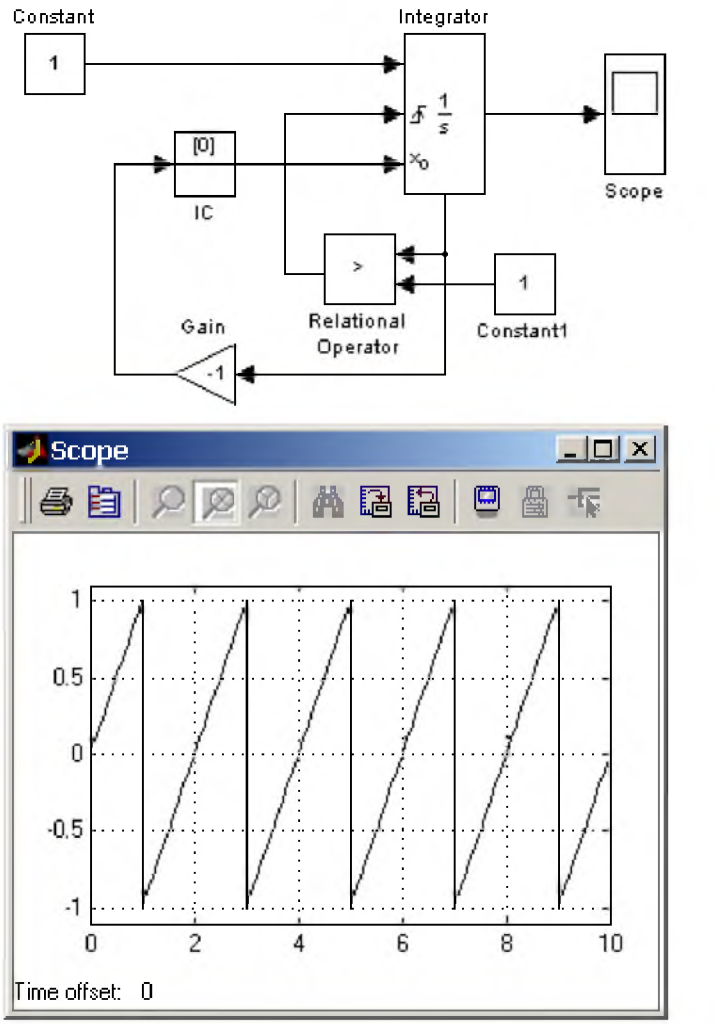


Рис. 9.3.5. Генератор двуполярного пилообразного сигнала

на основе интегратора.

9.3.3. Блок Memory

Назначение:

Выполняет задержку входного сигнала на один временной такт.

Параметры:

- **Initial condition** – начальное значение выходного сигнала.
- **Inherit sample time** (флажок) – Наследовать шаг модельного времени. Если этот флажок установлен, то блок **Memory** использует шаг модельного времени (**Sample time**) такой же, как и в предшествующем блоке.

На рис. 9.3.6 показан пример использования блока **Memory** для задержки дискретного сигнала на один временной такт.

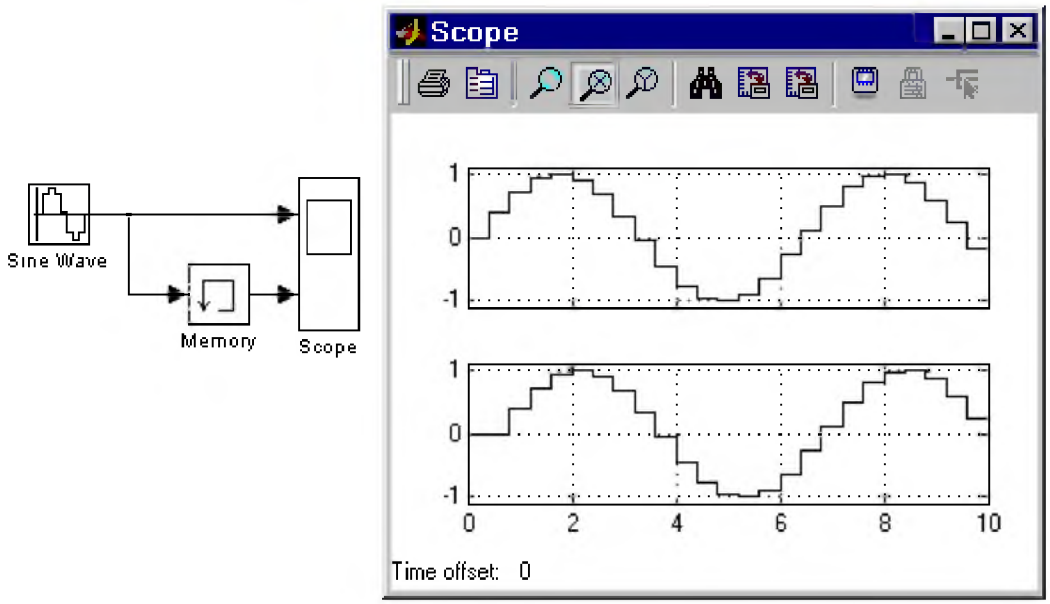


Рис. 9.3.6. Применение блока для задержки сигнала на один временной такт

9.3.4. Блок фиксированной задержки сигнала Transport Delay

Назначение:

Обеспечивает задержку входного сигнала на заданное время.

Параметры:

1. **Time Delay** — Время задержки сигнала (не отрицательное значение).
2. **Initial input** — Начальное значение выходного сигнала.
3. **Buffer size** — Размер памяти, выделяемой для хранения задержанного сигнала. Задается в байтах числом, кратным **8** (по умолчанию **1024**).
4. **Pade order (for linearization)** — Порядок ряда Паде, используемого при аппроксимации выходного сигнала. Задается целым положительным числом.

При выполнении моделирования значение сигнала и соответствующее ему модельное время сохраняются во внутреннем буфере блока **Transport Delay**. По истечении времени задержки значение сигнала, извлекается из буфера и передается на выход блока. В том случае, если шаги модельного времени не совпадают со значениями моментов времени для записанного в буфер сигнала, блок **Transport Delay** выполняет аппроксимацию выходного сигнала.

В том случае, если начального значения объема памяти буфера не хватит для хранения задержанного сигнала, **Simulink** автоматически выделит дополнительную память. После завершения моделирования в командном окне **MATLAB** появится сообщение с указанием нужного размера буфера.

На рис. 9.3.7 показан пример использования блока **Transport Delay** для задержки прямоугольного сигнала на **0.5** с.

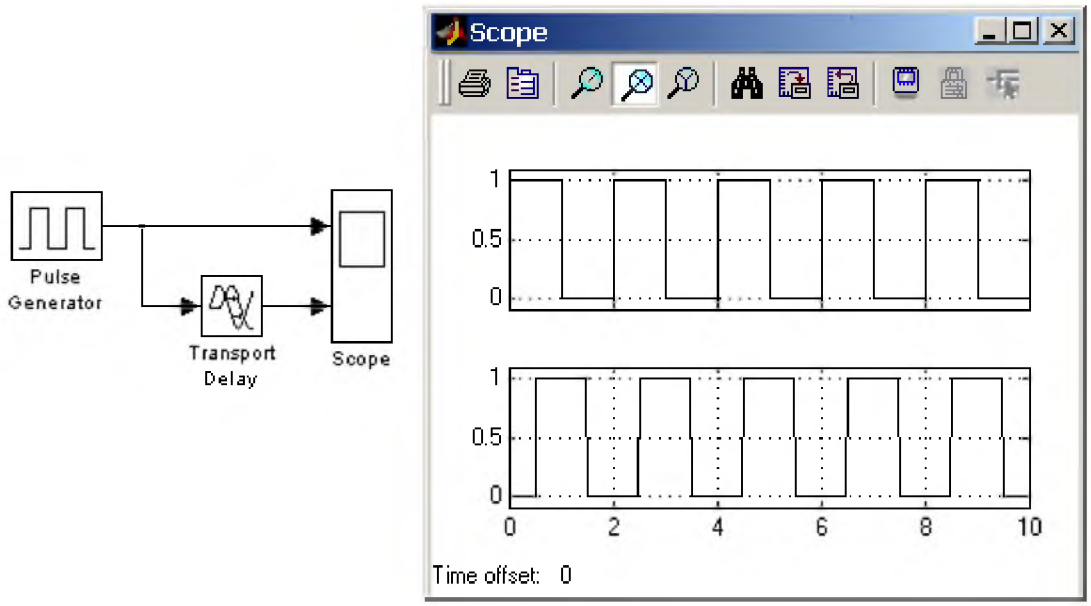


Рис. 9.3.7. Пример использования блока **Transport Delay** для задержки сигнала.

9.3.5. Блок управляемой задержки сигнала Variable Transport Delay

Назначение:

Выполняет задержку входного сигнала, заданную величиной сигнала управления.

Параметры:

1. **Maximum delay** — Максимальное значение времени задержки сигнала (не отрицательное значение).
2. **Initial input** — Начальное значение выходного сигнала.
3. **Buffer size** — Размер памяти, выделяемой для хранения задержанного сигнала. Задается в байтах числом, кратным **8** (по умолчанию **1024**).
4. **Pade order (for linearization)** — Порядок ряда Паде, используемого при аппроксимации выходного сигнала. Задается целым положительным числом.

Блок управляемой задержки **Variable Transport Delay** работает аналогично блоку постоянной задержки сигнала **Transport Delay**.

В том случае, если значение управляющего сигнала задающего величину задержки превышает значение, заданное параметром **Maximum delay,** то задержка выполняется на величину **Maximum delay**.

На рис. 9.3.8 показан пример использования блока **Variable Transport Delay**. Величина времени задержки сигнала изменяется от **0.5**с до **1**с в момент времени равный **5**с.

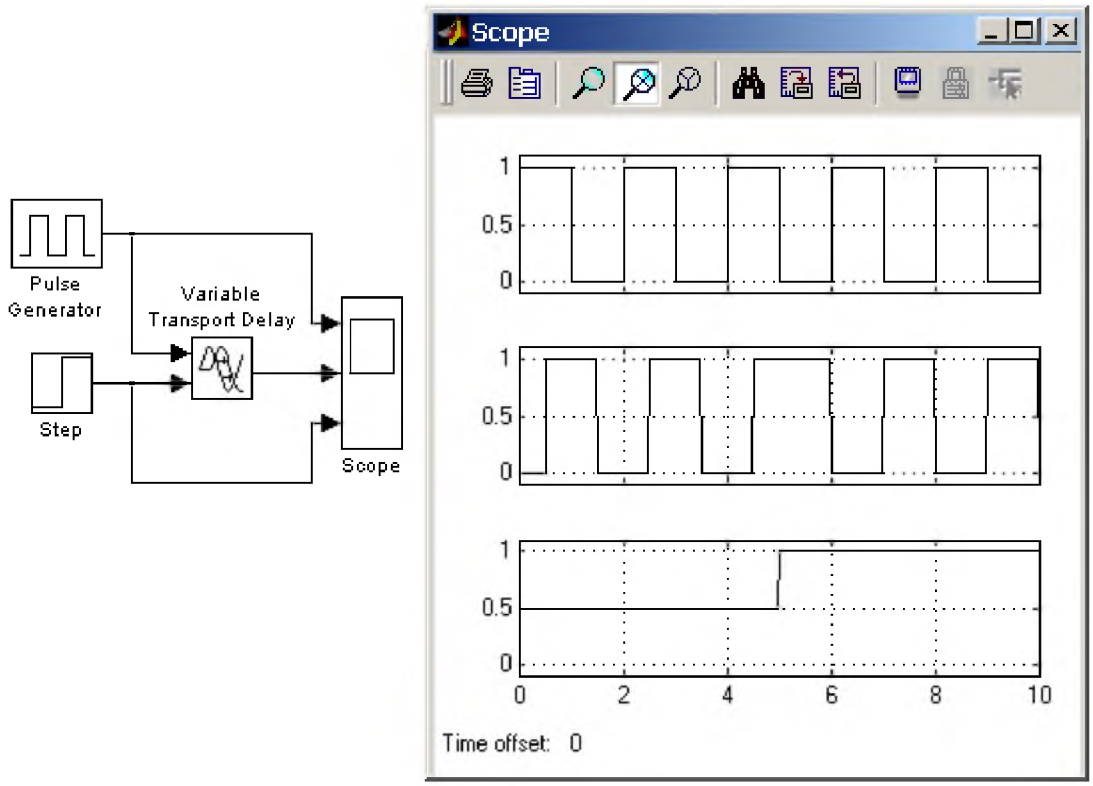


Рис. 9.3.8. Пример использования блока **Variable Transport Delay**.

9.3.6. Блок передаточной функции Transfer Fcn

Назначение:

Блок передаточной характеристики **Transfer Fcn** задает передаточную функцию в виде отношения полиномов:

$$H(s) = \frac{y(s)}{u(s)} = \frac{num(s)}{den(s)} = \frac{num(1)s^{nn-1} + num(2)s^{nn-2} + \ldots + num(nn)}{den(1)s^{nd-1} + den(2)s^{nd-2} + \ldots + den(nd)},$$

где

nn и ***nd*** – порядок числителя и знаменателя передаточной функции,
num – вектор или матрица коэффициентов числителя,
den – вектор коэффициентов знаменателя.

Параметры:

1. **Numerator** — вектор или матрица коэффициентов полинома числителя
2. **Denominator** вектор коэффициентов полинома знаменателя
3. **Absolute tolerance** — Абсолютная погрешность.

Порядок числителя не должен превышать порядок знаменателя.

Входной сигнал блока должен быть скалярным. В том случае, если коэффициенты числителя заданы вектором, то выходной сигнал блока будет также скалярным (как и входной сигнал). На рис. 9.3.8 показан пример моделирования колебательного звена с помощью блока **Transfer Fcn**.

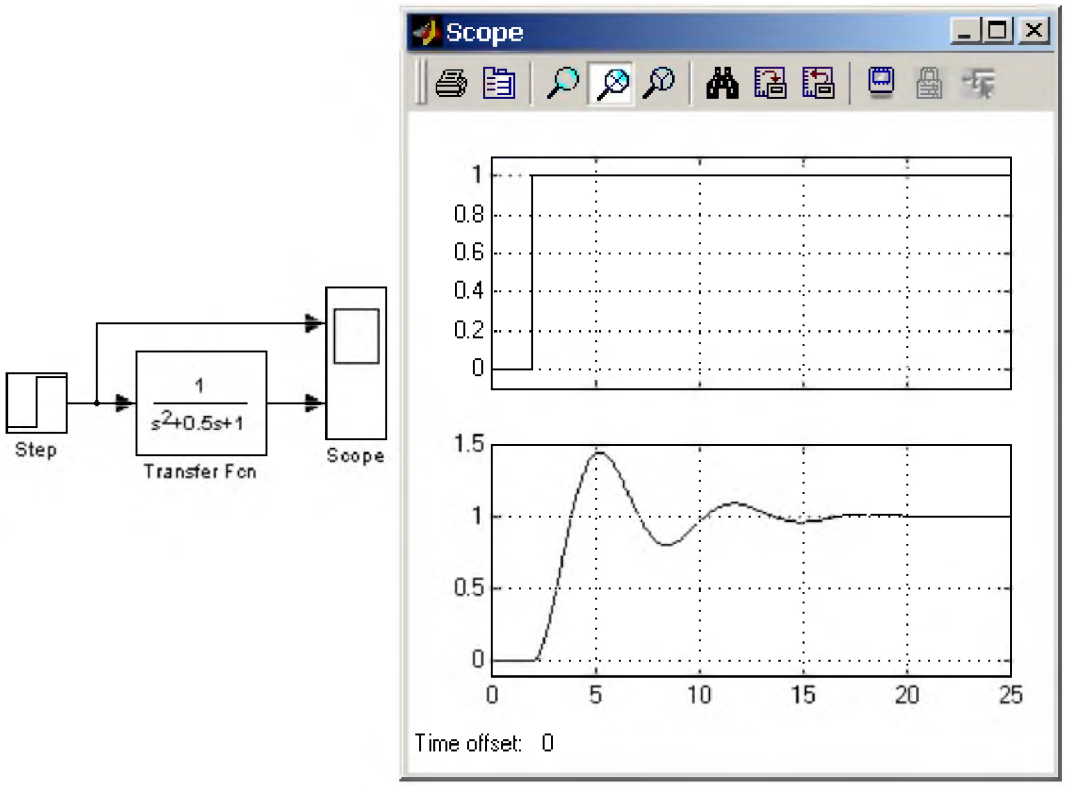


Рис. 9.3.8. Пример моделирования колебательного звена.

Если коэффициенты числителя заданы матрицей, то блок **Transfer Fcn** моделирует векторную передаточную функцию, которую можно интерпретировать как несколько передаточных функций имеющих одинаковые полиномы знаменателя, но разные полиномы числителя. При этом выходной сигнал блока является векторным и количество строк матрицы числителя задает размерность выходного сигнала.

На рис. 9.3.9 показан пример блока **Transfer Fcn** задающий векторную передаточную функцию. Там же показана модель полностью аналогичная рассматриваемой по своим свойствам, но состоящая из отдельных блоков **Transfer Fcn.**

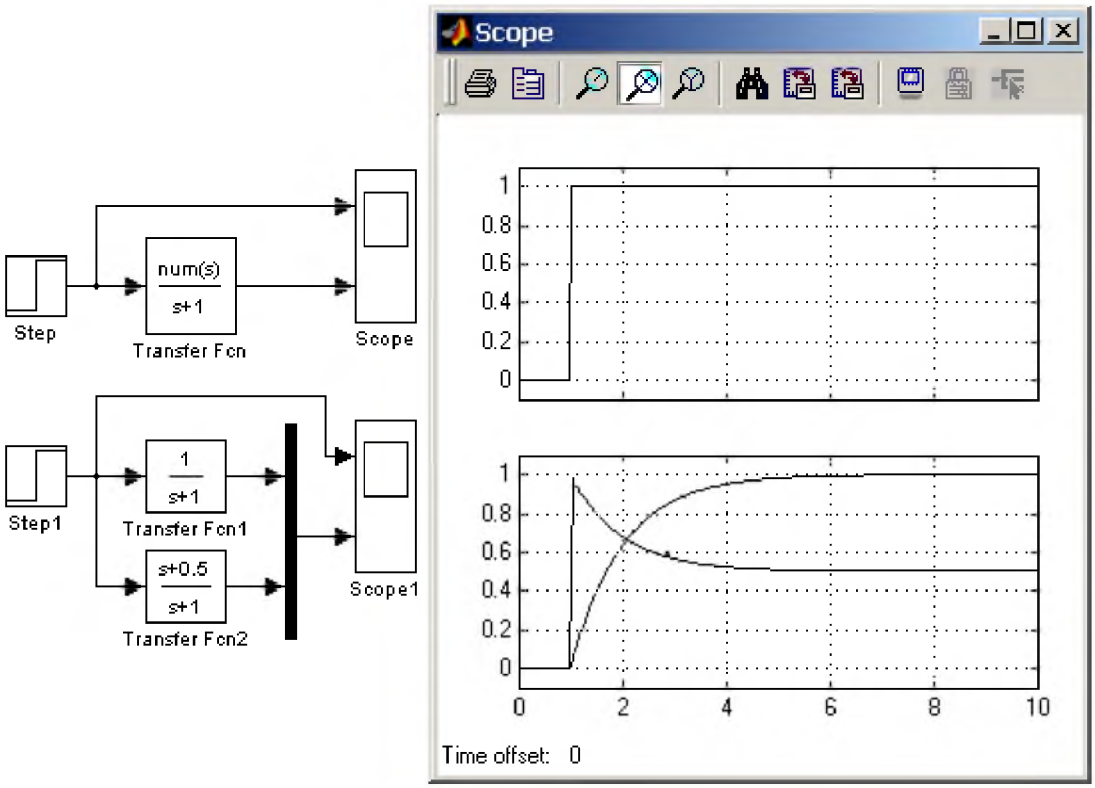


Рис. 9.3.9. Пример моделирования векторной передаточной функции и ее аналог.

Начальные условия при использовании блока **Transfer Fcn** полагаются нулевыми. Если же требуется, чтобы начальные условия не были нулевыми, то необходимо с помощью функции **tf2ss** (инструмент **Control System Toolbox**) перейти от передаточной функции к модели в пространстве состояний и моделировать динамический объект с помощью блока **StateSpace**.

9.3.7. Блок передаточной функции ZeroPole

Назначение:

Блок **ZeroPole** определяет передаточную функцию с заданными полюсами и нулями:

$$H(s) = K\frac{Z(s)}{P(s)} = K\frac{(s-Z(1))(s-Z(2))\ldots(s-Z(m))}{(s-P(1))(s-P(2))\ldots(s-P(n))},$$

где

Z – вектор или матрица нулей передаточной функции (корней полинома числителя),
P – вектор полюсов передаточной функции (корней полинома знаменателя),
K – коэффициент передаточной функции, или вектор коэффициентов, если нули передаточной функции заданы матрицей. При этом размерность вектора ***K*** определяется числом строк матрицы нулей.

Параметры:

1. **Zeros** – Вектор или матрица нулей.
2. **Poles** – Вектор полюсов.

3. **Gain –** Скалярный или векторный коэффициент передаточной функции.
4. **Absolute tolerance** — Абсолютная погрешность.

Количество нулей не должно превышать число полюсов передаточной функции.

В том случае, если нули передаточной функции заданы матрицей, то блок **ZeroPole** моделирует векторную передаточную функцию.

Нули или полюса могут быть заданы комплексными числами. В этом случае нули или полюса должны быть заданы комплексносопряженными парами полюсов или нулей, соответственно.

Начальные условия при использовании блока **ZeroPole** полагаются нулевыми.

На рис. 9.3.10 показан пример использования блока **ZeroPole**. В примере передаточная функция имеет один действительный нуль и два комплексносопряженных полюса.

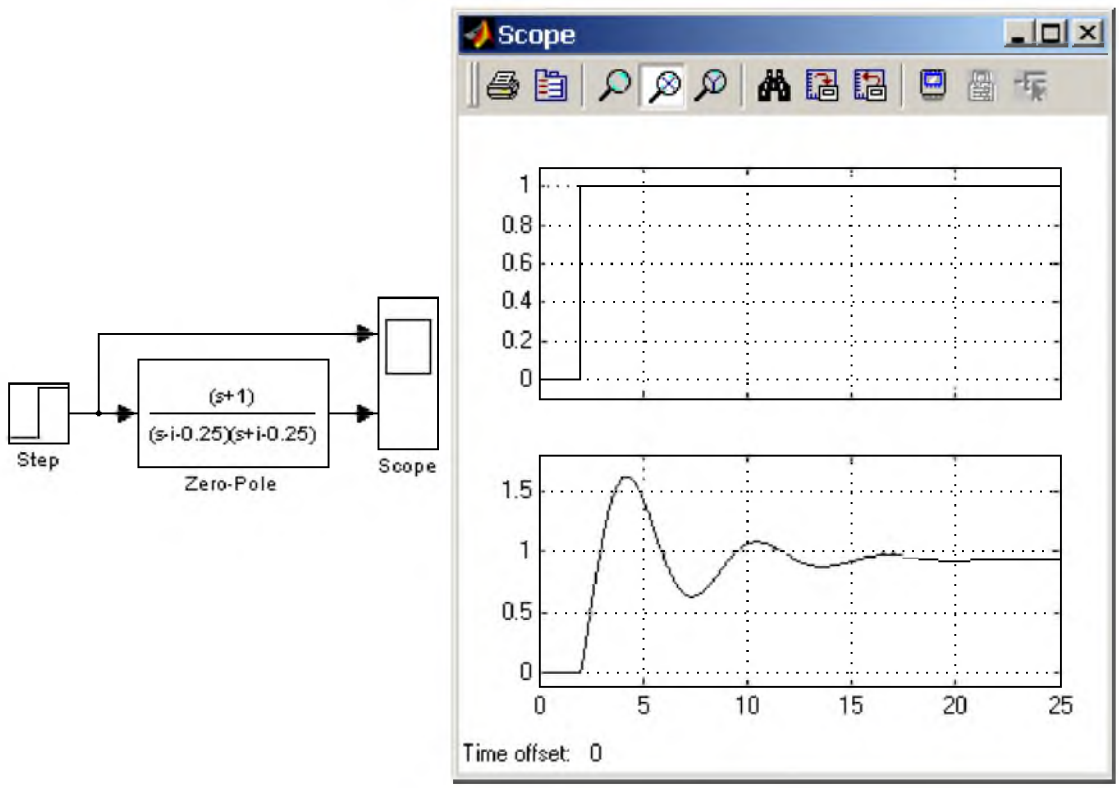


Рис. 9.3.10. Пример использования блока **ZeroPole**.

9.3.8. Блок модели динамического объекта StateSpace

Назначение:

Блок создает динамический объект, описываемый уравнениями в пространстве состояний:

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du,$$

где

x – вектор состояния,
u – вектор входных воздействий,
y – вектор выходных сигналов,
A, B, C, D матрицы: системы, входа, выхода и обхода, соответственно.

Размерность матриц показана на рис. 9.3.11 (**n** – количество переменных состояния, **m** – число входных сигналов, **r** – число выходных сигналов).

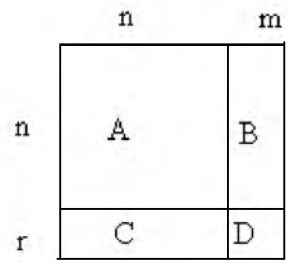


Рис. 9.3.11. Размерность матриц блока **StateSpace**

Параметры:

1. **A** –Матрица системы.
2. **B** – Матрица входа.
3. **C** – Матрица выхода
4. **D** – Матрица обхода
5. **Initial condition –** Вектор начальных условий.
6. **Absolute tolerance** — Абсолютная погрешность.

На рис. 9.3.11 показан пример моделирования динамического объекта с помощью блока **StateSpace**. Матрицы блока имеют следующие значения:

$$A=\begin{vmatrix}0 & 1\\ -5 & -2\end{vmatrix},\quad B=\begin{vmatrix}0\\ 3\end{vmatrix},\quad C=\begin{vmatrix}0 & 1\end{vmatrix},\quad D=\begin{vmatrix}0\end{vmatrix}.$$

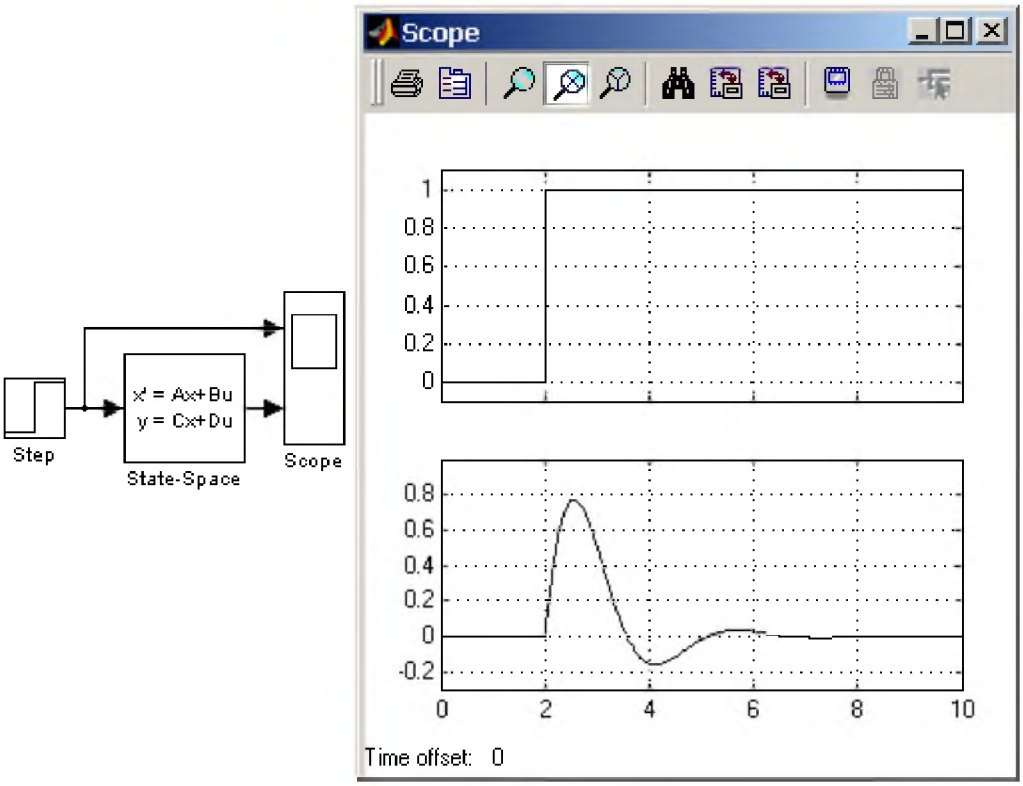


Рис. 9.3.12. Пример использования блока **StateSpace**.

9.4. Discrete – дискретные блоки

9.4.1. Блок единичной дискретной задержки Unit Delay

Назначение:

Выполняет задержку входного сигнала на один шаг модельного времени.

Параметры:

1. **Initial condition** – Начальное значение для выходного сигнала.
2. **Sample time** – Шаг модельного времени.

Входной сигнал блока может быть как скалярным, так и векторным. При векторном входном сигнале задержка выполняется для каждого элемента вектора. Блок поддерживает работу с комплексными и действительными сигналами.

На рис. 9.4.1 показан пример использования блока для задержки дискретного сигнала на один временной шаг, равный **0.1с.**

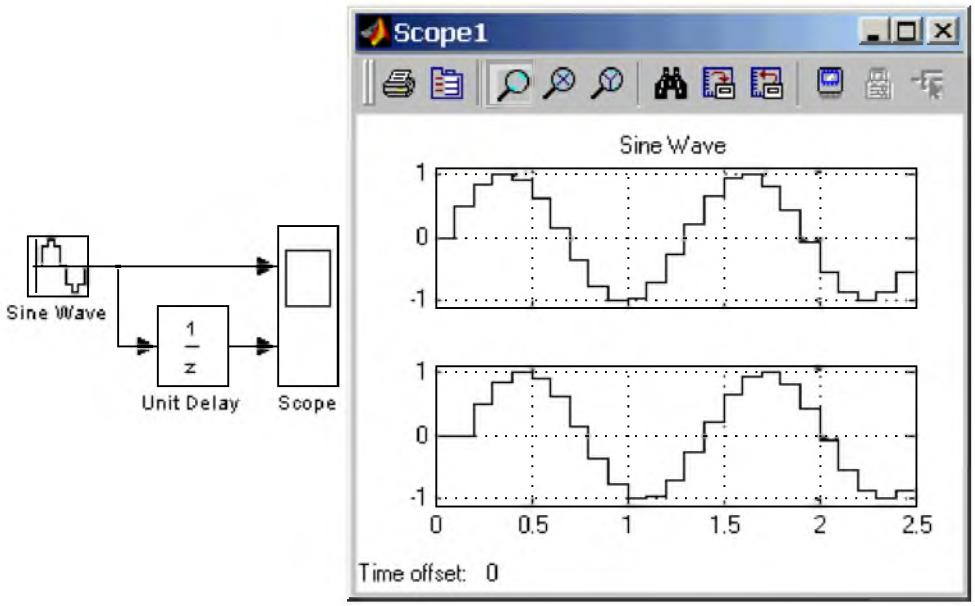


Рис. 9.4.1. Пример использования блока **Unit Delay**

9.4.2. Блок экстраполятора нулевого порядка ZeroOrder Hold

Назначение:

Блок выполняет дискретизацию входного сигнала по времени.

Параметры:

Sample time – Величина шага дискретизации по времени.

Блок фиксирует значение входного сигнала в начале интервала квантования и поддерживает на выходе это значение до окончания интервала квантования. Затем выходной сигнал изменяется скачком до величины входного сигнала на следующем шаге квантования.

На рис. 9.4.2 показан пример использования блока **ZeroOrder Hold** для формирования дискретного сигнала.

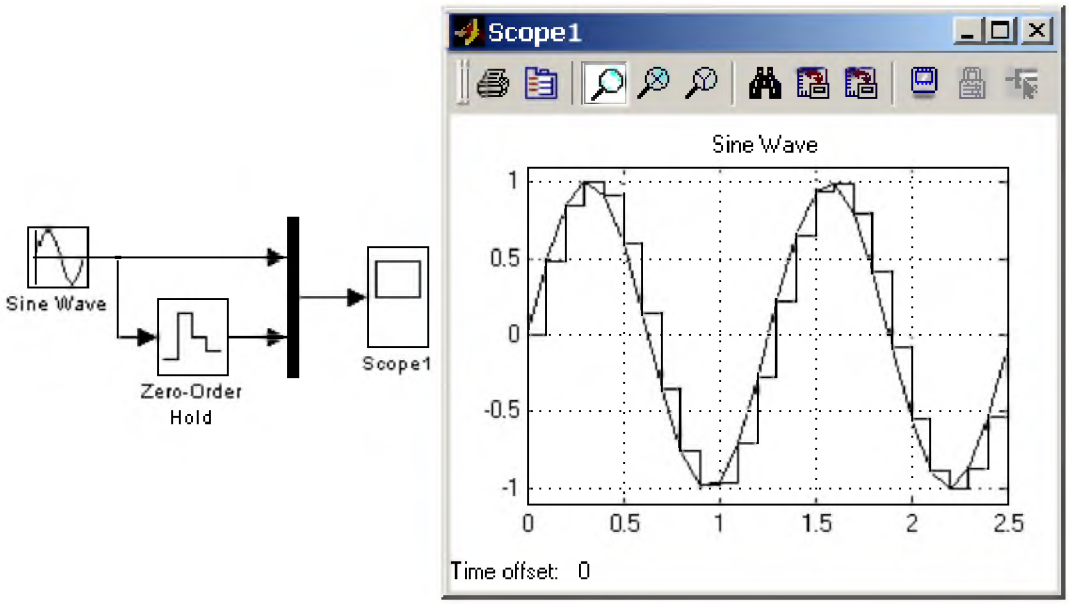
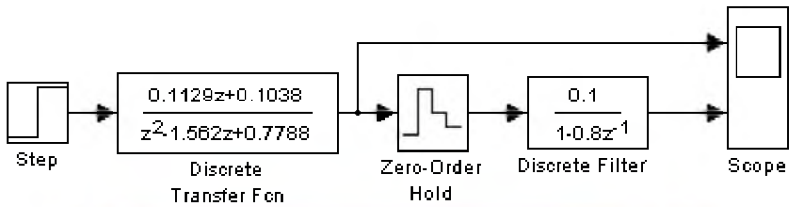


Рис. 9.4.2. Пример формирования дискретного сигнала с помощью блока **ZeroOrder Hold**

Блок экстраполятора нулевого порядка может использоваться также для согласования работы дискретных блоков имеющих разные интервалы квантования. На рис. 9.4.3 показан пример такого использования блока **ZeroOrder Hold.** В примере блок **Discrete Transfer Fcn** имеет параметр **Sample time** = **0.4** , а для блока **Discrete Filter** этот же параметр установлен равным **0.8**.



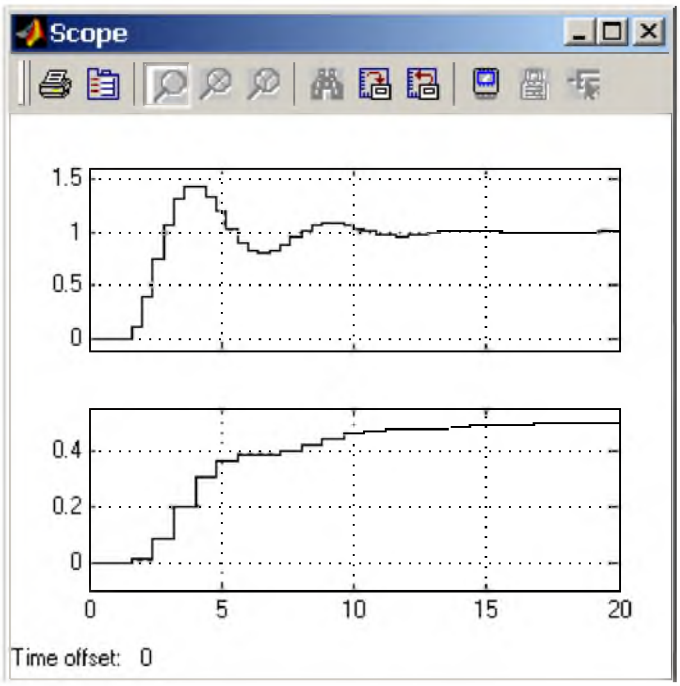


Рис. 9.4.3. Использование блока **ZeroOrder Hold** для согласования работы дискретных блоков.

9.4.3. Блок экстраполятора первого порядка FirstOrder Hold

Назначение:

Блок задает линейное изменение выходного сигнала на каждом такте дискретизации, в соответствии с крутизной входного сигнала на предыдущем интервале дискретизации.

Параметры:

Sample time – Величина шага дискретизации по времени.

Пример экстраполяции синусоидального сигнала этим блоком показан на рис. 9.4.4.

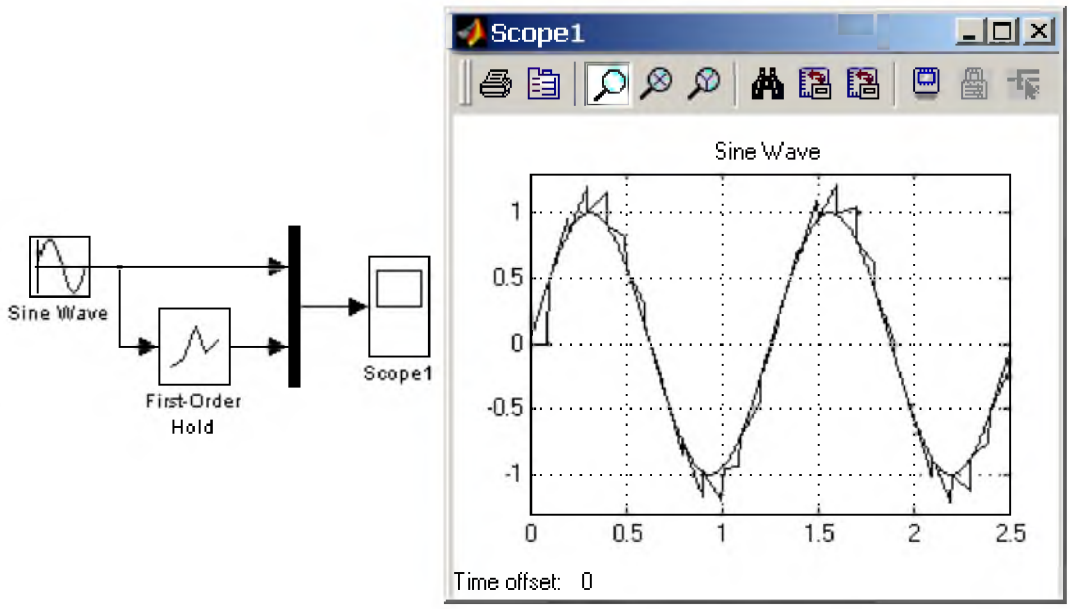


Рис. 9.4.4. Использование блока **FirstOrder Hold**

9.4.4. Блок дискретного интегратора DiscreteTime Integrator

Назначение:

Блок используется для выполнения операции интегрирования в дискретных системах.

Параметры:

1. **Integration method** – Метод численного интегрирования:
   - **Forward Euler** Прямой метод Эйлера.

     Метод использует аппроксимацию **T/(z1)** передаточной функции **1/s.** Выходной сигнал блока рассчитывается по выражению:
     y(k) = y(k–1) + T*u(k–1),
     y – выходной сигнал интегратора,

u – входной сигнал интегратора,
T – шаг дискретизации,
k – номер шага моделирования.

- **Backward Euler** – Обратный метод Эйлера.

  Метод использует аппроксимацию **T*z/(z–1)** передаточной функции **1/s.** Выходной сигнал блока рассчитывается по выражению:
  y(k) = y(k–1) + T*u(k).

- **Trapeziodal** – Метод трапеций.

  Метод использует аппроксимацию **T/2*(z+1)/(z–1)** передаточной функции **1/s.** Выходной сигнал блока рассчитывается по выражению:
  x(k) = y(k–1) + T/2 * u(k–1).

2. **Sample time** — Шаг дискретизации по времени.

Остальные параметры дискретного интегратора те же, что и у блока аналогового интегратора **Integrator** (библиотека **Continuous**).

На рис. 9.4.5 показан пример демонстрирующий все три способа численного интегрирования блока **DiscreteTime Integrator**. Как видно из рисунка изображение блока меняется в зависимости от выбранного метода интегрирования.

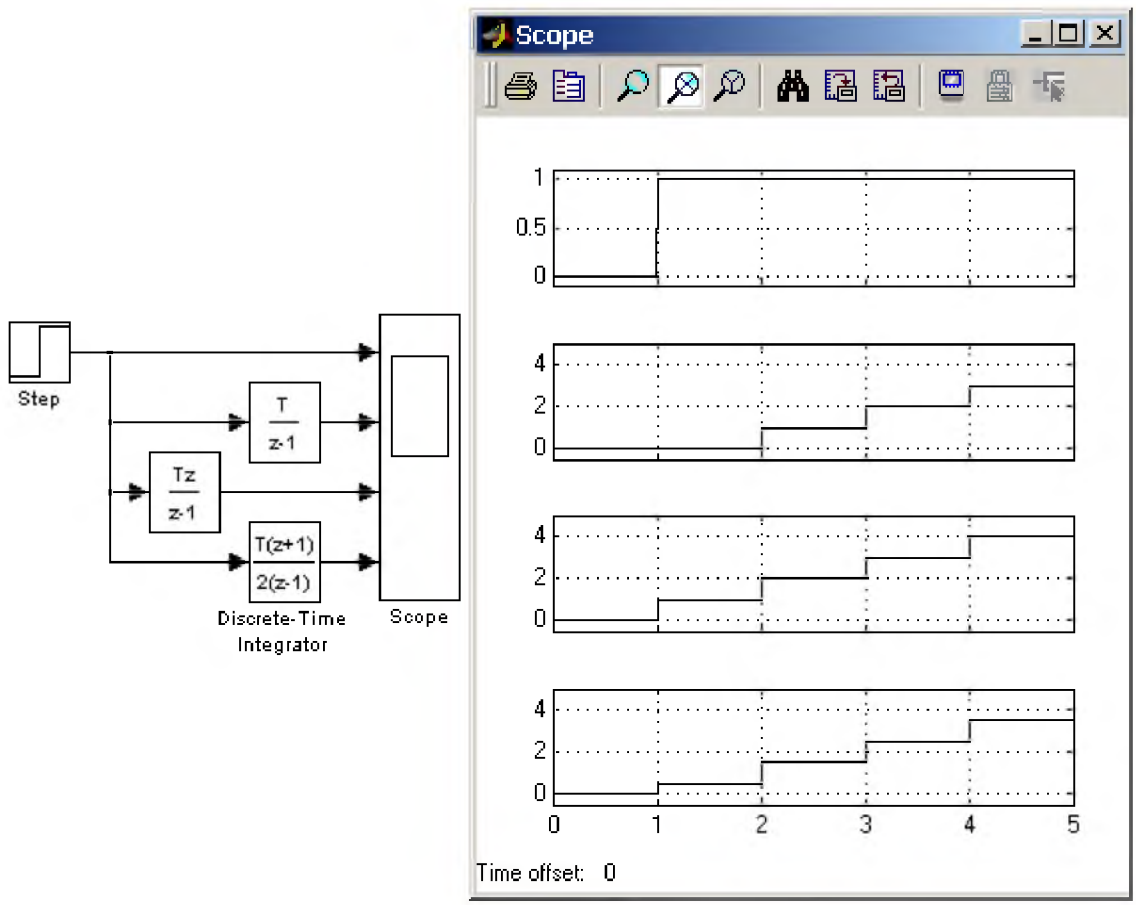


Рис. 9.4.5. Выполнение интегрирования блоками **DiscreteTime Integrator**, реализующими разные численные методы.

9.4.5. Дискретная передаточная функция Discrete Transfer Fcn

Назначение:

Блок **Discrete Transfer Fcn** задает дискретную передаточную функцию в виде отношения полиномов:

$$H(z) = \frac{num(z)}{den(z)} = \frac{num_0 z^n + num_1 z^{n-1} + \ldots + num_m z^{n-m}}{den_0 z^n + den_1 z^{n-1} + \ldots + den_n},$$

где

m+1 и ***n+1*** – количество коэффициентов числителя и знаменателя, соответственно.
num – вектор или матрица коэффициентов числителя,
den – вектор коэффициентов знаменателя.

Параметры:

1. **Numerator** — Вектор или матрица коэффициентов числителя
2. **Denominator** – Вектор коэффициентов знаменателя
3. **Sample time** — Шаг дискретизации по времени.

Порядок числителя не должен превышать порядок знаменателя.

Входной сигнал блока должен быть скалярным. В том случае, если коэффициенты числителя заданы вектором, то выходной сигнал блока будет скалярным (также как и входной сигнал). На рис. 9.4.6 показан пример использования блока **Discrete Transfer Fcn**. В примере рассчитывается реакция на единичное ступенчатое воздействие дискретного аналога колебательного звена: $\frac{1}{s^2+0.5s+1}$.

Шаг дискретизации выбран равным **0.5** с.

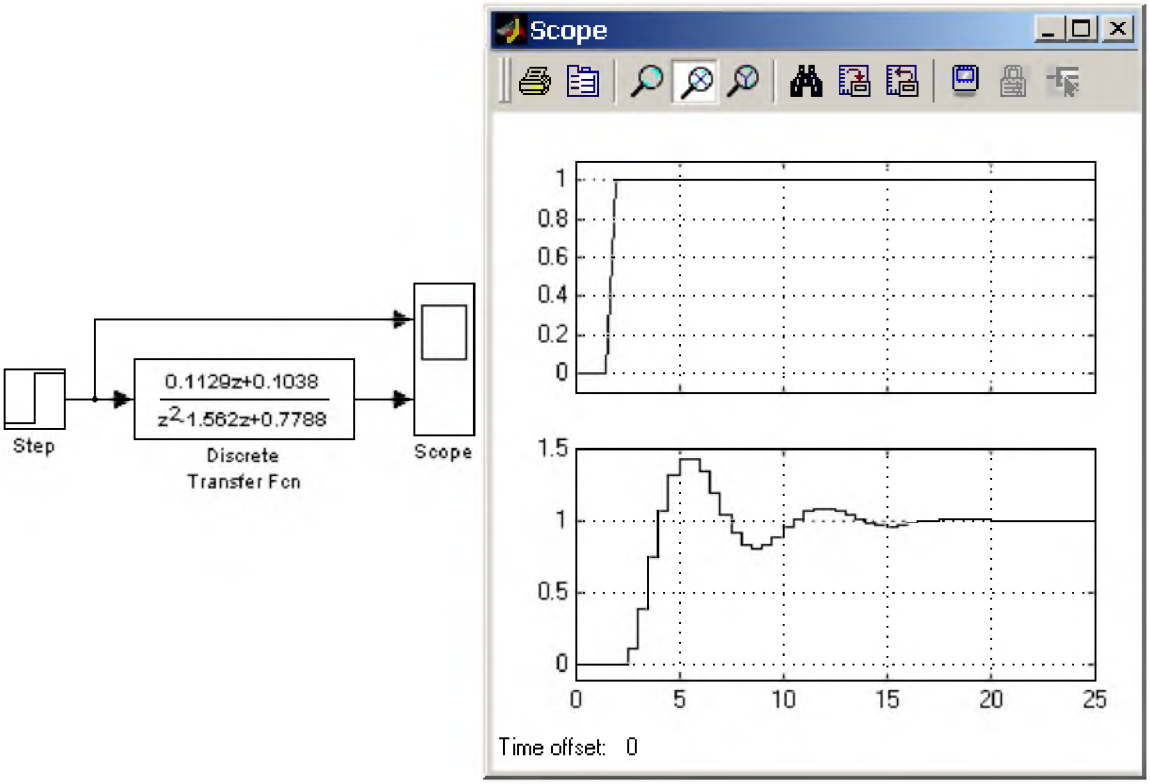


Рис. 9.4.6. Использование блока **Discrete Transfer Fcn**

9.4.6. Блок дискретной передаточной функции Discrete ZeroPole

Назначение:

Блок Discrete **ZeroPole** определяет дискретную передаточную функцию с заданными полюсами и нулями:

$$H(z) = K\frac{Z(z)}{P(z)} = K\frac{(z-Z_1)(z-Z_2)\ldots(z-Z_m)}{(z-P_1)(z-P_2)\ldots(z-P_n)},$$

где

Z – вектор или матрица нулей передаточной функции,
P – вектор полюсов передаточной функции,
K – коэффициент передаточной функции, или вектор коэффициентов, если нули передаточной функции заданы матрицей. При этом размерность вектора ***K*** определяется числом строк матрицы нулей.

Параметры:

1. **Zeros –** Вектор или матрица нулей.
2. **Poles –** Вектор полюсов.
3. **Gain –** Скалярный или векторный коэффициент передаточной функции.
4. **Sample time** — Шаг дискретизации по времени.

Количество нулей не должно превышать число полюсов передаточной функции.

В том случае, если нули передаточной функции заданы матрицей, то блок **Discrete ZeroPole** моделирует векторную передаточную функцию.

Нули или полюса могут быть заданы комплексными числами. В этом случае нули или полюса должны быть заданы комплексносопряженными парами полюсов или нулей, соответственно.

Начальные условия при использовании блока **Discrete ZeroPole** полагаются нулевыми.

На рис. 9.4.7 показан пример использования блока **Discrete ZeroPole**. С помощью рассматриваемого блока моделируется дискретный аналог передаточной функции

$$\frac{1}{(s+0.25-0.968i)(s+0.25+0.968i)}$$

Шаг дискретизации выбран равным **0.5** с.

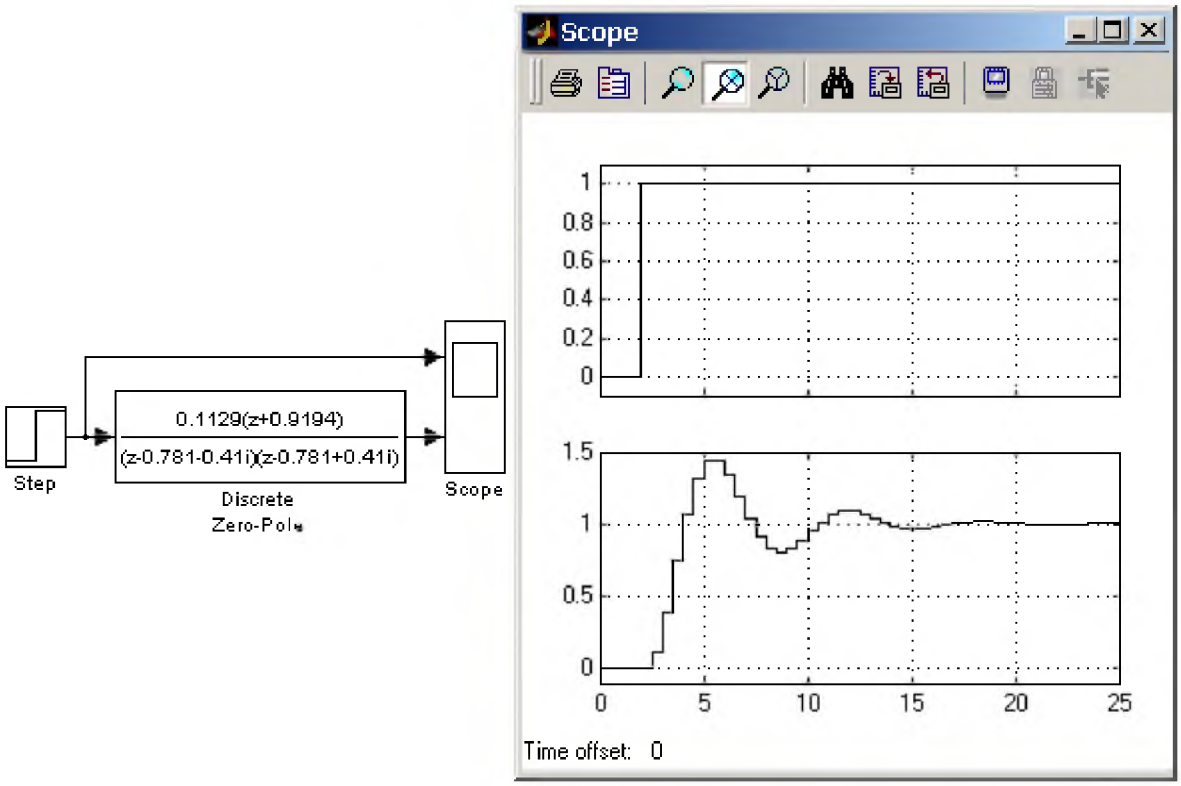


Рис. 9.4.7. Использование блока **Discrete ZeroPole**.

9.4.7. Блок дискретного фильтра Discrete Filter

Назначение:

Блок дискретного фильтра **Discrete Filter** задает дискретную передаточную функцию от обратного аргумента (**1/z**):

$$H(1/z) = \frac{num(1/z)}{den(1/z)} = \frac{num_0 z^0 + num_1 z^{-1} + num_2 z^{-2} + \ldots + num_m z^{-m}}{den_0 z^0 + den_1 z^{-1} + den_2 z^{-2} + \ldots + den_n z^{-n}},$$

m+1 и ***n+1*** – количество коэффициентов числителя и знаменателя, соответственно.
num – вектор или матрица коэффициентов числителя,
den – вектор коэффициентов знаменателя.

Параметры:

1. **Numerator** — Вектор или матрица коэффициентов числителя
2. **Denominator** –Вектор коэффициентов знаменателя
3. **Sample time** — Шаг дискретизации по времени.

На рис. 9.4.8 показан пример использования блока **Discrete Filter**. С помощью рассматриваемого блока моделируется дискретный аналог передаточной функции:

$$\frac{1}{4s+1}$$

Шаг дискретизации выбран равным **0.5** с.

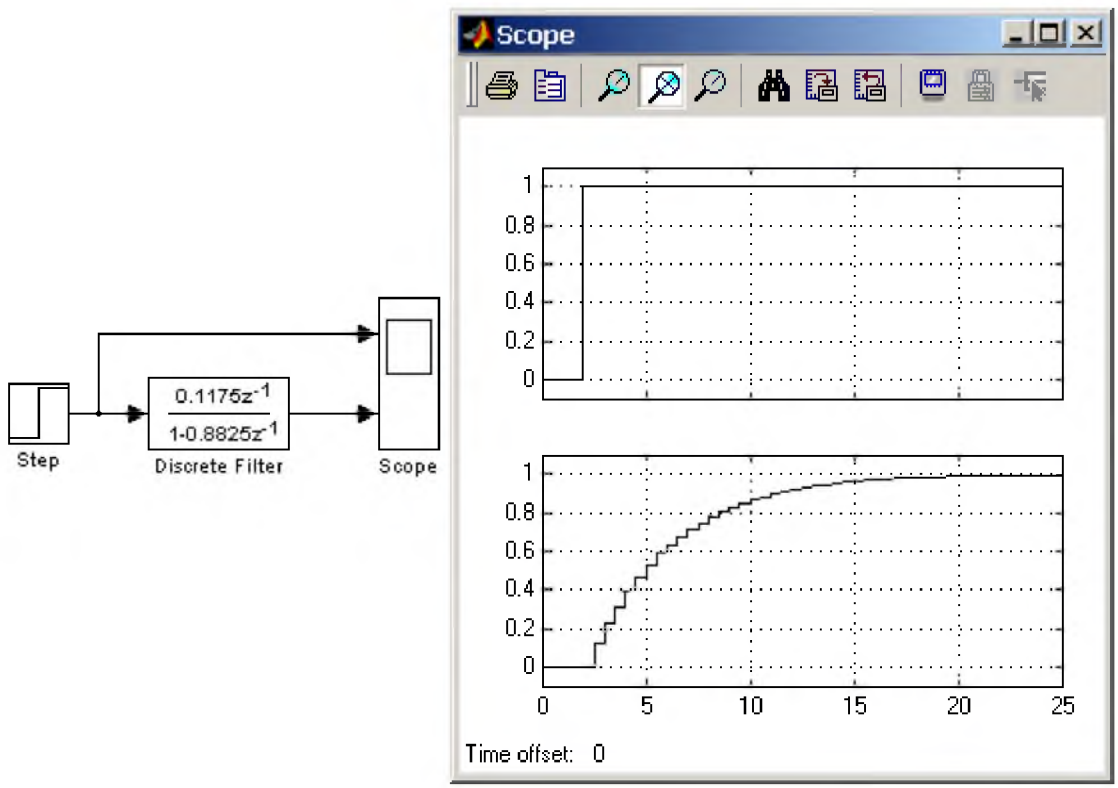


Рис. 9.4.8. Использование блока **Discrete Filter**.

9.4.8. Блок модели динамического объекта Discrete StateSpace

Назначение:

Блок создает динамический объект, описываемый уравнениями в пространстве состояний:

$$x(n+1) = Ax(n) + Bu(n)$$
$$y(n) = Cx(n) + Du(n)$$
,

где

x – вектор состояния,

u – вектор входных воздействий,

y – вектор выходных сигналов,

A, B, C, D матрицы: системы, входа, выхода и обхода, соответственно,

n – номер шага моделирования.

Размерность матриц показана на рис. 9.4.9 (**n** – количество переменных состояния, **m** – число входных сигналов, **r** – число выходных сигналов).

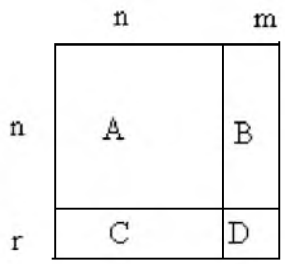


Рис. 9.4.9. Размерность матриц блока **Discrete StateSpace**

Параметры:

1. **A** –Матрица системы.
2. **B** – Матрица входа.
3. **C** – Матрица выхода
4. **D** – Матрица обхода
5. **Initial condition** – Вектор начальных условий.
6. **Sample time** — Шаг дискретизации по времени.

На рис. 9.4.10 показан пример моделирования динамического объекта с помощью блока **Discrete StateSpace**. Матрицы блока имеют следующие значения:

$$A = \begin{vmatrix} 0.9135 & 0.1594 \\ -0.7971 & 0.5947 \end{vmatrix}, \quad B = \begin{vmatrix} 0.05189 \\ 0.4782 \end{vmatrix}, \quad C = |0 \quad 1|, \quad D = |0|.$$

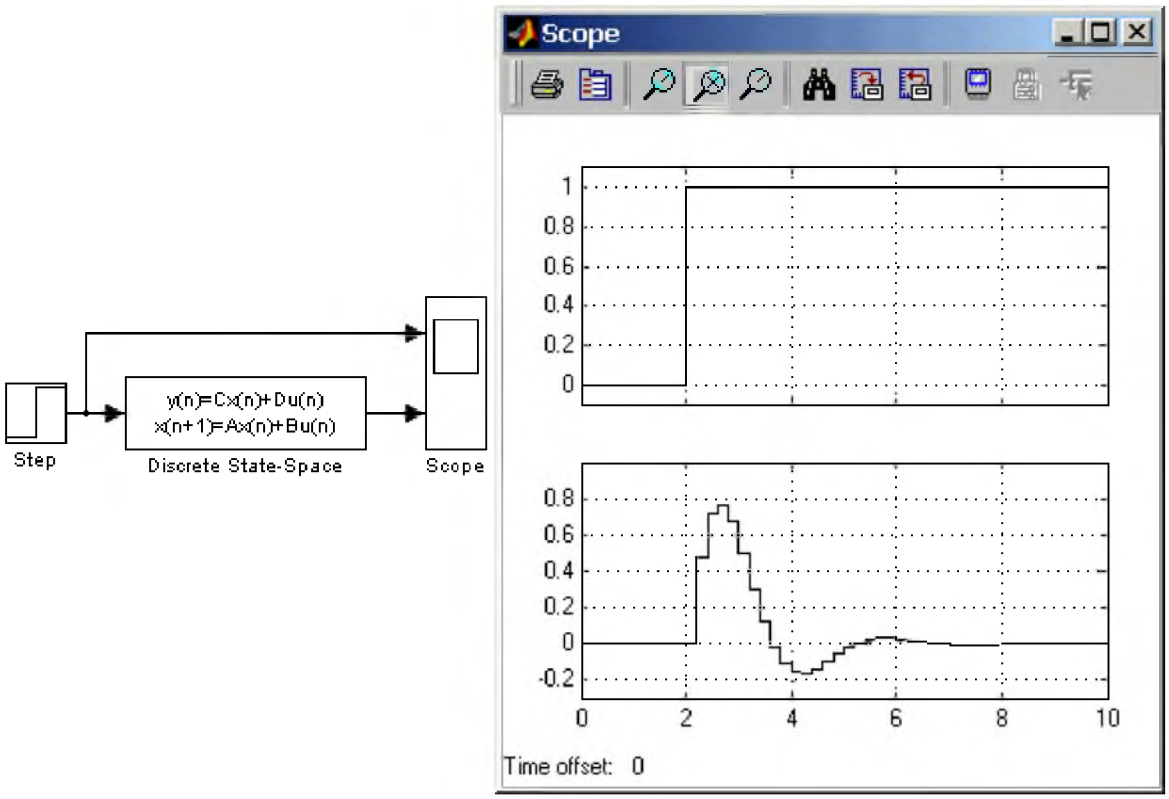


Рис. 9.4.10. Пример использования блока **Discrete StateSpace**.

9. Библиотека блоков Simulink

9.5. Nonlinear нелинейные блоки

9.5.1. Блок ограничения Saturation

Назначение:

Выполняет ограничение величины сигнала.

Параметры:

1. **Upper limit** Верхний порог ограничения.
2. **Lower limit** Нижний порог ограничения.
3. **Treat as gain when linearizing (флажок)** Трактовать как усилитель с коэффициентом передачи равным 1 при линеаризации.

Выходной сигнал блока равен входному если его величина не выходит за порог ограничения. По достижении входным сигналом уровня ограничения выходной сигнал блока перестает изменяться и остается равным порогу. На рис. 9.5.1 показан пример использования блока для ограничения синусоидального сигнала. На рисунке приводятся временные диаграммы сигналов и зависимость выходного сигнала блока от входного.

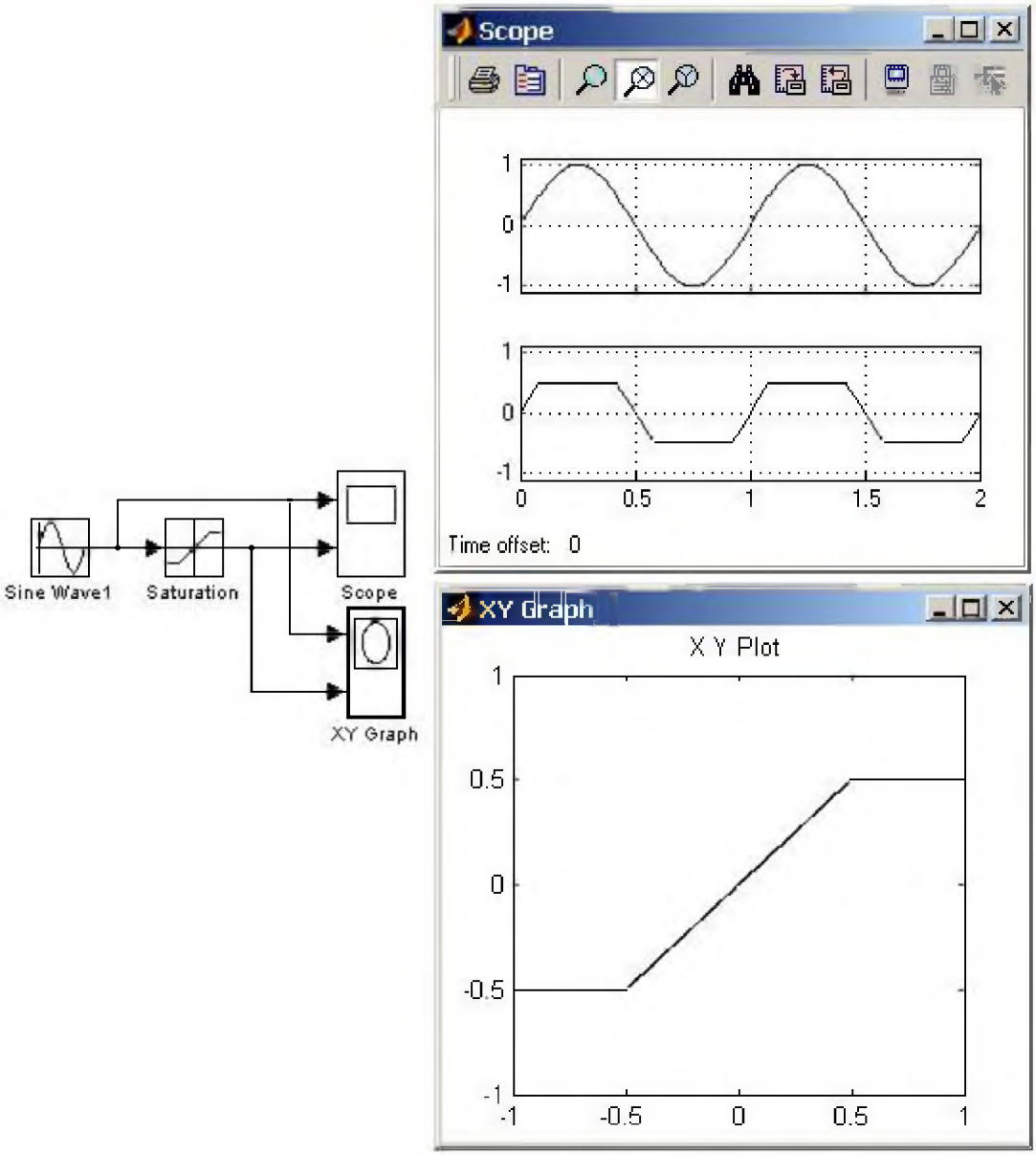


Рис. 9.5.1. Пример использования блока **Saturation**

9.5.2. Блок с зоной нечувствительности Dead Zone

Назначение:

Реализует нелинейную зависимость типа "зона нечувствительности (мертвая зона)".

Параметры:

1. **Start of dead zone** Начало зоны нечувствительности (нижний порог).
2. **End of dead zone** Конец зоны нечувствительности (верхний порог).
3. **Saturate on integer overflow (флажок)** Подавлять переполнение целого. При установленном флажке ограничение сигналов целого типа выполняется корректно.
4. **Treat as gain when linearizing (флажок)** Трактовать как усилитель с коэффициентом передачи равным 1 при линеаризации.

Выходной сигнал блока вычисляется в соответствии со следующим алгоритмом:

- Если величина входного сигнала находится в пределах зоны нечувствительности, то выходной сигнал блока равен нулю.
- Если входной сигнал больше или равен верхнему входному порогу зоны нечувствительности, то выходной сигнал равен входному минус величина порога.
- Если входной сигнал меньше или равен нижнему входному порогу зоны нечувствительности, то выходной сигнал равен входному минус величина порога.

На рис. 9.5.2 показан пример использования блока **Dead Zone**

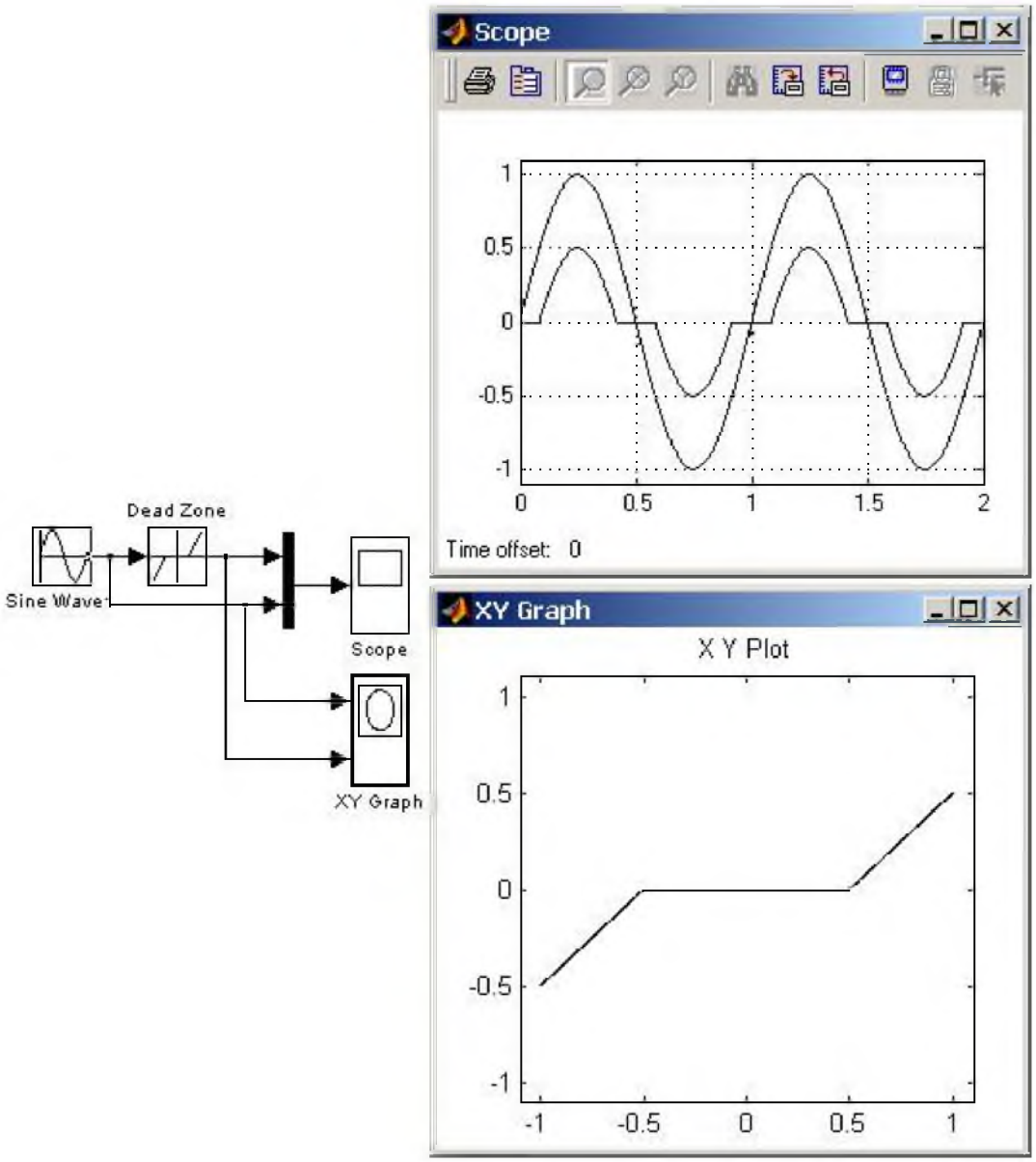


Рис. 9.5.2. Пример использования блока **Dead Zone**

9.5.3. Релейный блок Relay

Назначение:

Реализует релейную нелинейность.

Параметры:

1. **Switch on point** Порог включения. Значение, при котором происходит включение реле.
2. **Switch off point** Порог выключения. Значение, при котором происходит выключение реле.
3. **Output when on** Величина выходного сигнала во включенном состоянии.
4. **Output when off** Величина выходного сигнала в выключенном состоянии.

Выходной сигнал блока может принимать два значения. Одно из них соответствует включенному состоянию реле, второе выключенному. Переход их одного состояния в другое происходит скачком при достижении входным сигналом порога включения или выключения реле. В том случае если пороги включения и выключения реле имеют разные значения, то блок реализует релейную характеристику с гистерезисом. При этом значение порога включения должно быть больше, чем значение порога выключения.

На рис. 9.5.3 показан пример использования блока Relay. На временных диаграммах видно, что включение реле происходит при достижении входным сигналом величины 0.5, а выключение при 0.5.

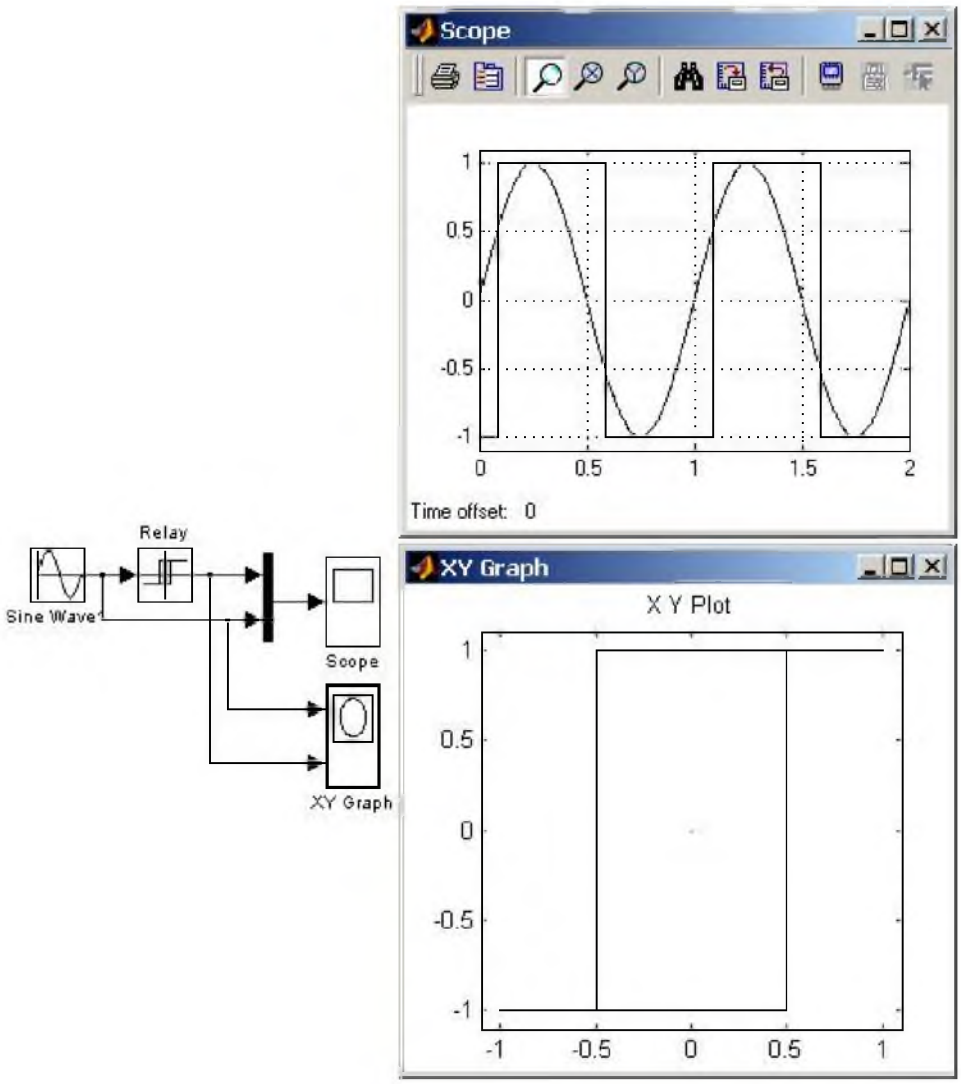


Рис. 9.5.3. Пример использования блока **Relay**

9.5.4. Блок ограничения скорости изменения сигнала Rate Limiter

Назначение:

Блок обеспечивает ограничение скорости изменения сигнала (первой производной).

Параметры:

1. **Rising slew rate** Уровень ограничения скорости при увеличении сигнала.
2. **Falling slew rate** Уровень ограничения скорости при уменьшении сигнала.

Вычисление производной сигнала выполняется по выражению:

$$rate = \frac{u(i) - y(i-1)}{t(i) - t(i-1)},$$

где
u(i) значение входного сигнала на текущем шаге,
t(i) значение модельного времени на текущем шаге,
y(i1) значение выходного сигнала на предыдущем шаге,
t(i1) значение модельного времени на предыдущем шаге.

Вычисленное значение производной сравнивается со значениями уровней ограничения скорости Rising slew rate и Falling slew rate. Если значение производной больше, чем значение параметра Rising slew rate, то выходной сигнал блока вычисляется по выражению:

$$y(i) = \Delta t \cdot R + y(i-1),$$

где R уровень ограничения скорости при увеличении сигнала.

Если значение производной меньше, чем значение параметра Falling slew rate, то выходной сигнал блока вычисляется по выражению:

$$y(i) = \Delta t \cdot F + y(i-1),$$

где F уровень ограничения скорости при уменьшении сигнала.

Если значение производной лежит в пределах между нижним и верхним уровнями ограничения, то выходной сигнал блока равен входному:

$$y(i) = u(i).$$

На рис. 9.5.4 показан пример использования блока Rate Limiter, при подаче на его вход прямоугольного периодического сигнала.

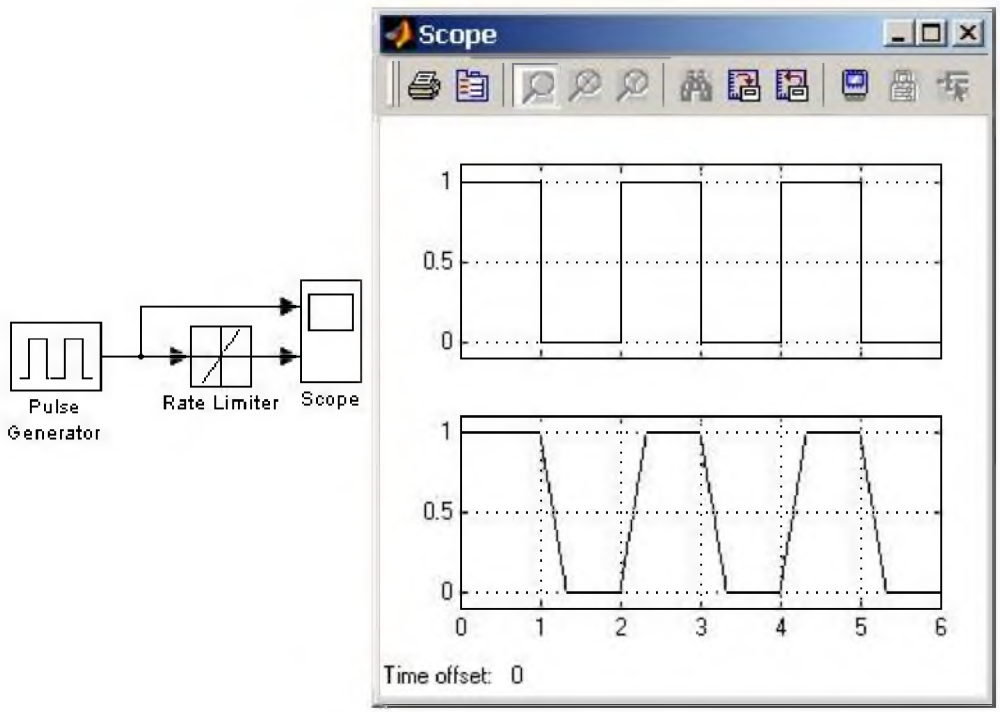


Рис. 9.5.4. Пример использования блока **Rate Limiter**

9.5.5. Блок квантования по уровню Quantizer

Назначение:

Блок обеспечивает квантование входного сигнала с одинаковым шагом по уровню.

Параметры:

Quantization interval шаг квантования по уровню.

На рис. 9.5.5 показан пример использования блока Quantizer, выполняющего квантование по уровню синусоидального сигнала. Шаг квантования задан равным 0.5.

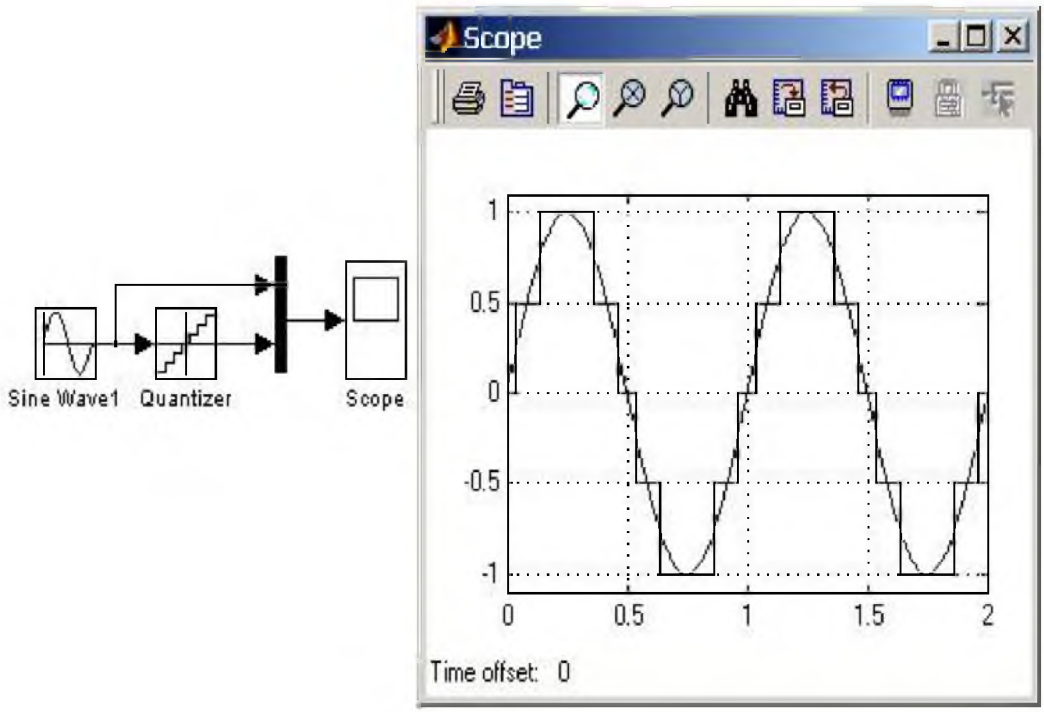


Рис. 9.5.5. Пример использования блока **Quantizer**

9.5.6. Блок сухого и вязкого трения Coulomb and Viscous Friction

Назначение:

Моделирует эффекты сухого и вязкого трения.

Параметры:

1. **Coulomb friction value (Offset)–** Величина сухого трения.
2. **Coefficient of viscous friction (Gain) –** Коэффициент вязкого трения.

Блок реализует нелинейную характеристику, соответствующую выражению:

```
y = sign(u) * (Gain * abs(u) + Offset)
```

где ***u*** – входной сигнал,
y – выходной сигнал,
Gain – коэффициент вязкого трения ,
Offset – Величина сухого трения.

На рис. 9.5.6 показан пример использования блока **Coulomb and Viscous Friction**. Оба параметра блока заданы равными **1**.

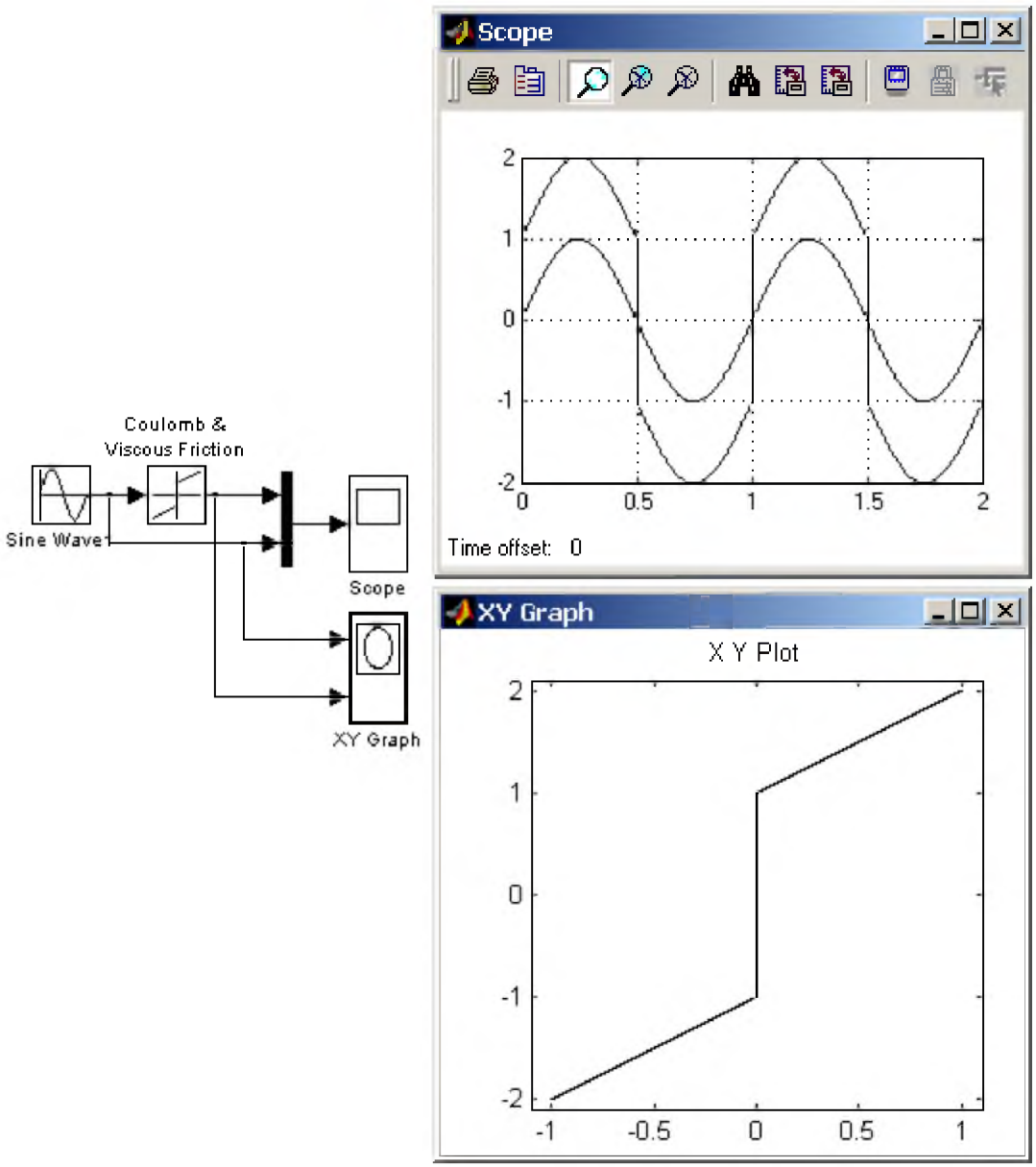


Рис. 9.5.6. Пример использования блока **Coulomb and Viscous Friction**

9.5.7. Блок люфта Backlash

Назначение:

Моделирует нелинейность типа “люфт”.

Параметры:

1. **Deaband width** – Ширина люфта.
2. **Initial output** – Начальное значение выходного сигнала.

Сигнал на выходе будет равен заданному значению **Initial output**, пока входной сигнал при возрастании не достигнет значения **(Deaband width)/2** (где **U** – входной сигнал), после чего выходной сигнал будет равен **U(Deaband width)/2.** После того как, произойдет смена направления изменения

входного сигнала, он будет оставаться неизменным, пока входной сигнал не изменится на величину **(Deaband width)/2,** после чего выходной сигнал будет равен **U+(Deaband width)/2**.

На рис. 9.5.7 показан пример работы блока **Backlash**. Входной сигнал блока гармонический с линейно возрастающей амплитудой.

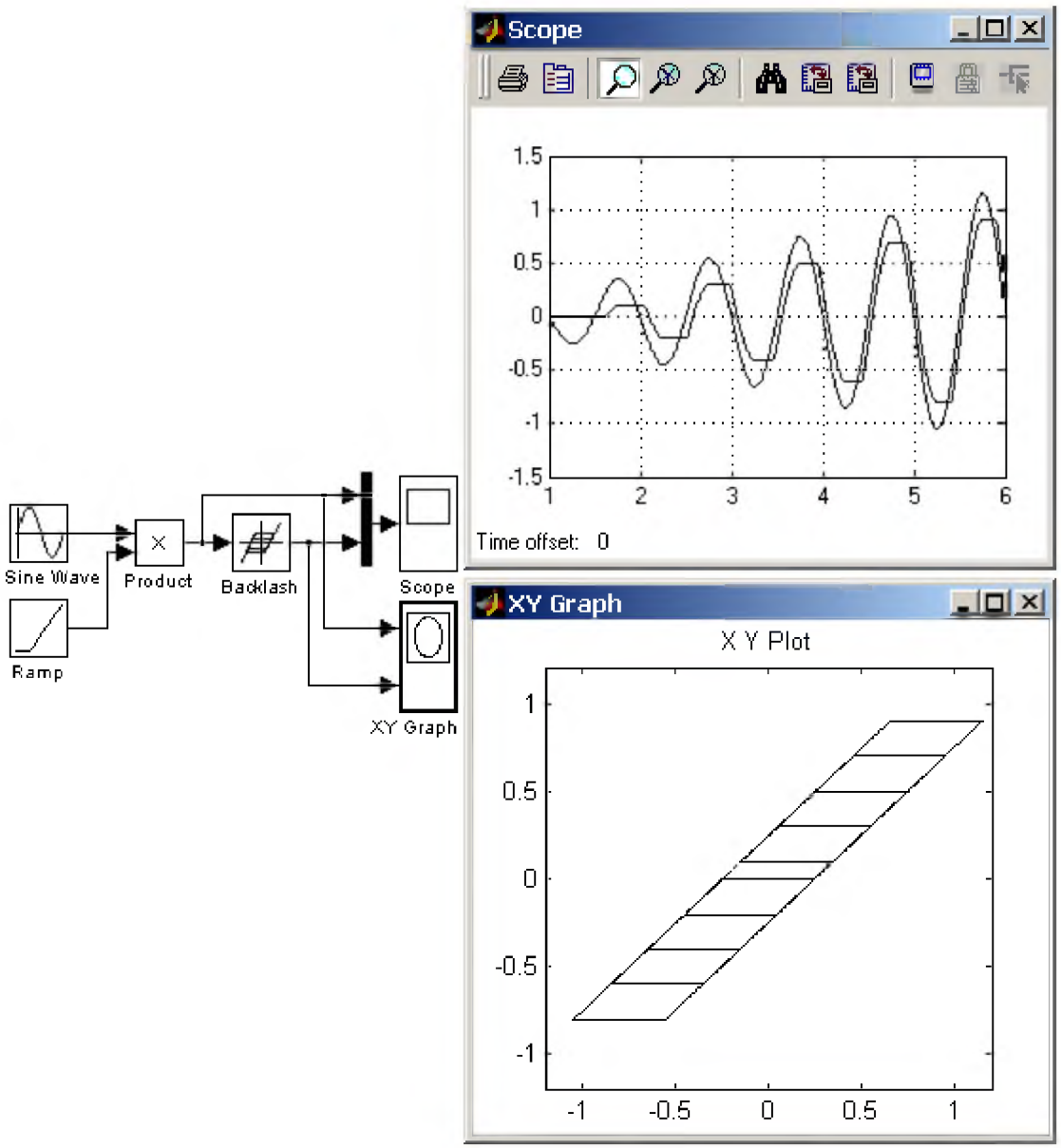


Рис. 9.5.7. Пример использования блока **Backlash**

9.5.8. Блок переключателя Switch

Назначение:

Выполняет переключение входных сигналов по сигналу управления.

Параметры:

Threshold – Порог управляющего сигнала.

Блок работает следующим образом:
Если сигнал управления, подаваемый на средний вход меньше, чем величина порогового значения **Threshold**, то на выход блока проходит сигнал с первого (верхнего) входа. Если сигнал управления превысит пороговое значение, то на выход блока будет поступать сигнал со второго (нижнего) входа.

На рис. 9.5.8 показан пример работы блока **Switch**. В том случае, когда сигнал на управляющем входе ключа равен **1**, на выход блока проходит гармонический сигнал, если же управляющий сигнал равен нулю, то на выход проходит сигнал нулевого уровня от блока **Ground**. Пороговое значение управляющего сигнала задано равным **0.5**.

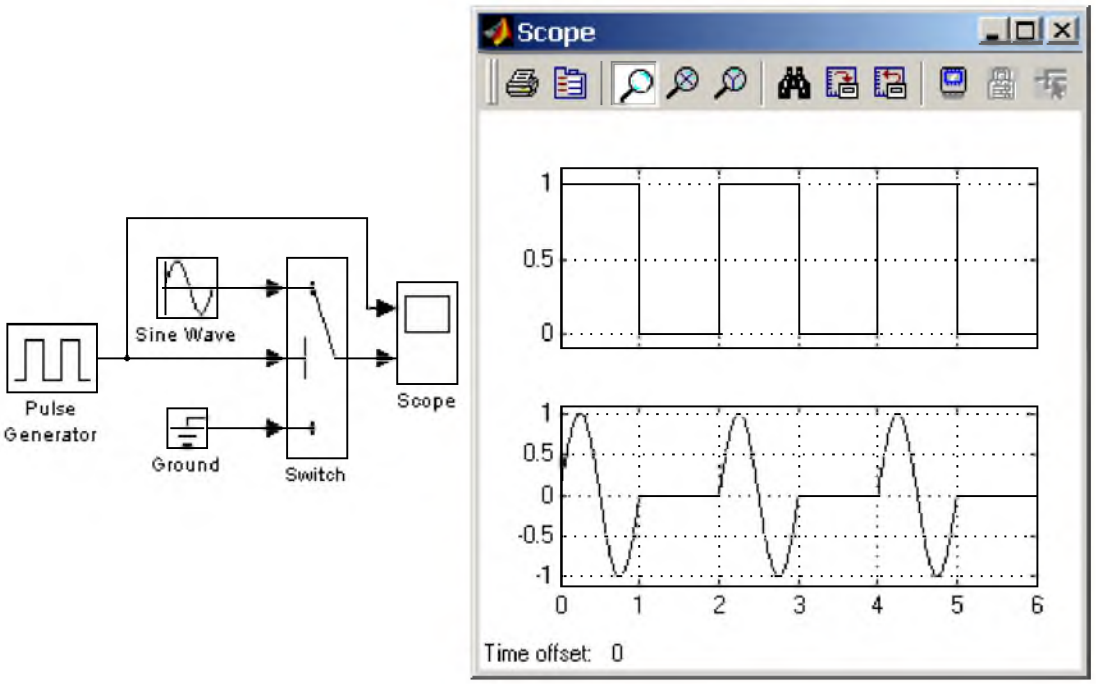


Рис. 9.5.8. Применение переключателя **Switch**

9.5.9. Блок многовходового переключателя Multiport Switch

Назначение:

Выполняет переключение входных сигналов по сигналу управления, задающему номер активного входного порта.

Параметры:

Number of inputs – Количество входов.

Блок многовходового переключателя **Multiport Switch,** пропускает на выход сигнал с того входного порта, номер которого равен текущему значению управляющего сигнала. Если управляющий сигнал не является сигналом целого типа, то блок **Multiport Switch** производит отбрасывание дробной части числа, при этом в командном окне **Matlab** появляется предупреждающее сообщение.

На рис. 9.5.9 показан пример работы блока **Multiport Switch**. Управляющий сигнал переключателя имеет три уровня и формируется с помощью блоков **Constant**, **Step**, **Step1** и **Sum**. На выход блока **Multiport Switch,** в зависимости от уровня входного сигнала**,** проходят гармонические сигналы, имеющие разные частоты.

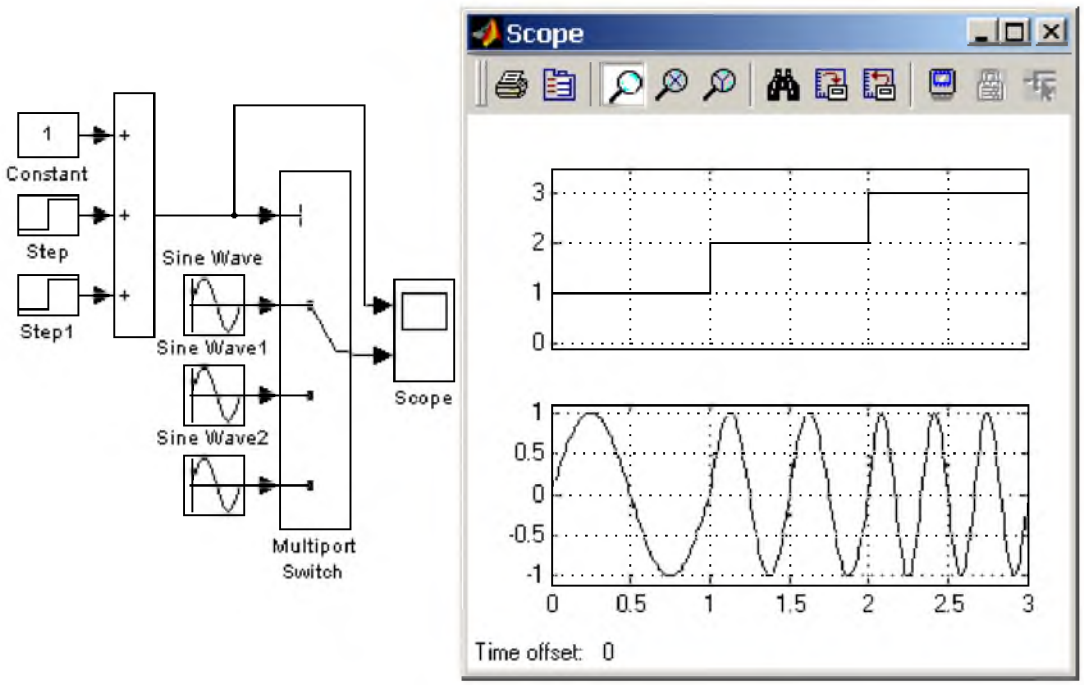


Рис. 9.5.9. Применение переключателя **Multiport Switch.**

Количество входов блока **Multiport Switch** можно задать равным **1**. В этом случае на вход блока необходимо подать векторный сигнал, а сам блок будет пропускать на выход тот элемент вектора, номер которого совпадает с уровнем управляющего сигнала.

На рис. 9.5.10 показан пример использования блока **Multiport Switch** при векторном сигнале. Временные диаграммы работы для данного примера совпадают с рассмотренными в предыдущем примере.

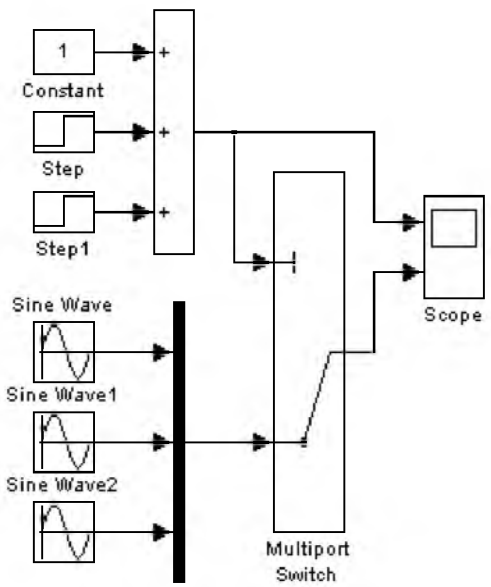


Рис. 9.5.10. Применение переключателя **Multiport Switch** при векторном входном сигнале.

9.5.10. Блок ручного переключателя Manual Switch

Назначение:

Выполняет переключение входных сигналов по команде пользователя.

Параметры:

Нет.

Командой на переключение является двойной щелчок левой клавишей “мыши” на изображении блока. При этом изображение блока изменяется, показывая, какой входной сигнал в данный момент проходит на выход блока. Переключение блока можно выполнять как до начала моделирования, так и в процессе расчета.

На рис. 9.5.11 показан пример использования блока **Manual Switch**.

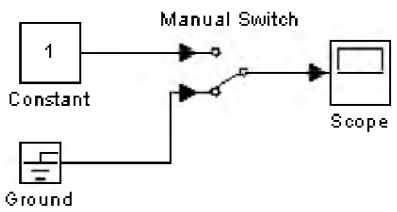


Рис. 9.5.11. Пример использования блока **Manual Switch.**

9.6. Math – блоки математических операций

9.6.1. Блок вычисления модуля Abs

Назначение:

Выполняет вычисление абсолютного значения величины сигнала.

Параметры:

- **Saturate on integer overflow** (флажок) – Подавлять переполнение целого. При установленном флажке ограничение сигналов целого типа выполняется корректно.

Пример использования блока **Abs**, вычисляющего модуль текущего значения синусоидального сигнала, показан на рис. 9.6.1.

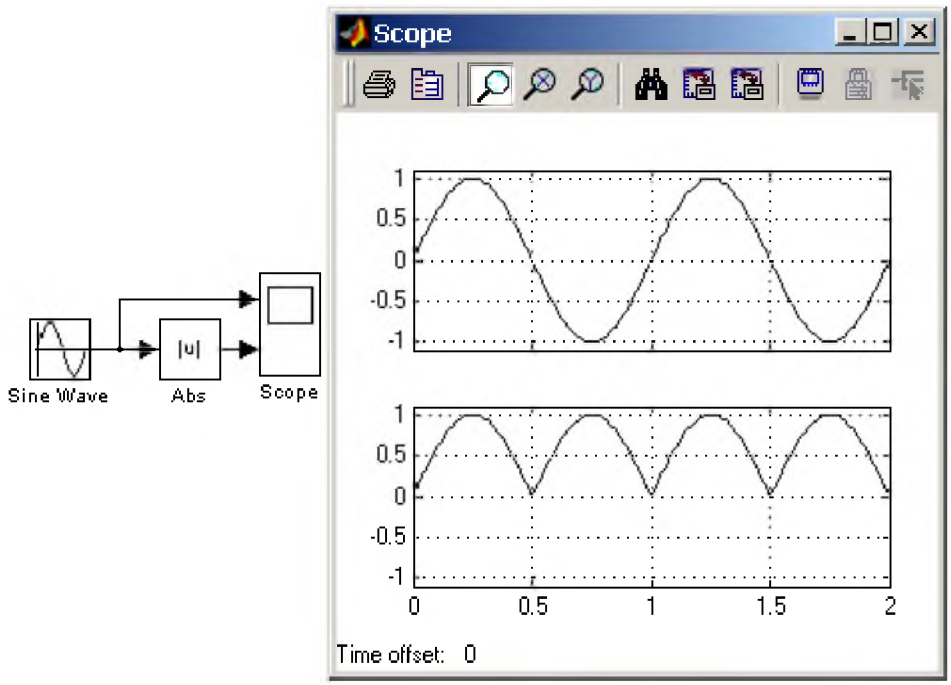


Рис. 9.6.1. Пример использования блока **Abs**

Блок **Abs** может использоваться также для вычисления модуля сигнала комплексного типа. На рис. 9.6.2 показан пример вычисления модуля комплексного сигнала вида:

$$u = \cos(\omega \cdot t) + i \cdot \sin(\omega \cdot t)$$

Модуль этого сигнала (как и следовало ожидать) равен **1** для любого момента времени.

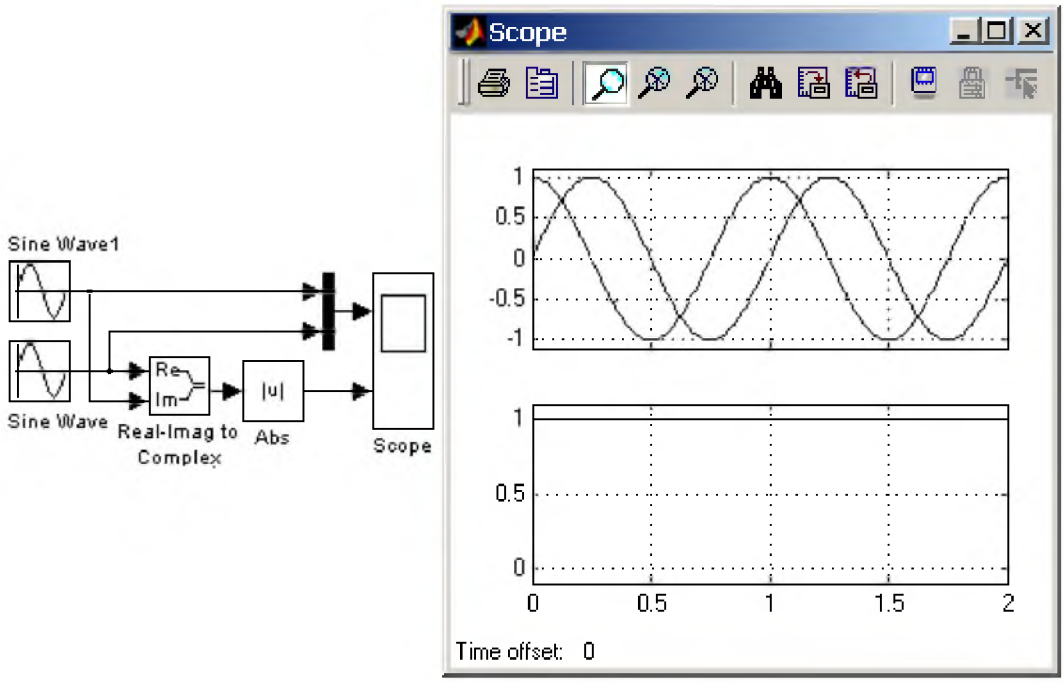


Рис. 9.6.2. Пример использования блока **Abs** для вычисления модуля комплексного сигнала

9.6.2. Блок вычисления суммы Sum

Назначение:

Выполняет вычисление суммы текущих значений сигналов.

Параметры:

1. **Icon shape** – Форма блока. Выбирается из списка.
   round – окружность,
   rectangular – прямоугольник.
2. **List of sign** – Список знаков. В списке можно использовать следующие знаки:
   + (плюс), (минус) и | (разделитель знаков).
3. **Saturate on integer overflow** (флажок) – Подавлять переполнение целого. При установленном флажке ограничение сигналов целого типа выполняется корректно.

Количество входов и операция (сложение или вычитание) определяется списком знаков параметра **List of sign**, при этом метки входов обозначаются соответствующими знаками. В параметре **List of sign** можно также указать число входов блока. В этом случае все входы будут суммирующими.

Если количество входов блока превышает **3**, то удобнее использовать блок **Sum** прямоугольной формы.

Блок может использоваться для суммирования скалярных, векторных или матричных сигналов. Типы суммируемых сигналов должны совпадать. Нельзя, например, подать на один и тот же суммирующий блок сигналы целого и действительного типов.

Если количество входов блока больше, чем один, то блок выполняет поэлементные операции над векторными и матричными сигналами. При этом количество элементов в матрице или векторе должно быть одинаковым.

Если в качестве списка знаков указать цифру **1** (один вход), то блок можно использовать для определения суммы элементов вектора.

Примеры использования блока **Sum** показаны на 9.6.3.

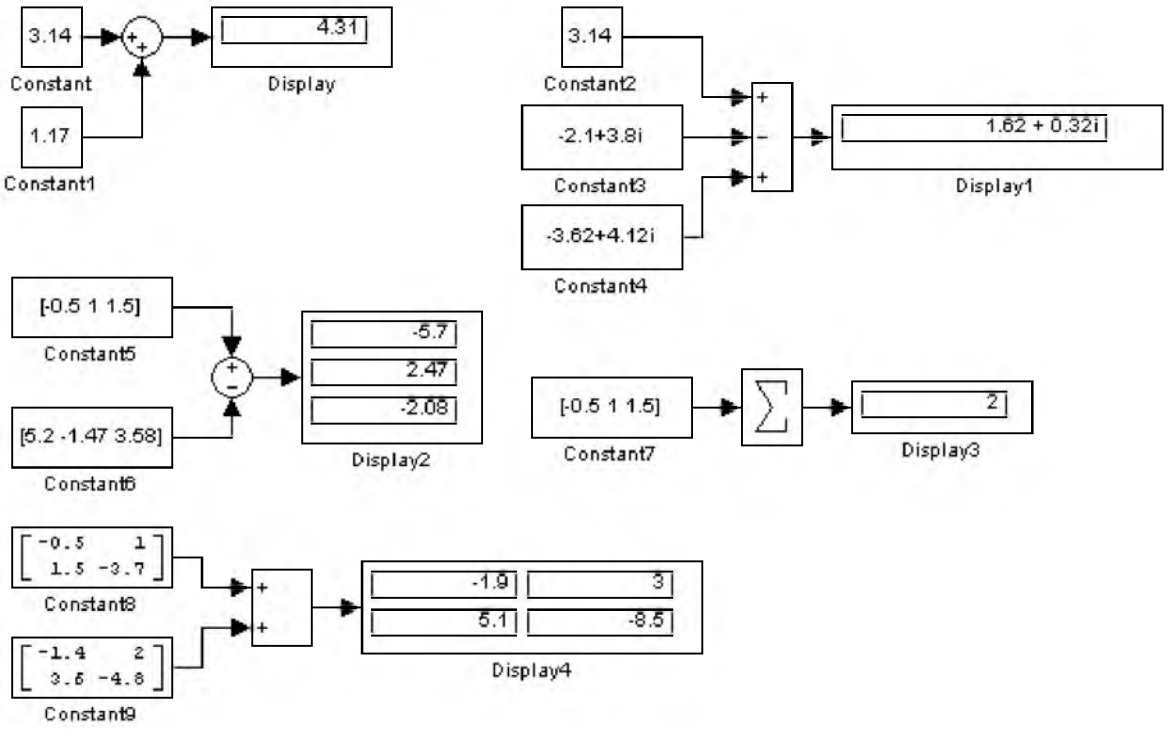


Рис. 9.6.3. Примеры использования блока **Sum**

9.6.3. Блок умножения Product

Назначение:

Выполняет вычисление произведения текущих значений сигналов.

Параметры:

1. **Number of inputs** – Количество входов. Может задаваться как число или как список знаков. В списке знаков можно использовать знаки ***** (умножить) и / (разделить).
2. **Multiplication** – Способ выполнения операции. Может принимать значения (из списка):
   Elementwise – Поэлементный.
   Matrix – Матричный.
3. **Saturate on integer overflow** (флажок) – Подавлять переполнение целого. При установленном флажке ограничение сигналов целого типа выполняется корректно.

Если параметр **Number of inputs** задан списком, включающим кроме знаков умножения также знаки деления, то метки входов будут обозначены символами соответствующих операций.

Блок может использоваться для операций умножения или деления скалярных векторных или матричных сигналов. Типы входных сигналов блока должны совпадать. Если в качестве количества входов указать цифру **1** (один вход), то блок можно использовать для определения произведения элементов вектора.

Примеры использования блока **Product** при выполнении скалярных и поэлементных операций показаны на 9.6.4.

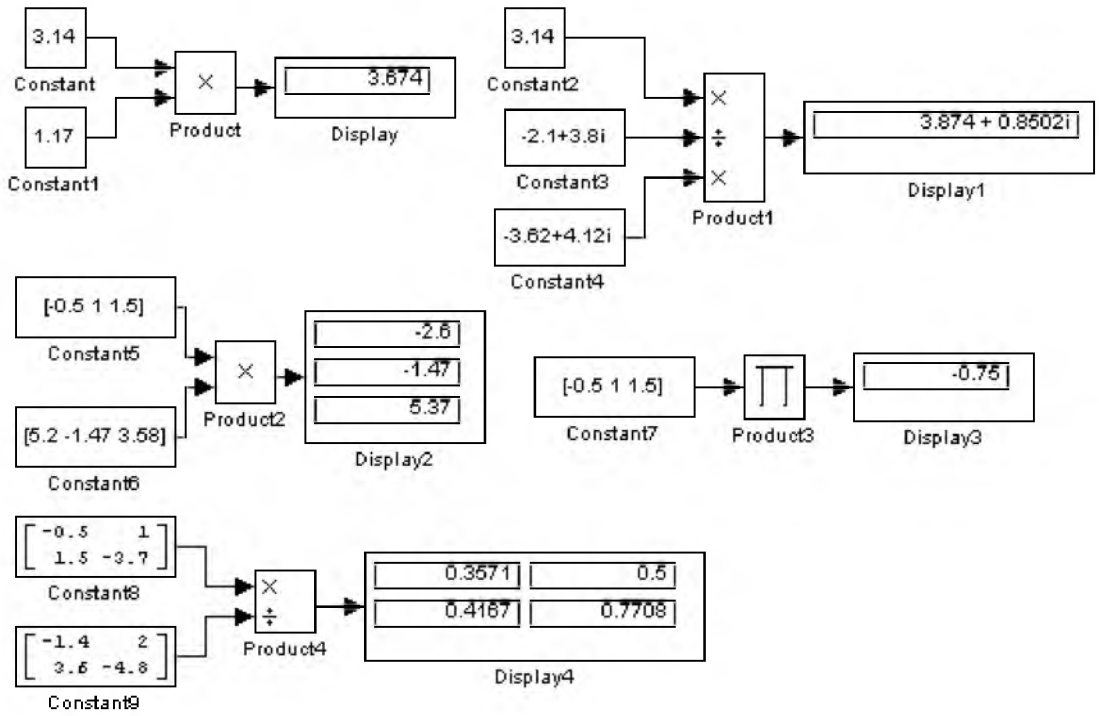


Рис. 9.6.4. Примеры использования блока **Product** при выполнении скалярных и поэлементных операций

При выполнении матричных операций необходимо соблюдать правила их выполнения. Например, при умножении двух матриц необходимо, чтобы количество строк первой матрицы равнялось количеству столбцов второй матрицы. Примеры использования блока **Product** при выполнении матричных операций показаны на рис. 9.6.5. В примере показаны операции формирования обратной матрицы, деление матриц, а также умножение матриц.

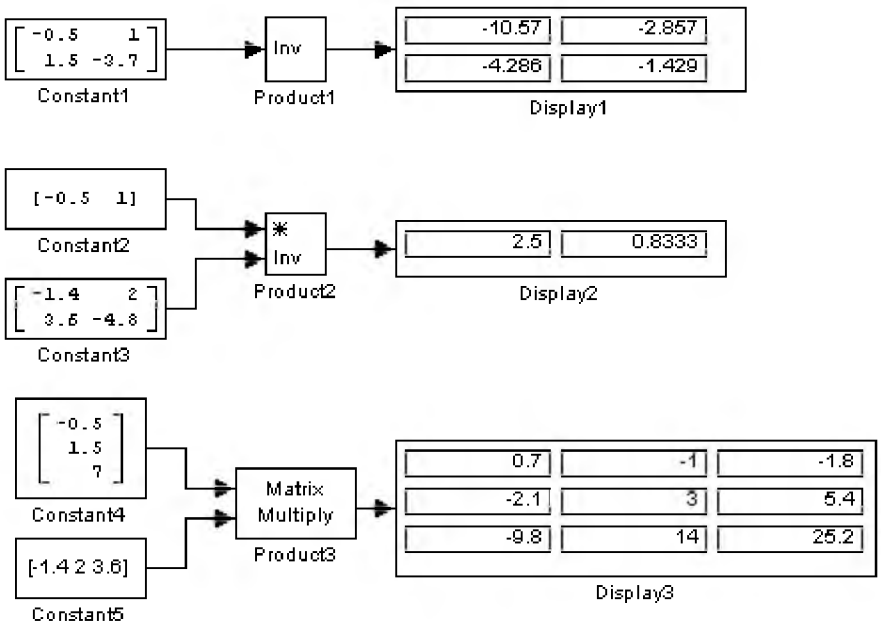


Рис. 9.6.5. Примеры использования блока **Product** при выполнении матричных операций

9.6.4. Блок определения знака сигнала Sign

Назначение:

Определяет знак входного сигнала.

Параметры:

Нет.

Блок работает в соответствии со следующим алгоритмом:

- Если входной сигнал блока положителен, то выходной сигнал равен **1**.
- Если входной сигнал блока отрицателен, то выходной сигнал равен **1**.
- Если входной сигнал блока равен **0**, то выходной сигнал также равен **0**.

Рис. 9.6.6. иллюстрирует работу блока **Sign**.

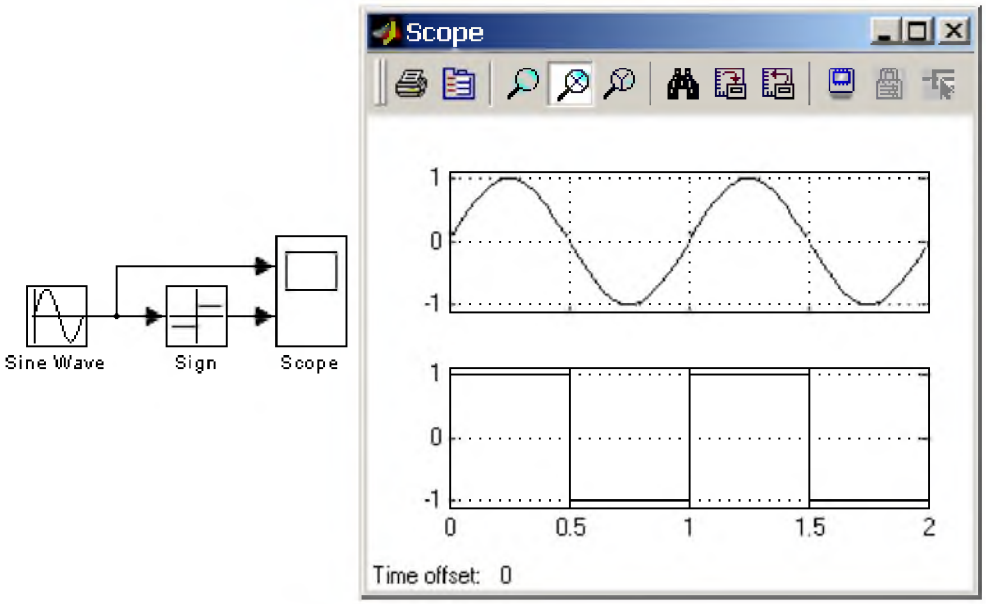


Рис. 9.6.6. Пример использования блока **Sign**

9.6.5. Усилители Gain и Matrix Gain

Назначение:

Выполняют умножение входного сигнала на постоянный коэффициент.

Параметры:

1. **Gain** – Коэффициент усиления.
2. **Multiplication –** Способ выполнения операции. Может принимать значения (из списка):
   Elementwise K*u– Поэлементный.
   Matrix K*u – Матричный. Коэффициент усиления является левосторонним операндом.
   Matrix u*K – Матричный. Коэффициент усиления является правосторонним операндом.
3. **Saturate on integer overflow** (флажок) – Подавлять переполнение целого. При установленном флажке ограничение сигналов целого типа выполняется корректно.

Блоки усилителей **Gain** и **Matrix Gain** есть один и тот же блок, но с разными начальными установками параметра **Multiplication**.

Параметр блока **Gain** может быть положительным или отрицательным числом, как больше, так и меньше **1**. Коэффициент усиления можно задавать в виде скаляра, матрицы или вектора, а также в виде вычисляемого выражения.

В том случае если парметр **Multiplication** задан как **Elementwise K*u**, то блок выполняет операцию умножения на заданный коэффициент скалярного сигнала или каждого элемента векторного сигнала. В противном случае блок выполняет операцию матричного умножения сигнала на коэффициент заданный матрицей.

По умолчанию коэффициент усиления является действительным числом типа **double**.

Для операции поэлементного усиления входной сигнал может быть скалярным, векторным или матричным любого типа, за исключением логического (**boolean**). Элементы вектора должны иметь одинаковый тип сигнала. Выходной сигнал блока будет иметь тот же самый тип, что и входной сигнал. Параметр блока **Gain** может быть скаляром, вектором или матрицей либого типа, за исключением логического (**boolean**).

При вычислении выходного сигнала блок **Gain** использует следующие правила:

- Если входной сигнал действительного типа, а коэффициент усиления комплексный, то выходной сигнал будет комплексным.
- Если тип входного сигнала отличается от типа коэффициента усиления, то **Simulink** пытается выполнить приведение типа коэффициента усиления к типу входного сигнала. В том случае, если такое приведение невозможно, то расчет будет остановлен с выводом сообщения об ошибке. Такая ситуация может возникнуть, например, если входной сигнал есть беззнаковое целое (**uint8**), а параметр **Gain** задан отрицательным числом.

Примеры использования блока **Gain** при выполнении скалярных и поэлементных операций показаны на 9.6.7.

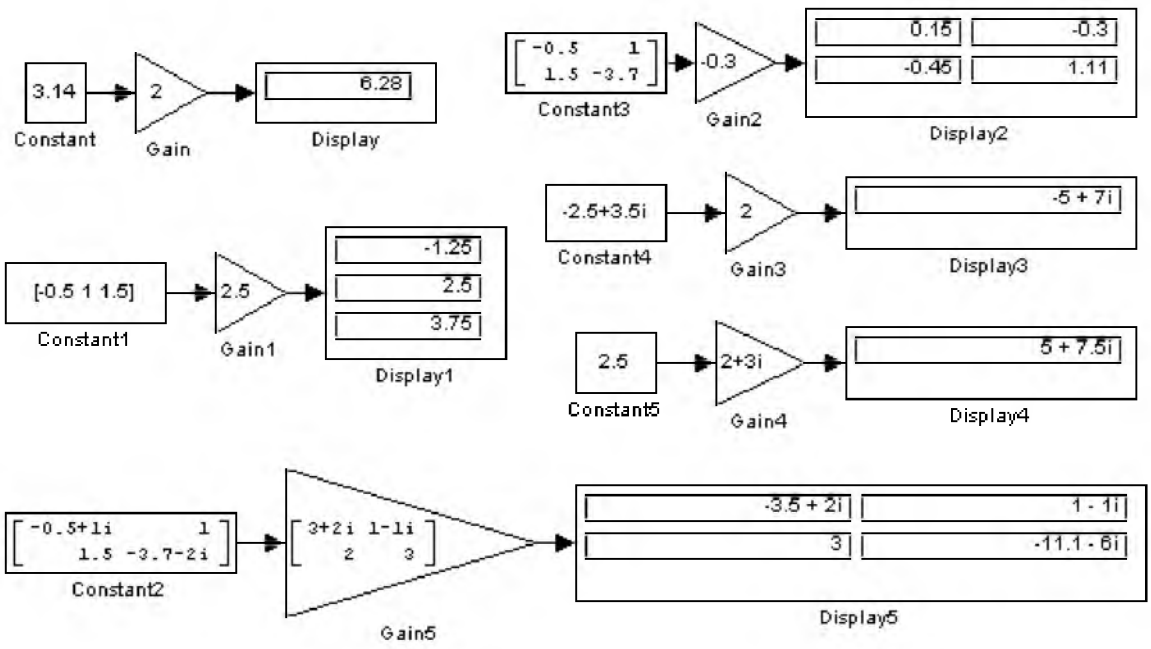


Рис. 9.6.7. Примеры использования блока **Gain.**

Для операций матричного усиления (матричного умножения входного сигнала на заданный коэффициент) входной сигнал и коэффициент усиления должны быть скалярными, векторными или матричными значениями комплексного или действительного типа **single** или **double**.

Примеры использования блока **Matrix Gain** при выполнении матричных операций показаны на рис. 9.6.8.

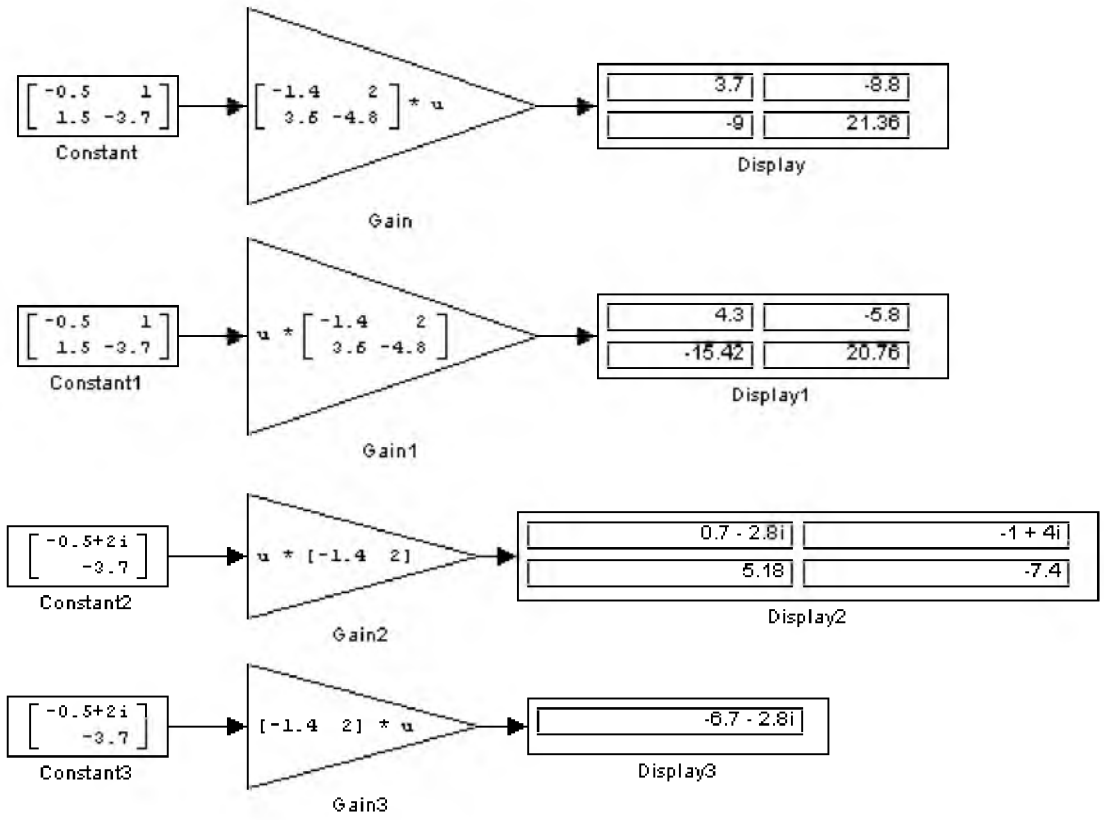


Рис. 9.6.8. Примеры использования блока **Matrix Gain**

9.6.6. Ползунковый регулятор Slider Gain

Назначение:

Обеспечивает изменение коэффициента усиления в процессе расчета.

Параметры:

1. **Low** – Нижний предел коэффициента усиления.
2. **High** – Верхний предел коэффициента усиления.

Для изменения коэффициента усиления блока **Slider Gain** необходимо передвинуть ползунок регулятора. Перемещение ползунка вправо приведет к увеличению коэффициента усиления, перемещение влево – к уменьшению. Изменение коэффициента усиления будет выполняться в пределах диапазона заданного параметрами **Low** и **High.**

Если щелкнуть с помощью мыши на левой или правой стрелках шкалы регулятора, то коэффициент усиления изменится на **1%** от установленного диапазона. Если щелкнуть с помощью мыши на самой шкале регулятора слева или справа от ползунка, то коэффициент усиления изменится на **10%** от установленного диапазона. Можно также просто задать требуемое значение коэффициента в среднем окне блока.

Блок может выполнять поэлементное усиление векторного или матричного сигнала. Входной сигнал может быть комплексным.

Примеры использования блока **Slider Gain** показаны на рис. 9.6.9.

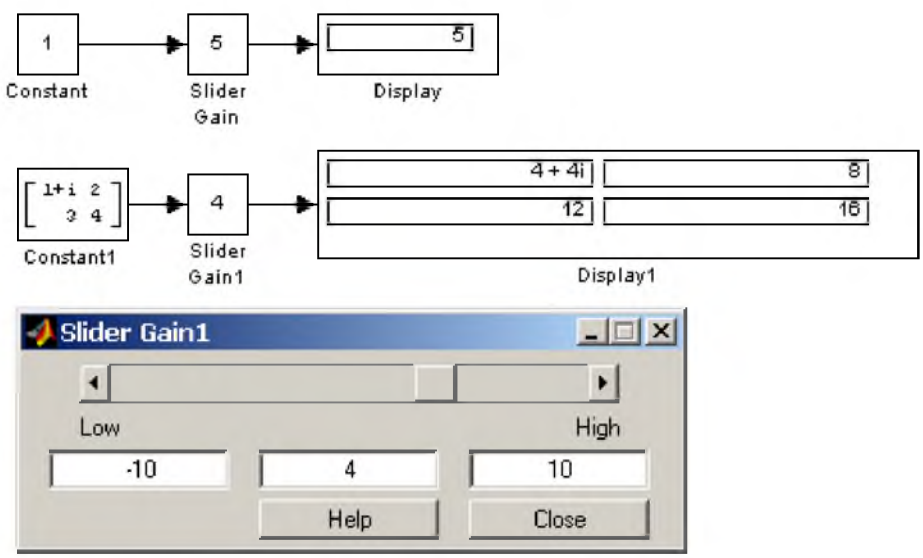


Рис. 9.6.9. Примеры использования блока **Slider Gain**

9.6.7. Блок скалярного умножения Dot Product

Назначение:

Выполняет вычисление скалярного произведения (свертку) двух векторов.

Параметры:

Нет.

Блок выполняет вычисление выходного сигнала в соответствии с выражением:

```
y = sum(conj(u1) .* u2 )
```

где **u1** и **u2** – входные векторы,
conj – операция вычисления комплексносопряженного числа,
sum – операция вычисления суммы.

Если оба входных вектора являются действительными, то выходной сигнал также будет действительным. Если хотя бы один из входных векторов содержит комплексный сигнал, то выходной сигнал будет комплексным.

Примеры, иллюстрирующие работу блока **Dot Product**, показаны на рис. 9.6.10.

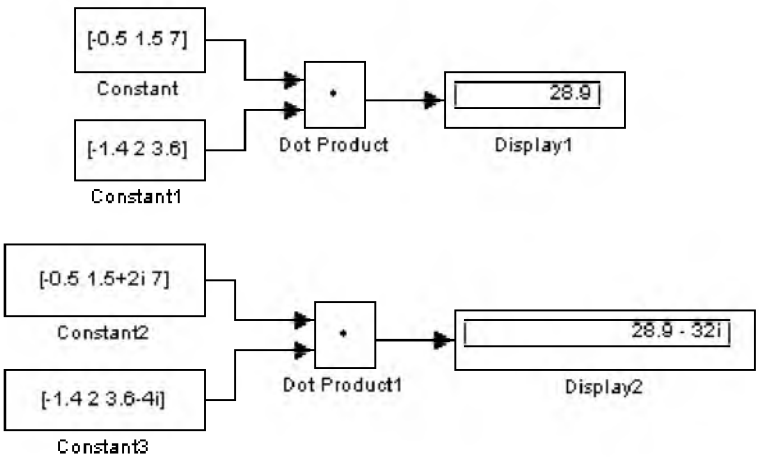


Рис. 9.6.10. Примеры использования блока **Dot Product**

9.6.8. Блок вычисления математических функций Math Function

Назначение:

Выполняет вычисление математической функции.

Параметры:

1. **Function** – Вид вычисляемой функции (выбирается из списка):
   exp – Экспоненциальная функция
   log – Функция натурального логарифма
   10^u – Вычисление степени **10**
   log10 – Функции логарифма
   magnitude^2 – Вычисление квадрата модуля входного сигнала
   square – Вычисление квадрата входного сигнала
   sqrt – Квадратный корень
   pow – Возведение в степень
   conj – Вычисление комплексносопряженного числа
   reciprocal – Вычисление частного от деления входного сигнала на **1**
   hypot –Вычисление корня квадратного из суммы квадратов входных сигналов (гипотенузы прямоугольного треугольника по значениям катетов)
   rem – Функция, вычисляющая остаток от деления первого входного сигнала на второй
   mod – Функция, вычисляющая остаток от деления с учетом знака
   transpose – Транспонирование матрицы
   hermitian – Вычисление эрмитовой матрицы.
2. **Output signal type** – Тип выходного сигнала (выбирается из списка):
   auto – Автоматическое определение типа
   real –Действительный сигнал
   complex Комплексный сигнал.

Тип выходного сигнала в зависимости от типа входного сигнала, вычисляемой функции и параметра блока **Output signal type** приведен в таблице 9.6.1.

Таблица 9.6.1.

Функция	Входной Сигнал	Выходной Сигнал		
		Auto	Real	Complex
Exp, log, 10^u, log10, square, sqrt, pow, reciprocal, conjugate, transpose, hermitian	real complex	real complex	real error	complex complex
magnitude squared	real complex	real real	real real	complex complex
hypot, rem, mod	real complex	real error	real error	complex error

Примеры использования блока **Math Function** показаны на рис. 9.6.11.

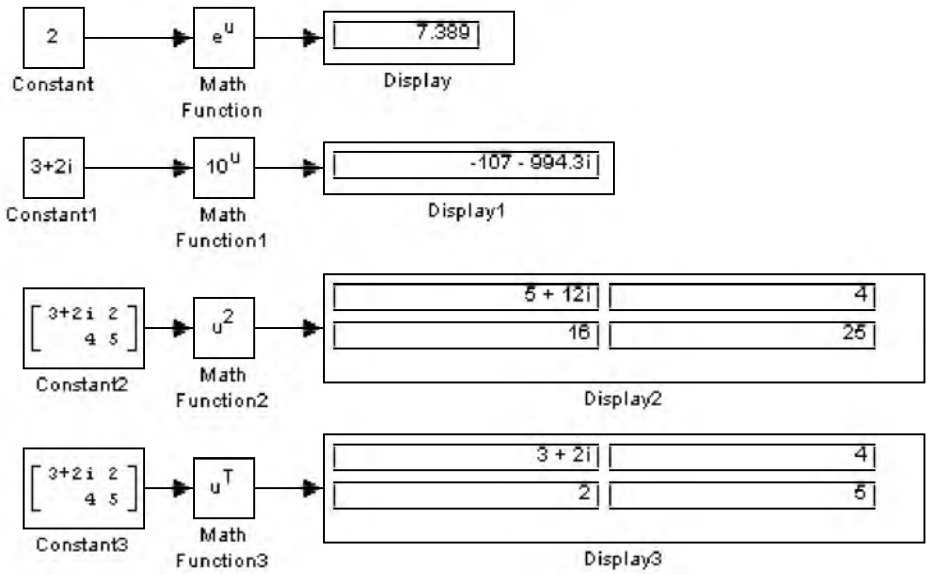


Рис. 9.6.11. Примеры использования блока **Math Function**

9.6.9. Блок вычисления тригонометрических функций Trigonometric Function

Назначение:

Выполняет вычисление тригонометрической функции.

Параметры:

1. **Function –** Вид вычисляемой функции (выбирается из списка): **sin, cos, tan, asin, acos, atan, atan2, sinh, cosh** и **tanh.**
2. **Output signal type** – Тип выходного сигнала (выбирается из списка):
   auto – Автоматическое определение типа.
   real –Действительный сигнал.
   complex Комплексный сигнал.

При векторном или матричном входном сигнале блок выполняет поэлементное вычисление заданной функции.

Примеры использования блока **Trigonometric Function** показаны на рис. 9.6.12.

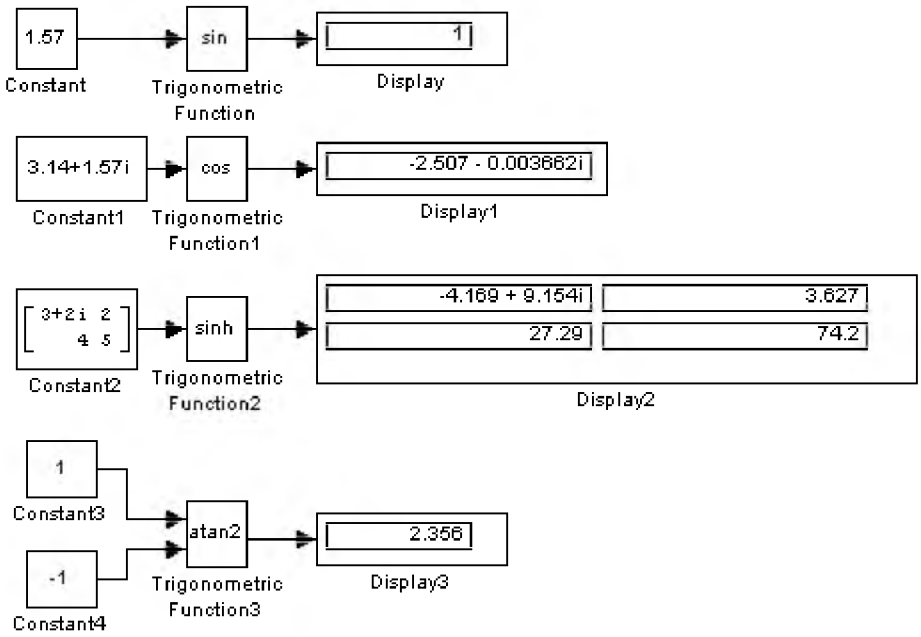


Рис. 9.6.12. Примеры использования блока **Trigonometric Function**

9.6.10. Блок вычисления действительной и (или) мнимой части комплексного числа Complex to RealImag

Назначение:

Вычисляет действительную и (или) мнимую часть комплексного числа.

Параметры:

Output – Выходной сигнал (выбирается из списка):

- **Real** – Действительная часть
- **Image** – Мнимая часть
- **RealAndImage** – Действительная и мнимая часть.

Входной сигнал блока может быть скалярным, векторным или матричным сигналом.

Примеры использования блока **Complex to RealImag** показаны на рис. 9.6.13.

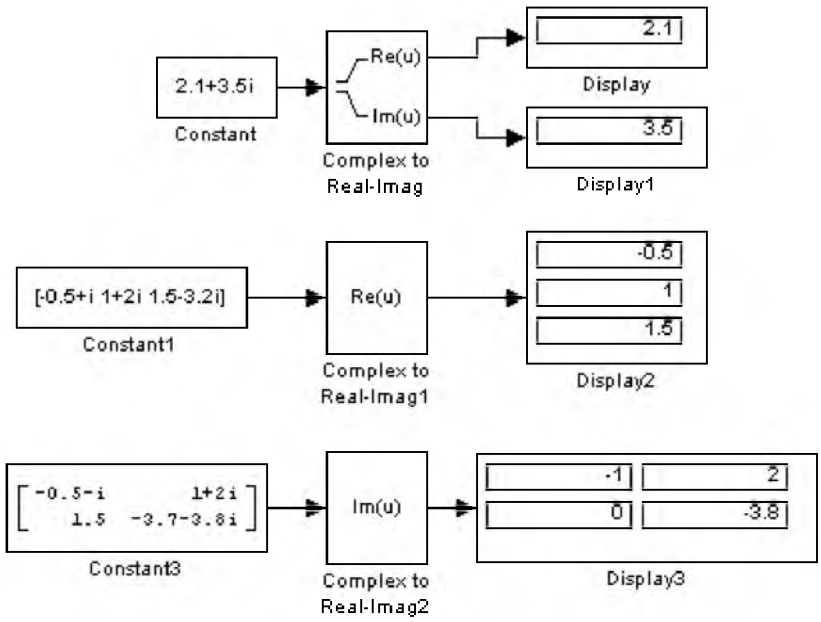


Рис. 9.6.13. Примеры использования блока **Complex to RealImag**

9.6.11. Блок вычисления модуля и (или) аргумена комплексного числа Complex to MagnitudeAngle

Назначение:

Вычисляет модуль и (или) аргумент комплексного числа.

Параметры:

Output – Выходной сигнал (выбирается из списка):

- **Magnitude** – Модуль.
- **Angle** – Аргумент.
- **MagnitudeAndAngle** – Модуль и аргумент.

Входной сигнал блока может быть скалярным, векторным или матричным сигналом.

Примеры использования блока **Complex to MagnitudeAngle** показаны на рис. 9.6.14.

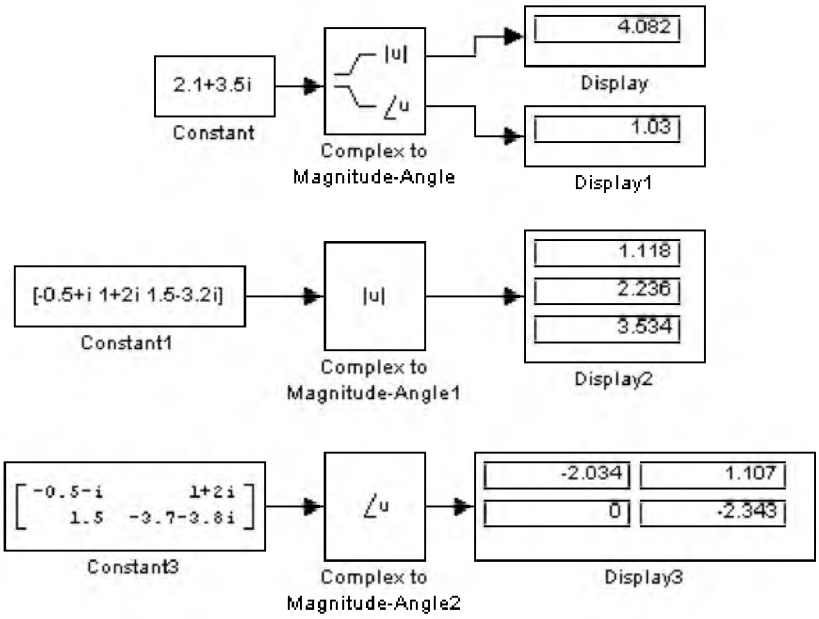


Рис. 9.6.14. Примеры использования блока **Complex to MagnitudeAngle**

9.6.12. Блок вычисления комплексного числа по его действительной и мнимой части RealImag to Complex

Назначение:

Вычисляет комплексное число по его действительной и мнимой части.

Параметры:

1. **Input** – Входной сигнал (выбирается из списка):
   Real – Действительная часть.
   Image – Мнимая часть.
   RealAndImage – Действительная и мнимая часть.
2. **Image part** – Мнимая часть. Параметр доступен, если параметр **Input** объявлен как **Real**.
3. **Real part** – Действительная часть. Параметр доступен, если параметр **Input** объявлен как **Image**.

Входные сигналы блока могут быть скалярными, векторными или матричными. Параметры **Image part** и **Real part** должны задаваться как векторы или матрицы, если входной сигнал является вектором или матрицей.

Примеры использования блока **RealImag to Complex** показаны на рис. 9.6.15.

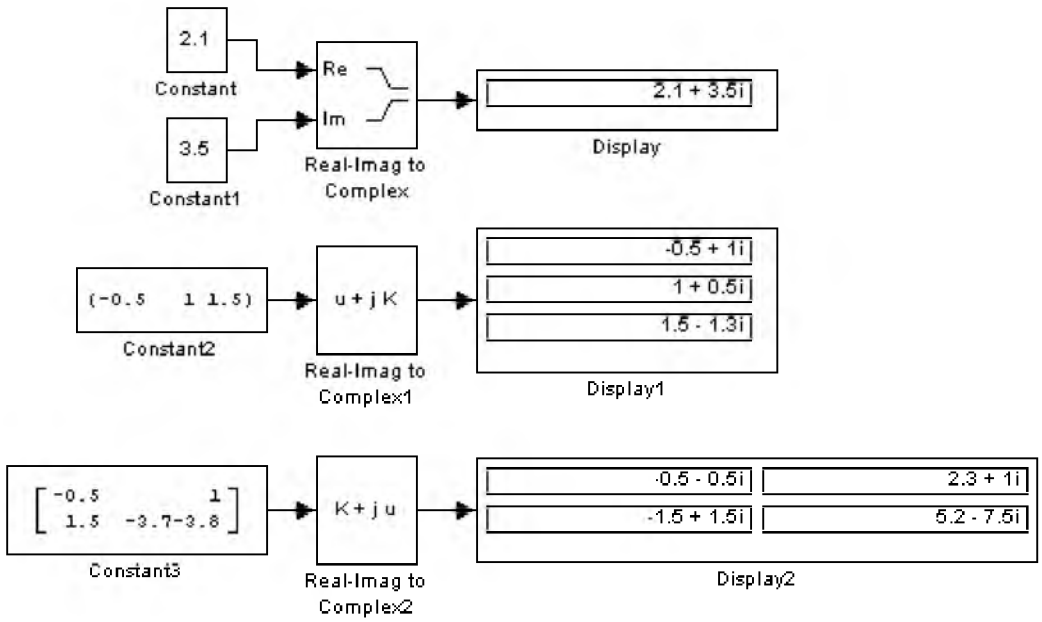


Рис. 9.6.15. Примеры использования блока **RealImag to Complex**

9.6.13. Блок вычисления комплексного числа по его модулю и аргументу MagnitudeAngle to Complex

Назначение:

Вычисляет комплексное число по его модулю и аргументу.

Параметры:

1. **Input** – Входной сигнал (выбирается из списка):
   Magnitude – Модуль.
   Angle – Аргумент.
   MagnitudeAndAngle – Модуль и аргумент.
2. **Angle** – Аргумент. Параметр доступен, если параметр **Input** объявлен как **Magnitude**.
3. **Magnitude** – Модуль. Параметр доступен, если параметр **Input** объявлен как **Angle**.

Входные сигналы блока могут быть скалярными, векторными или матричными.

Параметры **Angle** и **Magnitude** должны задаваться как векторы или матрицы, если входной сигнал является вектором или матрицей.

Примеры использования блока **MagnitudeAngle to Complex** показаны на рис. 9.6.16.

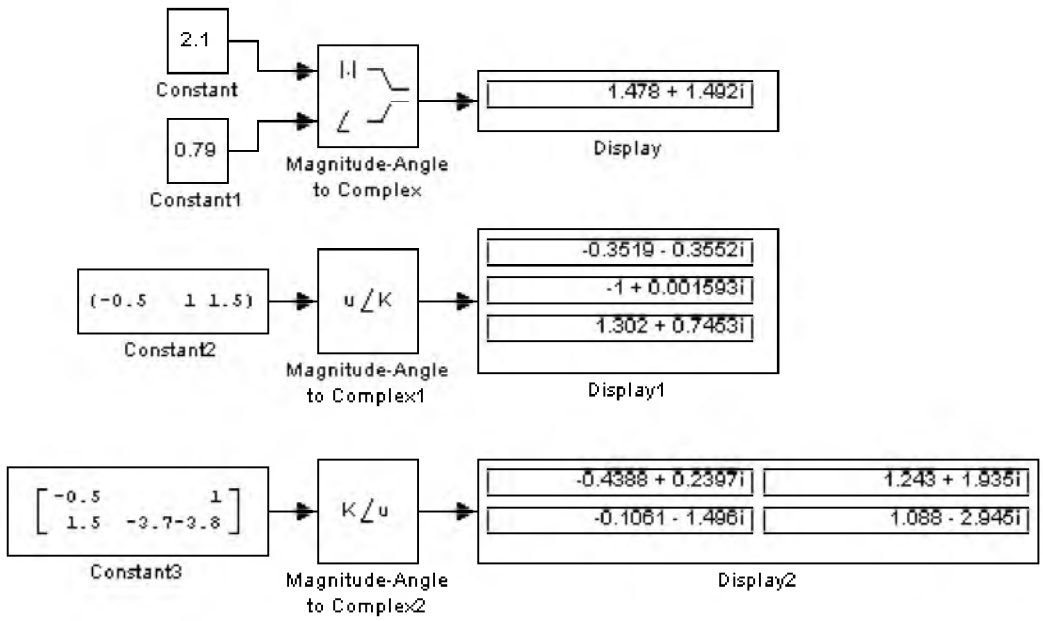


Рис. 9.6.16. Примеры использования блока **MagnitudeAngle to Complex**

9.6.14. Блок определения минимального или максимального значения MinMax

Назначение:

Определяет максимальное или минимальное значение из всех сигналов, поступающих на его входы.

Параметры:

1. **Function** Выходной параметр. Выбирается из списка:
   min – Минимальное значение.
   max – Максимальное значение.
2. **Number of input ports** – Количество входных портов.

Входные сигналы блока могут быть скалярными или векторными. Блок определяет максимальное или минимальное значение из всех скалярных сигналов, поступающих на его входы. Если входные сигналы являются векторными, то блок выполняет поэлементную операцию поиска минимального или максимального значения. В этом случае размерности векторов должны совпадать. Если количество входных портов блока задано равным **1**, то блок может использоваться для нахождения минимального или максимального значения во входном векторе.

Примеры использования блока **MinMax** показаны на рис. 9.6.17.

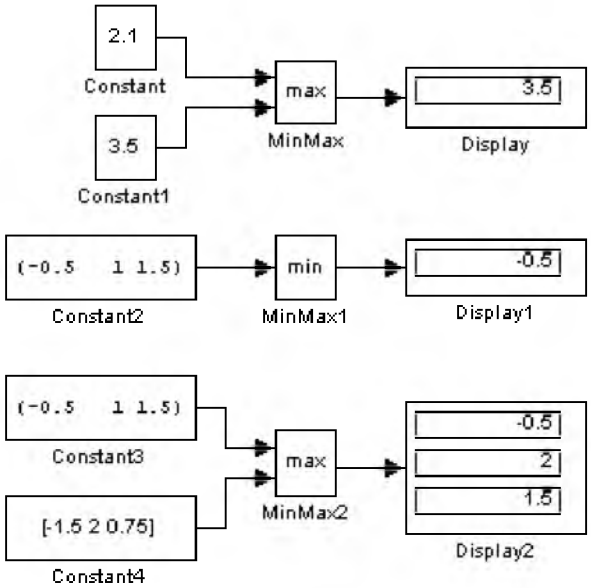


Рис. 9.6.17. Примеры использования блока **MinMax**

9.6.15. Блок округления числового значения Rounding Function

Назначение:

Выполняет операцию округления числового значения.

Параметры:

Function – Способ округления (выбирается из списка):

- **floor** – Округление до ближайшего меньшего целого.
- **ceil** – Округление до ближайшего большего целого.
- **round** – Округление до ближайшего целого.
- **fix** – Округление отбрасыванием дробной части.

Входные сигналы блока могут быть скалярными, векторными или матричными действительного и комплексного типа. При векторном или матричном входном сигнале блок выполняет поэлементные операции.

Выходной сигнал блока будет иметь тип **double** или **single**.

Примеры использования блока **Rounding Function** показаны на рис. 9.6.18.

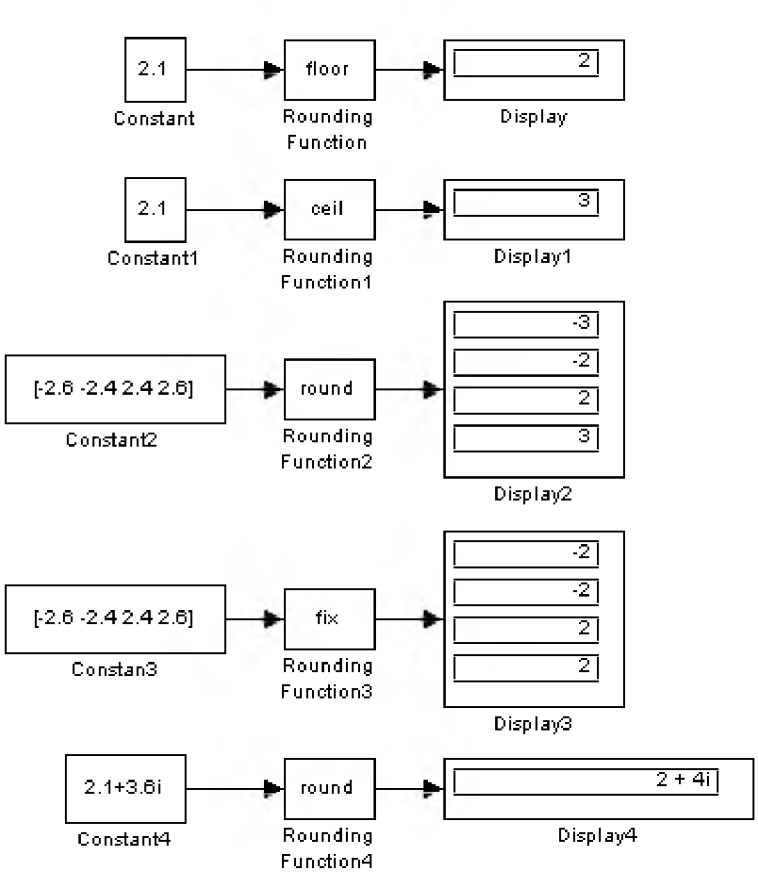


Рис. 9.6.18. Примеры использования блока **Rounding Function**

9.6.16. Блок вычисления операции отношения Relational Operator

Назначение:

Блок сравнивает текущие значения входных сигналов.

Параметры:

Relational Operator – Тип операции отношения (выбирается из списка):

- = = Тождественно равно.
- ~ = Не равно.
- < Меньше.
- < = Меньше или равно.
- > = Больше или равно.
- > Больше.

В операции отношения первым операндом является сигнал, подаваемый на первый (верхний) вход блока, а вторым операндом – сигнал, подаваемый на второй (нижний) вход. Выходным сигналом блока является **1**, если результат вычисления операции отношения есть **“ИСТИНА”** и **0,** если результат – **“ЛОЖЬ”**.

Входные сигналы блока могут быть скалярными, векторными или матричными. Если оба входных сигнала – векторы или матрицы, то блок выполняет поэлементную операцию сравнения, при этом

размерность входных сигналов должна совпадать. Если один из входных сигналов – вектор или матрица, а другой входной сигнал – скаляр, то блок выполняет сравнение скалярного входного сигнала с каждым элементом массива. Размерность выходного сигнала, в этом случае, будет определяться размерностью векторного или матричного сигнала, подаваемого на один из входов.

Для операций = = (тождественно равно) и ~ = (не равно) допускается использовать комплексные входные сигналы.

Входные сигналы также могут быть логического типа (**boolean**).

Примеры использования блока **Relational Operator** показаны на рис. 9.6.19.

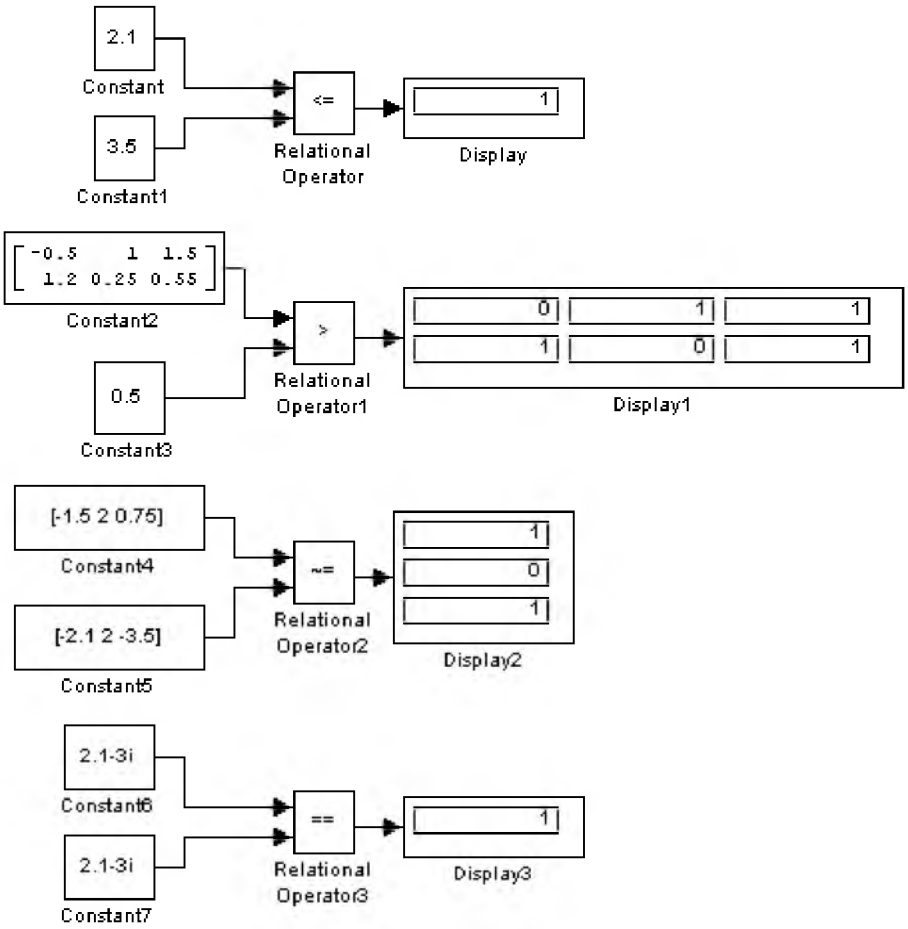


Рис. 9.6.19. Примеры использования блока **Relational Operator**

9.6.17. Блок логических операций Logical Operation

Назначение:

Реализует одну из базовых логических операций.

Параметры:

1. **Operator –** Вид реализуемой логической операции (выбирается из списка):

- **AND** – Логическое умножение (операция **И**).
- **OR** – Логическое сложение (операция **ИЛИ**).
- **NAND** – Операция **ИНЕ**.
- **NOR** – Операция **ИЛИНЕ**.
- **XOR** – Исключающее **ИЛИ** (операция сложения по модулю **2**).
- **NOT** – Логическое отрицание (**НЕ**).

2. **Number of input ports –** Количество входных портов.

Выходным сигналом блока является **1**, если результат вычисления логической операции есть **“ИСТИНА”** и **0,** если результат – **“ЛОЖЬ”**.

Входные сигналы блока могут быть скалярными, векторными или матричными. Если входные сигналы – векторы или матрицы, то блок выполняет поэлементную логическую операцию, при этом размерность входных сигналов должна совпадать. Если часть входных сигналов – векторы или матрицы, а другая часть входных сигналов – скаляры, то блок выполняет логическую операцию для скалярных входных сигналов и каждого элемента векторных или матричных сигналов. Размерность выходного сигнала, в этом случае, будет определяться размерностью векторных или матричных входных сигналов.

При выполнении логической операции отрицания блок будет иметь лишь один входной порт.

Входные сигналы могут быть как действительного, так и логического типа (**boolean**).

Примеры использования блока **Logical Operation** показаны на рис. 9.6.20.

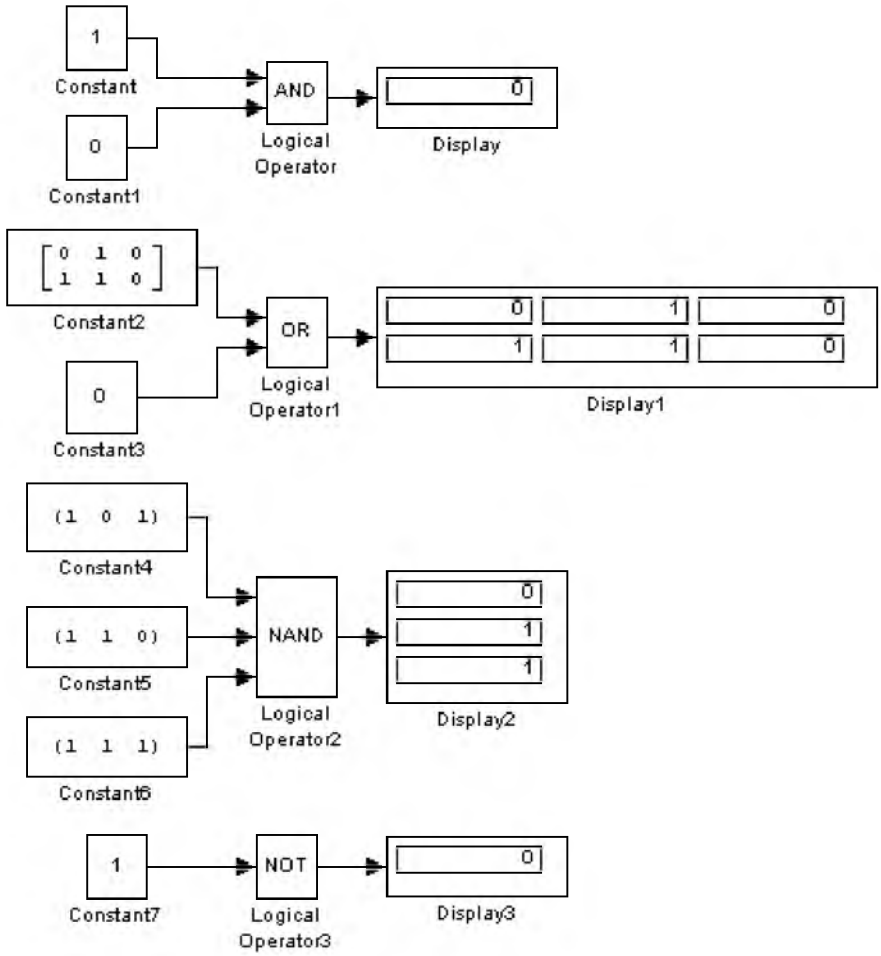


Рис. 9.6.20. Примеры использования блока **Logical Operation**

9.6.18. Блок побитовых логических операций Birwise Logical Operator

Назначение:

Реализует одну из базовых логических операций по отношению к целому числу в двоичном представлении.

Параметры:

1. **Bitwise operator** – Вид реализуемой логической операции (выбирается из списка):

- **AND** – Логическое умножение (операция **И**).
- **OR** – Логическое сложение (операция **ИЛИ**).
- **XOR** – Исключающее **ИЛИ** (операция сложения по модулю **2**).
- **NOT** – Логическое отрицание (**НЕ**).
- **SHIFT_LEFT –** Поразрядный сдвиг влево.
- **SHIFT_RIGHT –** Поразрядный сдвиг вправо.

2. **Second operand** – Второй операнд. Задается шестнадцатеричным числом в символьном виде.

Одним из операндов блока **Birwise Logical Operator** является сигнал, подаваемый на вход блока, а вторым – параметр блока **Second operand**.

Входными сигналами блока должны быть беззнаковые переменные типа **uint8, uint16** или **uint32**.

Входной сигнал блока может быть скалярным, векторным или матричным. Если входной сигнал – вектор или матрица и второй операнд также вектор или матрица, то блок выполняет поэлементную логическую операцию, при этом размерность операндов должна совпадать. Если один из операндов – вектор или матрица, а другой операнд – скаляр, то блок выполняет логическую операцию для скалярного операнда и каждого элемента векторного или матричного операнда. Размерность выходного сигнала, в этом случае, будет определяться размерностью векторного или матричного операнда.

При выполнении логической операции отрицания блок будет иметь лишь один операнд (входной сигнал).

Примеры использования блока **Birwise Logical Operator** показаны на рис. 9.6.21.

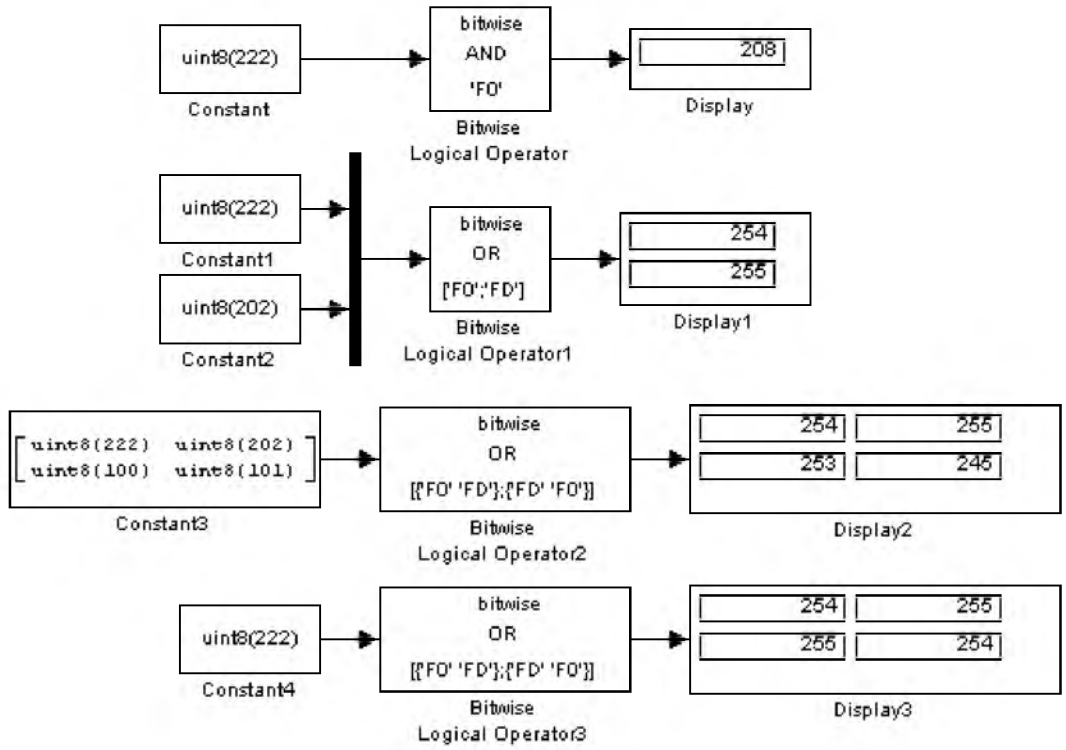


Рис. 9.6.21. Примеры использования блока **Birwise Logical Operator**

9.6.19. Блок комбинаторной логики Gombinatorical Logic

Назначение:

Преобразует входные сигналы в соответствии с таблицей истинности.

Параметры:

Truth table – Таблица истинности.

Блок **Combinatorical Logic** обеспечивает преобразование входного сигнала в соответствии с правилами, определяемыми таблицей истинности. Таблица истинности представляет собой список возможных выходных значений блока. Такое описание работы устройств принято в теории конечных автоматов. Число строк в таблице истинности определяется соотношением:

```
number of rows = 2 ^ (number of inputs)
```

где
number of inputs – число входных сигналов,
number of rows – число строк таблицы истинности.

Входные сигналы при составлении таблицы истинности считаются заданными. Они определяют индекс (номер) строки, в которой записываются выходные значения блока. Индекс каждой строки определяется выражением:

$$row\ index = 1 + u(m)*2^0 + u(m-1)*2^1 + \ldots + u(1)*2^{m-1}$$

где
row index – индекс строки,
m – количество входных сигналов (элементов во входном векторе),
u(1) – первый входной сигнал (первый элемент входного вектора),
u(m) – последний входной сигнал (последний элемент входного вектора).

Например, в случае операции логического **И** (**AND**) для двух операндов выражение, определяющее индекс строки будет выглядеть следующим образом:

$$row\ index = 1 + u(2) \cdot 2^0 + u(1) \cdot 2^1$$

Ниже приведен пример формирования таблицы истинности операции логического **И** (**AND**) для двух операндов:

Таблица 9.6.2

Вход 2	**Вход 1**	**Выражение для индекса строки**	**Значение индекса строки**	**Таблица истинности (Выход)**
0	0	$1+0\cdot 2^0+0\cdot 2^1$	1	0
1	0	$1+1\cdot 2^0+0\cdot 2^1$	2	0
0	1	$1+0\cdot 2^0+1\cdot 2^1$	3	0
1	1	$1+1\cdot 2^0+1\cdot 2^1$	4	1

На рис. 9.6.22 показан пример реализации операции логического **И** с помощью блока **Combinatorical Logic**. Параметр блока **Truth table** задан выражением **[0;0;0;1]**.

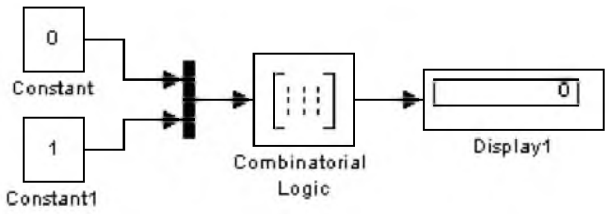


Рис. 9.6.22. Пример использования блока **Combinatorical Logic**

9.6.20. Блок алгебраического контура Algebraic Constraint

Назначение:

Выполняет поиск корней алгебраических уравнений.

Параметры:

Initial guess – Начальное значение выходного сигнала.

Блок находит такое значение выходного сигнала, при котором значение входного сигнала становится равным нулю. При этом входной сигнал должен быть прямо или опосредованно связан с входным сигналом.

На рис. 9.6.23 показан пример решения системы нелинейных уравнений вида:

$$\begin{cases} x^2 + y^2 = 6 \\ x + y = 2 \end{cases}$$

Поскольку данная система уравнений имеет два решения, то начальные значения блоков **Algebraic Constraint** заданы в виде векторов. Для первого (верхнего) блока начальное значение задано вектором **[1 1]**, а для второго (нижнего) блока – вектором **[1 1]**.

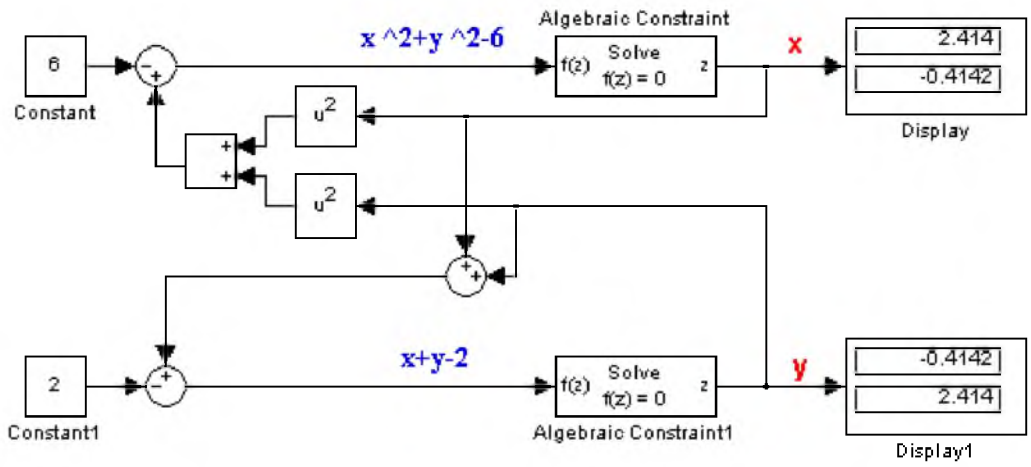


Рис. 9.6.23. Пример использования блока **Algebraic Constraint**

Блок **Algebraic Constraint** может использоваться также и для решения нелинейных матричных уравнений. На рис. 9.6.24 показан пример решения нелинейного матричного уравнения вида:

$$X^2 + 2 \cdot X + 1 = \begin{vmatrix} 13 & 4 & 4 \\ 4 & 9 & -3 \\ 4 & -3 & 57 \end{vmatrix}$$

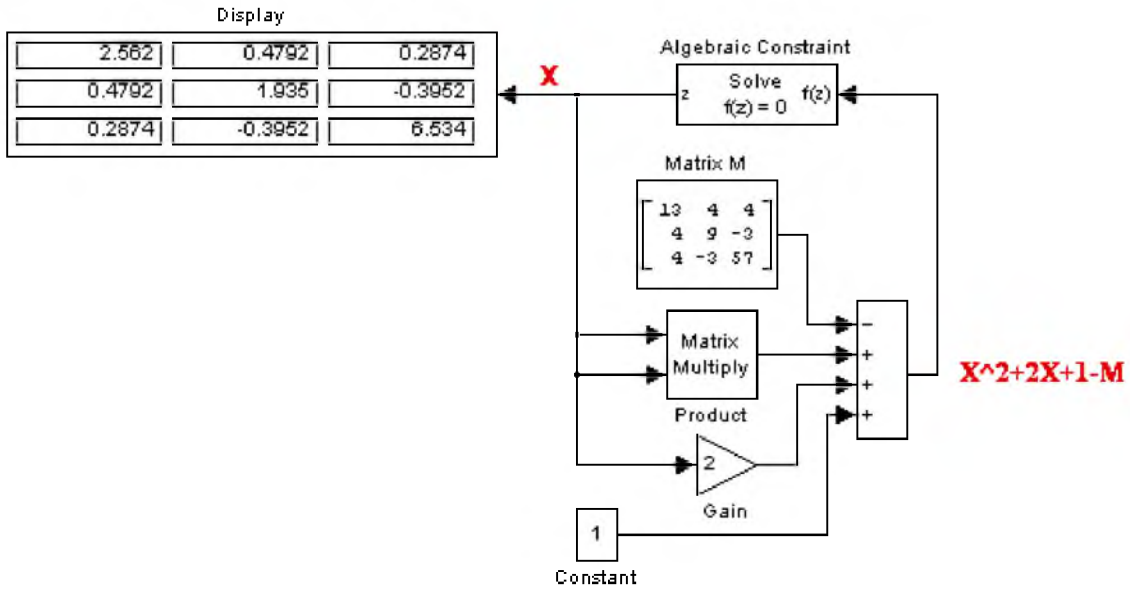


Рис. 9.6.23. Пример использования блока **Algebraic Constraint** для решения нелинейного матричного уравнения.

9.7. Signal&Systems блоки преобразования сигналов и вспомогательные блоки

9.7.1. Мультиплексор (смеситель) Mux

Назначение:

Объединяет входные сигналы в вектор.

Параметры:

1. **Number of Inputs** Количество входов.
2. **Display option** Способ отображения. Выбирается из списка:
   - **bar** Вертикальный узкий прямоугольник черного цвета.
   - **signals** Прямоугольник с белым фоном и отображением меток входных сигналов.
   - **none** Прямоугольник с белым фоном без отображения меток входных сигналов.

Входные сигналы блока могут быть скалярными и (или) векторными.

Если среди входных сигналов есть векторы, то количество входов можно задавать как вектор с указанием числа элементов каждого вектора. Например, выражение **[2 3 1]** определяет три входных

сигнала, первый сигнал вектор из двух элементов, второй сигнал вектор из трех элементов, и последний сигнал скаляр. В том случае, если размерность входного вектора не совпадает с указанной в параметре **Number of Inputs**, то после начала расчета **Simulink** выдаст сообщение об ошибке. Размерность входного вектора можно задавать как **1** (минус один). В этом случае размерность входного вектора может быть любой.

Параметр **Number of Inputs** можно задавать также в виде списка меток сигналов, например: **Vector1, Vector2, Scalar.** В этом случае метки сигналов будут отображаться рядом с соответствующими соединительными линиями.

Сигналы, подаваемые на входы блока должны быть одного типа (действительного или комплексного).

Примеры использования блока **Mux** показаны на рис. 9.7.1.

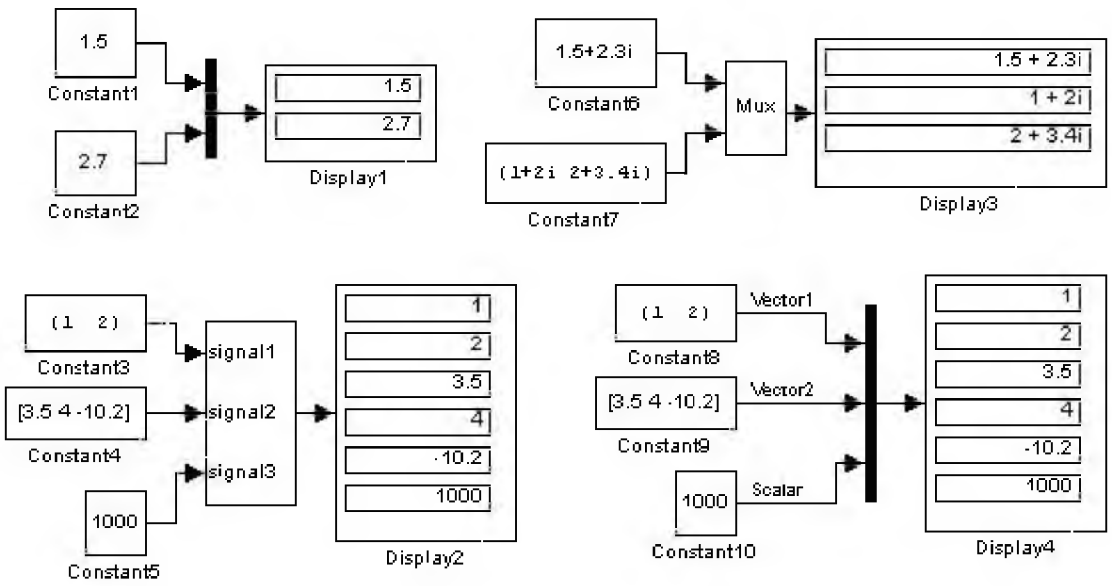


Рис. 9.7.1. Примеры использования блока **Mux**

9.7.2. Демультиплексор (разделитель) Demux

Назначение:

Разделяет входной векторный сигнал на отдельные составляющие.

Параметры:

1. **Number of Outputs** Количество выходов.
2. **Bus Selection Mode** (флажок) Режим разделения векторных сигналов.

Входным сигналами в обычном режиме является вектор, сформированный любым способом. Выходными сигналами являются скаляры или векторы, количество которых и размерность определяется параметром **Number of Outputs** и размерностью входного вектора.

Если количество выходов **P** (значение параметра **Number of Outputs**) равно размерности входного сигнала **N**, то блок выполняет разделение входного вектора на отдельные элементы.

Если количество выходов **P** меньше, чем размерность входного сигнала **N**, то размерность первых **P1** выходных сигналов равна отношению **N/P**, округленному до ближайшего большего числа, а размерность последнего выходного сигнала равна разности между размерностью входного сигнала и суммой размерностей первых **P1** выходов. Например, если размерность входного сигнала равна **8**, а количество выходов равно **3**, то первые два выходных вектора будут иметь размерность **ceil(8/3) = 3**, а последний выходной вектор будет иметь размерность **8 (3+3) = 2**.

Параметр **Number of Outputs** может быть задан также с помощью вектора, определяющего размерность каждого выходного сигнала. Например, выражение **[2 3 1]** определяет три выходных сигнала, первый сигнал вектор из двух элементов, второй сигнал вектор из трех элементов, и последний сигнал скаляр. Размерность можно также задавать как **1** (минус один). В этом случае размерность соответствующего выходного сигнала определяется как разность между размерностью входного вектора и суммой размерностей заданных выходных сигналов. Например, если размерность входного вектора равна **6**, а параметр **Number of Outputs** задан выражением [**1 1 3**], то второй выходной сигнал будет иметь размерность **6 (3+1) = 2.**

Примеры использования блока **Demux** показаны на рис. 9.7.2.

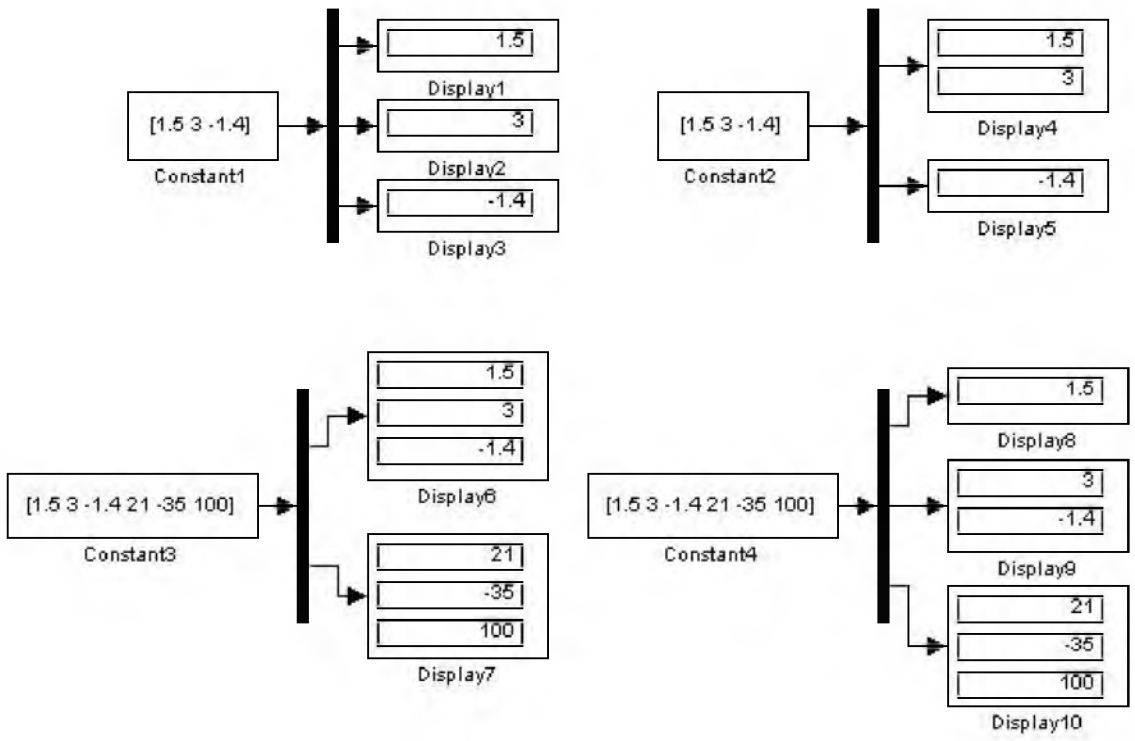


Рис. 9.7.2. Примеры использования блока **Demux**

В режиме **Bus Selection Mode** блок **Demux** работает не с отдельными элементами векторов, а с векторными сигналами в целом. Входной сигнал в этом режиме должен быть сформирован блоком **Mux** или другим блоком **Demux**. Параметр **Number of Outputs** в этом случае задается в виде скаляра, определяющего количество выходных сигналов, либо в виде вектора, каждый элемент которого определяет количество векторных сигналов в данном выходном сигнале. Например, при входном сигнале, состоящем из трех векторов параметр **Number of Outputs**, заданный вектором **[2 1]**, определит два выходных сигнала, первый из которых будет содержать два векторных сигнала, а второй один.

Примеры использования блока **Demux** в режиме **Bus Selection Mode** показаны на рис. 9.7.3.

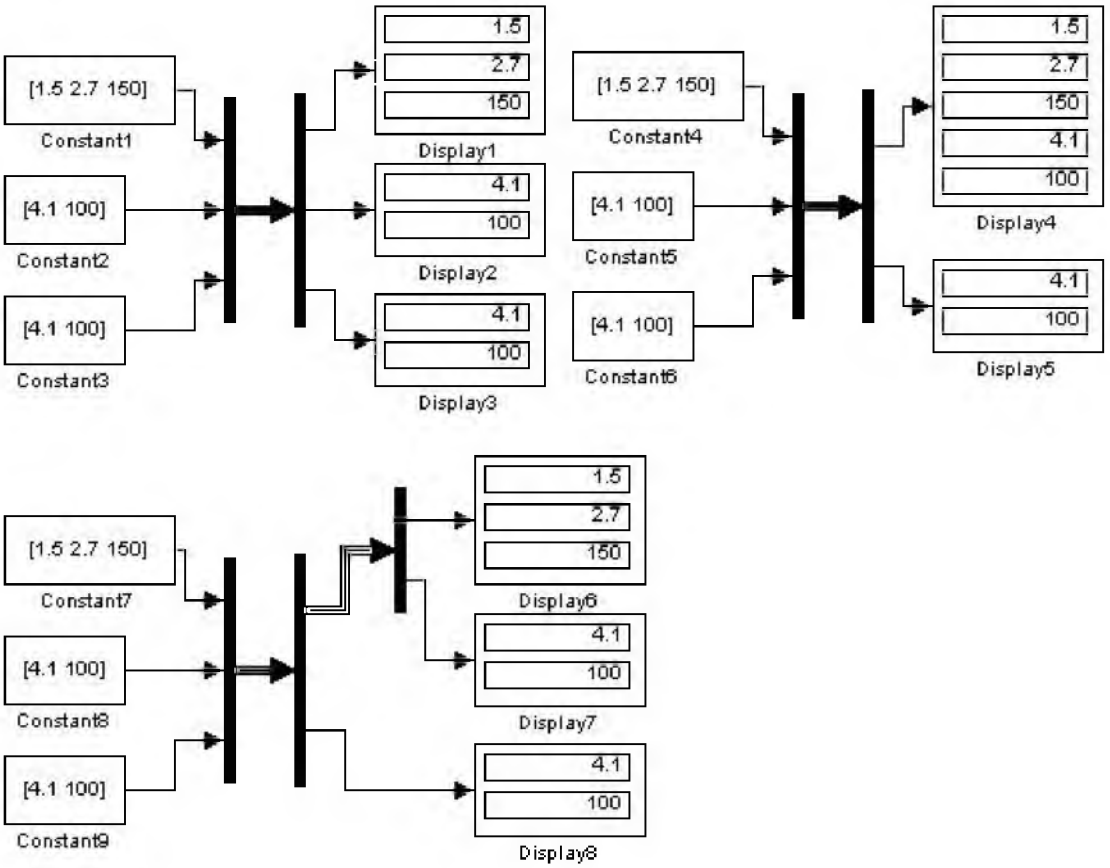


Рис. 9.7.3. Примеры использования блока **Demux** в режиме **Bus Selection Mode**

9.7.3. Блок шинного формирователя Bus Creator

Назначение:

Формирует шину из сигналов различных типов.

Параметры:

1. **Signal naming options** Способ именования сигнала. Выбирается из списка:
   - **Inherit bus signal names from input ports** Наследовать имена входных сигналов.
   - **Require input signal names to match signals below** Требуется ввести имена сигналов.
2. **Number of inputs ports** Количество входных портов.
3. **Signals in bus** Список сигналов, объединяемых в шину.
4. **Rename selected signal** Новое имя выделенного сигнала. Параметр доступен, если выбрана опция **Require input signal names to match signals below**.

Блок позволяет объединять любые сигналы (векторные, матричные, комплексные, действительные и целые разных типов) в единую шину. Такая шина позволяет сократить количество соединительных линий в модели. Для разделения шины на отдельные составляющие необходимо использовать блок **Bus Selector**.

Окно параметров блока позволяет отыскать блок, который является источником сигнала. Для такого поиска необходимо выделить название сигнала в списке **Signals in bus** и нажать с помощью мыши кнопку **Find**. Блок являющийся источником выбранного сигнала будет выделен цветом.

На рис. 9.7.4 показан пример формирования шины с помощью блока **Bus Creator** и окно параметров этого блока. Там же показан, выделенный цветом, источник сигнала **signal 2** блок **Constant3**, найденный с помощью изложенной выше процедуры.

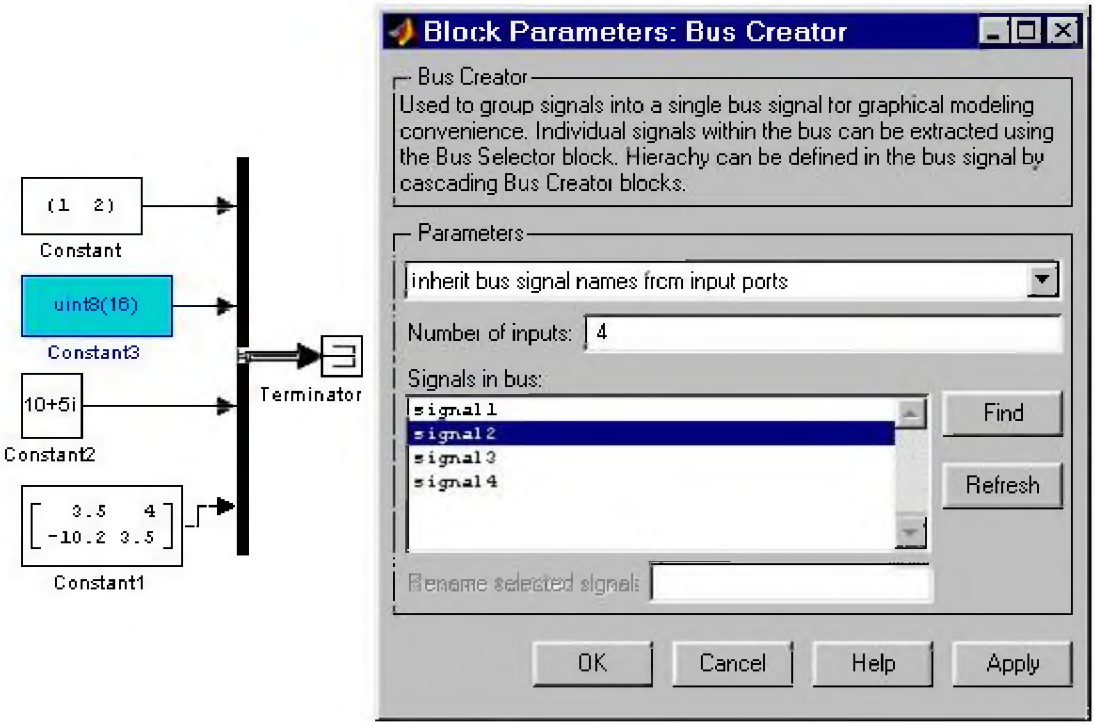


Рис. 9.7.4. Пример использования блока **Bus Creator**.

9.7.4. Блок шинного селектора Bus Selector

Назначение:

Выделяет из шины требуемые сигналы.

Параметры:

1. **Signals in the bus** Имеющиеся в шине сигналы (входные сигналы).
2. **Selected signals** Выделенные сигналы (выходные сигналы).
3. **Muxed output** (флажок) Объединение выходных сигналов в один.

Шина может быть сформирована блоком **Mux** или **Bus Creator**.

Для извлечения сигнала из шины необходимо открыть окно параметров блока, выделить сигнал в окне **Signals in the bus** и, с помощью кнопки **Select**, скопировать имя сигнала в окно **Selected signals**. Для удаления сигнала из списка **Selected signals** необходимо выделить его имя в правом списке окна параметров блока и, затем, воспользоваться кнопкой **Remove**.

С помощью кнопок **Up** и **Down** можно изменить порядок расположения сигналов в шине, перемещая их в окне **Selected signals** вверх или вниз, соответственно.

Установка параметра **Muxed output** позволяет объединить сигналы в шину.

На рис. 9.7.5 показаны примеры использования блока **Bus Selector** и окно его параметров.

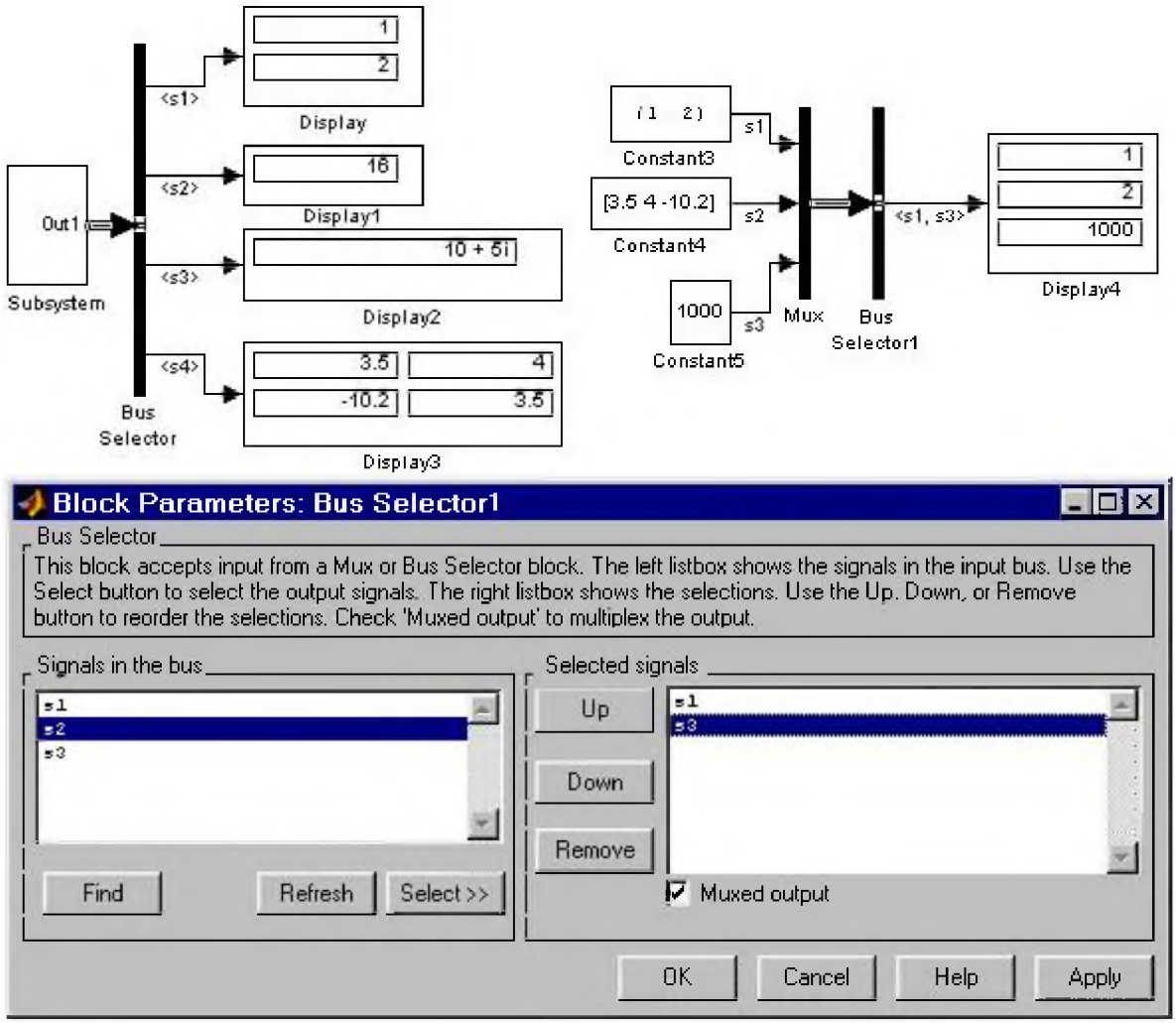


Рис. 9.7.5. Примеры использования блока **Bus Selector**.

9.7.5. Блок селектора Selector

Назначение:

Выбирает из вектора или матрицы требуемые элементы.

Параметры:

1. **Input Type** – Тип входного сигнала. Выбирается из списка:
   - **vector** – Вектор.
   - **matrix** – Матрица.

Список параметров блока изменяется в зависимости от типа входного сигнала.

2. **Source of element indices** – Источник индексов элементов вектора. Выбирается из списка:
   - **internal** – Внутренний. Индексы выбираемых элементов вектора задаются параметром **Elements**.
   - **external** – Внешний. Индексы элементов вектора задаются с помощью внешнего входного сигнала.
3. **Elements** – Список индексов элементов входного вектора, передаваемых на выход блока. Задается в виде вектора. Значение параметра **–1** (минус один) предписывает выбор всех элементов вектора.
4. **Input port width** – Размерность входного вектора.
5. **Source of row indices** – Источник индексов строк элементов матрицы.
6. **Rows** – Список индексов строк матрицы.
7. **Source of column indices** – Источник индексов столбцов элементов матрицы.
8. **Columns** – Список индексов столбцов матрицы.

Внешний вид блока изменяется в зависимости от установленных параметров блока. При выборе внешних источников индексов элементов на изображении блока появляются дополнительные входы, обозначенные следующими символами:

- **E** – Вход сигнала, задающего индексы выбираемых элементов вектора.
- **R** – Вход сигнала, задающего индексы строк матрицы.
- **C** – Вход сигнала, задающего индексы столбцов матрицы.

Блок выбирает во входном векторе или матрице и передает на выход только те сигналы, которые определены в параметрах блока или заданы внешним входным сигналом.

На рис. 9.7.6 приведены примеры использования блока **Selector** для различных вариантов настройки блока.

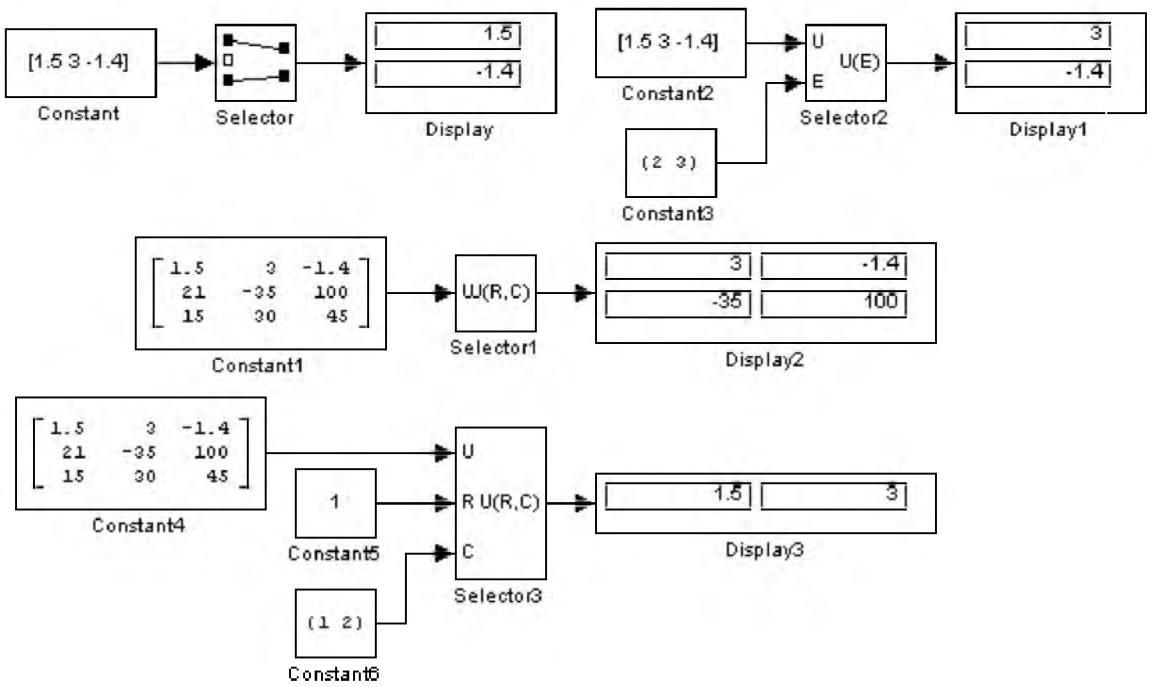


Рис. 9.7.6. Примеры использования блока **Selector**.

9.7.6. Блок присвоения новых значений элементам массива Assignment

Назначение:

Заменяет элементы вектора или матрицы.

Параметры:

1. **Input Type** – Тип входного сигнала. Выбирается из списка:
   - **vector** – Вектор.
   - **matrix** – Матрица.

   Список параметров блока изменяется в зависимости от типа входного сигнала.

2. **Source of element indices** – Источник индексов элементов вектора. Выбирается из списка:
   - **internal** – Внутренний. Индексы выбираемых элементов вектора задаются параметром **Elements**.
   - **external** – Внешний. Индексы элементов вектора задаются с помощью внешнего входного сигнала.
3. **Elements** – Список индексов элементов входного вектора, передаваемых на выход блока. Задается в виде вектора. Значение параметра **–1** (минус один) предписывает выбор всех элементов вектора.
4. **Source of row indices** – Источник индексов строк элементов матрицы.
5. **Rows** – Список индексов строк матрицы.
6. **Source of column indices** – Источник индексов столбцов элементов матрицы.
7. **Columns** – Список индексов столбцов матрицы.

Блок выполняет замену отдельных элементов первого входного массива на элементы второго входного массива в соответствии со списком индексов. Список индексов может задаваться как параметр блока или считываться из внешнего управляющего сигнала.

Внешний вид блока изменяется в зависимости от установленных параметров блока. При выборе внешних источников индексов элементов на изображении блока появляются дополнительные входы, обозначенные следующими символами:

- **E** – Вход сигнала, задающего индексы выбираемых элементов вектора.
- **R** – Вход сигнала, задающего индексы строк матрицы.
- **C** – Вход сигнала, задающего индексы столбцов матрицы.

На рис. 9.7.7 приведены примеры использования блока **Assignment** для различных вариантов настройки блока.

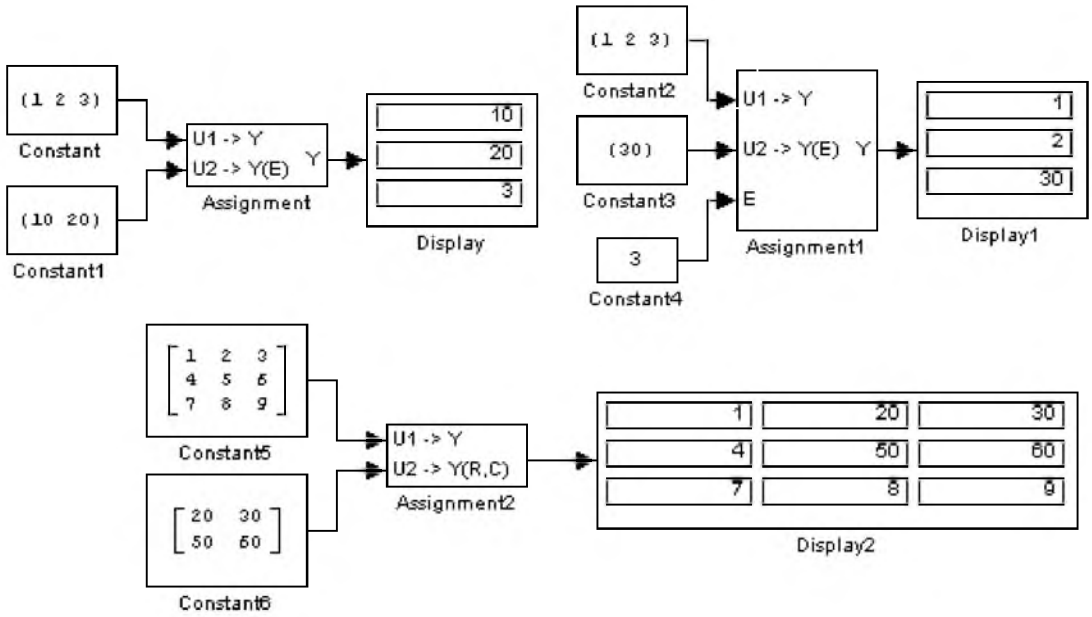


Рис. 9.7.7. Примеры использования блока **Assignment**.

9.7.7. Блок объединения сигналов Merge

Назначение:

Блок выполняет объединение входных сигналов в единый векторный сигнал.

Параметры:

1. **Number of inputs** – Количество входов.
2. **Initial output** – Начальное значение выходного сигнала. Если этот параметр не задан, то на выход блока проходит сигнал, значение которого было вычислено последним.
3. **Allow unequal port widths** (флажок) – Разрешить неодинаковую размерность входных портов.
4. **Input port offsets** – Смещение входного сигнала. Задается в виде вектора, каждое значение которого определяет расположение соответствующего сигнала в выходном векторе.

Блок передает на выход значение сигнала вычисленное последним.

С помощью параметра **Input port offsets** можно регулировать расположение входных сигналов в результирующем векторе.

Размерность выходного сигнала определяется в соответствии с выражением:

$$\max(w_1+o_1,\ w_2+o_2,\ \ldots\ w_n+o_n),$$

где

$\mathbf{w_k}$ – размерность **k**го входного сигнала,
$\mathbf{o_k}$ – смещение **k**го входного сигнала.

На рис. 9.7.8 приведен пример использования блока **Merge** для объединения двух векторов. Параметр **Input port offsets** в примере задан вектором **[0 3]**.

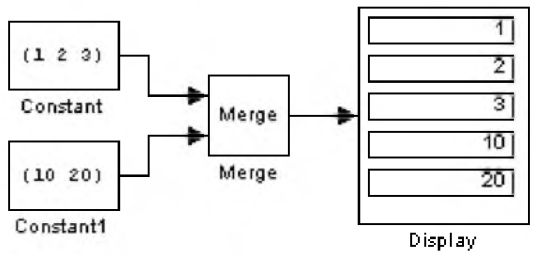


Рис. 9.7.8. Пример использования блока **Merge** для объединения входных сигналов.

Следующий пример (рис. 9.7.9) демонстрирует свойство блока пропускать на выход сигнал, который был вычислен последним. В примере использованы блоки управляемых подсистем **Enabled Subsystem**, которые выполняют вычисления только в том случае, если на управляющий вход подсистемы подан не нулевой сигнал. В данном примере подсистема не выполняет какиелибо вычисления, а лишь пропускает сигнал со своего входа на выход. Таким образом, на выход блока **Merge** поочередно проходят гармонический либо пилообразный сигналы.

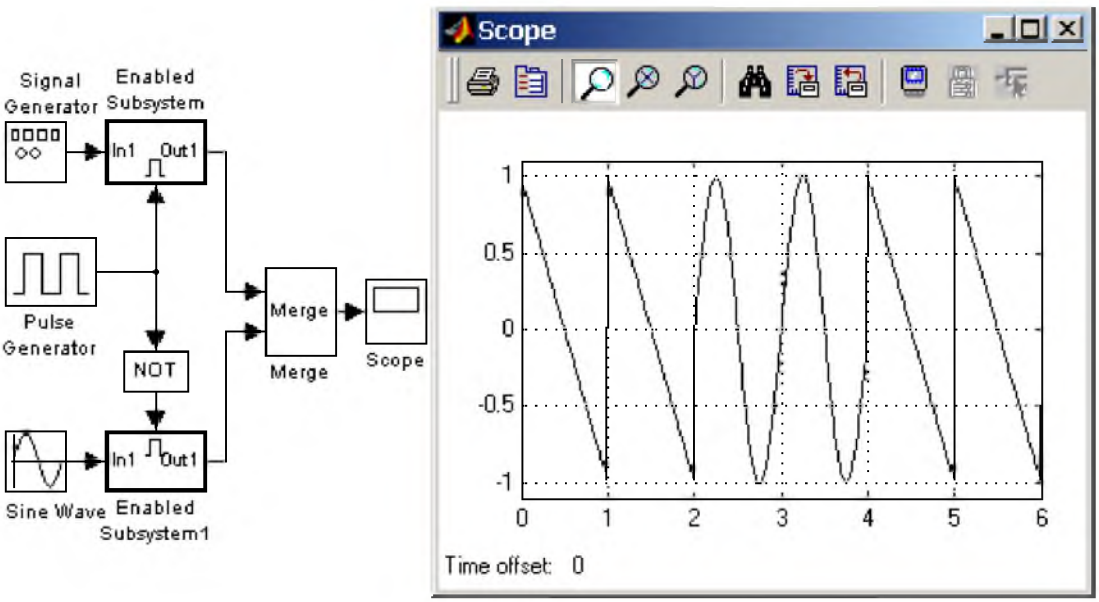


Рис. 9.7.9. Пример использования блока **Merge**

9.7.8. Блок объединения сигналов в матрицу Matrix Concatenation

Назначение:

Блок выполняет объединение (конкатенацию) входных векторов или матриц.

Параметры:

1. **Number of inputs** – Количество входов.
2. **Concatenation method** – Способ объединения. Выбирается из списка:

- **Horizontal** – Горизонтальный. Массивы объединяются добавлением новых массивов справа.
- **Vertical** – Вертикальный. Массивы объединяются добавлением новых массивов снизу.

Примеры использования блока **Matrix Concatenation** приведены на рис. 9.7.10.

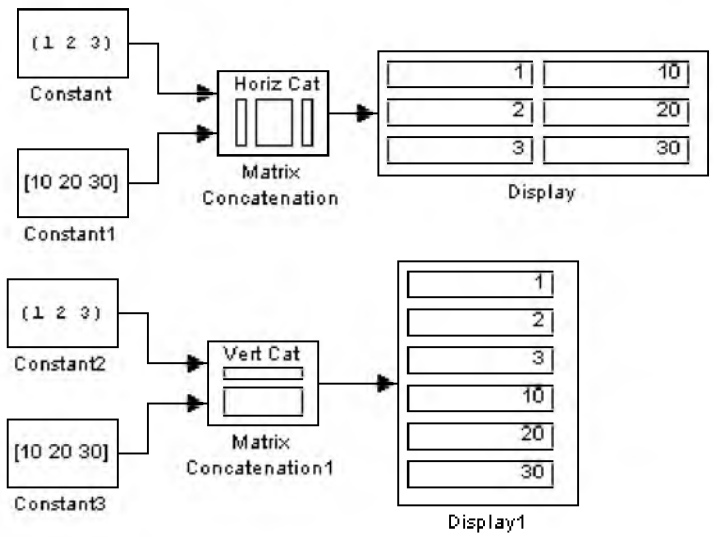


Рис. 9.7.10. Примеры использования блока **Matrix Concatenation**.

9.7.9. Блок передачи сигнала Goto

Назначение:

Блок выполняет передачу сигнала к блоку **From**.

Параметры:

1. **Tag** – Идентификатор сигнала.
2. **Tag visibility** – Признак видимости. Выбирается из списка:
   - **local** – Сигнал передается в пределах локальной подсистемы.
   - **scoped** – Сигнал передается в пределах локальной подсистемы и подсистемах нижнего уровня иерархии.
   - **global** – Сигнал передается в пределах всей модели.

Использование блока **Goto** совместно с блоком **From** обеспечивает передачу сигнала без линии связи. Для передачи могут использоваться сигналы любого типа.

В зависимости от выбранного параметра **Tag visibility** изменяется внешний вид блока:

- Идентификатор сигнала помещается в квадратные скобки, если признак видимости имеет значение **local**. Например, **[A],** где **A** – идентификатор сигнала.
- Идентификатор сигнала помещается в фигурные скобки, если признак видимости имеет значение **scoped**. Например, **{A}**.
- Идентификатор сигнала отображается на пиктограмме блока без дополнительных символов, если признак видимости имеет значение **global**.

На рис. 9.7.11. показан “беспроводной” способ передачи сигнала от источника синусоидального сигнала к блоку **Scope** в подсистему.

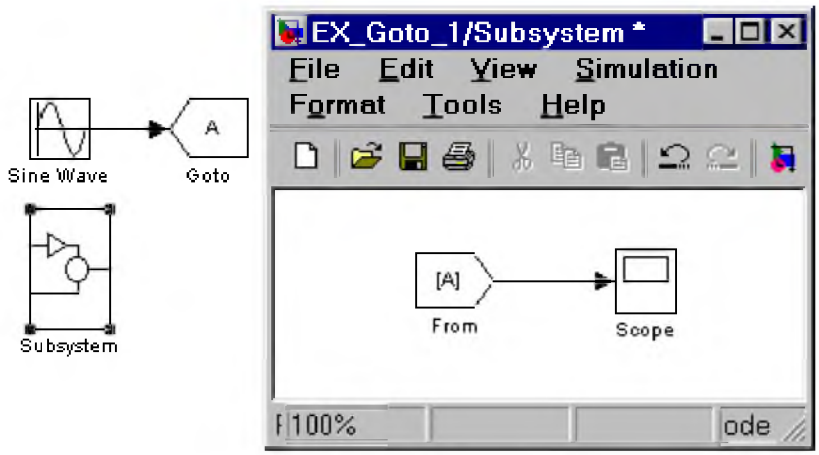


Рис. 9.7.11. Применение блока **Goto**.

9.7.10. Блок приема сигнала From

Назначение:

Блок выполняет прием сигнала от блока **Goto**.

Параметры:

Goto tag – Идентификатор принимаемого сигнала. Должен совпадать с идентификатором указанным в соответствующем блоке **Goto**.

Использование блока **From** совместно с блоком **Goto** обеспечивает передачу сигнала без линии связи.

Признак видимости сигнала отображается на пиктограмме блока таким же способом, что и у блока **Goto**.

В модели может быть сколь угодно много блоков **From**, принимающих сигнал от одного блока **Goto**.

На рис. 9.7.12. показан пример использования блоков **From** в модели. В примере один блок **Goto** передает сигнал трем блокам **From** (двум в основной модели и одному в подсистеме).

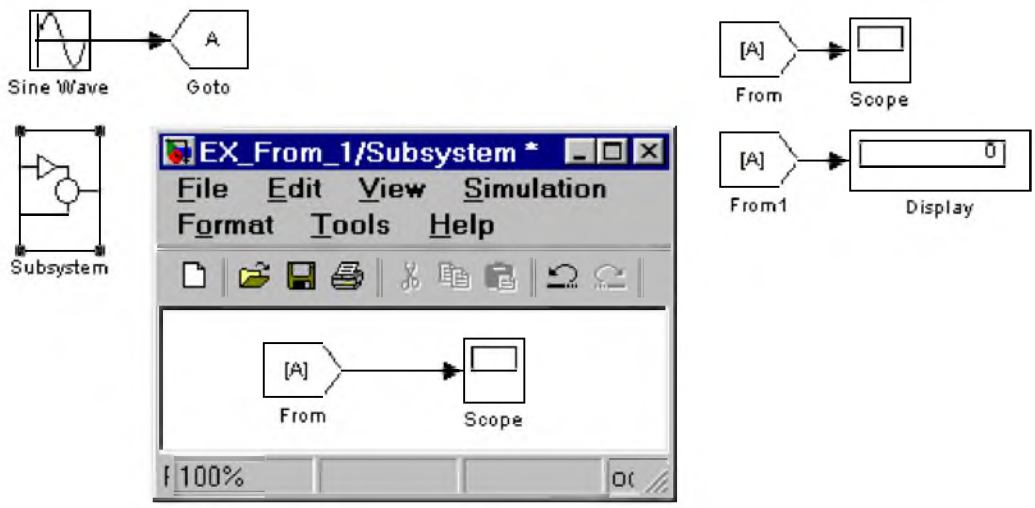


Рис. 9.7.12. Применение блока **From**.

9.7.11. Блок признака видимости сигнала Goto Tag Visibility

Назначение:

Блок отображает признак видимости сигнала передаваемого блоком **Goto**.

Параметры:

Goto tag – Идентификатор сигнала передаваемого блоком **Goto**.

Блок необходимо включать в состав модели или подсистемы в том случае, если для передаваемых сигналов задана область видимости **scoped**. Блок помещается в те подсистемы, на которые распространяется область видимости передаваемых данных. Блок не участвует в передаче сигнала, а лишь отображает имя передаваемого сигнала.

Пример использования блока показан на рис. 9.7.13.

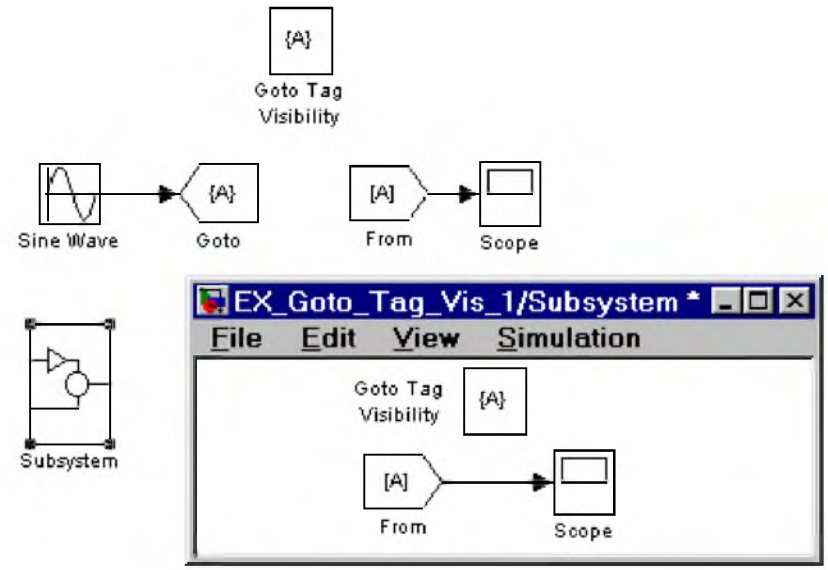


Рис. 9.7.13. Применение блока **Goto Tag Visibility**.

9.7.12. Блок создания общей области памяти Data Store Memory

Назначение:

Блок создает поименованную область памяти для хранения данных.

Параметры:

1. **Data store name** – Имя области памяти.
2. **Initial value** — Начальное значение.
3. **Interpret vector parameters as 1D** (флажок) – Интерпретировать вектор параметров данных как одномерный вектор.

Блок используется совместно с блоками **Data Store Write** (запись данных) и **Data Store Read** (считывание данных).

Параметр **Initial value** задает не только начальное значение сигнала, но и его размерность. Например, если начальное значение сигнала задано матрицей **[0 1; 2 3]**, то сохраняемый сигнал должен быть матрицей **2x2**.

Если блок **Data Store Memory** расположен в модели верхнего уровня, то заданную им область памяти можно использовать как в самой модели, так и во всех подсистемах нижнего уровня иерархии. Если блок **Data Store Memory** расположен в подсистеме, то заданную им область памяти можно использовать в данной подсистеме и всех подсистемах нижнего уровня иерархии.

Блок работает с действительными сигналами типа **double**.

Пример использования блока **Data Store Memory** совместно с блоками **Data Store Write** и **Data Store Read** показан на рис. 9.7.14 (п.9.17.14).

9.7.13. Блок записи данных в общую область памяти Data Store Write

Назначение:

Блок записывает данные в поименованную область памяти.

Параметры:

1. **Data store name** – Имя области памяти.
2. **Sample time –** Шаг модельного времени.

Операция записи выполняется для значения сигнала полученного на предыдущем шаге расчета.

В модели могут использоваться несколько блоков **Data Store Write**, выполняющих запись в одну область памяти. Однако, если, запись производится на одном и том же шаге расчета, то результат будет не предсказуем.

Пример использования блока **Data Store Write** совместно с блоками **Data Store Memory** и **Data Store Read** показан на рис. 9.7.14 (п.9.17.14).

9.7.14. Блок считывания данных из общей области памяти Data Store Read

Назначение:

Блок считывает данные из поименованной области памяти.

Параметры:

1. **Data store name** – Имя области памяти.
2. **Sample time** – Шаг модельного времени.

Операция считывания выполняется на каждом шаге расчета.

В модели могут использоваться несколько блоков **Data Store Read**, выполняющих считывание данных из одной и той же области памяти. Пример использования блока **Data Store Read** совместно с блоками **Data Store Memory** и **Data Store Write** показан на рис. 9.7.14. В примере используется триггерная подсистема, выполняющая вычисления по переднему фронту управляющего сигнала. Таким образом, запись значений в общую область памяти происходит только в моменты изменения управляющего сигнала в положительном направлении. В остальные моменты времени значения данных в области памяти не изменяются.

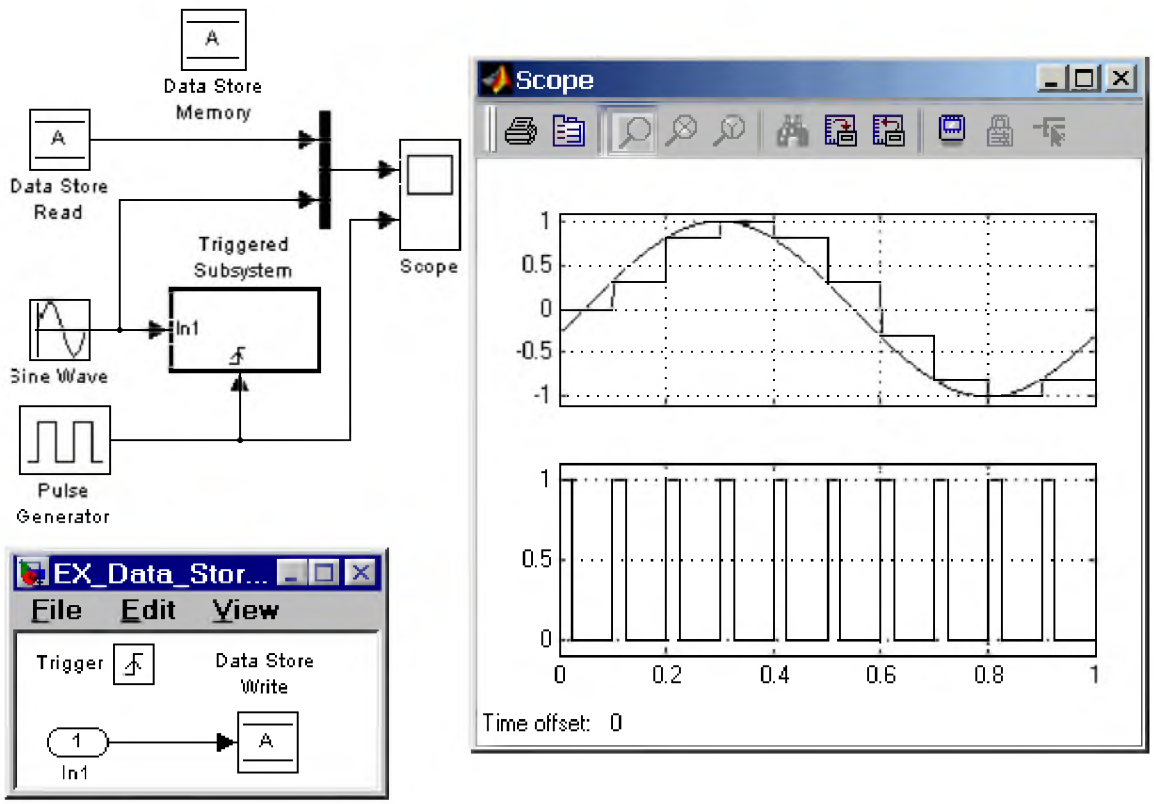


Рис. 9.7.14. Использование блоков **Data Store Memory**, **Data Store Write** и **Data Store Read**.

9.7.15. Блок преобразования типа сигнала Data Type Conversion

Назначение:

Блок преобразует тип входного сигнала.

Параметры:

1. **Data type** – Тип данных выходного сигнала. Может принимать значения (выбираются из списка): **auto**, **double**, **single**, **int8**, **int16**, **int32**, **uint8**, **uint16**, **uint32** и **boolean**.
2. **Saturate on integer overflow** (флажок) – Подавлять переполнение целого. При установленном флажке ограничение сигналов целого типа выполняется корректно.

Значение **auto** параметра **Data type** используется в том случае, если необходимо установить тип данных такой же, как у входного порта блока получающего сигнал от данного блока.

Входной сигнал блока может быть действительным или комплексным. В случае комплексного входного сигнала выходной сигнал также будет комплексным.

Блок работает со скалярными, векторными и матричными сигналами.

На рис. 9.7.15. показаны примеры использования блока **Data Type Conversion**.

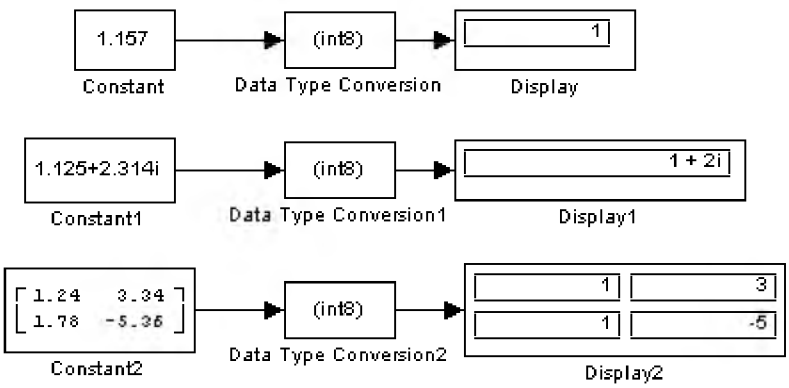


Рис. 9.7.15. Использование блока **Data Type Conversion**

9.7.16. Блок преобразования размерности сигнала Reshape

Назначение:

Блок изменяет размерность векторного или матричного сигнала.

Параметры:

1. **Output dimensionality** – Вид размерности выходного сигнала. Выбирается из списка:
   - **1D array** – Одномерный массив (вектор).
   - **Column vector** – Векторстолбец.
   - **Row vector** – Векторстрока
   - **Customize** – Матрица или вектор заданной размерности. Для векторного выходного сигнала параметр задается как скаляр, определяющий число элементов выходного вектора. Для матричного выходного сигнала параметр задается как вектор,

определяющий количество строк и столбцов выходной матрицы. Значение параметра должно соответствовать количеству элементов во входном массиве. В случае матричных сигналов данные выбираются из столбцов входной матрицы и последовательно заносятся в столбцы выходной матрицы.

2. **Output dimensions** – Значение размерности выходного сигнала. Параметр доступен, если вид размерности установлен как **Customize**.

Примеры использования блока **Reshape** показаны на рис. 9.7.16.

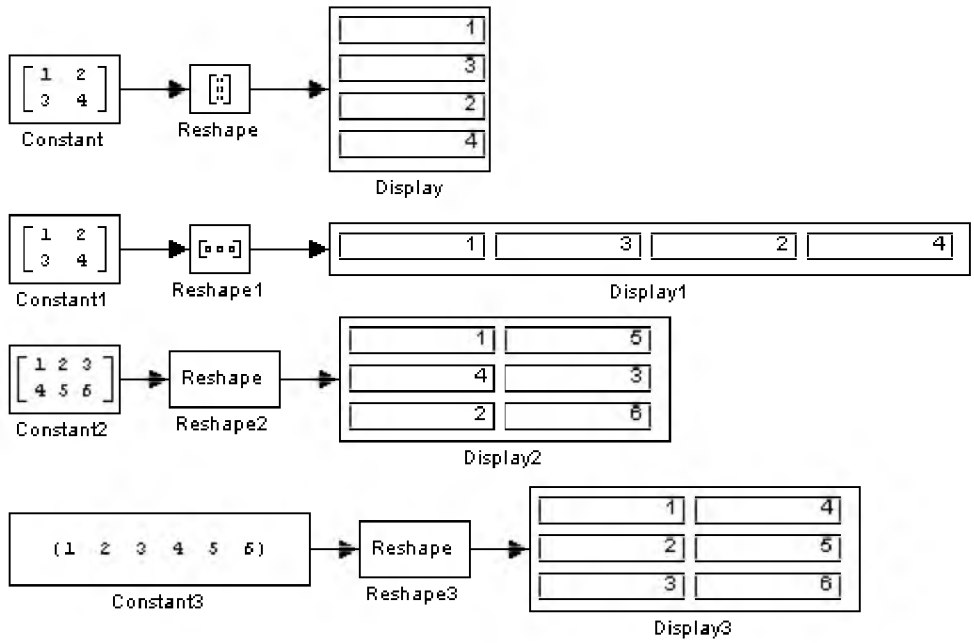


Рис. 9.7.16. Примеры использования блока **Reshape**

9.7.17. Блок определения размерности сигнала Width

Назначение:

Вычисляет размерность входного сигнала.

Параметры:

Нет.

Входным сигналом блока может быть действительный или комплексный сигнал любого типа.

Выходной сигнал блока имеет тип **double**.

Примеры использования блока **Width** показаны на рис. 9.7.17.

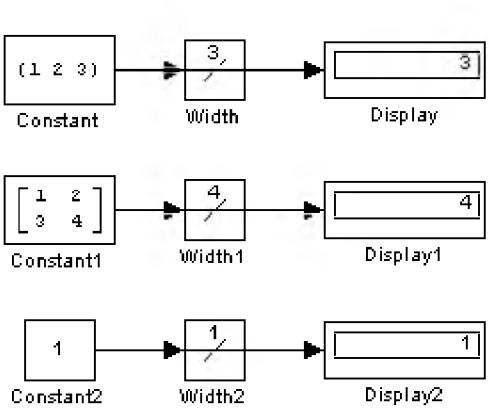


Рис. 9.7.17. Примеры использования блока **Width**

9.7.18. Блок определения момента пересечения порогового значения Hit Crossing

Назначение:

Определяет момент времени, когда входной сигнал пересекает заданное пороговое значение.

Параметры:

1. **Hit crossing offset** – Порог. Значение, пересечение которого входным сигналом требуется идентифицировать.
    - **Hit crossing direction** – Направление пересечения. Выбирается из списка:
    - **rising** – Возрастание.
    - **failing** – Убывание.
    - **either** – Оба направления.
2. **Show output port** (флажок) – Показать выходной порт. В том случае, если этот флажок снят, то точка пересечения сигналом порогового уровня находится, но выходной сигнал блоком не генерируется.

В момент пересечения порогового уровня блок вырабатывает единичный сигнал длительностью в один шаг модельного времени.

Пример использования блока **Hit Crossing** показан на рис. 9.7.18. Блок определяет моменты пересечения в обоих направлениях синусоидальным сигналом уровня **0.5**.

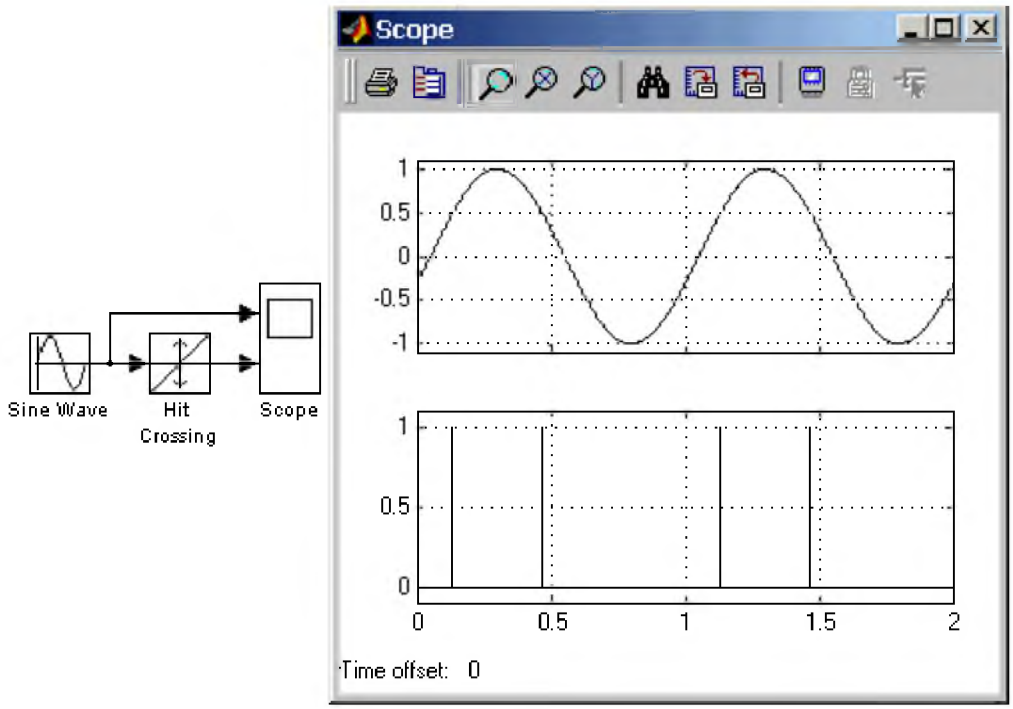


Рис. 9.7.18. Пример использования блока **Hit Crossing**

9.7.19. Блок установки начального значения сигнала IC

Назначение:

Задает начальное значение сигнала.

Параметры:

Initial value – Начальное значение.

Выходной сигнал блока **IC** равен значению параметра **Initial value** на первом шаге расчета вне зависимости от величины входного сигнала блока. На остальных расчетных шагах входной сигнал проходит на выход блока без какихлибо изменений.

Пример использования блока **IC** показан на рис. 9.7.19. В примере начальное значение сигнала задано равным **0.5**. Шаг расчета задан равным **1c**.

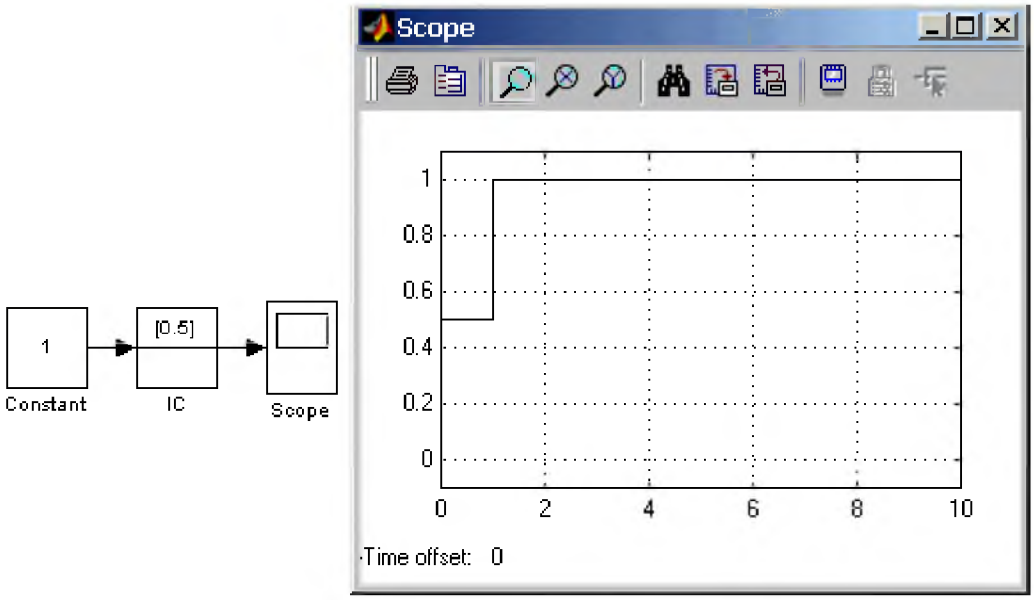


Рис. 9.7.19. Пример использования блока **IC**

9.7.20. Блок проверки сигнала Signal Specification

Назначение:

Выполняет проверку сигнала на соответствие заданным для сигнала параметрам.

Параметры:

1. **Dimension** Размерность сигнала. Задается скаляром, если входной сигнал векторный или матрицей вида **[m n]** (**m** – количество строк, **n** – количество столбцов), если входной сигнал – матрица. Если значение параметра задано как **–1** (минус **1**), то проверка не производится.
2. **Sample time** – Шаг модельного времени. Задается вектором вида **[period offset]**, где **period** – значение шага модельного времени, **offset** – смещение. Если значение параметра задано как **–1** (минус **1**), то проверка не производится. Можно также задавать значение **–1** (минус **1**) и отдельно для параметров **period** или **offset**. В этом случае не будет проводиться проверка именно этих параметров.
3. **Data Type** Тип данных. Выбирается из списка: **auto** (проверка не производится)**, double, single, int8, uint8, int16, uint16, int32, uint32** или **boolean.**
4. **Signal type** – Тип сигнала. Выбирается из списка: **auto** (проверка не производится)**, real** или **complex**.

На пиктограмме блока отображаются проверяемые параметры сигнала и их значения. Пример использования блока **Signal Specification** показан на рис. 9.7.20.

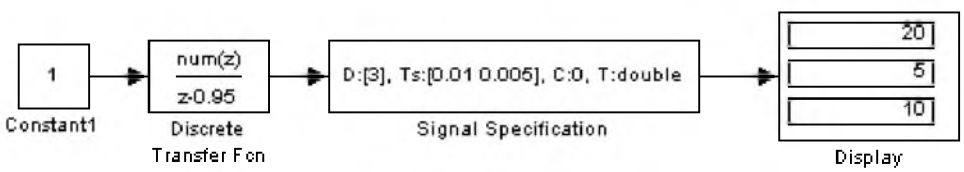


Рис. 9.7.20. Пример использования блока **Signal Specification**

9.7.21. Датчик свойств сигнала Probe

Назначение:

Блок позволяет получить численные значения параметров сигнала.

Параметры:

1. **Probe width** (флажок) – Определение числа элементов в векторном или матричном сигнале.
2. **Probe Sample time** (флажок) – Определение значения эталонного времени.
3. **Probe Complex Signal** (флажок) – Определение типа сигнала (возвращает **1**, если сигнал представлен в комплексном виде, и **0** в противном случае).
4. **Probe signal dimension** (флажок) – Определение размерности сигнала.
   Контролируются те параметры, для которых установлены флажки. Числом отмеченных флажков задается число выходов блока.

Установка флажка для какоголибо параметра приводит к появлению на изображении блока порта, с которого можно считывать значение данного параметра сигнала.

Пример использования блока **Probe** показан на рис. 9.7.21.

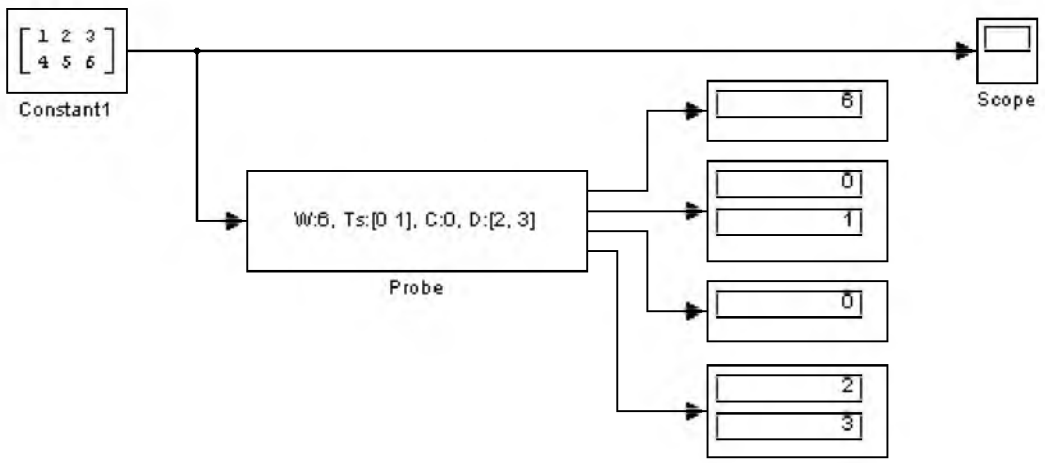


Рис. 9.7.21. Пример использования блока **Probe**

9.7.22. Блок, задающий количество итераций FunctionCall Generator

Назначение:

Блок позволяет задать количество итераций на каждом шаге модельного времени для управляемой подсистемы.

Параметры:

1. **Sample time** – Шаг модельного времени.
2. **Number of iterations** – Количество итераций.

Блок используется совместно с управляемыми подсистемами **FunctionCall Subsystem** или **Triggered Subsystem.** Для управляющих блоков внутри этих подсистем параметр **Trigger type** должен иметь значение **functioncall**.

Пример использования блока **FunctionCall Generator** показан на рис. 9.7.22. В примере использована управляемая подсистема, выходной сигнал которой увеличивается на единицу при каждом ее вызове. Для первой подсистемы блок **FunctionCall Generator** задает количество итераций на каждом шаге равное **1**, а для второй – равное **3**.

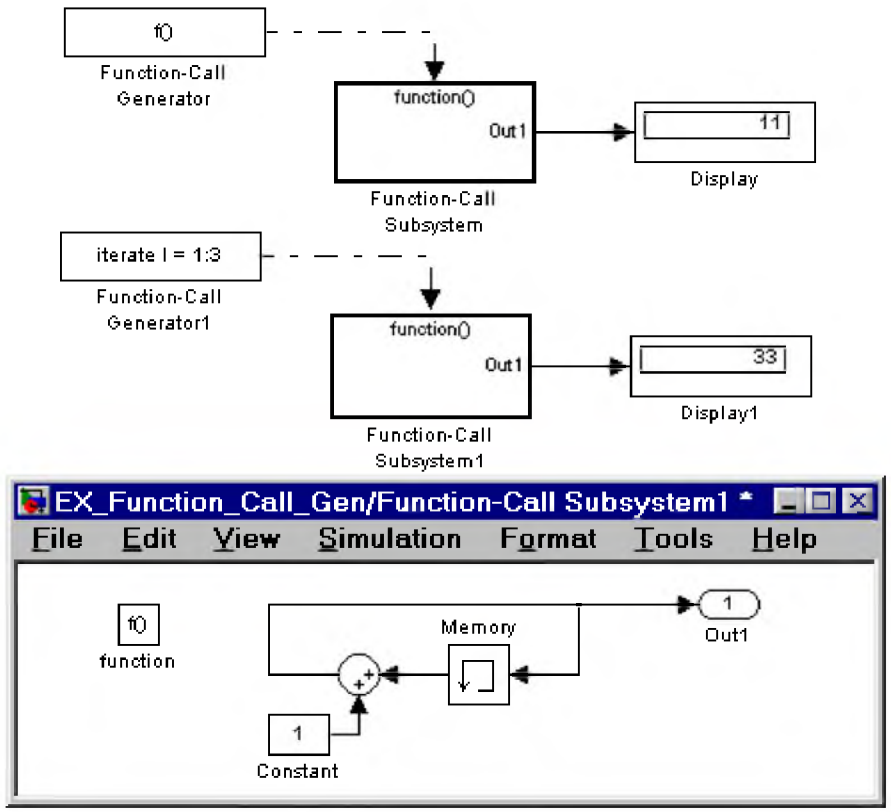


Рис. 9.7.22. Пример использования блока **FunctionCall Generator**

9.7.23. Информационный блок Model Info

Назначение:

Блок отображает информацию о модели.

Параметры:

1. **Model properties** – Свойства модели:

- **Created** – Дата и время создания модели.
- **Creator –** Данные об авторе
- **Modified by** – Данные о пользователе вносившем изменения.
- **ModifiedDate** – Дата изменения.
- **ModifiedComment** – Описание изменений.
- **ModelVersion** – Версия модели.
- **Description** – Описание модели.
- **LastModificationDate** – Дата последнего изменения.

2. **Horizontal text allignment –** Способ выравнивания текста по горизонтали. Выбирается из списка:
   - **Center –** По центру.
   - **Left –** По левому краю.
   - **Right –** По правому краю.
3. **Show block frame** (флажок) **–** Отобразить рамку блока.

Для отображения данных на пиктограмме блока необходимо с помощью кнопки 

скопировать нужный параметр из окна **Model properties** в окно редактирования. В блоке может отображаться статическая информация, которую пользователь вносит сам (например, данные об авторе, описание модели и т.п.) и динамически обновляемая информация (например, дата создания модели, дата последней модификации и т.п.). Динамически обновляемая информация представляется в окне блока как ссылка на переменную, которая ее содержит. Ссылка имеет вид **%<имя_переменной>**. Например, ссылка **%<LastModificationDate>** означает, что в требуемой позиции будет выведено значение переменной **LastModificationDate**, содержащей дату последней модификации модели.

На пиктограмме блока отображается также часть информации заданная с помощью команды **Model Properties** меню **File** окна модели.

Пример использования блока **Model Info** показан на рис. 9.7.23. Там же показано окно параметров данного блока.

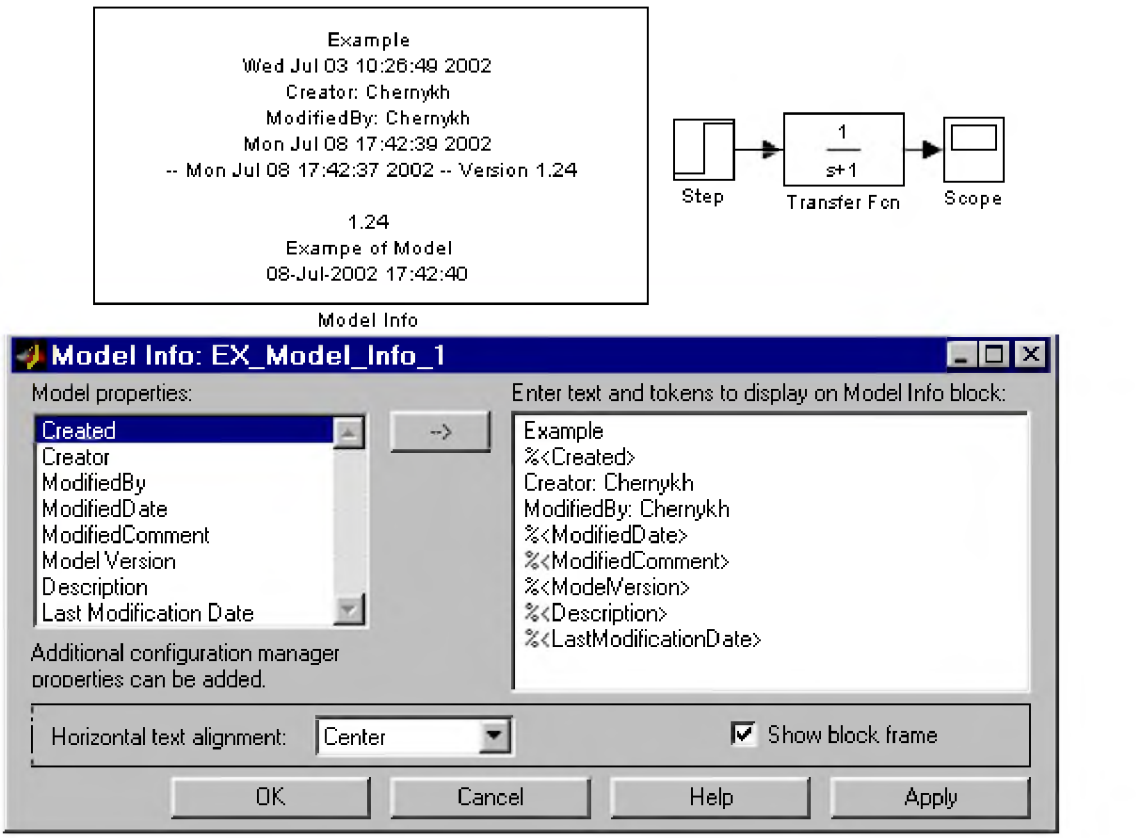


Рис. 9.7.23. Пример использования блока **Model Info**

9.8. Function & Tables – блоки функций и таблиц

9.8.1. Блок задания функции Fcn

Назначение:

Задает выражение в стиле языка программирования **C .**

Параметры:

Expression – Выражение, используемое блоком для вычисления выходного сигнала на основании входного. Это выражение составляется по правилам, принятым для описания функций на языке **C**.

В выражении можно использовать следующие компоненты:

1. Входной сигнал. Входной сигнал в выражении обозначается **u**, если он является скаляром. Если входной сигнал – вектор, необходимо указывать номер элемента вектора в круглых скобках. Например, **u(1)** и **u(3)** – первый и третий элементы входного вектора.
2. Константы.
3. Арифметические операторы (+ – * /).
4. Операторы отношения (= = **!**= > < >= <=).
5. Логические операторы (**&&** | | **!**).

6. Круглые скобки.
7. Математические функции: **abs, acos, asin, atan, atan2, ceil, cos, cosh, exp, fabs, floor, hypot, ln, log, log10, pow, power, rem, sgn, sin, sinh, sqrt, tan,** и **tanh.**
8. Переменные из рабочей области. Если переменная рабочей области является массивом, то ее элементы должны указываться с помощью индексов в круглых скобках. Напрмер, **A(1,1)** - первый элемент матрицы **A**.

Операторы отношения и логические операторы возвращают значения в виде логического нуля (**FALSE**) или логической единицы (**TRUE**).

Операторы, допускаемые к использованию в выражении, имеют следующий приоритет (в порядке убывания):

1. ( )
2. + – (унарные)
3. Возведение в степень
4. !
5. /
6. + – (бинарные)
7. > < <= >=
8. = !=
9. &&
10. | |

Блок не поддерживает матричные и векторные операции. Выходной сигнал блока всегда – скаляр.

Примеры использования блока **Fcn** показаны на рис. 9.8.1.

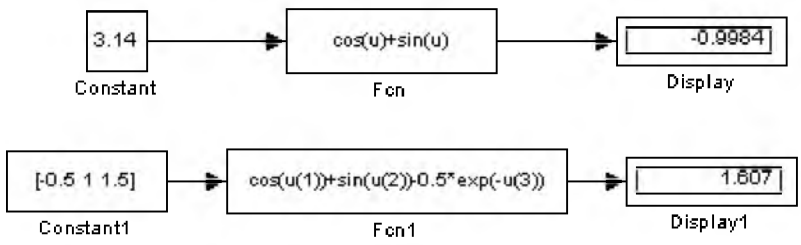


Рис. 9.8.1. Примеры использования блока **Fcn**

9.8.2. Блок задания функции MATLAB Fcn

Назначение:

Задает выражение в стиле языка программирования **MATLAB**.

Параметры:

1. **MATLAB function** – Выражение на языке **MATLAB**.
2. **Output dimensions** – Размерность выходного сигнала. Значение параметра **–1** (минус один) предписывает блоку определять размерность автоматически.

3. **Output signal type** – Тип выходного сигнала. Выбирается из списка:
    - **real** – Действительный сигнал.
    - **complex** – Комплексный сигнал.
    - **auto** – Автоматическое определение типа сигнала.
4. **Collapse 2-D results to 1-D** – Преобразование двумерного выходного сигнала к одномерному.

Входной сигнал в выражении обозначается **u**, если он является скаляром. Если входной сигнал – вектор, необходимо указывать номер элемента вектора в круглых скобках. Например, **u(1)** и **u(3)** – первый и третий элементы входного вектора. Если выражение состоит из одной функции, то ее можно задать без указания параметров. Выражение может содержать также собственные функции пользователя, написанные на языке **MATLAB** и оформленные в виде **m**-файлов. Имя **m**-файла не должно совпадать с именем модели (**mdl**-файлом).

Рис. 9.8.2 демонстрирует применение блока **MATLAB Fcn.** В примере используется функция **My_Matlab_Fcn_1**, вычисляющая сумму и произведение двух элементов входного вектора. Текст функции (файл **My_Matlab_Fcn_1.m**) приведен ниже:

```
function y=My_Matlab_Fcn_1(x,k);
y(1)=x*k;
y(2)=x + k;
```

Выражение для вызова функции, заданное параметром **MATLAB function**, имеет вид:
My_Matlab_Fcn_1(u(1),u(2)) .

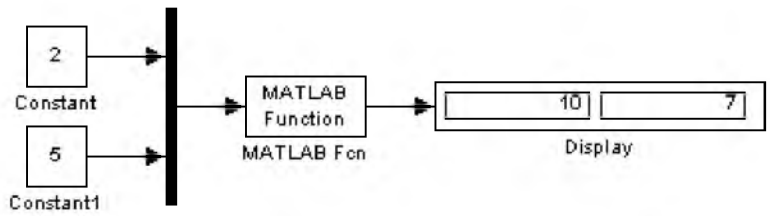


Рис. 9.8.2. Примеры использования блока **MATLAB Fcn**

9.8.3. Блок задания степенного многочлена Polynomial

Назначение:

Задает степенной многочлен.

Параметры:

Polynomial coefficients – Вектор коэффициентов полинома. Коэффициенты расположены в векторе по убыванию степени независимой переменной. Например, для полинома $\mathbf{x^2+2x+5}$ необходимо задать вектор коэффициентов **[1 2 5]**. Коэффициенты должны быть действительного типа.

Блок вычисляет значение полинома по его коэффициентам и величине входного сигнала. Если входной сигнал вектор или матрица блок вычисляет результат для каждого элемента массива.

На рис. 9.8.3 показаны примеры использования блока **Polynomial.** В примерах для первого полиномиального блока коэффициенты заданы вектором **[1 2 5]**, а для второго – **[1 2 0 5]**.

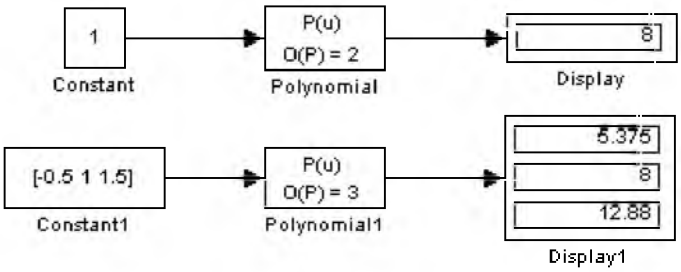


Рис. 9.8.3. Примеры использования блока **Polynomial**

9.8.4. Блок одномерной таблицы Look-Up Table

Назначение:

Задает в табличной форме функцию одной переменной.

Параметры:

1. **Vector of input values** – Вектор значений входного сигнала. Может быть задан в виде дискретных значений (например, **[1 2 7 9]**), либо в виде непрерывного диапазона (например, **[0:10]**). Элементы вектора или граница диапазона могут быть заданы в виде вычисляемого выражения, например **[tan(5) sin(3)]**.
2. **Vector of output values** – Вектор выходных значений, соответствующий вектору входных значений.

Блок работает в соответствии со следующими правилами:

1. Если входной сигнал равен одному из элементов вектора входных значений (**Vector of input values**), то выходное значение блока будет равно соответствующему элементу вектора выходных значений (**Vector of output values**). Например, пусть вектор входных значений равен **[0 1 2 5]**, а вектор выходных значений **[-5 –10 3 100],** тогда при входном сигнале равном **1** выходной сигнал будет равен **–10**.
2. Если входной сигнал не совпадает ни с одним из элементов вектора входных значений, то блок выполняет линейную интерполяцию между двумя ближайшими к нему элементами.
3. Если входной сигнал выходит за границы вектора входных значений, то блок выполняет линейную экстраполяцию по двум крайним элементам.

График функции, заданный с помощью настроек блока отображается на его пиктограмме.

Входной сигнал блока может быть векторным. В этом случае блок выполняет поэлементную операцию.

На рис. 9.8.4 показан пример использования блока **Look-Up Table**. В примере вектор входных значений равен **[-5:5**], а вектор выходных значений равен **tanh([-5:5])**.

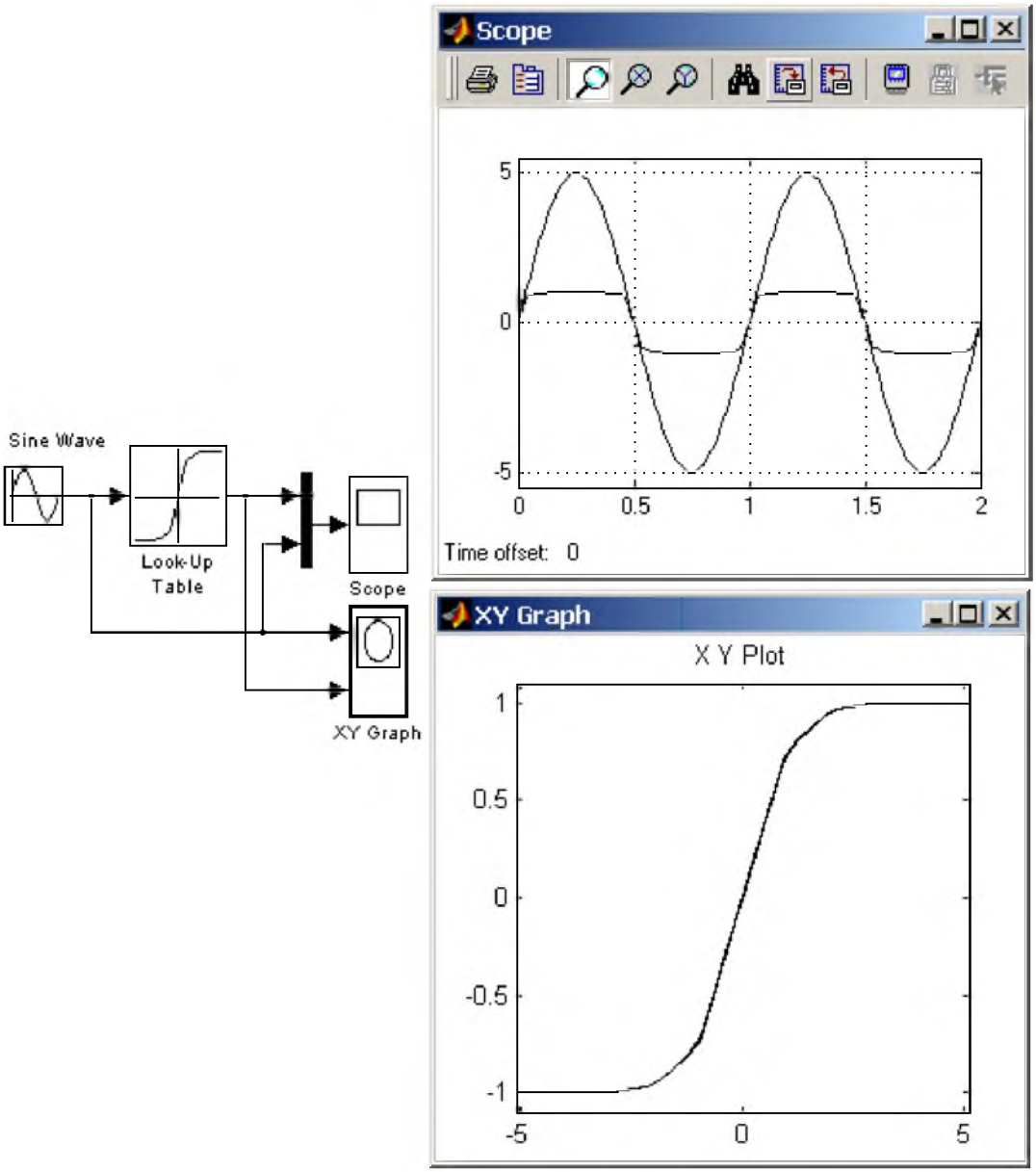


Рис. 9.8.4. Пример использования блока **Look-Up Table**

9.8.5. Блок двумерной таблицы Look-Up Table(2D)

Назначение:

Задает в табличной форме функцию двух переменных.

Параметры:

1. **Row** – Строка. Вектор значений первого аргумента. Задается аналогично параметру **Vector of input values** одномерной таблицы. Элементы вектора должны быть упорядочены по возрастанию.
2. **Column** – Столбец. Вектор значений второго аргумента. Задается аналогично предыдущему параметру.

3. **Table –** Таблица значений функции. Задается в виде матрицы. Количество строк должно быть равно числу элементов вектора **Row**, а количество столбцов – числу элементов вектора **Column**.

Правила формирования таблицы значений функции показаны в Табл.9.8.1.

Таблица 9.8.1.

		Второй аргумент (Column)		
		3	**7**	**9**
Первый аргумент (Row)	**2**	**10**	**20**	**30**
	4	**40**	**50**	**60**
	8	**70**	**80**	**90**

Для приведенной таблицы значения параметров блока будут следующими:

Row – [2 4 8] ,
Column – [3 7 9] ,
Table – [10 20 30;40 50 60;70 80 90] .

Пример использования блока **Look-Up Table(2D)** показан на рис. 9.8.5. Параметры блока заданы в соответствии с Табл. 9.8.1.

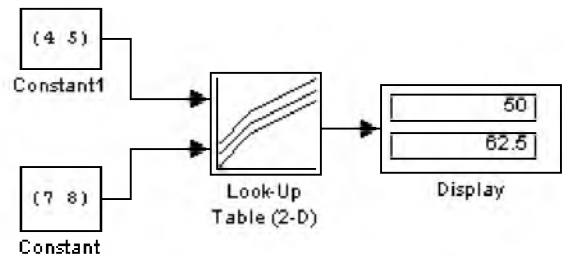


Рис. 9.8.5. Пример использования блока **Look-Up Table(2D)**

9.8.6. Блок многомерной таблицы Look-Up Table (n-D)

Назначение:

Задает в табличной форме функцию многих переменных.

Параметры:

1. **Number of table dimensions** – Количество размерностей таблицы (аргументов функции). Значение параметра выбирается из списка: **1**, **2**, **3**, **4**, **More…**(Много).
2. **First input (row) breakpoint set** – Вектор значений первого аргумента (строка). Задается аналогично параметру **Row** двумерной таблицы.

3. **Second (column) input breakpoint set** – Вектор значений второго аргумента (столбец). Задается аналогично предыдущему параметру.
4. **Third input breakpoint set** – Вектор значений третьего аргумента. Параметр доступен, если количество размерностей таблицы задано больше **2**.
5. **Fourth input breakpoint set** – Вектор значений четвертого аргумента. Параметр доступен, если количество размерностей таблицы задано больше **3**.
6. **Fifth..Nth input breakpoint sets (cell array)** – Массив значений пятого и остальных аргументов (массив ячеек). Параметр доступен, если количество размерностей таблицы задано больше **4**.
7. **Explicit number of dimensions** – Точное количество размерностей таблицы (аргументов функции). Параметр доступен и его необходимо задавать, если параметр **Number of table dimensions** имеет значение **More.**
8. **Index search method** – Метод поиска по индексам. Принимает значения из списка:

- **Evenly Spaced Points** – Поиск для равноотстоящих индексов. Дает наилучший результат по скорости поиска, если векторы аргументов имеют равноотстоящие друг от друга значения (например, **[10 20 30]**).
- **Linear Search** – Линейный поиск. Дает наилучший результат, если значения входных сигналов на текущем шаге расчета отличаются от предыдущих значений незначительно.
- **Binary Search** – Двоичный поиск. Дает наилучший результат, если значения входных сигналов на текущем шаге расчета значительно отличаются от предыдущих значений.

9. **Begin index searches using previous index results** (флажок) – Начинать поиск, используя результаты предыдущего поиска.
10. **Use one (vector) input port instead of N ports** (флажок) – Использовать многомерный входной порт вместо нескольких одномерных.
11. **Table data** – Таблица значений функции. Задается по правилам формирования многомерных массивов.
12. **Interpolation method** – Метод интерполяции. Выбирается из списка:

- **None** – Интерполяция не выполняется.
- **Linear** – Линейная интерполяция.
- **Cubic Spline** – Кубическая сплайн-интерполяция.

13. **Extrapolation method** – Метод экстраполяции. Выбирается из списка: **None**, **Linear** или **Cubic Spline**.
14. **Action for out of range input** – Реакция на выход входного сигнала за границы вектора значений аргумента. Выбирается из списка:

- **None** – Реакция отсутствует.
- **Warning** – Вывод предупреждающего сообщения в командной строке **MATLAB**.
- **Error** – Вывод сообщения об ошибке в командной строке **MATLAB** и остановка расчета.

Пример использования блока **Look-Up Table (n-D)** для задания функции двух аргументов показан на рис. 9.8.6. Параметры блока заданы в соответствии с Табл. 9.8.1. Для расчета выходных значений задана кубическая сплайн-интерполяция.

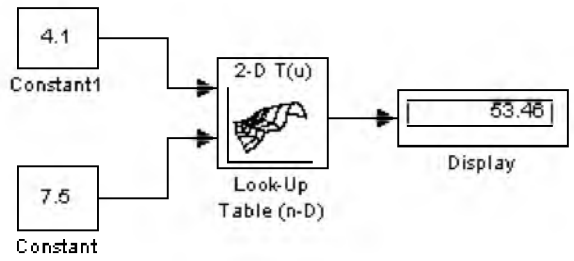


Рис. 9.8.6. Пример использования блока **Look-Up Table (n-D)**

9.8.7. Блок таблицы с прямым доступом Direct Loop-Up Table (n-D)

Назначение:

Задает многомерную таблицу с прямым доступом к ее элементам. Индексация элементов начинается с нуля.

Параметры:

1. **Number of table dimensions –** Количество размерностей таблицы (аргументов функции). Значение параметра выбирается из списка: **1**, **2**, **3**, **4**, **More…**(Много).
2. **Explicit number of dimensions –** Точное количество размерностей таблицы (аргументов функции). Параметр доступен, и его необходимо задавать, если параметр **Number of table dimensions** имеет значение **More.**
3. **Inputs select this object from table –** Задать вид выходного сигнала. Выбирается из списка:

- **Element** – Элемент. Если на выходе блока необходимо получить отдельный элемент таблицы, то на вход блока должны подаваться значения всех индексов элемента.
- **Column** – Столбец. Если на выходе необходимо получить столбец, то на вход блока необходимо подавать на один индекс меньше, по сравнению с предыдущим вариантом.
- **D Matrix** – Матрица. В этом случае на вход блока подается на два индекса меньше, по сравнению с первым вариантом.

4. **Make table an input –** Таблица значений функции задается через отдельный вход блока, а не параметром **Table data**.
5. **Table data –** Таблица значений функции. Задается по правилам формирования многомерных массивов.
6. **Action for out of range input –** Реакция на выход входного сигнала за границы вектора значений аргумента. Выбирается из списка:

- **None** – Реакция отсутствует.
- **Warning** – Вывод предупреждающего сообщения в командной строке **MATLAB**.
- **Error** – Вывод сообщения об ошибке в командной строке **MATLAB** и остановка расчета.

Примеры использования блока **Look-Up Table (n-D)** для задания функции двух аргументов показаны на рис. 9.8.7. В первом случае таблица значений функции (**[10 20 30;40 50 60;70 80 90]**) задана в параметрах блока, а во втором –подается через отдельный вход (установлен флажок **Make table an input**)

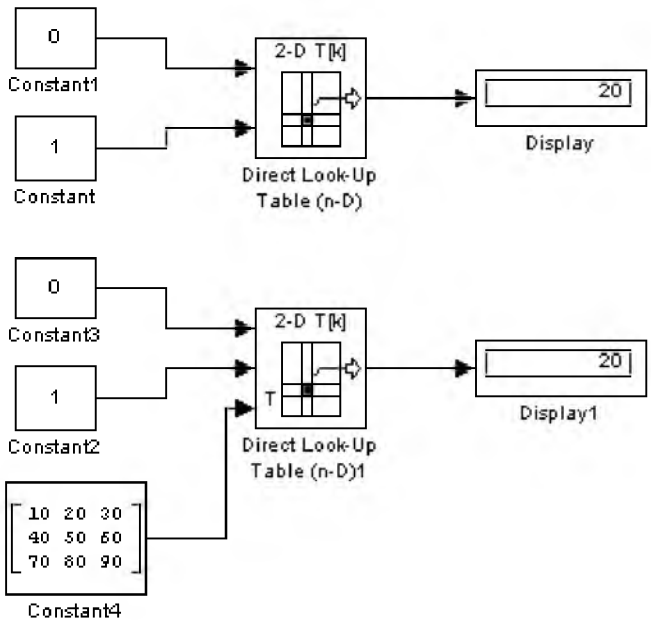


Рис. 9.8.7. Пример использования блока **Direct Loop-Up Table (n-D**)

9.8.8. Блок работы с индексами PreLook-Up Index Search

Назначение:

Вычисляет значение индекса и относительную величину входного сигнала. Используется совместно с блоком **Interpolation (n-D) using PreLook-Up**.

Параметры:

1. **Breakpoint data** – Вектор узловых точек. Данный параметр аналогичен вектору входного сигнала блоков задающих табличные функции.
2. **Index search method** - Метод поиска индексов. Выбирается из списка:

- **Evenly Spaced Points –** Поиск для равноотстоящих индексов.
- **Linear Search –** Линейный поиск.
- **Binary Search –** Двоичный поиск.

3. **Begin index search using previous index result** (флажок) – Начало поиска индекса с последнего результата.
4. **Output only the index** (флажок) – Вывод только индексов.
5. **Process out of range input** – Тип процесса при выходе входного сигнала за заданные пределы. Выбирается из списка:

- **Clip to Range** – Ограничить предельным значением.
- **Linear Extrapolation** – Линейная экстраполяция.

6. **Action for out of range input –** Реакция на выход входного сигнала за границы вектора узловых точек. Выбирается из списка:

- **None** – Реакция отсутствует.

- **Warning** – Вывод предупреждающего сообщения в командной строке **MATLAB**.
- **Error** – Вывод сообщения об ошибке в командной строке **MATLAB** и остановка расчета.

Выходным сигналом блока является вектор, первый элемент вектора – найденный индекс, а второй – относительная величина входного сигнала. Блок находит индекс того элемента, значение которого не превышает величину входного сигнала. Например, для вектора узловых точек **[0 5 10 20 50 100]** и входного сигнала равного **55** найденный индекс будет равен **4** .

Относительная величина входного сигнала рассчитывается в соответствии с выражением:

$$h = \frac{x - A(i)}{A(i+1) - A(i)},$$

где

x – входной сигнал,

i – найденный индекс,

A – вектор узловых точек.

Для приведенного выше примера относительная величина входного сигнала будет равна **0.1**.

Пример поясняющий работу блока, показан на рис. 9.8.8.

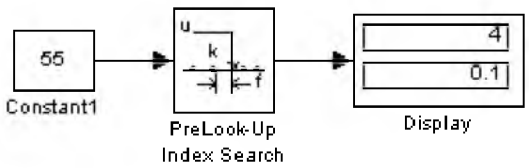


Рис. 9.8.8. Пример использования блока **PreLook-Up Index Search**

9.8.9. Блок интерполяции табличной функции Interpolation (n-D) using PreLook-Up

Назначение:

Вычисляет значение табличной функции по значению индекса и относительной величине входного сигнала. Используется совместно с блоками **PreLook-Up Index Search**.

Параметры:

1. **Number of table dimensions** – Количество размерностей таблицы (аргументов функции). Значение параметра выбирается из списка: **1**, **2**, **3**, **4**, **More…**(Много).
2. **Explicit number of dimensions** – Точное количество размерностей таблицы (аргументов функции). Параметр доступен, и его необходимо задавать, если параметр **Number of table dimensions** имеет значение **More.**

3. **Table data** – Таблица значений функции. Задается по правилам формирования многомерных массивов.
4. **Interpolation method** – Метод интерполяции. Выбирается из списка:

- **None** – Интерполяция не выполняется.
- **Linear** – Линейная интерполяция.

5. **Extrapolation method** – Метод экстраполяции. Выбирается из списка: **None** или **Linear**.
6. **Action for out of range input** – Реакция на выход входного сигнала за границы вектора значений аргумента. Выбирается из списка:

- **None** – Реакция отсутствует.
- **Warning** – Вывод предупреждающего сообщения в командной строке **MATLAB**.
- **Error** – Вывод сообщения об ошибке в командной строке **MATLAB** и остановка расчета.

Пример поясняющий работу блока, показан на рис. 9.8.9. Таблица значений функции задана матрицей **[10 20 30;40 50 60;70 80 90]**.

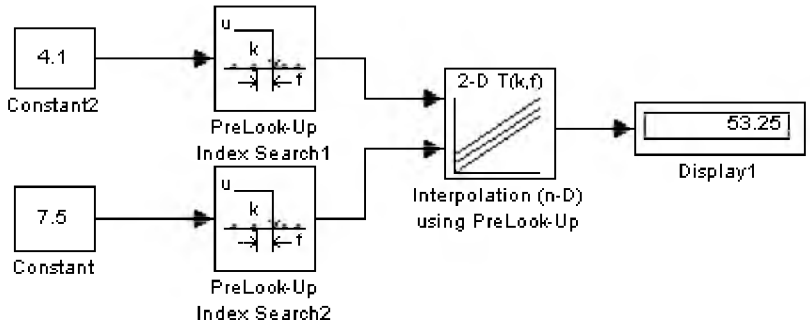


Рис. 9.8.9. Пример использования блока **Interpolation (n-D) using PreLook-Up**

Раздел библиотеки **Function & Tables** содержит еще два блока - **S-Function** и **S-Function Builder**. Они будут подробно рассмотрены в разделе, посвященном созданию **S-функций**.

9.9. Subsystem – подсистемы.

Подсистема это фрагмент **Simulink**-модели, оформленный в виде отдельного блока. Использование подсистем при составлении модели имеет следующие положительные стороны:

1. Уменьшает количество одновременно отображаемых блоков на экране, что облегчает восприятие модели (в идеале модель полностью должна отображаться на экране монитора).
2. Позволяет создавать и отлаживать фрагменты модели по отдельности, что повышает технологичность создания модели.
3. Позволяет создавать собственные библиотеки.
4. Дает возможность синхронизации параллельно работающих подсистем.
5. Позволяет включать в модель собственные справочные средства.
6. Дает возможность связывать подсистему с каким-либо **m**-файлом, обеспечивая запуск этого файла при открытии подсистемы (нестандартное открытие подсистемы).

Использование подсистем и механизма их блоков позволяет создавать блоки, не уступающие стандартным по своему оформлению (собственное окно параметров блока, пиктограмма, справка и т.п.).

Количество подсистем в модели не ограничено, кроме того подсистемы могут включать в себя другие подсистемы. Уровень вложенности подсистем друг в друга также не ограничен.

Связь подсистемы с моделью (или подсистемой верхнего уровня иерархии) выполняется с помощью входных (блок **Inport** библиотеки **Sources**) и выходных (блок **Outport** библиотеки **Sinks**) портов. Добавление в подсистему входного или выходного порта приводит к появлению на изображении подсистемы метки порта, с помощью которой внешние сигналы передаются внутрь подсистемы или выводятся в основную модель. Переименование блоков **Inport** или **Outport** позволяет изменить метки портов, отображаемые на пиктограмме подсистемы со стандартных (**In** и **Out**) на те, которые нужны пользователю.

Подсистемы могут быть виртуальными (**Subsystem**) и монолитными (**Atomic Subsystem**). Отличие этих видов подсистем заключается в порядке выполнения блоков во время расчета. Если подсистема является виртуальной, то **Simulink** игнорирует наличие границ отделяющих такую подсистему от модели при определении порядка расчета блоков. Иными словами в виртуальной системе сначала могут быть рассчитаны выходные сигналы нескольких блоков, затем выполнен расчет блоков в основной модели, а затем вновь выполнен расчет блоков входящих в подсистему. Монолитная подсистема считается единым (неделимым) блоком и **Simulink** выполняет расчет всех блоков в такой подсистеме, не переключаясь на расчеты других блоков в основной модели. Изображение монолитной подсистемы имеет более толстую рамку по сравнению с виртуальной подсистемой.

Подсистемы могут быть также управляемыми или неуправляемыми. Управляемые подсистемы всегда являются монолитными. Управляемые подсистемы имеют дополнительные (управляющие) входы, на которые поступают сигналы активизирующие данную подсистему. Управляющие входы расположены сверху или снизу подсистемы. Когда управляемая подсистема активизирована – она выполняет вычисления. В том случае если управляемая подсистема пассивна, то она не выполняет вычисления, а значения сигналов на ее выходах определяются настройками выходных портов.

Для создания в модели подсистемы можно воспользоваться двумя способами:

1. Скопировать нужную подсистему из библиотеки **Subsystem** в модель.
2. Выделить с помощью мыши нужный фрагмент модели и выполнить команду **Create Subsystem** из меню **Edit** окна модели. Выделенный фрагмент будет помещен в подсистему, а входы и выходы подсистемы будут снабжены соответствующими портами. Данный способ позволяет создать виртуальную неуправляемую подсистему. В дальнейшем, если это необходимо, можно сделать подсистему монолитной, изменив ее параметры, или управляемой, добавив управляющий элемент из нужной подсистемы находящейся в библиотеке. Отменить группировку блоков в подсистему можно командой **Undo**.

Рис. 9.9.1 иллюстрирует процесс создания подсистемы вторым способом. На рис. 9.9.2 показан результат этого процесса. В примере использована модель управляемого функционального генератора.

Рис. 9.9.1 Создание подсистемы

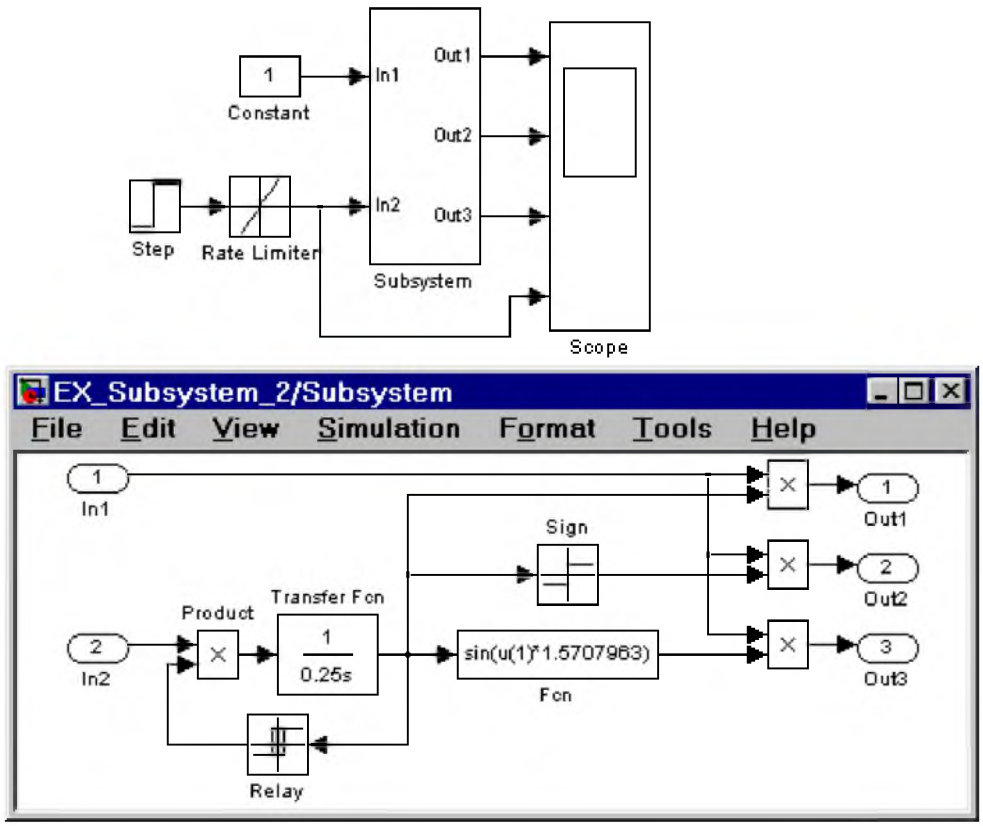


Рис. 9.9.2 Модель, использующая подсистему

9.9.1. Виртуальная и монолитная подсистемы Subsystem и Atomic Subsystem

Доступ к окну параметров подсистемы осуществляется через меню **Edit** командой **Block Parameters…**

Параметры:

1. **Show port labels –** Показать метки портов.
2. **Treat as atomic unit** (флажок) – Считать подсистему монолитной. Таким образом, блоки виртуальной и монолитной подсистем – это один и тот же блок, отличающийся значением данного параметра.

3. **Access** – Доступность подсистемы для изменений. Выбирается из списка:
    - **ReadWrite** – Пользователь может открывать и изменять подсистему.
    - **ReadOnly** – Пользователь может открывать подсистему только для просмотра.
    - **NoReadOrWrite** – Пользователь не может открывать и изменять подсистему.
4. **Name of error callback function** – Имя функции используемой для обработки ошибок возникающих в данной подсистеме.

Остальные параметры подсистемы доступны при разработке приложений с использованием **Real-Time Workshop** и рассмотрены в документации на это приложение.

Находящийся в библиотеке блок **Subsystem** (или **Atomic Subsystem**) содержит входной и выходной порты и линию связи между ними.

После того как блок подсистемы скопирован из библиотеки в модель, он становится доступным для редактирования.

9.9.2. Управляемая уровнем сигнала подсистема Enabled Subsystem

Подсистема **Enabled Subsystem** (в дальнейшем **E**-подсистема) активизируется при наличии положительного сигнала на управляющем входе. Если входной сигнал векторный, то подсистема активизируется, если хотя бы один элемент принимает положительное значение. Величина выходного сигнала в том случае, если система заблокирована, определяется настройками выходных портов подсистемы (блоки **Outport**). В том случае если параметр **Output when disabled** (вид сигнала на выходе подсистемы) выходного порта имеет значение **held**, то выходной сигнал подсистемы равен последнему рассчитанному ею значению, если же этот параметр имеет значение **reset**, то выходной сигнал подсистемы равен значению задаваемому параметром **Initial output** (начальное значение).

Свойства **E**-подсистемы определяются параметрами блока **Enable**, который может находиться в любом месте данной подсистемы. Его параметры перечислены ниже.

Параметры:

1. **States when enabling** – Состояние при запуске. Параметр задает состояние подсистемы при каждом запуске. Выбирается из списка:
    - **held** – Использовать предыдущее состояние (последнее состояние когда система была активна).
    - **reset** – Использовать начальное (исходное) состояние.
2. **Show output port** (флажок) – Показать выходной порт. При установленном флажке на пиктограмме блока **Enable** появляется дополнительный выходной порт, сигнал с которого может быть использован для управления блоками внутри подсистемы.

На рис. 9.9.3 показан пример модели с подсистемой и схема этой подсистемы. В примере параметр **States when enabling** блока **Enable** имеет значение **held**. Параметр **Output when disabled** первого выходного порта подсистемы имеет значение **reset**, а второго – **held**. Как видно из временных диаграмм при выключении подсистемы сигнал первого выходного порта равен начальному значению (нулю), а сигнал второго выходного порта равен последнему рассчитанному значению в момент активности подсистемы.

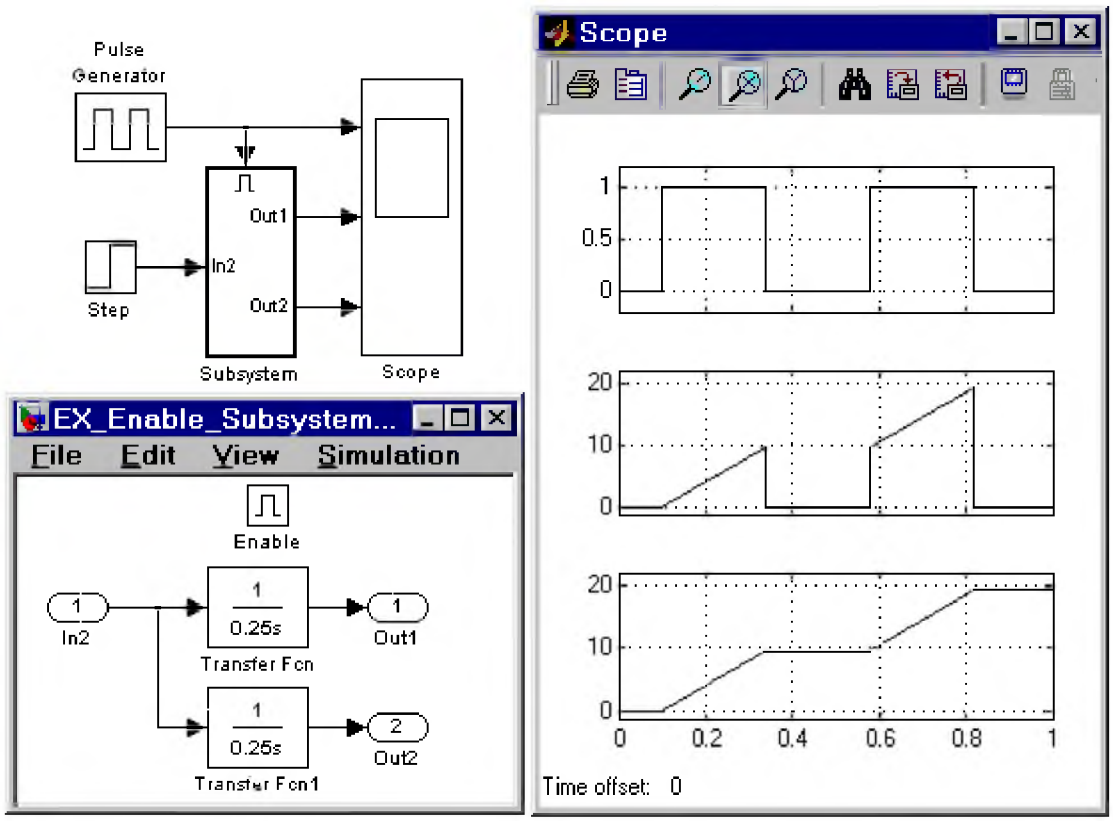


Рис. 9.9.3 Модель, использующая **E**-подсистему

Пример на рис. 9.9.4 отличается от предыдущего настройкой блока **Enable** подсистемы. В данном примере параметр **States when enabling** блока **Enable** имеет значение **reset**. На временных диаграммах видно, что при выключении подсистемы происходит ее сброс до начального состояния.

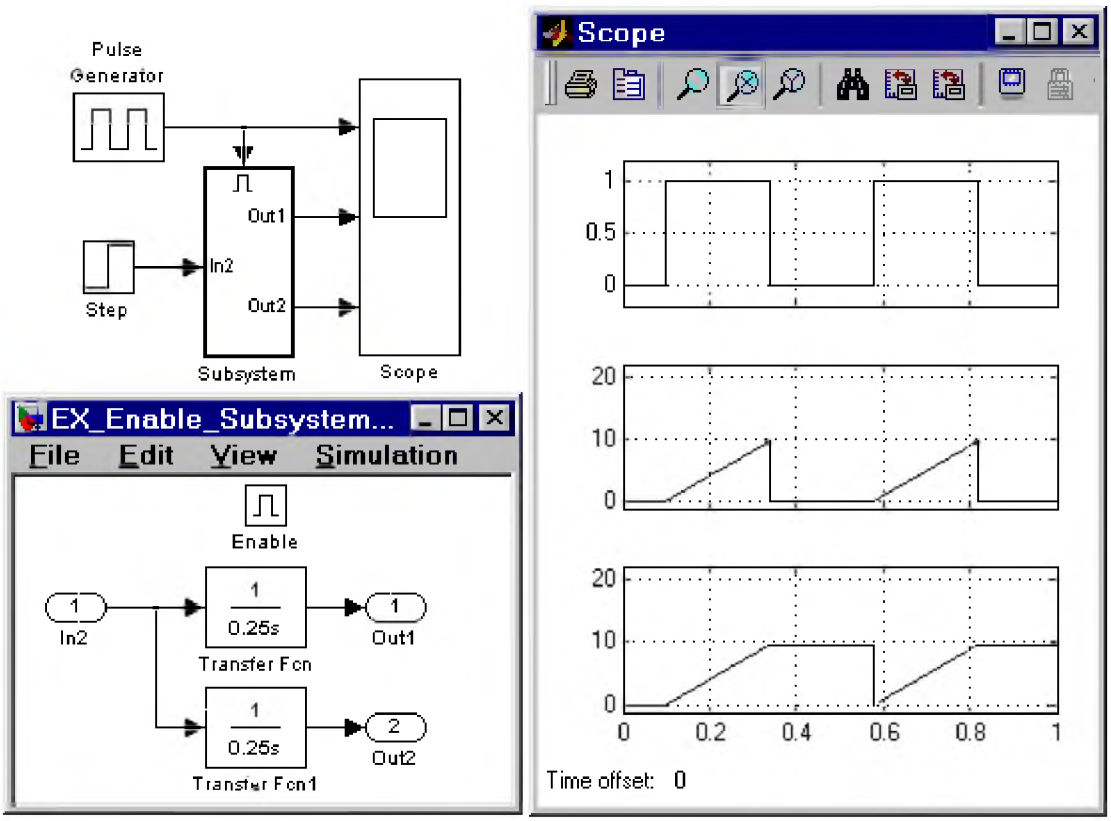


Рис. 9.9.4 Модель, использующая **E**-подсистему

9.9.3. Управляемая фронтом сигнала подсистема Triggered Subsystem

Подсистема **Triggered Subsystem** (в дальнейшем **T**-подсистема) включается фронтом (перепадом уровня) управляющего сигнала и выполняет вычисления только на том шаге моделирования, где произошло это изменение. Если входной сигнал векторный, то подсистема активизируется, если хотя бы в одном элементе изменяется уровень сигнала. Возврат **T**-подсистемы в исходное состояние не производится (подсистема сохраняет последнее значение до следующего запуска), поэтому параметр **States when enabling** выходных портов имеет значение **held**, и недоступен для изменения.

В **T**-подсистеме могут использоваться блоки, для которых модельное время является наследуемым параметром от предыдущего блока (например, **Gain** или **Logical Operator**), а также дискретные блоки, для которых параметр **sample time** имеет значение **–1** (минус один).

Свойства **T**-подсистемы определяются параметрами блока **Trigger**, который может находиться в любом месте данной подсистемы. Его параметры перечислены ниже.

Параметры:

1. **Trigger type** – Тип триггера. Выбирается из списка:
   - **rising** – Активизация подсистемы положительным фронтом.
   - **falling** – Активизация подсистемы отрицательным фронтом.
   - **either** – Активизация подсистемы как положительным, так и отрицательным фронтом.
   - **function-call** – Активизация подсистемы определяется логикой работы заданной **S**-функции.
2. **Show output port** (флажок) – Показать выходной порт.

На рис. 9.9.5 показан пример модели с **T**-подсистемой. Сама **T**-подсистема содержит лишь один усилитель с коэффициентом передачи равным **1**. Как видно из временных диаграмм, подсистема срабатывает по положительному фронту управляющего сигнала. Выходной сигнал подсистемы остается неизменным до следующего положительного фронта управляющего сигнала.

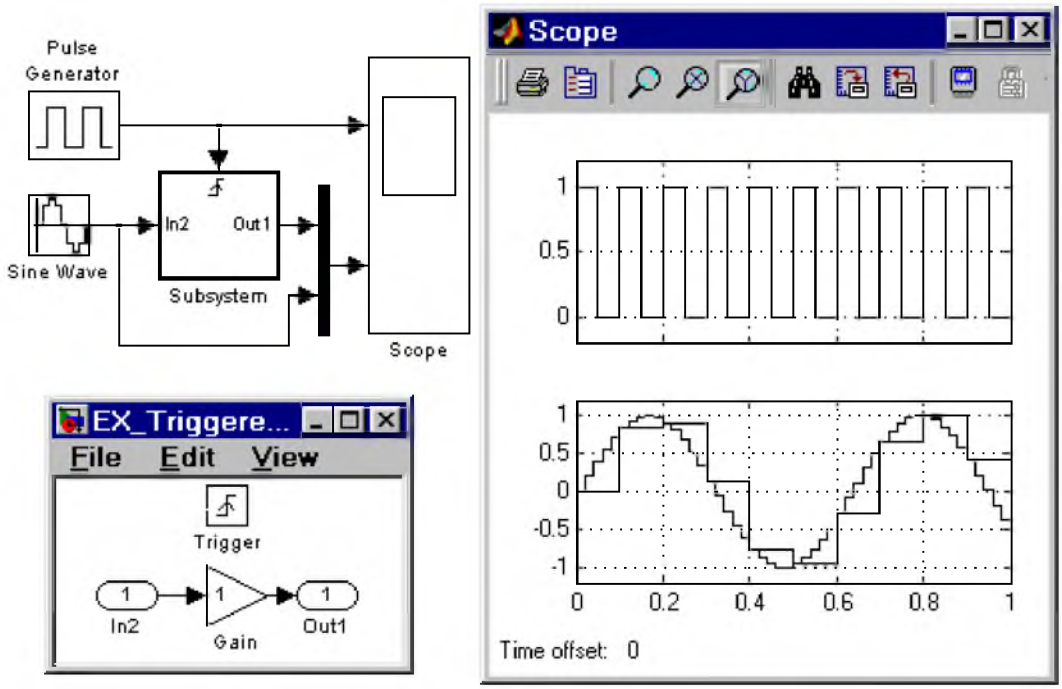


Рис. 9.9.5 Модель, использующая **T**-подсистему

9.9.4. Управляемая уровнем и фронтом сигнала подсистема Enabled and Triggered Subsystem

Подсистема **Enabled and Triggered Subsystem** (в дальнейшем **ET**-подсистема) включается фронтом сигнала поступающего на **T**-вход системы при наличии положительного сигнала на **E**-входе системы. Так же как и **Triggered Subsystem** эта подсистема выполняет вычисления только на том шаге моделирования, где произошло изменение управляющего сигнала на **T**-входе. Параметр **States when enabling** блока **Enable** не оказывает влияния на работу **ET**-подсистемы.

Оба управляющих сигнала могут быть векторными.

Пример **ET**-подсистемы дан на рис. 9.9.6.

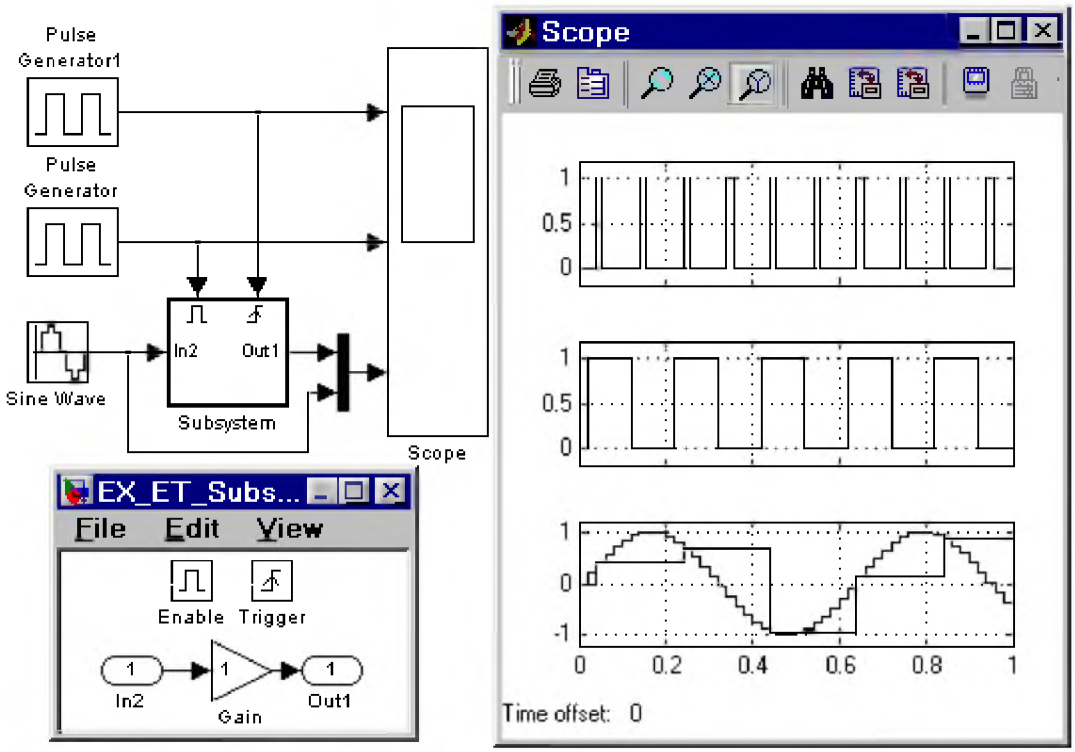


Рис. 9.9.6 Модель, использующая **ET**-подсистему

9.9.5. Управляемая S-функцией подсистема Function-call subsystem

Function-call subsystem (в дальнейшем **FC**-подсистема) является **T**-подсистемой, предназначенной для использования совместно с **S-функцией** написанной на языке **C.** Используя специальные средства, можно обеспечить выполнение подсистемы во время выполнения **S-функции.** На время выполнения **FC**-подсистемы работа **S-функции** останавливается, а по окончании выполнения **FC**-подсистемы работа **S-функции** возобновляется. Таким образом, **FC**-подсистема обеспечивает создание **S-функций,** запускающих подсистемы составленные из **Simulink**-блоков. Механизм создания таких **S-функций** описан в документации **Simulink**, посвященной созданию **S-функций**.

Для работы с **FC**-подсистемой можно использовать также **Function-Call Generator** и средства пакета событийного моделирования **Stateflow**.

9.9.6. Блок условного оператора If

Назначение:

Обеспечивает формирование управляющих сигналов для подсистем **If Action**

Subsystem. Блок является аналогом оператора **if-else** языка программирования **C**.

Параметры:

1. **Number of inputs** – Количество входов.

2. **If expression** – Условное выражение. Условное выражение может включать в себя следующие знаки: **<. <=, ==, ~=, >, >=, &, |, [ ],** а также унарный минус. Если записанное условное выражение истинно, то на выходном **If**-порту блока формируется управляющий сигнал.
3. **Elseif expressions** – Одно или список альтернативных условных выражений разделенных запятыми, вычисляющихся, если условное выражение **If expression** ложно. Каждому условному выражению, записанному в списке **Elseif expressions** соответствует выходной **Elseif**-порт на котором формируется управляющий сигнал, если соответствующее условное выражение истинно. При этом алгоритм вычисления альтернативных условных выражений таков, что если одно из альтернативных условных выражений окажется истинным, то следующие в списке выражения не проверяются. Альтернативное условное выражение может включать в себя те же знаки, что и выражение **If expression**.
4. **Show else condition** (флажок) – Показать **Else**-порт. На **Else**-порту формируется управляющий сигнал, если условное выражение и все альтернативные условные выражения ложны.

На пиктограмме блока отображаются условные выражения, записанные в его параметрах. Добавление каждого нового альтернативного условного выражения приводит к появлению нового **Elseif** выходного порта.

Если входные сигналы блока являются скалярами, то для их обозначения в выражениях используется запись вида **u1**, **u2** ,**u3** и т.д. Если входные сигналы векторные, то для обозначения элементов вектора используются выражения вида **u1(1)**, **u1(2)**, **u2(1), u2(2)** и т.д.

На рис. 9.9.7 показан пример использования блока **If** совместно с подсистемами **If Action Subsystem**. В примере первая подсистема пропускает через себя входной сигнал если входной сигал блока **If** больше **1**, вторая – если входной сигнал меньше **–1** (минус один), и третья – если входной сигнал лежит в интервале от **-1** до **+1**.

C-код, соответствующий алгоритму работы блока **If** в приведенном примере выглядит следующим образом:

```
if (u1 > 1) {
If Action Subsystem 1;
}
elseif (u1 < -1){
If Action Subsystem 2;
}
else {
If Action Subsystem 3;
}
```

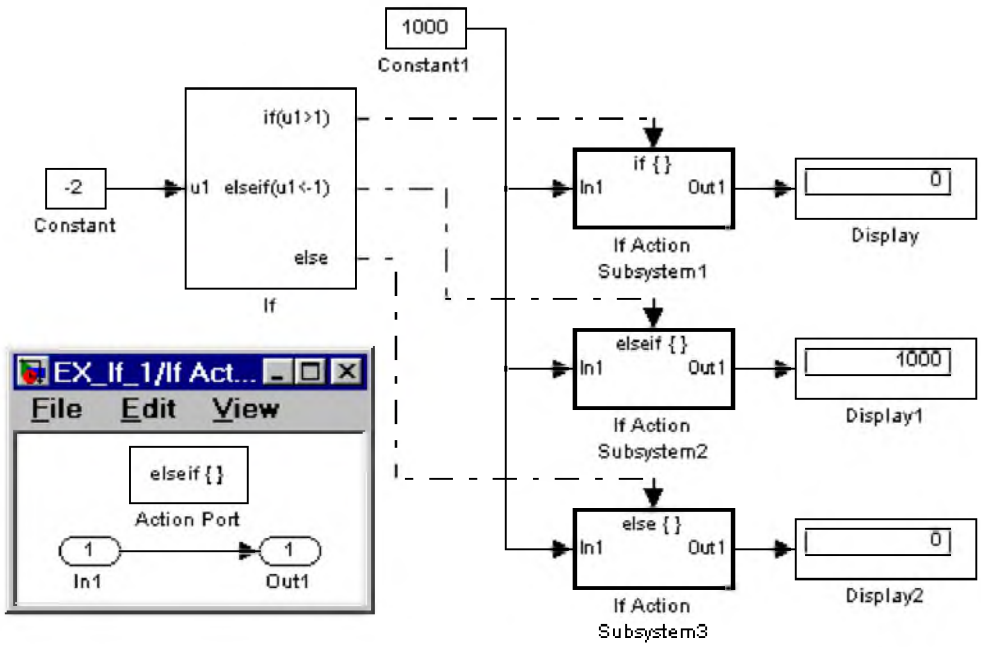


Рис. 9.9.7 Использование блока **If** совместно с подсистемами **If Action Subsystem**

9.9.7. Блок переключателя Switch Case

Назначение:

Обеспечивает формирование управляющих сигналов для подсистем **Case Action Subsystem**. Блок является аналогом оператора **Switch** языка программирования **C**.

Параметры:

1. **Case conditions** – Список значений входных сигналов (целое число). Каждому значению соответствует отдельный выходной **Case-**порт. Если значение входного сигнала, поступающего на вход блока **Switch Case**, совпадает с каким либо значением из списка, то на соответствующем выходе блока формируется управляющий сигнал. Если входной сигнал не является целым, то его дробная часть отбрасывается. В выражении **Case conditions** можно использовать квадратные скобки, если необходимо вырабатывать управляющий сигал на каком-либо порту для нескольких значений входного сигнала. Например, выражение **{1,[7,9]}** задает два выходных **Case-**порта. На первом из них управляющий сигнал формируется, если входной сигнал блока равен **1**, а на втором, – если входной сигнал равен **7** или **9**. В выражении **Case conditions** можно использовать также диапазоны значений. Например, выражение **{1:5}** определяет, что для единственного выходного **Case-**порта выходной сигал будет вырабатываться, если входной сигнал блока равен **1**, **2**, **3**, **4** или **5**.
2. **Show default case** (флажок) – Показать **default case-**порт. На выходе **default case-**порта формируется управляющий сигнал, если входной сигнал блока не совпадает ни с одним значением, перечисленным в списке **Case conditions**.

На рис. 9.9.8 показан пример использования блока **Switch Case** совместно с подсистемами **Switch Case Action Subsystem**. В примере первая подсистема пропускает через себя входной сигнал, если входной

сигал блока **Switch Case** равен **1**, вторая – если входной сигнал равен **–1** (минус один), и третья – если входной сигнал не равен ни **-1** ни **+1**.

C-код, соответствующий алгоритму работы блока **Switch Case** в приведенном примере выглядит следующим образом:

```
switch (u1) {
case 1:
Switch Case Action Subsystem 1;
break;
case -1:
Switch Case Action Subsystem 2;
break;
default:
Switch Case Action Subsystem 3;
}
```

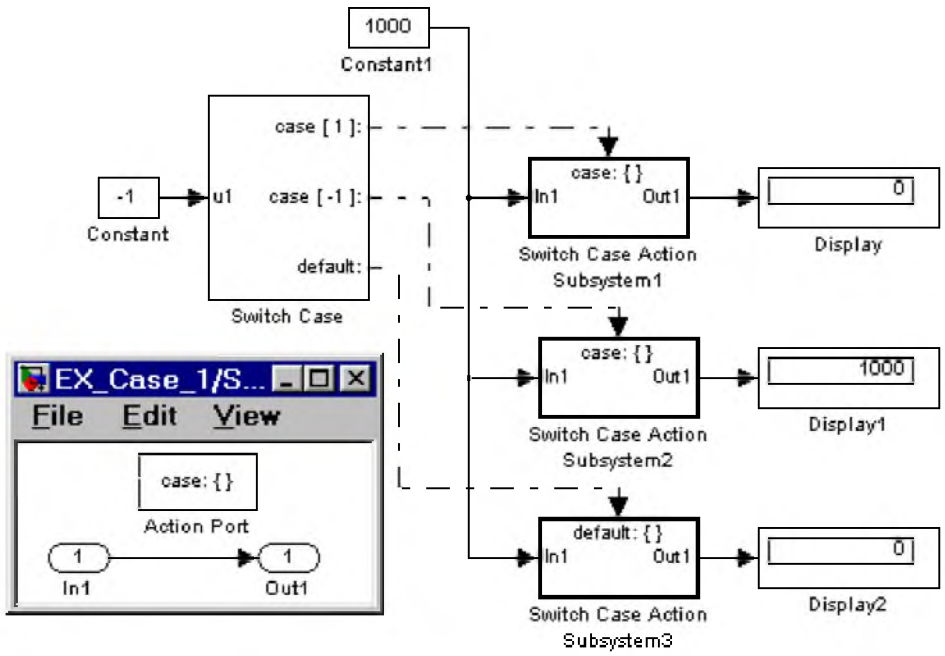


Рис. 9.9.8 Использование блока **Switch Case** совместно с подсистемами **Switch Case Action Subsystem**

9.9.8. Управляемая по условию подсистема Action Subsystem

Подсистема предназначена для работы под управлением блоков **If** или **Switch Case**. В первом случае она называется **If Action Subsystem**, а во втором **Switch Case Action Subsystem**.

Параметры подсистемы определяются настройками ее выходных портов, а также настройкой блока **Action Port**, наличие которого в подсистеме и превращает ее в **Action Subsystem**.

Блок имеет один *параметр* настройки:

States when execution is resumed – Состояние подсистемы системы при следующем возобновлении работы. Значение параметра выбирается из списка:

- **held** – Использовать предыдущее состояние (последнее состояние когда система была активна).
- **reset** – Использовать начальное (исходное) состояние.

Рассматриваемый параметр оказывает такое же действие на поведение подсистемы как параметр **States when enabling** блока **Enable**.

9.9.9. Управляемая подсистема For Iterator Subsystem

Управляемая подсистема **For Iterator Subsystem** представляет собой подсистему, которая выполняется неоднократно в течение одного такта моделирования. Количество повторений должно быть известно заранее и может задаваться внешним источником сигнала или с помощью параметра блока. Основные свойства подсистемы задает итерационный блок **For Iterator**. Блок является аналогом оператора цикла **For** языка программирования **C**.

Блока **For** может находиться в любом месте подсистемы. Его параметры перечислены ниже.

Параметры:

1. **States when starting** – Состояние подсистемы при следующем запуске. Значение параметра выбирается из списка:
    - **held** – Использовать предыдущее состояние (последнее состояние когда система была активна).
    - **reset** – Использовать начальное (исходное) состояние.
2. **Source of number of iterations** (флажок) – Источник задающий количество итераций.
    - **internal** – Внутренний.
    - **external** – Внешний.
3. **Number of iterations** – Количество итераций. Параметр доступен, если выбран внутренний источник числа итераций.
4. **Show iteration number port** – Отобразить на пиктограмме блока выходной порт, с которого снимается сигнал номера итерации.
5. **Output data type** – Тип данных выходного сигнала порта. Значение параметра выбирается из списка: **int32**, **int16**, **int8** и **double**.

На рис. 9.9.9 показан пример использования **For Iterator Subsystem**. В примере выполняется накопление суммы значений с шагом равным **10**. Количество итераций задается внешним источником и равно **20**.

C-код, соответствующий алгоритму работы **For Iterator Subsystem** в приведенном примере выглядит следующим образом:

```
sum = 0;
iterations = 20;
sum_increment = 10;
for (i = 0; i < iterations; i + +) {
sum = sum + sum_increment;
}
```

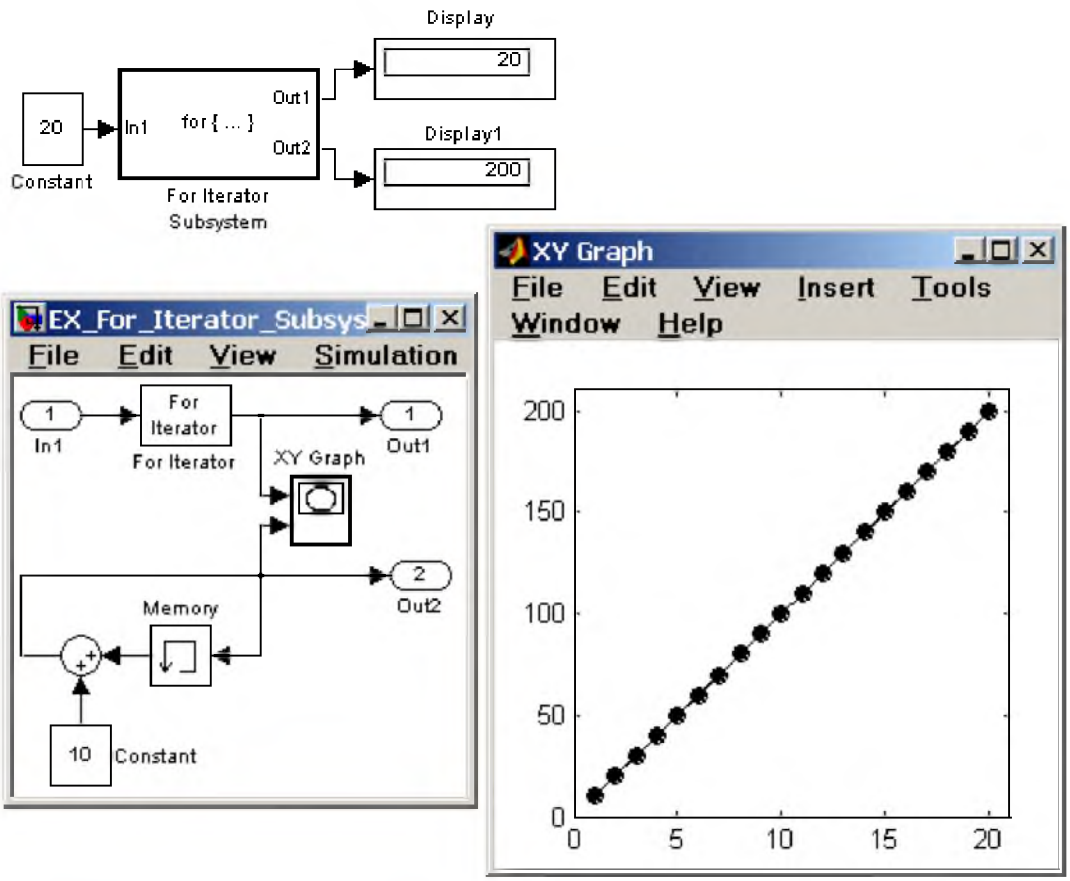


Рис. 9.9.9 Применение **For Iterator Subsystem**

Скачать пример (EX_For_Iterator_Subsystem_1.zip)

9.9.10. Управляемая подсистема While Iterator Subsystem

Управляемая подсистема **While Iterator Subsystem** представляет собой подсистему, которая выполняется неоднократно в течение одного такта моделирования. Количество повторений заранее не известно. Цикл прекращается, если значение логического сигнала на управляющем входе подсистемы станет равно **FALSE**. Основные свойства подсистемы задает итерационный блок **While Iterator**. Блок является аналогом оператора цикла **while (do-while)** языка программирования **C**.

Свойства **While Iterator Subsystem** определяются параметрами блока **While Iterator.** Его параметры перечислены ниже.

Параметры:

1. **Maximum number of iterations** – Максимальное количество итераций. Если значение параметра равно **-1** (минус один), то количество итераций не ограничивается.
2. **While loop type** (флажок) – Тип цикла. Выбирается из списка:
   - **while** – Цикл **while**.
   - **do-while** – Цикл **do-while**.

3. **States when starting** – Состояние подсистемы системы при следующем запуске. Значение параметра выбирается из списка:
    - **held** – Использовать предыдущее состояние (последнее состояние когда система была активна).
    - **reset** – Использовать начальное (исходное) состояние.
4. **Show iteration number port** – Отобразить на пиктограмме блока выходной порт, с которого снимается сигнал номера итерации.
5. **Output data type** – Тип данных выходного сигнала порта. Значение параметра выбирается из списка: **int32**, **int16**, **int8** и **double**.

Входной порт **IC** позволяет задать начальное значение сигнала прекращающего выполнение цикла **while**. При использовании цикла **do-while** подсистема будет выполнена хотя бы один раз (поскольку проверка условия в этом случае производится в конце цикла).

На рис. 9.9.10 показан пример использования **While Iterator Subsystem**. В примере выполняется накопление суммы значений с шагом равным **10**. Выполнение цикла прекращается, когда величина суммы достигнет значения **100**.

C-код, соответствующий алгоритму работы **While Iterator Subsystem** в приведенном примере выглядит следующим образом:

```
sum = 0;
IC = 1;
iteration_number = 0;
cond = IC;
while (cond != 0) {
iteration_number = iteration_number + 1;
sum = sum + sum_increment;
if (sum > 100 OR iterations > max_iterations) cond = 0;
}
```

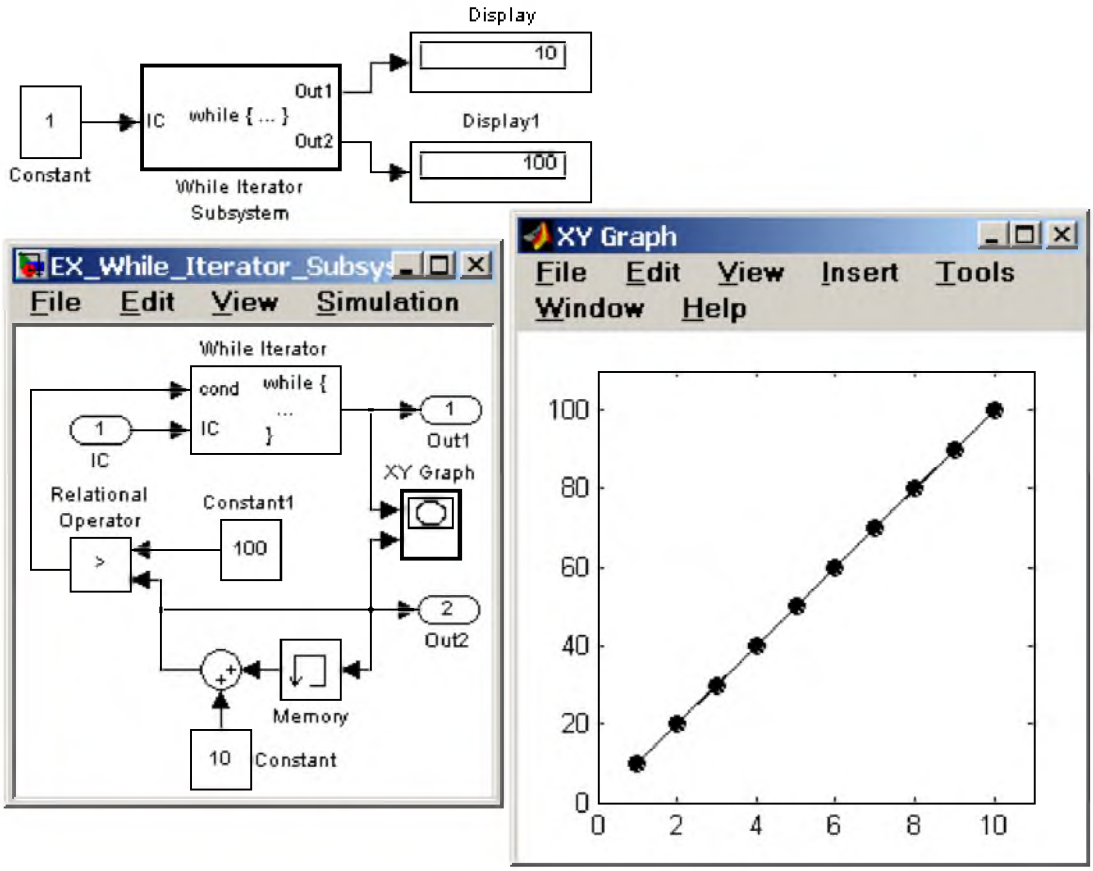


Рис. 9.9.10 Применение **While Iterator Subsystem**

9.9.11. Конфигурируемая подсистема Configurable Subsystem

Блок **Configurable Subsystem** позволяет создавать подсистему, обеспечивающую выбор конфигурации этой подсистемы. Например, в систему управления каким-либо объектом можно поставить конфигурируемую подсистему, наполнив ее различными вариантами регуляторов, и затем, перед проведением расчета, выбирать нужный вариант регулятора.

Для реализации такого механизма конфигурирования необходимо:

1. Создать библиотеку (**File/New/Library**).
2. Добавить в созданную библиотеку блок **Configurable Subsystem** и все необходимые варианты конфигурации подсистемы. Каждый из вариантов должен представлять собой стандартный блок **Simulink** либо маскированную подсистему (подсистему, имеющую собственное окно установки параметров).
3. Открыть **Configurable Subsystem** и выполнить ее настройку, отметив флажками нужные варианты и выбрав отображаемые входные и выходные порты подсистемы. Пример окна диалога **Configuration dialog** показан на рис. 9.9.11.

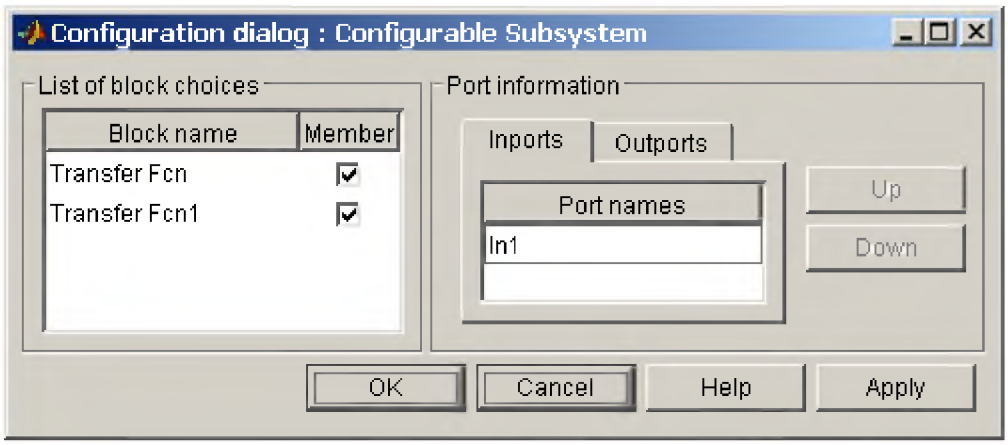


Рис. 9.9.11 Окно диалога **Configuration dialog**

4. Поместить в окно модели блок **Configurable Subsystem** из только что созданной библиотеки.
5. С помощью команды контекстного меню (вызывается нажатием правой клавиши мыши на объекте) **Block choice** (вариант блока) выбрать нужный вариант конфигурации. При открытии конфигурируемой подсистемы в окне модели будет автоматически открываться окно параметров того блока, который выбран командой **Block choice**.

Пример модели, использующей конфигурируемую подсистему, и библиотека конфигурируемой подсистемы показаны на рис. 9.9.12. В примере конфигурируемая подсистема состоит из апериодического и колебательного звеньев, которые могут выбираться при указании нужного варианта.

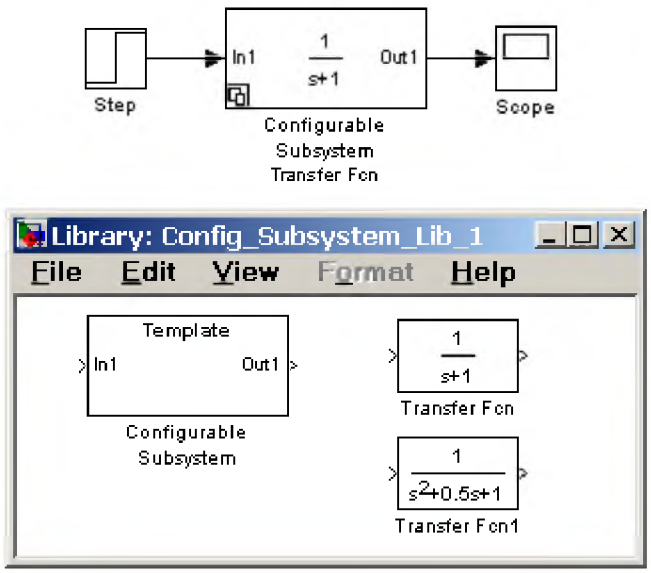


Рис. 9.9.12 Применение **Configurable Subsystem.**

9.10. Маскирование подсистем

9.10.1. Общие сведения

Механизм маскирования подсистем позволяет оформить подсистему как полноценный библиотечный блок, т.е. снабдить подсистему собственным окном параметров, пиктограммой, справочной системой и т.п.

Маскирование подсистем дает пользователю следующие преимущества:

1. Расширяет возможности пользователя по управлению параметрами модели.
2. Позволяет создавать более понятный интерфейс подсистемы.
3. Повышает наглядность блок-диаграммы.
4. Расширяет возможности построения сложных моделей.
5. Повышает защищенность модели от несанкционированной модификации.

Для выполнения маскирования имеющейся подсистемы необходимо предварительно выполнить следующие действия:

1. Определить какие параметры подсистемы должны задаваться пользователем в будущем окне параметров. Задать эти параметры в подсистеме с помощью идентификаторов (имен).
2. Определить каким образом параметр должен задаваться в окне диалога (с помощью строки ввода, выбором из раскрывающегося списка или установкой флажка).
3. Разработать эскиз пиктограммы блока.
4. Создать комментарии (справку) по использованию подсистемы.

Маскирование подсистемы выполняется с помощью **Mask Editor** (редактор маски). Для запуска редактора маски необходимо выделить маскируемую подсистему и выполнить команду **Mask Subsystem…** из меню **Edit**. Можно также воспользоваться контекстным меню. После запуска **Mask Editor** на экран будет выведено окно редактора (рис. 9.10.1), имеющее 3 вкладки: **Icon** (Пиктограмма), **Initialization** (Инициализация)**, Documentation** (Документация). Первая из вкладок обеспечивает создание пиктограммы подсистемы, вторая – дает возможность создать окно диалога для ввода параметров и третья – позволяет ввести описание блока и создать его справку.

В верхней части всех вкладок имеется поле **Mask Type**, с помощью которого можно задать имя блока. В нижней части окна имеется 5 кнопок управления редактором:

1. **OK** – Сохранить внесенные изменения и закрыть окно.
2. **Cancel** – Отменить внесенные изменения и закрыть окно.
3. **Unmask** – Снять маску с подсистемы. До закрытия файла модели маску можно восстановить, воспользовавшись командой **Edit Mask…** из меню **Edit**.
4. **Help** – Открыть окно справки редактора маски.
5. **Apply** – Сохранить внесенные изменения без закрытия окна редактора.

Повторный вызов редактора маски для уже маскированной подсистемы осуществляется командой **Edit Mask…** из меню **Edit** (или аналогичной командой из контекстного меню).

После того как маскирование системы будет выполнено, двойной щелчок на ее изображении будет открывать окно параметров подсистемы, а не окно модели. Открыть саму подсистему (окно модели) для редактирования или просмотра можно командой **Look under mask** из меню **Edit** или контекстного меню.

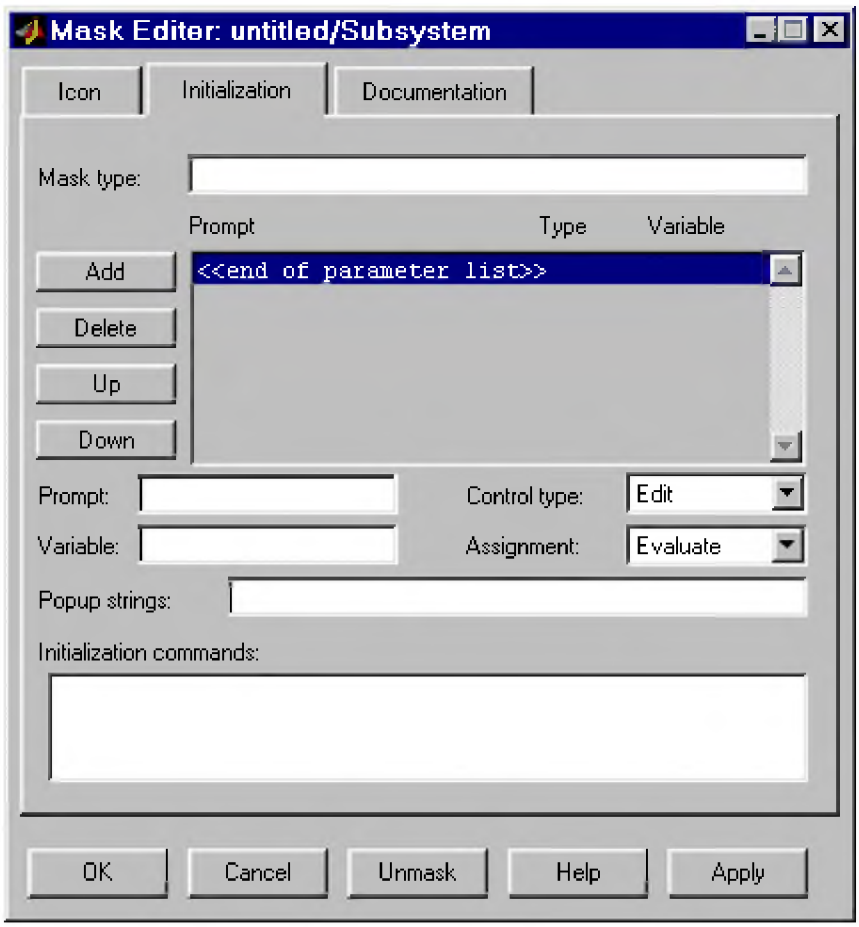


Рис. 9.10.1 Окно редактора маски **Mask Editor**

9.10.2. Создание окна параметров

Окно параметров создается с помощью вкладки **Initialization** (Инициализация) редактора маски. Для создания поля ввода параметра с его описанием необходимо выполнить следующие действия:

1. Нажать кнопку **Add** (Добавить).
2. Ввести описание параметра в поле **Prompt** (Подсказка). В качестве описания параметра обычно используется его название в виде текста, например, “**Gain**”, “**Constant value**” и т.п.
3. Указать идентификатор параметра в поле **Variable** (Переменная). Естественно, что это должен быть один из тех идентификаторов, который использовался при задании параметров блоков внутри подсистемы (хотя это не обязательно, поскольку параметр может быть использован и для модификации самого окна диалога). Все переменные, идентификаторы которых заданы на вкладке **Initialization**, помещаются в **Mask Workspace** – локальную рабочую область маски и являются доступными только внутри подсистемы.
4. Выбрать тип элемента интерфейса задающего параметр из списка **Control Type**:
   - **Edit** – Редактируемое поле ввода.
   - **Checkbox** – Флажок.
   - **Popup** – Раскрывающийся список. В этом случае в графе **Popup Strings** (Элементы списка) необходимо ввести элементы списка, разделенные символом вертикальной черты. Например, выражение **alpha**|**beta**|**gamma** задаст список из трех элементов: **alpha**,**beta** и **gamma**.
5. Выбрать формат параметра из списка **Assiggment**:

- **Evaluate** – Вычисляемый. Выбирается, если параметр должен иметь числовое значение. В данное поле можно будет ввести выражение в соответствии с правилами языка **MATLAB**. Формат **Evaluate** позволяет также использовать числовую форму значения переменной в том случае, если тип элемента интерфейса выбран в виде флажка или раскрывающегося списка. Так, например, для раскрывающегося списка **alpha|beta|gamma** значение связанной со списком переменной будет равно **1**, если в списке выбрано **alpha**, **2** – если в списке выбрано **beta**, и **3** – если в списке выбрано **gamma**. Для элемента интерфейса **Checkbox** вычисляемые значения будут равны **1** (при установленном флажке) и **0** (при снятом флажке).
- **Literal** – Текстовый. Выбирается, если параметр должен быть строкой символов.

6. Ввести команды инициализации в графе **Initialization commands**. Команды инициализации представляют собой обычные команды на языке **MATLAB** и могут включать операторы и **m**-функции. Такие команды задают переменные, которые будут находиться в рабочей области маскированной подсистемы. Эти переменные доступны внутри подсистемы и могут быть использованы в качестве параметров блоков входящих в состав подсистемы, а также для создания пиктограммы подсистемы. Команды инициализации выполняются в следующих случаях:
   - При открытии окна модели.
   - При запуске модели на выполнение.
   - При выполнении команды **Edit/Update diagram**.
   - При вращении блока маскированной подсистемы (в этом случае команды инициализации обеспечивают перерисовку пиктограммы).
   - При автоматическом изменении пиктограммы, зависящей от параметров блока.

В качестве примера маскированной подсистемы рассмотрим функциональный генератор. Схема модели генератора показана на рис. 9.10.2.

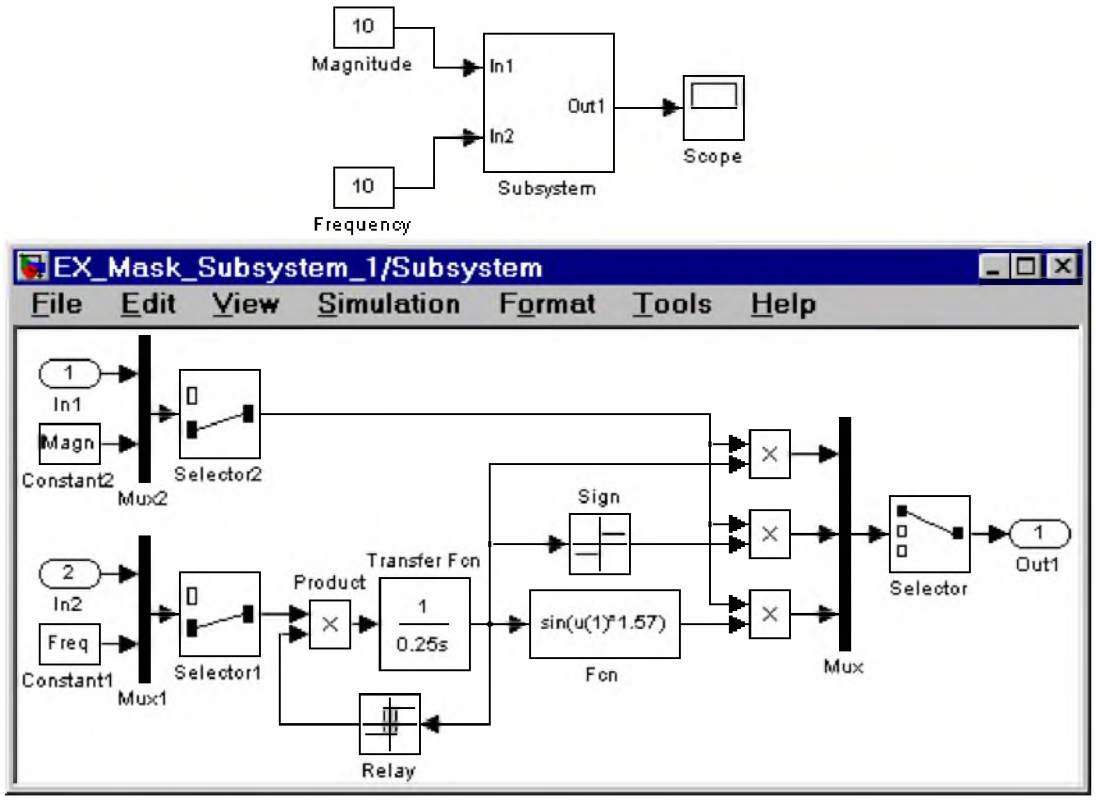


Рис. 9.10.2 Функциональный генератор.

Модель генератора обладает следующими возможностями:

1. Значения амплитуды и частоты сигнала могут задаваться либо как параметры генератора в его окне диалога, либо от внешних источников через входные порты.
2. Форма выходного сигнала генератора (треугольник, прямоугольник или синусоида) задается в окне диалога.

Вид окна диалога, созданного с помощью редактора маски показан на рис. 9.10.3.

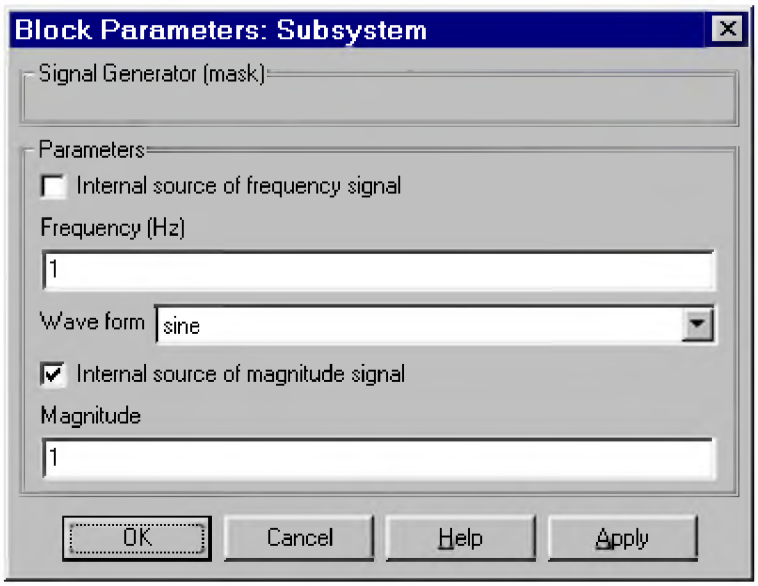


Рис. 9.10.3 Окно параметров генератора

Название параметра, идентификатор связанной с ним переменной, тип элемента интерфейса и формат параметра приведены в таблице 9.10.1.

Таблица 9.10.1.

N	**Prompt**	**Variable**	**Control Type**	**Assiggment**	**Назначение**
1	Internal source of frequency signal	Internal_freq	Checkbox	Evaluate	Задает тип источника сигнала задания на частоту: внутренний или внешний.
2	Frequency (Hz)	Freq	Edit	Evaluate	Задает величину задания на частоту внутреннего источника
3	Wave	Wave_form	Popup	Evaluate	Задает форму

	form				выходного сигнала: треугольник, прямоугольник или синусоида
4	Internal source of magnitude signal	Internal_magn	Checkbox	Evaluate	Задает тип источника сигнала задания на амплитуду: внутренний или внешний.
5	Magnitude	Magn	Edit	Evaluate	Задает величину задания на амплитуду внутреннего источника

Окно редактора маски с открытой вкладкой **Initialization**, в котором создано окно параметров генератора, показано на рис. 9.10.4.

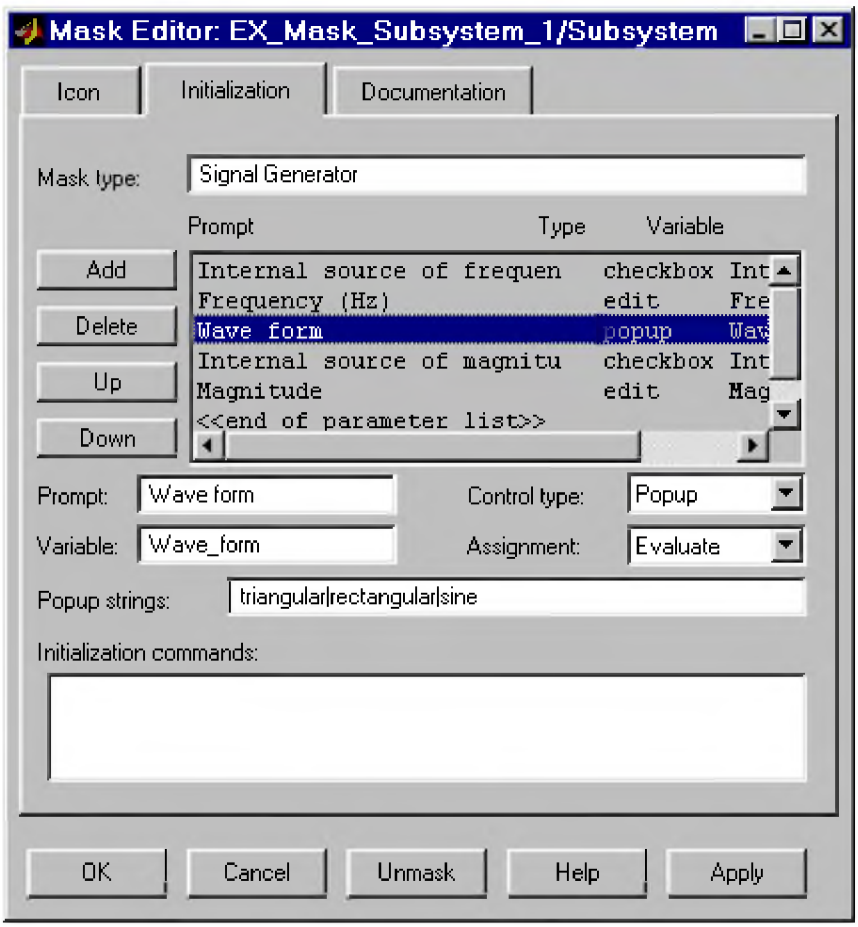


Рис. 9.10.4 Окно редактора маски на этапе создания окна параметров

Выбор типа источников задания на частоту (внутренний или внешний) осуществляется с помощью блока **Selector1** (см. рис. 9.10.2). Для этого значение параметра **Elements** блока **Selector1** задано как **[(Internal_freq+1)]**. Таким образом, если флажок параметра **Internal source of frequency signal** установлен, то числовое значение переменной **Internal_freq** равно **1** и на выход селектора проходит сигнал от внутреннего источника, если же флажок снят, то на выход селектора проходит сигнал от входного порта системы (т.е. от внешнего по отношению к генератору источника). Аналогичным образом с помощью переменной **Internal_magn** выполняется выбор источника сигнала задания на амплитуду.

Выбор формы выходного сигнала выполняется также с помощью блока **Selector**. Треугольный, прямоугольный и синусоидальный сигналы объединяются в вектор с помощью блока **Mux**, а затем в зависимости от числового значения переменной **Wave_form**, блок **Selector** выполняет выбор нужного элемента входного вектора. Значение параметра **Elements** блока **Selector** задано как **[Wave_form]**. Таким образом, если, например, параметр генератора **Wave form** имеет значение **Sine**, то числовое значение переменной **Wave_form** равно **3**, и, следовательно, на выход селектора проходит третий элемент входного вектора, т.е. синусоидальный сигнал.

9.10.3. Создание пиктограммы подсистемы

Пиктограмма подсистемы создается с помощью вкладки **Icon** (Пиктограмма) редактора маски. Окно редактора маски с открытой вкладкой **Icon** показано на рис. 9.10.5.

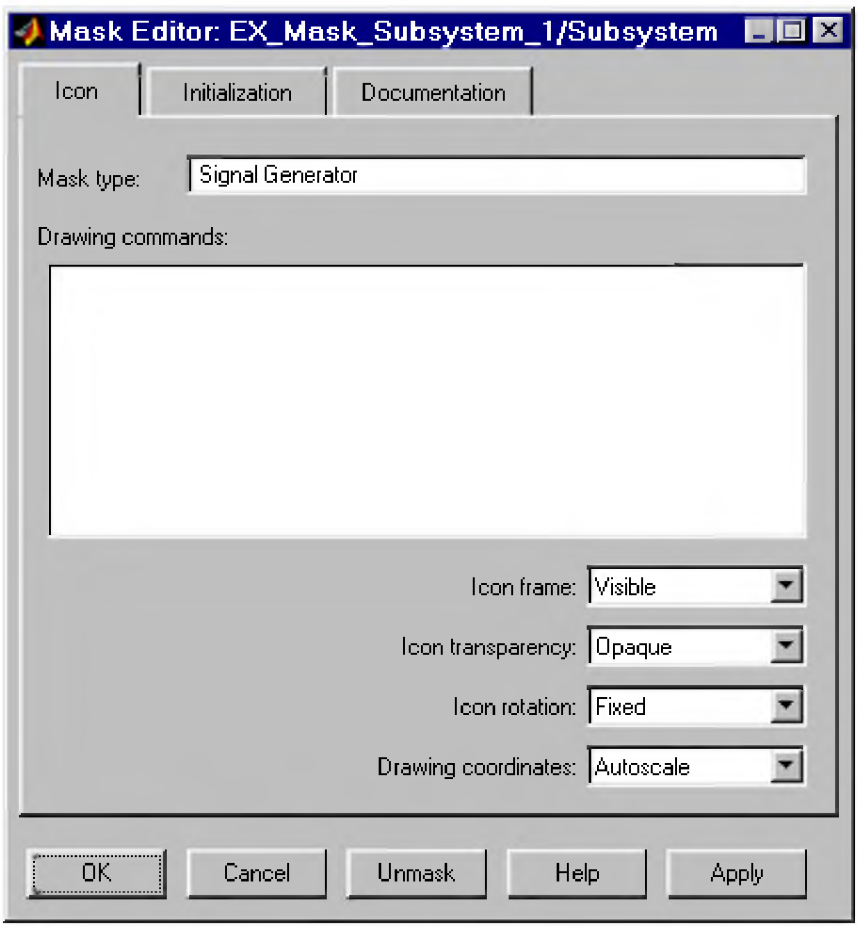


Рис. 9.10.5 Вкладка **Icon** редактора маски

Вкладка содержит следующие элементы:

1. **Drawing commands** – Область ввода команд рисования. Команды рисования являются выражениями допустимыми в языке **MATLAB**.
2. **Icon frame** – Список позволяющий выбрать способ отображения рамки пиктограммы:
   - **Visible** – Рамка видна.
   - **Invisible** – Рамка не видна.
3. **Icon transparency -** Список позволяющий установить прозрачность пиктограммы:
   - **Opaque** – Пиктограмма не прозрачна.
   - **Transparent**– Пиктограмма прозрачна.
4. **Icon rotation -** Список позволяющий задать возможность вращения пиктограммы:
   - **Fixed** – Положение пиктограммы фиксировано.
   - **Rotates** – Пиктограмма может вращаться вместе с блоком.
5. **Drawing coordinates** – Список, задающий условия масштабирования пиктограммы.
   - **Autoscale** – Автоматическое масштабирование. Рисунок занимает максимально возможную площадь внутри пиктограммы.
   - **Normalized** – Нормализованное масштабирование. Координаты левого нижнего угла пиктограммы **(0,0)**, координаты правого верхнего угла **(1,1)**.
   - **Pixel** – Координаты рисунка задаются в пикселах.

9.10.3.1. Команды вывода текста

Для вывода текста могут использоваться следующие команды:

- **disp('text')** или **disp(variablename)** – Вывод текста **'text'** или значения символьной переменной **variablename** в центре пиктограммы.
- **text(x, y, 'text')** или **text(x, y, variablename)** – Вывод текста **'text'** или значения символьной переменной **variablename** начиная с позиции, заданной координатами **x** и **y**.
- **text(x, y, 'text', 'horizontalAlignment', halign,'verticalAlignment', valign)**-Вывод текста **'text'** в позиции заданной координатами **x** и **y** и с указанием способов выравнивания относительно этой позиции по вертикали или горизонтали. Параметр **halign** может принимать значения: **'left', 'right'** или **'center'**. Параметр **valign** может принимать значения: **'base'**, **'bottom'** или **'middle'**.
- **fprintf('text')** или **fprintf('format', variablename)** – Форматированный вывод (по правилам языка **C**) текста **'text'** или значения символьной переменной **variablename** в центре пиктограммы.
- **port_label(port_type, port_number, label)** – Вывод на пиктограмме метки порта. Например, выражение **port_label('input', 1, 'a')** выводит на пиктограмме метку **a** первого входного порта.

Для вывода текста в несколько строк допускается использование сочетания символов **\n** для перехода на новую строку.

Примеры маскированных подсистем с различными вариантами текстовых надписей даны на рис. 9.10.6. Значения текстовых переменных заданы на вкладке **Initialization** в графе **Initialization commands**.

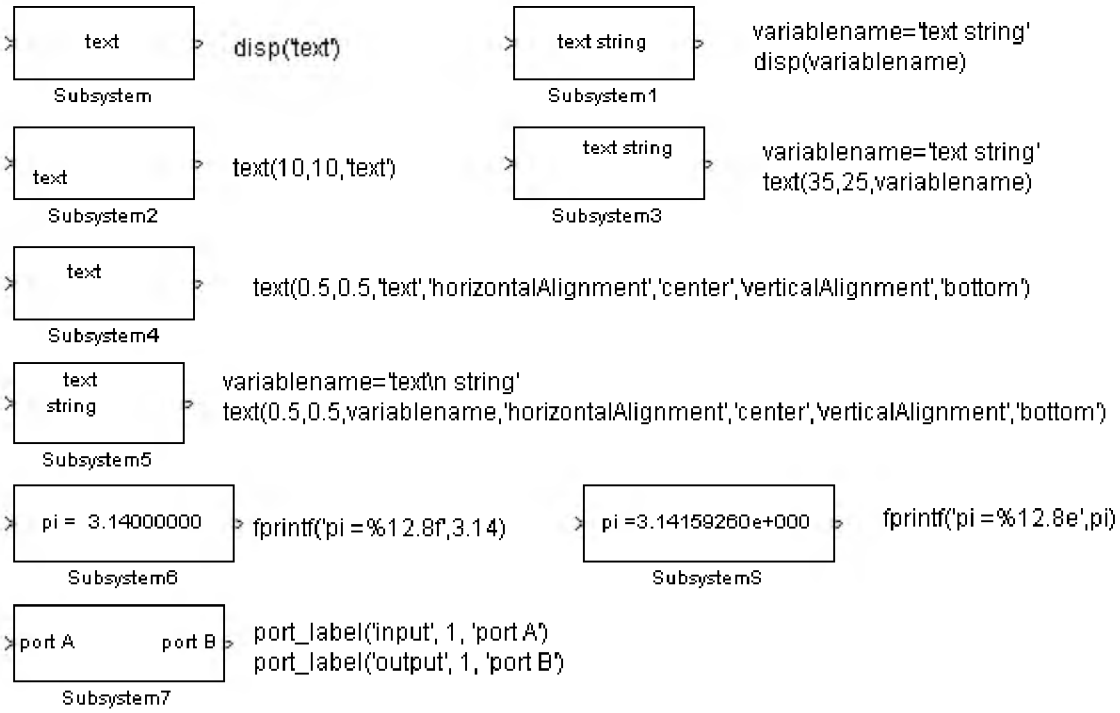


Рис. 9.10.6. Варианты текстовых надписей на пиктограммах

9.10.3.2. Команды построения графиков

Для построения графиков на пиктограмме могут использоваться следующие команды:

- **plot(Y)** – В том случае, если **Y** является вектором, то строится график по оси абсцисс которого откладывается значение индекса элемента, а по оси ординат значение самого элемента. В том случае если **Y** является матрицей – строятся линии для каждого столбца. По оси абсцисс в этом случае также откладывается значение индекса элемента.
- **plot(X1,Y1,X2,Y2,...)** – Строится графики вида **Y1(X1)**, **Y2(X2)** и т.д.

Примеры маскированных подсистем с различными вариантами графиков представлены на рис. 9.10.7. Значения переменных заданы на вкладке **Initialization** в графе **Initialization commands**.

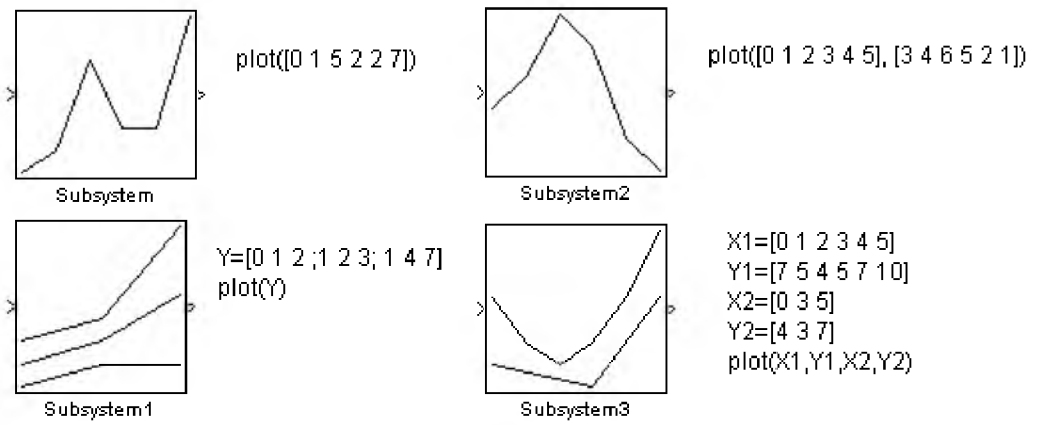


Рис. 9.10.7. Варианты графиков на пиктограммах

9.10.3.3. Команды отображения передаточных функций

Для отображения на пиктограмме передаточной функции используются следующие команды:

- **dpoly(num, den)** – Отображение дробно-рациональной передаточной функции (**num** – вектор коэффициентов числителя, **den** – вектор коэффициентов знаменателя). Оператор Лапласа будет отображен с помощью символа **s**.
- **dpoly(num, den, 'character') -** Отображение дробно-рациональной передаточной функции. Оператор Лапласа будет отображен с помощью символа **character**.
- **dpoly(num, den, 'z')** Отображение дискретной дробно-рациональной передаточной функции.
- **dpoly(num, den, 'z-') -** Отображение дискретной дробно-рациональной передаточной функции от обратного аргумента.
- **droots(z, p, k) -** Отображение **Zpk**-формы передаточной функции. Для рассматриваемого выражения может быть добавлен четвертый аргумент в виде **'z'** или **'z-'** для отображения дискретных передаточных функций.

Примеры маскированных подсистем с различными вариантами отображения передаточных функций показаны на рис. 9.10.8.

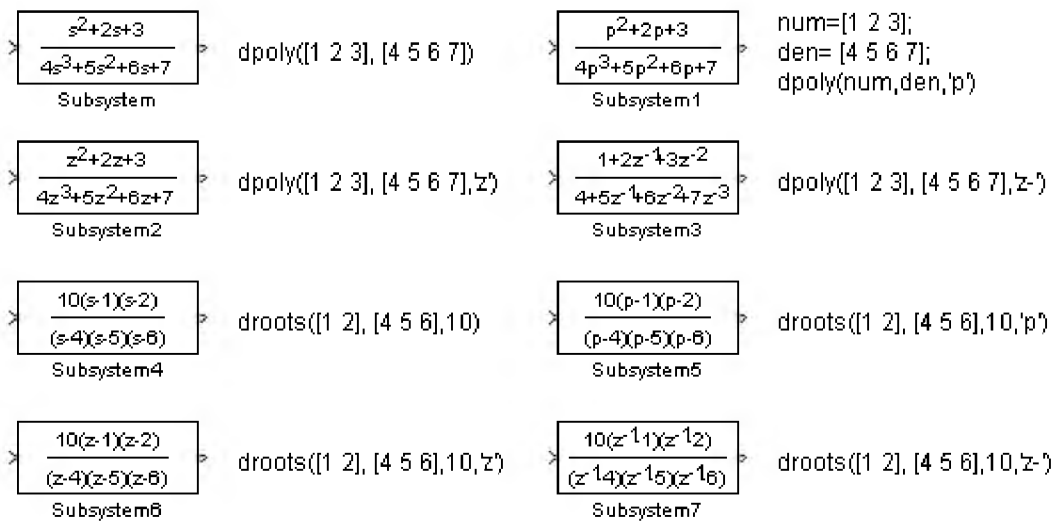


Рис. 9.10.8. Варианты отображения передаточных функций на пиктограммах

9.10.3.4. Команды отображения рисунка из графического файла

Для отображения на пиктограмме рисунка из графического файла используются следующие команды:

- **image(imread('filename'))** – Отображение рисунка из файла с полным именем **filename**. Для правильной работы этой команды необходимо поместить рисунок в туже папку, где находится файл модели, и сделать эту папку рабочей. Допускается также совместно с именем файла указывать его полный путь.

- **image(a, [x, y, w, h])** – Отображение рисунка содержащегося в переменной **a**. Ширина и высота рисунка задаются параметрами **w** и **h**, соответственно. Левый нижний угол рисунка расположен в точке с координатами **x**,**y**. Считывание рисунка из файла может быть выполнено командой **a = imread('filename'))**.
- **image(a, [x, y, w, h], rotation)** – Команда аналогичная предыдущей, но позволяющая задавать поведение рисунка при вращении пиктограммы. Значение параметра **rotation** равное **'on'** позволяет поворачивать рисунок вместе с пиктограммой подсистемы.
- **patch(x, y)** – Отображение закрашенного многоугольника, координаты которого заданы векторами **x** и **y**. Цвет рисунка – черный.
- **patch(x, y, [r g b])** - Команда аналогичная предыдущей, но позволяющая задавать цвет рисунка. Параметры **r**,**g** и **b** задают соотношение красного, зеленого и синего цветов в рисунке. Значение параметров должно находиться в пределах от **0** до **1**.

Примеры маскированных подсистем с различными вариантами команд отображения рисунков показаны на рис. 9.10.9.

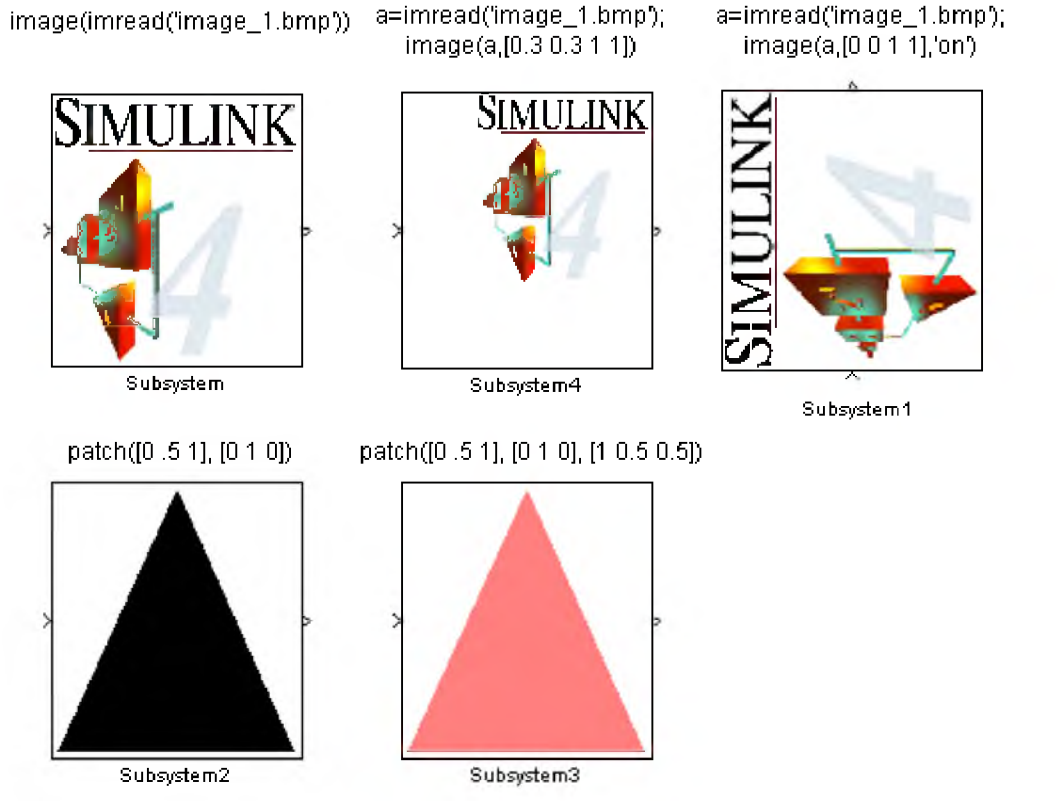


Рис. 9.10.9. Варианты отображения рисунков на пиктограммах.

9.10.3.5. Использование редактора пиктограмм iconedit

Для создания пиктограмм можно также использовать редактор пиктограмм **iconedit**. Для его вызова используется команда:

iconedit('modelname','Subsystem'),

где **modelname** – имя файла модели (без расширения),

Subsystem – имя подсистемы, для которой будет создаваться пиктограмма.

Пиктограмма создается по точкам, расположение которых указывается с помощью мыши. Между собой точки соединяются прямыми линиями. Для того, чтобы начать новую линию необходимо нажать клавишу **n** на клавиатуре. Для отмены создания последней точки используется клавиша **d**. Выход из редактора с автоматическим обновлением пиктограммы осуществляется клавишей **q**. По завершении работы с редактором необходимо также закрыть его окно рисования. Кроме обновления пиктограммы завершение работы с редактором пиктограмм сопровождается выводом в командной строке **MATLAB** графической команды, обеспечивающей построение пиктограммы.

Пример пиктограммы, созданный с помощью **iconedit**, его окно рисования, а также текст команды, обеспечивающей построение пиктограммы, показаны на рис. 9.10.10.

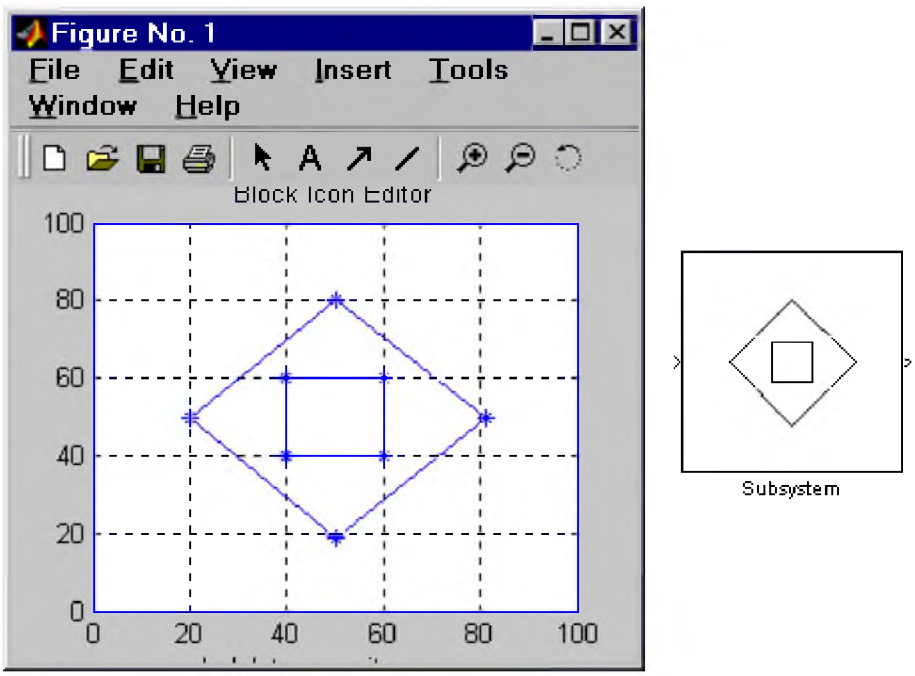


Рис. 9.10.10 Создание пиктограммы с помощью **iconedit**

9.10.3.6. Создание автоматически обновляемых пиктограмм

Создание автоматически обновляемой пиктограммы рассмотрим на примере функционального генератора (рис. 9.10.2). Генератор может вырабатывать сигнал трех видов: треугольный, прямоугольный и синусоидальный. Вполне логично было бы создать такую пиктограмму, на которой отображалась бы форма выбранного на текущий момент сигнала. Это достаточно легко сделать, поскольку за выбор формы сигнала в рабочей области маски отвечает переменная **Wave_form**. Числовое значение этой переменной равное **1** соответствует треугольному сигналу на выходе генератора, значение равное **2** соответствует прямоугольному сигналу, и **3** – синусоидальному.

Реализация поставленной задачи обеспечивается указанными ниже командами, которые необходимо ввести в графе **Initialization commands** редактора маски:

```
switch Wave_form
   case 1
% треугольный сигнал
   x=[-6.28 -4.71 -1.57 1.57 4.71 6.28 ];
   y=[0 1 -1 1 -1 0];

   case 2
% прЯмоугольный сигнал
   x=[-6.28 -6.28 -3.14 -3.14 0 0 3.14 3.14 6.28 6.28 ];
   y=[0 1 1 -1 -1 1 1 -1 -1 0];

   case 3
% синусоидальный сигнал
   x=(-314*2:314*2)/100;
   y=sin(x);
end;
```

Примечание: Здесь и в дальнейшем в текстах на языке **MATLAB** включены комментарии на русском языке, которые необходимо удалить при составлении выражений в среде **MATLAB**.

В зависимости от значения переменной **Wave_form** векторам **x** и **y** присваиваются разные значения, благодаря чему команда построения графика **plot(x,y);** , указанная в графе **Drawing commands** строит разные графики.

Пример, показывающий все три варианта пиктограммы генератора, представлен на рис. 9.10.11. В примере, дополнительно, строятся оси координат с помощью следующих команд:

```
plot([-6.28 -6.28],[1.2 -1.2]);
plot([-6.28 8],[0 0]);
```

Также в графе графе **Drawing commands** введена команда, рисующая одну точку в левом нижнем углу пиктограммы:

```
plot([-10,-10],[-1.2 -1.2]);
```

С помощью этой команды достигается относительное смещение графиков вправо. Таким образом в левой части пиктограммы появляется дополнительное свободное место для отображения меток входных портов (управление портами маскированной подсистемы будет рассмотрено позже).

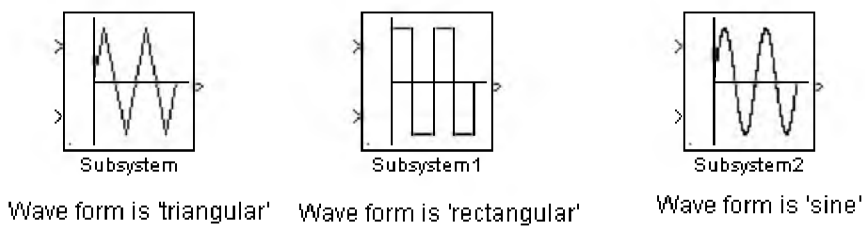


Рис. 9.10.11 Варианты пиктограммы функционального генератора.

9.10.4. Создание справки маскированной подсистемы

Для создания описания и справки маскированной подсистемы служит вкладка **Documentation** (Документация). Вкладка **Documentation** содержит две графы: **Block description** (Описание блока) и **Block Help** (Справка по блоку). Внешний вид редактора маски с открытой вкладкой **Documentation** показан на рис. 9.10.12.

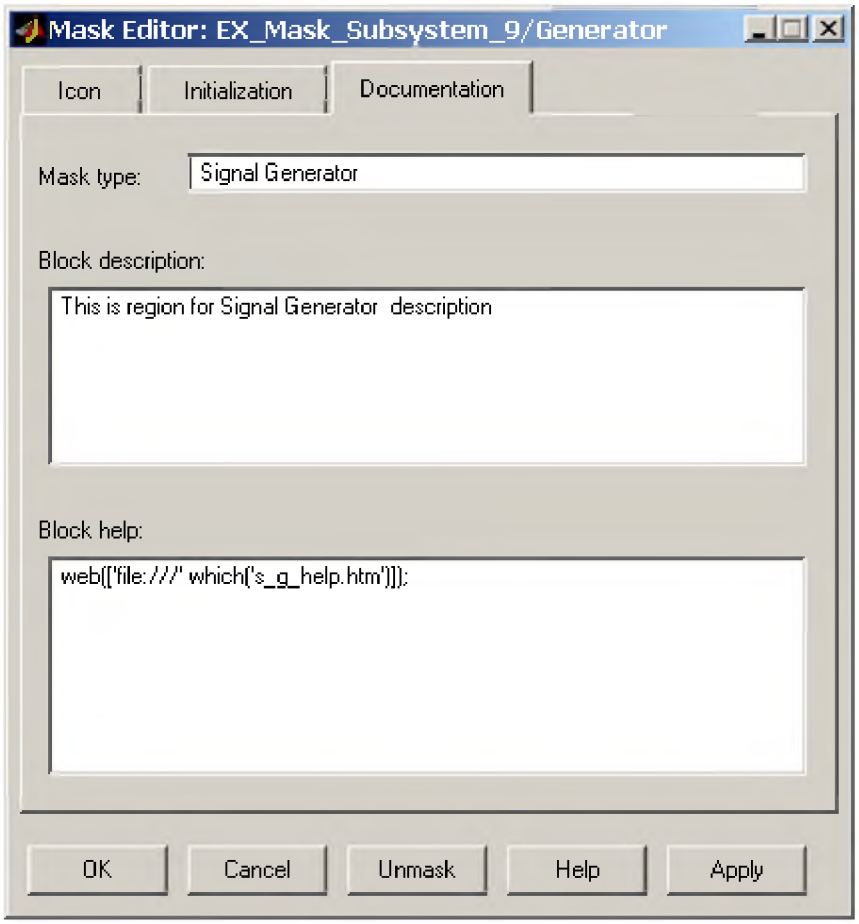


Рис. 9.10.12 Вкладка **Documentation** редактора маски

Текст, введенный в графу **Block description**, отображается в верхней части окна диалога и предназначен для краткого описания блока. В графу **Block Help** вносятся команды обеспечивающие загрузку файлов справки, созданных пользователем, в справочную систему при нажатии клавиши **Help** в окне параметров. Эти команды описаны в документации по **Simulink**. Наиболее удобным форматом файла справки является **htm** (**html**) – формат. Вызов справочного **htm**-файла осуществляется командой вида:

web(['file:///' which('helpfile.htm')]); ,

где

helpfile.htm – имя файла справки.

Для правильной работы справочной системы необходимо, чтобы файл справки находился в той же папке, что и файл модели, и данная папка являлась рабочей. Допускается также вместе с именем файла указывать его полный путь.

Пример окна справки подсистемы показан на рис. 9.10.13.

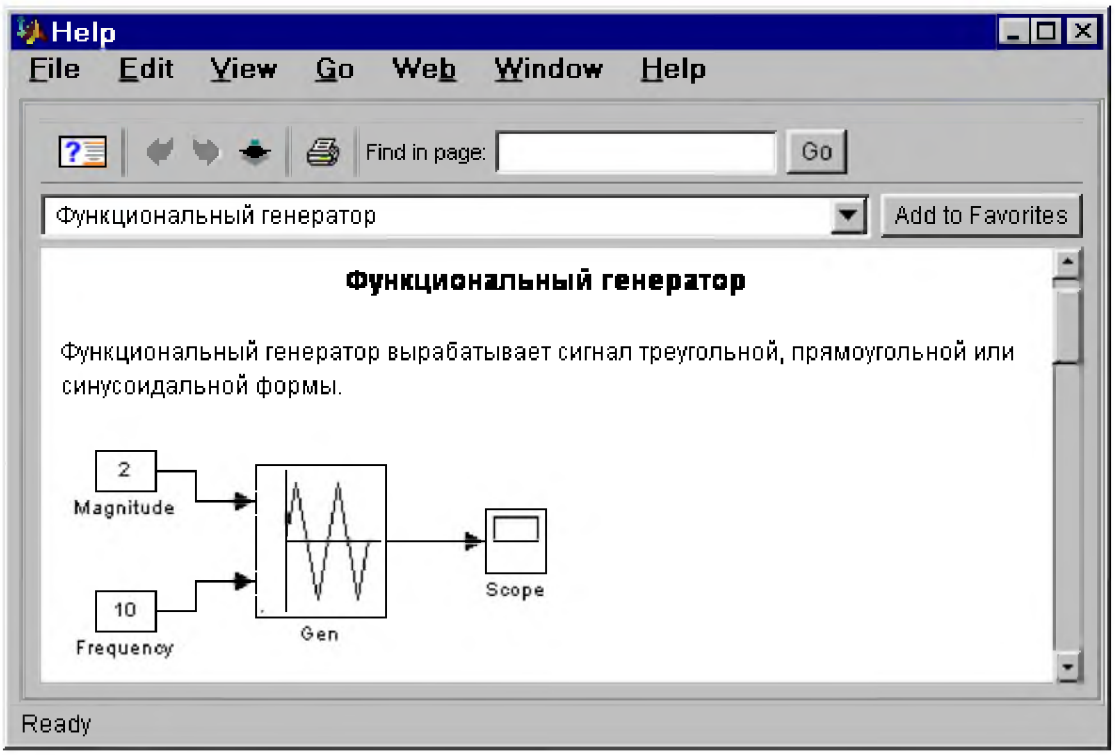


Рис. 9.10.13.Пример окна справки подсистемы

9.10.5. Создание динамически обновляемых окон диалога

Динамически обновляемое окно диалога это такое окно, внешний вид которого изменяется в зависимости от значения параметров заданных в самом окне. Например, для рассматриваемого в данной главе функционального генератора, в случае выбора внешних источников сигналов задания на частоту или амплитуду, графы, в которые вводятся значения частоты и амплитуды, могут отсутствовать или быть не активными. Для создания такого окна необходимо:

1. Выделить блок и ввести в командном окне MATLAB следующее выражение:

set_param(gcb, 'MaskSelfModifiable', 'on') . После чего модель необходимо сохранить. Данная команда дает разрешение на самомодификацию окна.

2. Ввести в командном окне команду вида:

set_param(gcb,'MaskCallbacks',{'parm1_callback', '','parm3_callback'}); ,

где в фигурных скобках указываются функции обрабатывающие событие изменения параметра. В данном примере функция **parm1_callback** обрабатывает событие при изменении первого параметра, а

функция **parm3_callback** обрабатывает событие при изменении третьего параметра. В том случае, если для какого-либо параметра такая обработка не нужна, функция не записывается, но два апострофа для данного параметра (пустая функция) все равно должны указываться. В данном примере обработка события для второго параметра отсутствует. Сама функция может быть любым допустимым выражением на языке **MATLAB**.

Применительно к рассматриваемому функциональному генератору эта команда выглядит следующим образом:

set_param(gcb,'MaskCallbacks',{'call_back_freq','','','call_back_magn',''}); .

Функция **call_back_freq** обрабатывает событие при установке или снятии флажка параметра **Internal source of frequency signal** (тип источника сигнала задания на частоту, переменная **Internal_freq**), а функция **call_back_magn** обрабатывает событие при установке или снятии флажка параметра **Internal source of magnitude signal** (тип источника сигнала задания на амплитуду, переменная **Internal_magn**).

После этого модель необходимо сохранить.

3. Разработать функции обработки.

Для рассматриваемого примера функция **call_back_freq** (файл **call_back_freq.m**) выглядит следующим образом:

```
Freq_param=get_param(gcb,'Internal_freq'); % Присвоение переменной Freq_param значениЯ
                % параметра Internal_freq (тип источника % сигнала заданиЯ на частоту: внутренний
или
                % внешний). Параметр Internal_freq является
                % вторым в списке параметров окна диалога.
if strcmp(Freq_param,'on'); % Если значение переменной Freq_param есть 'on'
                % (внутренний источник сигнала заданиЯ на частоту), то
   enable={'on','on','on','on','on'}; % всем элементам вектора enable присваиваются
                % значениЯ равные 'on' (все параметры окна диалога
                % должны быть активны).
else; % Если значение переменной Freq_param не равно 'on'
        % (внешний источник сигнала заданиЯ на частоту), то
   enable={'on','off','on','on','on'}; % второму элементу вектора enable присваивается
                % значение 'off' (второй параметр должен быть не % активным).
end; % Завершение конструкци if … else
set_param(gcb,'MaskEnables',enable); % Присвоение параметру маскированной подсистемы
                % MaskEnables значениЯ вектора enable.
                % Параметр MaskEnables устанавливает режим
                % активности параметров окна диалога маскированной
                % подсистемы.
```

Функция проверяет значение параметра **Internal_freq**. Если значение этого параметра есть **'on'**, то вектор **enable** имеет все элементы равные **'on'**, если же значение параметра **Internal_freq** равно **'off'** (используется внешний источник сигнала задания на частоту), то *второй* элемент вектора **enable** имеет значение **'off'** и функция **set_param(gcb,'MaskEnables',enable);** сделает *не активной* графу для ввода второго параметра (частота внутреннего источника).

Функция **call_back_magn** (файл **call_back_magn.m**) выглядит следующим образом:

```
Magn_param=get_param(gcb,'Internal_magn'); % Присвоение переменной Magn_param
                % значениЯ параметра Internal_magn (тип
                % источника сигнала заданиЯ на амплитуду:
                % внутренний или внешний).
                % Параметр Internal_magn ЯвлЯетсЯ пЯтым
                % в списке параметров окна диалога.
if strcmp(Magn_param,'on'); % Если значение переменной Magn_param есть 'on'
                % (внутренний источник сигнала заданиЯ на амплитуду),
   visible={'on','on','on','on','on'}; % то всем элементам вектора visible присваиваютсЯ
                % значениЯ равные 'on' (все параметры окна диалога
                % должны быть видимы).
else; % Если значение переменной Magn_param не равно 'on'
              % (внешний источник сигнала заданиЯ на на амплитуду), то
   visible={'on','on','on','on','off'}; % пЯтому элементу вектора visible присваиваетсЯ
                % значение 'off' (пЯтый параметр окна диалога должен
                % быть не видимым).
end; % Завершение конструкци if ... else
set_param(gcb,'MaskVisibilities',visible); % Присвоение параметру маскированной подсистемы
                % MaskVisibilities значениЯ вектора visible.
                % Параметр MaskVisibilities устанавливает режим
                % видимости параметров окна диалога маскированной
                % подсистемы.
```

Функция проверяет значение параметра **Internal_magn**. Если значение этого параметра есть **'on'**, то вектор **visible** имеет все элементы равные **'on'**, если же значение параметра **Internal_magn** равно **'off'** (используется внешний источник сигнала задания на амплитуду), то *пятый* элемент вектора **visible** имеет значение **'off'** и функция **set_param(gcb,'MaskVisibilities',visible);** сделает *не отображаемой* графу для ввода пятого параметра (амплитуда внутреннего источника).

Для правильной работы такого окна диалога необходимо чтобы файл модели и файлы функций обработки находились в одной папке, и данная папка являлась рабочей. Согласно документации по **Simulink** текст **callback**-функций можно записывать также явным образом в вызове функции **set_param**.

Внешний вид окна диалога генератора для случая использования внешних источников сигналов задания на частоту и амплитуду показан на рис. 9.10.14.

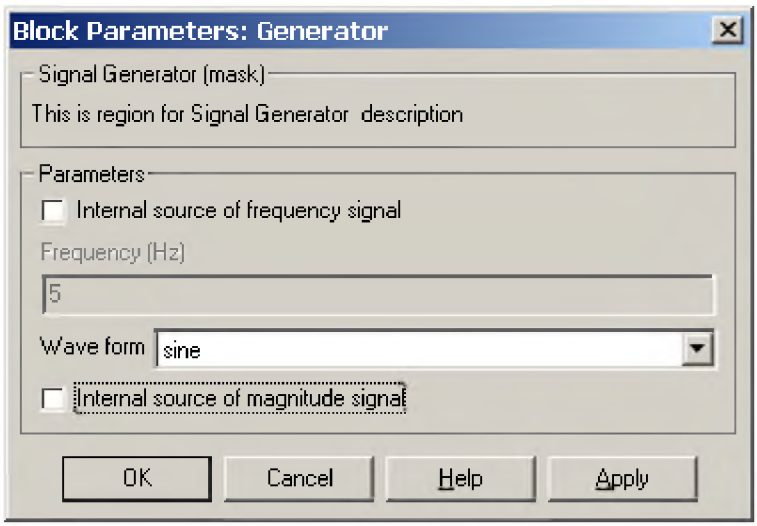


Рис. 9.10.14.Окно диалога генератора

При разработке динамически изменяемых окон диалога можно также переопределять с помощью функции **set_param** следующие параметры маскированной подсистемы:

- **MaskType** – Название блока.
- **MaskDescription** – Описание маскированной подсистемы.
- **MaskPromptString** – Названия параметров, задаваемые в окне диалога.
- **MaskValueString** – Значения параметров, задаваемые в окне диалога.

9.10.6. Управление портами маскированной подсистемы

В предыдущем параграфе рассматривалась методика создания динамически обновляемых окон диалога. В приведенном примере задание на амплитуду или частоту генератора сигналов может задаваться как параметр окна диалога блока, либо поступать от внешнего источника через входной порт подсистемы. При этом внешний вид блока, когда внешние источники не задействованы, оказывается точно таким же, как и при их использовании. Это неудобно, поскольку внешний вид блока (наличие на пиктограмме входных портов) вводит в заблуждение относительно фактических источников сигналов задания. Выходом из создавшейся ситуации является создание **callback**-функций убирающих или восстанавливающих входные порты в подсистеме. Основная идея при этом заключается в том, чтобы в случае, если внешний источник не используется – заменить входной порт подсистемы на блок **Ground**, а если внешний источник используется – выполнить обратную замену. Такие замены легко выполняются с помощью команд управления **Simulink**-моделью (команды подробно рассмотрены в документации на **Simulink**). Для повышения наглядности пиктограммы генератора стандартные названия входных портов **In1** и **In2** заменены на **M** (входной порт для сигнала задания на амплитуду) и **F** (входной порт сигнала задания на частоту). Пиктограмма генератора и его схема показаны на рис. 9.10.15.

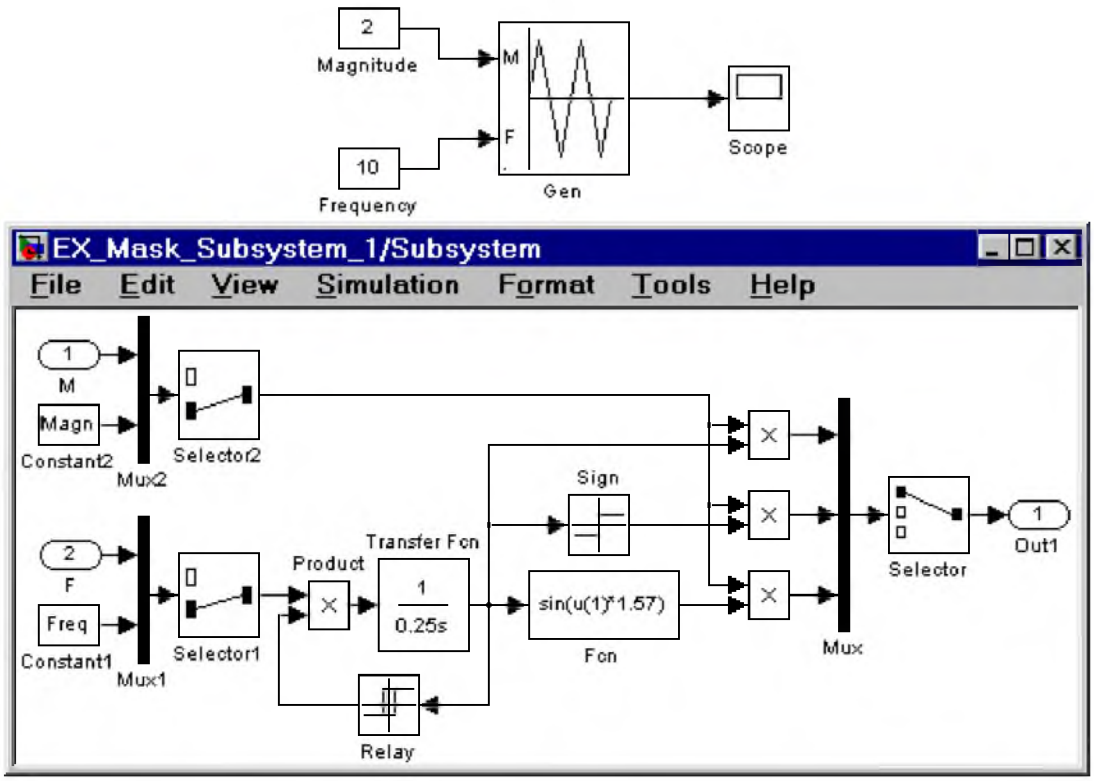


Рис. 9.10.15 Функциональный генератор.

Текст **callback**-функции задающей вид источника сигнала задания на частоту (файл **call_back_freq.m**) приведен ниже.

```
                % ПервАЯ часть (управление окном диалога)
Freq_param=get_param(gcb,'Internal_freq');
if strcmp(Freq_param,'on');
   enable={'on','on','on','on','on'};
else;
   enable={'on','off','on','on','on'};
end;
set_param(gcb,'MaskEnables',enable);
                % ВторАЯ часть (управление портами)
Magn_param=get_param(gcb,'Internal_magn'); % Присвоение переменной Magn_param
                % значениЯ параметра Internal_magn (тип источника
                % сигнала заданиЯ на амплитуду: внутренний или внешний).

In_2_BlockType=get_param([gcb,'/F'],'BlockType'); % Определение типа блока в подсистеме,
                % имеющего метку F .

if strcmp(Freq_param,'on')&(In_2_BlockType=='Inport');% Если значение переменной
                % Freq_param равно 'on' (внутренний источник сигнала
                % заданиЯ на частоту), а тип блока, имеющего метку F,
                % есть 'Inport', то
   replace_block(gcb,'Name','F','Ground','noprompt') % выполнЯетсЯ замена блока, имеющего
                % метку F (второго входного порта) на блок Ground.
                % Порт с меткой F с пиктограммы блока исчезает.
```

```
            % Величина заданиЯ на частоту генератора
            % определЯетсЯ параметром, задаваемым в окне диалога.
end; % Завершение конструкции if.

if strcmp(Freq_param,'off')&(In_2_BlockType=='Ground'); % Если значение переменной
            % Freq_param равно 'off' (внешний источник сигнала
            % заданиЯ на частоту), а тип блока, меющего метку F,
            % есть 'Ground', то
   replace_block(gcb,'Name','F','Inport','noprompt')% выполнЯетсЯ замена блока Ground
            % на блок Inport. Порт с меткой F на пиктограмме блока
            % поЯвлЯетсЯ. Величина заданиЯ на частоту генератора
            % определЯетсЯ сигналом поступающим на данный порт.
end; % Завершение конструкци if.

if strcmp(Freq_param,'off')&strcmp(Magn_param,'off') % Если значениЯ переменных
            % Freq_param и Magn_param равны 'off' (частота и амплитуда
            % генератора задаетсЯ внешними источниками), то должна быть
            % выполнена проверка правильности нумерации входных портов.
            % Порт, имеющий метку M должен быть первым, а порт, имеющий
            % метку F должен быть вторым.
   Port_1_param=get_param([gcb,'/M'],'port'); % Переменной Port_1_param присваиваетсЯ
            % значение номера порта имеющего метку M.
   Port_2_param=get_param([gcb,'/F'],'port'); % Переменной Port_2_param присваиваетсЯ
            % значение номера порта имеющего метку F.
   if (Port_1_param = = '2')&(Port_2_param = = '1'); % Если нумерациЯ портов нарушена, то
      replace_block(gcb,'Name','F','Ground','noprompt') % порт, имеющий метку F
            % (через который поступает задание на частоту) временно заменЯетсЯ
            % на блок Ground. При этом оставшемусЯ порту
            % автоматически присваиваетсЯ первый номер.
      replace_block(gcb,'Name','F','Inport','noprompt') % Блок, имеющий метку F заменЯетсЯ на
            % блок входного порта. При этом ему автоматически присваиваетсЯ
            % второй номер.
   end; % Завершение внутренней конструкци if.
end; % Завершение внешней конструкци if.
```

Первая часть функции управляет окном диалога (она полностью повторяет приведенную в предыдущем параграфе), а вторая часть выполняет управление входным портом **F** (задание на частоту).

Текст **callback**-функции задающей вид источника сигнала задания на амплитуду (файл **call_back_magn.m**) аналогичен тексту функции **call_back_freq** и приводится ниже без комментариев.

```
Magn_param=get_param(gcb,'Internal_magn');
if strcmp(Magn_param,'on');
   visible={'on','on','on','on','on'};
else;
   visible={'on','on','on','on','off'};
end;
set_param(gcb,'MaskVisibilities',visible);

Freq_param=get_param(gcb,'Internal_freq');
```

```
In_1_BlockType=get_param([gcb,'/M'],'BlockType');
if strcmp(Magn_param,'on')&(In_1_BlockType=='Inport');
   replace_block(gcb,'Name','M','Ground','noprompt')
end;
if strcmp(Magn_param,'off')&(In_1_BlockType= ='Ground');
   replace_block(gcb,'Name','M','Inport','noprompt')
end;

if strcmp(Freq_param,'off')&strcmp(Magn_param,'off')
   Port_1_param=get_param([gcb,'/M'],'port');
   Port_2_param=get_param([gcb,'/F'],'port');
   if (Port_1_param = = '2')&(Port_2_param = = '1')
      replace_block(gcb,'Name','F','Inport','noprompt')
   end;
end;
```

Первая часть функции **call_back_freq** управляет окном диалога, и аналогична приведенной в предыдущем параграфе. Вторая часть – управляет первым входным портом подсистемы. Вторая часть функции содержит также команды проверяющие правильность нумерации портов и восстанавливающие ее, если нумерация портов нарушена.

На рис.9.10.16 приведена модель генератора для случая, когда источник задания на амплитуду сигнала является внутренним, а источник задания на частоту – внешним.

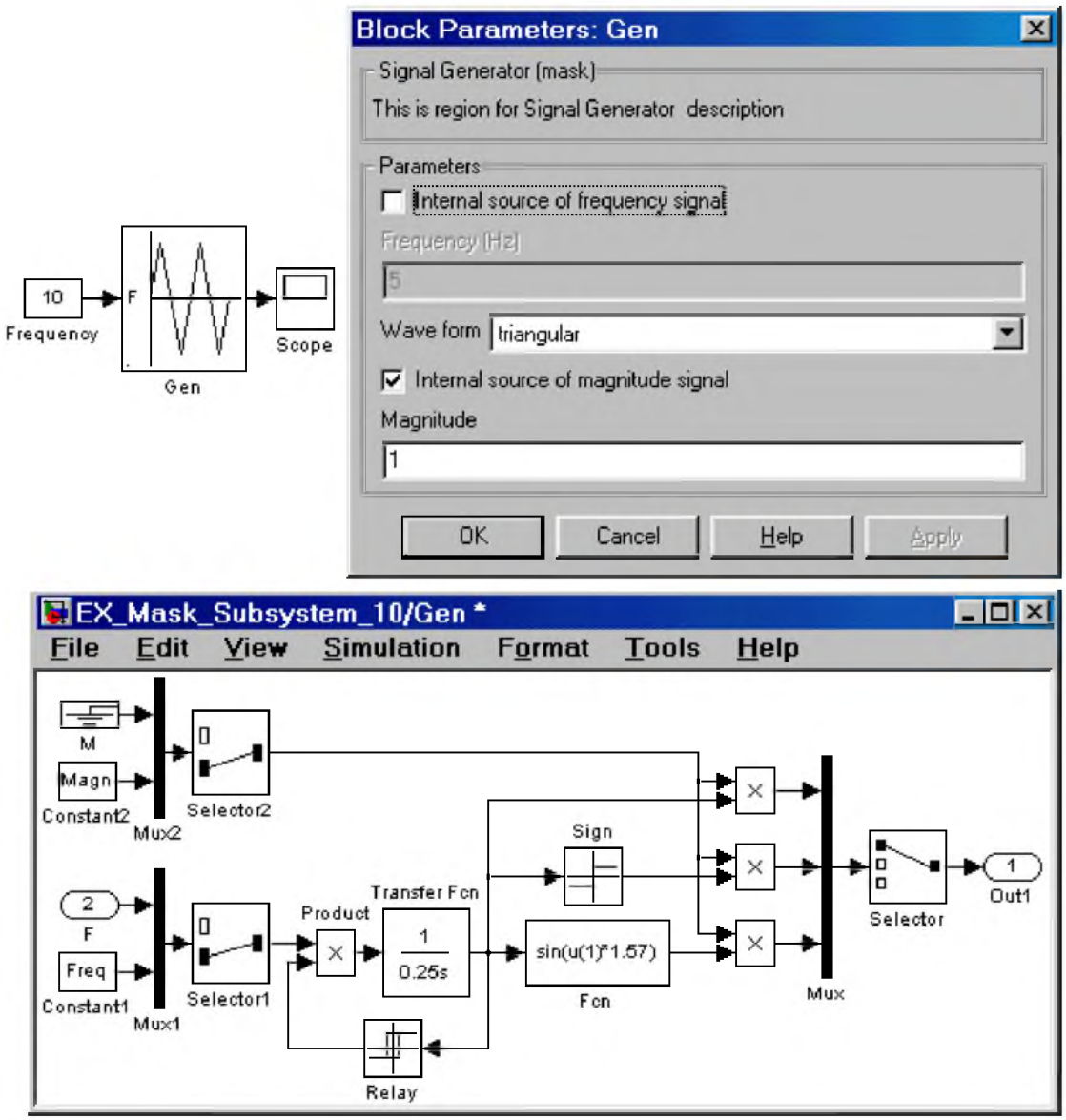


Рис. 9.10.16 Модель функционального генератора при использовании внутреннего источника задания на амплитуду сигнала.

На рисунке видно, что при выборе внутреннего источника сигнала задания на амплитуду (флажок **Internal source of magnitude signal** установлен) соответствующий входной порт на пиктограмме отсутствует, а в самой модели генератора входной порт **M** заменен блоком **Ground**. При этом задание на амплитуду сигнала поступает от блока **Constant2** внутри подсистемы, а задание на частоту выходного сигнала – от внешнего источника через входной порт с меткой **F**.

10. Редактор дифференциальных уравнений DEE

Simulink содержит специальный блок – **Differential Equation Editor** (редактор дифференциальных уравнений). С помощью этого блока можно задавать системы дифференциальных уравнений в явной форме Коши и выполнять их решение. Вызов редактора выполняется вводом команды **dee** в окне **MATLAB**.

Использование редактора рассмотрим на примере расчета переходных процессов в последовательном колебательном контуре. Задача заключается в нахождении тока протекающего в электрической цепи и напряжения на конденсаторе **C** после замыкания ключа. Схема цепи показана на рис. 10.1. Начальные условия полагаем нулевыми (ток в цепи отсутствует, и конденсатор не заряжен).

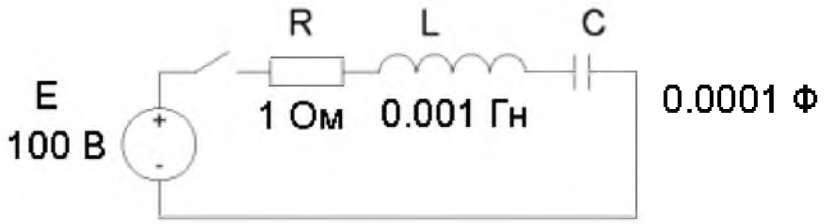


Рис.10.1. Расчетная электрическая схема

Предварительно составляем систему дифференциальных уравнений, описывающую электрическую цепь:

$$e = L\frac{di}{dt} + R \cdot i + u_c,$$

$$i = c\frac{du_c}{dt},$$

где i – ток в цепи, u_c - напряжение на конденсаторе

Записываем данную систему уравнений в явной форме Коши:

$$\frac{di}{dt} = (e - R \cdot i - u_c) / L,$$

$$\frac{du_c}{dt} = \frac{1}{c} i.$$

Вводим “машинные” переменные:

$i \rightarrow x(1),$
$u_c \rightarrow x(2),$
$e \rightarrow u(1).$

В итоге система уравнений примет вид:

$$\frac{dx(1)}{dt} = (u(1) - R \cdot x(1) - x(2)) / L,$$

$$\frac{dx(2)}{dt} = \frac{1}{c} \cdot x(1).$$

Введение “машинных” переменных, связано с тем, что редактор дифференциальных уравнений требует задавать в виде векторов входные воздействия (***u***) и переменные состояния (***x***) и имена этих векторов жестко заданы.

После получения системы дифференциальных уравнений с использованием “машинных” переменных, необходимо запустить редактор командой **dee** в окне **MATLAB**. Затем нужно поместить блок редактора в окно с создаваемой моделью, открыть окно редактора и ввести систему дифференциальных уравнений, начальные условия, а также алгебраические уравнения для расчета выходных сигналов (в рассматриваемой задаче выходные переменные равны переменным состояния). Также необходимо указать размерность вектора входного сигнала (**# of inputs**). Схема модели и окно редактора показаны на рис. 10.2. Там же приведены и результаты расчета.

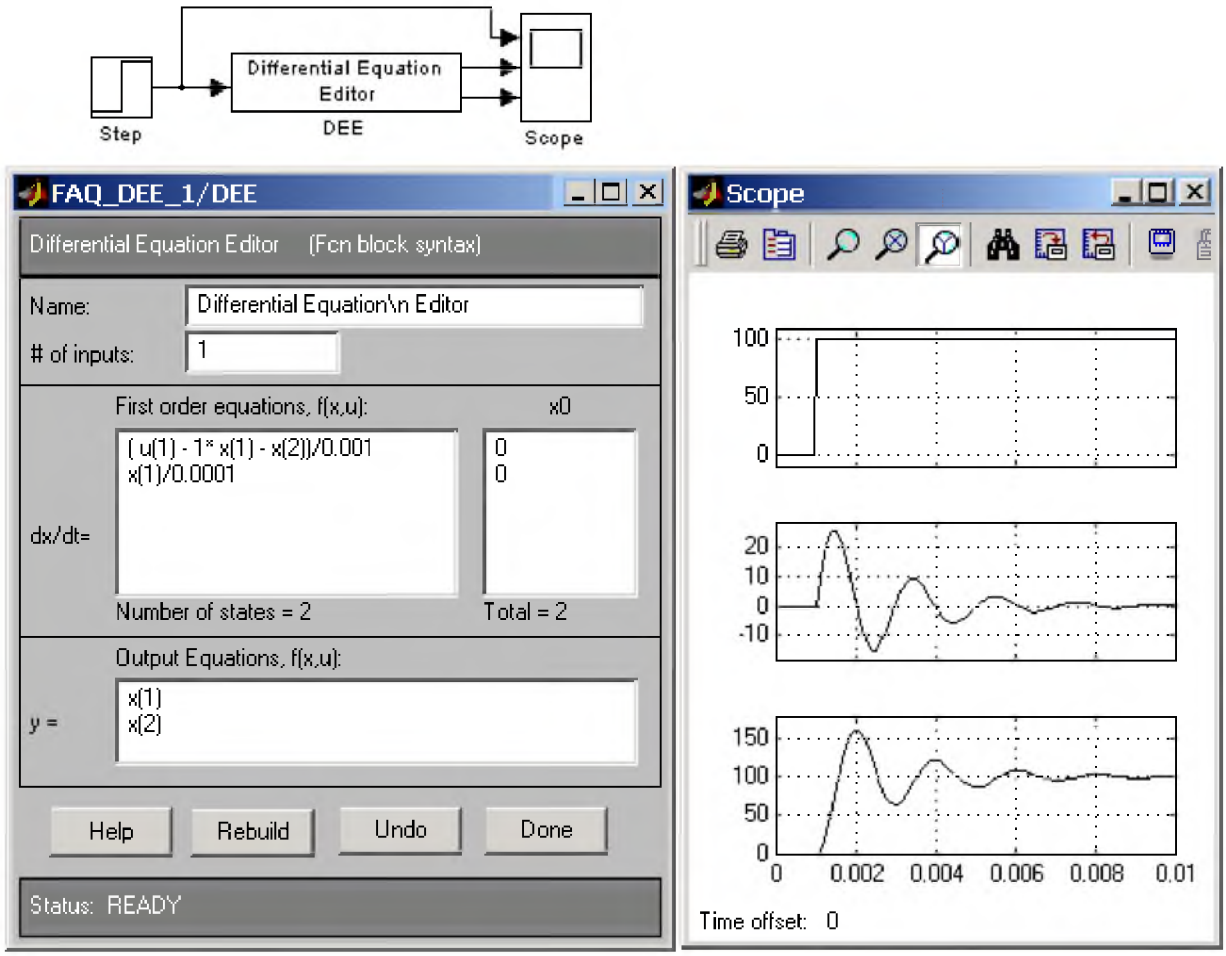


Рис. 10.2. Модель, использующая редактор дифференциальных уравнений

Значения постоянных коэффициентов системы уравнений можно задавать не только как числовые константы, но и использовать переменные рабочей области **MATLAB**.

Достоинством редактора **DEE** является также то, коэффициенты дифференциального уравнения могут быть переменными и задаваться также как и входные сигналы (через входной порт). В качестве

примера на рис. 10.3 показан вариант предыдущей модели, в котором величина сопротивления увеличивается в 10 раза в процессе расчета. В системе дифференциальных уравнений сопротивление записано как входной сигнал ***u(2)***.

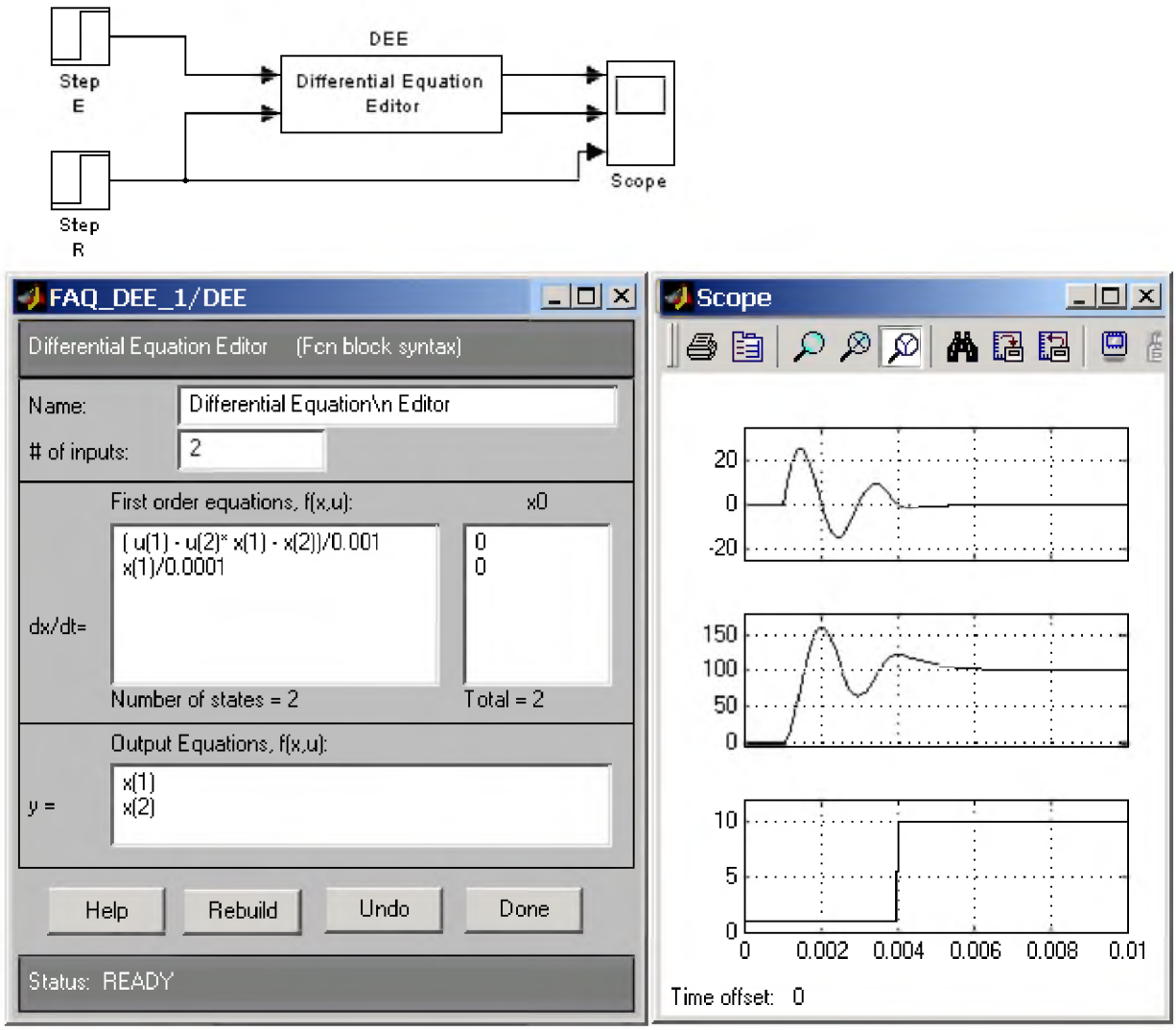


Рис. 10.3 Второй вариант модели

11. Использование Simulink LTI-Viewer для анализа динамических систем

Инструмент **Simulink LTI-Viewer** входит в состав пакета прикладных **программ Control System Toolbox** и предназначен для анализа линейных стационарных систем. С помощью данного инструмента можно легко построить частотные характеристики исследуемой системы, получить ее отклики на единичные ступенчатое и импульсное воздействия, найти нули и полюса системы и т.д.

Краткий алгоритм работы с **Simulink LTI-Viewer** приведен ниже.

11.1. Работа с Simulink LTI-Viewer

1. Выполнить команду **Tools\Linear Analysis...** окна **Simulink**-модели.

В результате выполнения команды откроется окно **Model_Inputs_and_Outputs** как это показано на рис. 11.1, а также пустое окно **Simulink LTI-Viewer** .

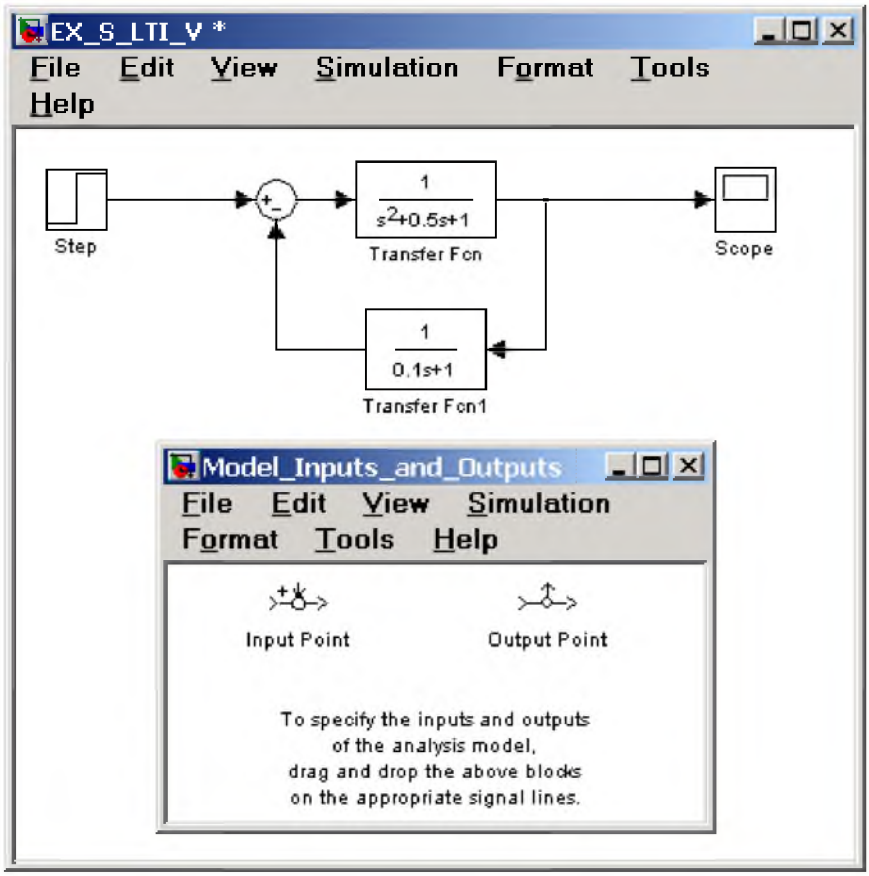


Рис. 11.1 Исследуемая модель и окно **Model_Inputs_and_Outputs** инструмента **Simulink LTI-Viewer**

2. Установить блок **Input Point** на входе и блок **Output Point** на выходе исследуемой системы, как это показано на рис. 11.2.

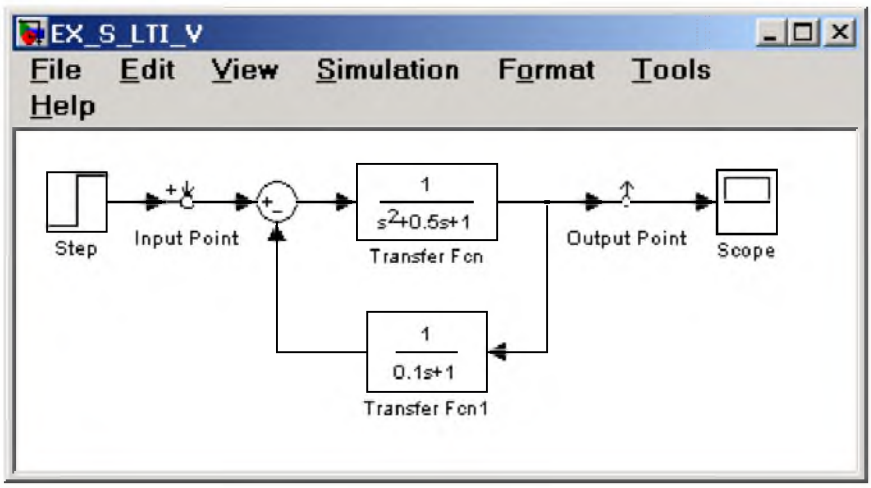


Рис. 11.2 Исследуемая модель с установленными блоками

Input Point и **Output Point**

3. В окне **LTI Viewer** выполнить команду **Simulink\Get Linearized Model**.

Данная команда выполняет линеаризацию модели и строит реакцию системы на единичное ступенчатое воздействие. Результат выполнения данного пункта показан на рис. 11.3

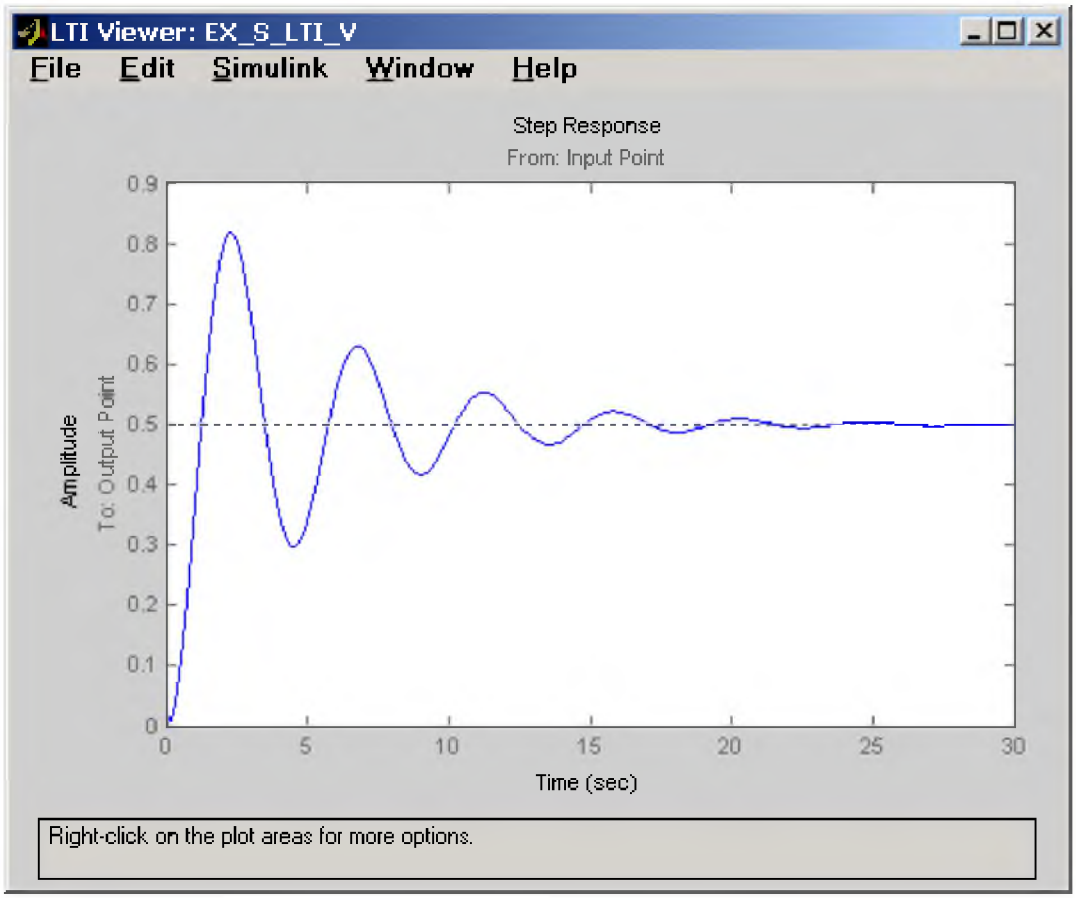


Рис. 11.3 Реакция системы на единичное ступенчатое воздействие.

Если система имеет несколько входов и выходов и для всех них установлены блоки **Input Point** и **Output Point**, то на графике будет отображено несколько окон показывающих реакцию на каждом выходе при воздействии на каждый вход.

5. Для получения остальных характеристик системы необходимо выполнить команду **Edit\Plot Configuration... в** окне **LTI Viewer**. В результате выполнения этой команды откроется окно **Plot Configuration**, показанное на рис. 11.4.

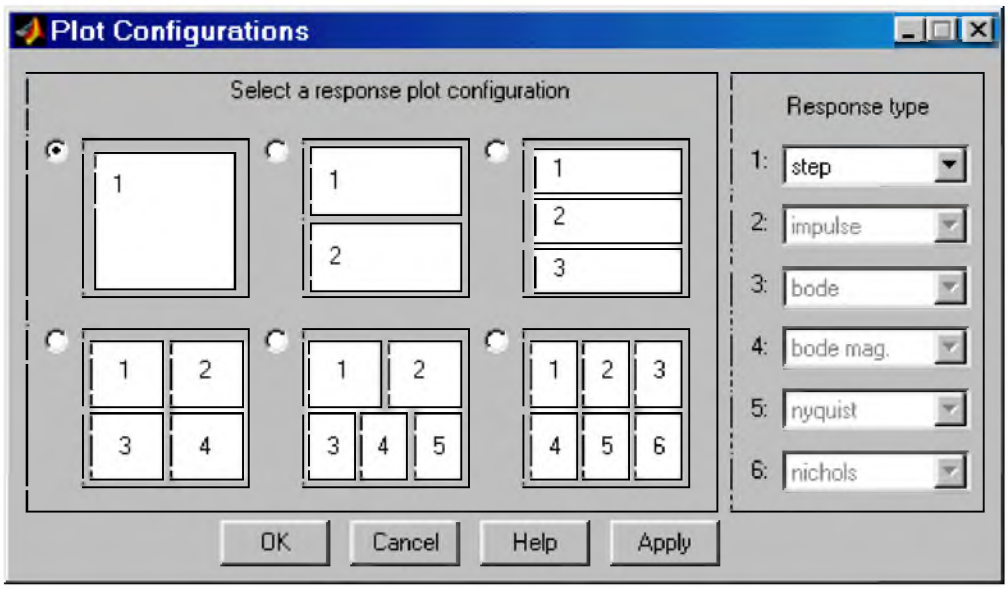


Рис. 11.4 Окно **Plot Configuration**

В открывшемся окне можно выбрать число отображаемых графиков (панель **Select a response plot configuration**) и вид отображаемых графиков (панель **Response type**). Для построения доступны следующие графики (диаграммы):

- **step** – Реакция на единичное ступенчатое воздействие.
- **impulse** – Реакция на единичное импульсное воздействие.
- **bode** – Логарифмические амплитудная и фазовая частотные характеристики.
- **bode mag** – Логарифмическая амплитудная частотная характеристика.
- **nyquist** – Диаграмма Найквиста.
- **nichols** – Годограф Николса.
- **sigma** – Сингулярные числа.
- **pole/zero** – Нули и полюса системы.

На рис. 11.5 приведен пример окна **Simulink LTI-Viewer** с несколькими различными характеристиками исследуемой системы.

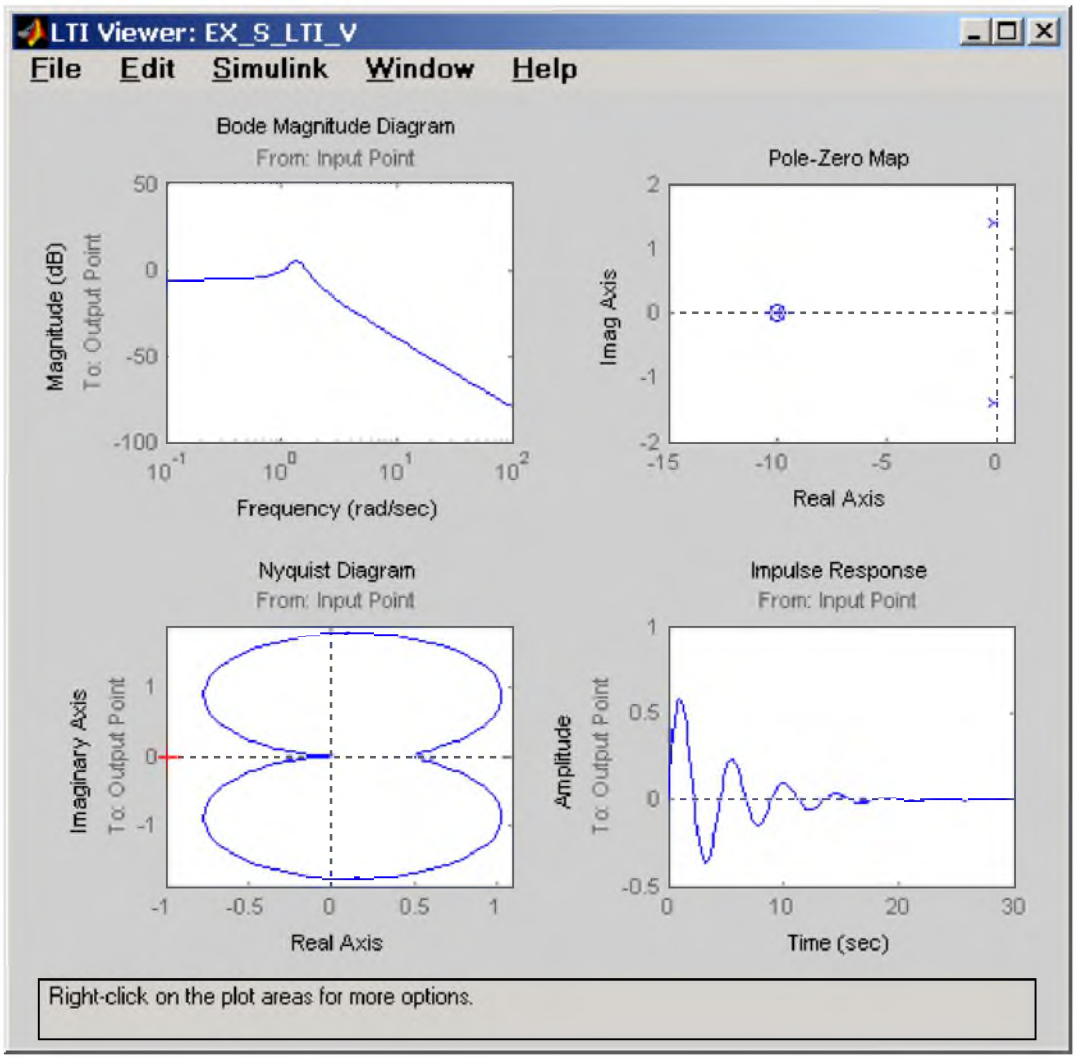


Рис. 11.5 Окно **Simulink LTI-Viewer** с несколькими графиками.

Настройку внешнего вида графиков можно выполнить с помощью команды **Edit\Line Styles…** (установка вида и цвета линий, вида маркеров).

11.2. Настройка Simulink LTI-Viewer

С помощью команды **Edit\Viewer Preferences…** выполняются следующие виды настройки:

1. Установка единиц измерения (вкладка **Units**). Вид окна при настройке единиц измерения показан на рис. 11.6.

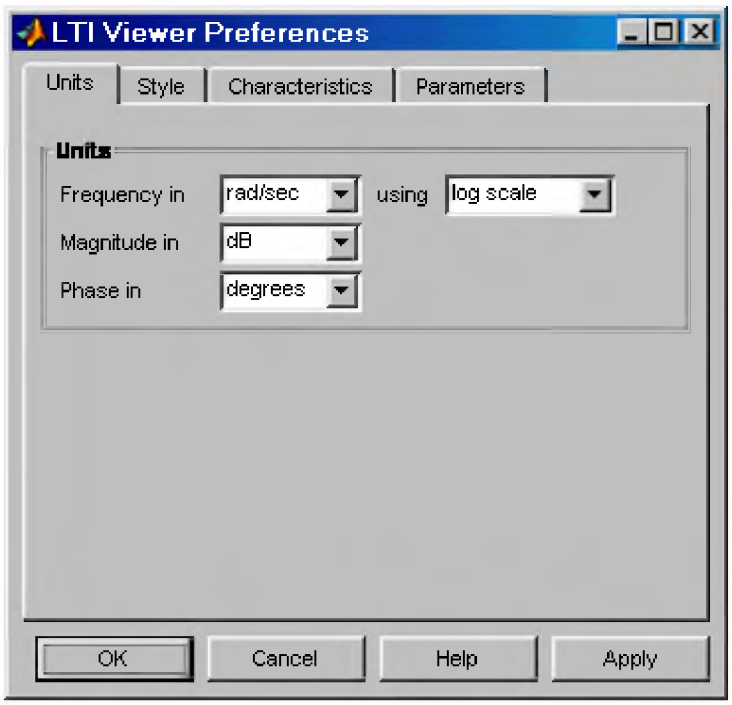


Рис. 11.6 Вкладка **Units**

Вкладка **Units** окна позволяет задать единицы измерения частоты (**рад/с** или **Гц**), уровня (**dB** или абсолютные единицы), фазы (градусы или радианы), а также установить вид шкалы частоты (логарифмический или линейный).

2. Установка стиля графиков (вкладка **Style**). На данной вкладке можно выполнить настройку шрифтов окна **Simulink LTI-Viewer** (панель **Fonts**), выбрать цвет осей графиков (панель **Colors**), а также задать нанесение линий сетки на графики (флажок **Show grids**). Внешний вид вкладки **Style** показан на рис. 11.7.

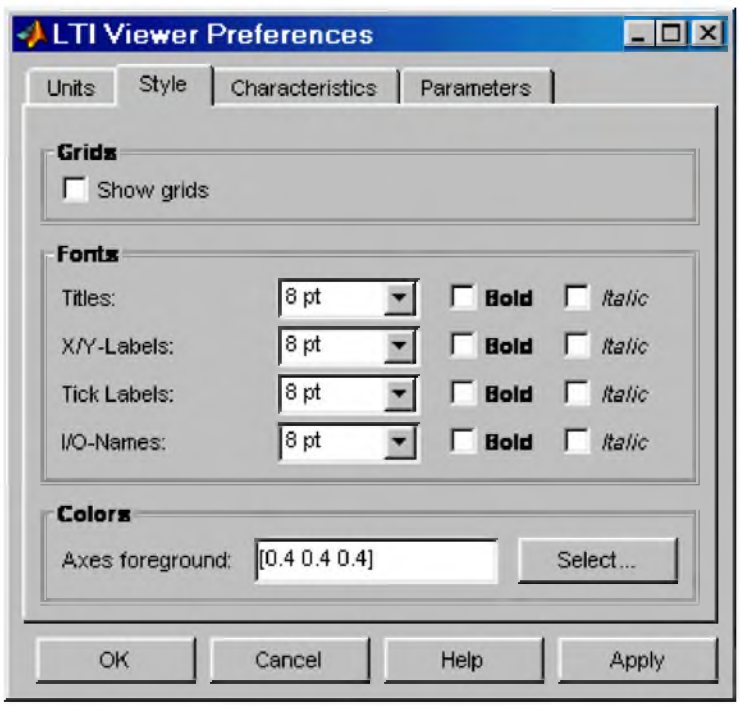


Рис. 11.7 Вкладка **Style**

3. Установка параметров расчета переходного процесса (вкладка **Characteristics**). Данная вкладка позволяет задать параметры установленные "по умолчанию" для вычисления времени нарастания и времени переходного процесса. По умолчанию **Simulink LTI-Viewer** вычисляет время переходного процесса как время, когда переходная функция входит в 2% зону и больше не выходит из нее (параметр **Show setting time within**). Также можно изменить параметры для вычисления времени переходного процесса (**Show rise time from**). На данной вкладке имеется также флажок **Unwrap phase**, установка которого позволяет избежать отображения разрывов в фазо-частотной характеристике, связанных с областью определения функции **arctg**, вычисляющей фазовый сдвиг. Внешний вид вкладки **Characteristics** показан на рис. 11.8.

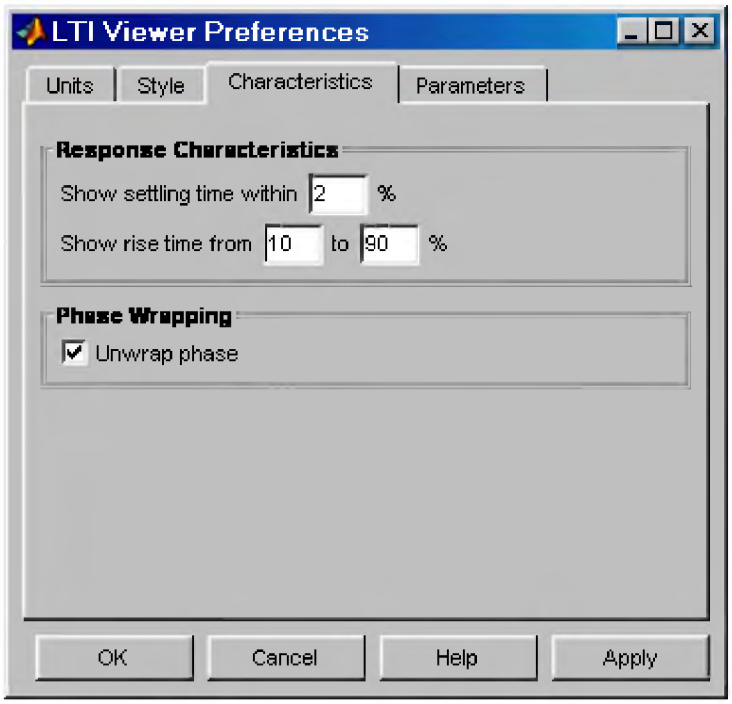


Рис. 11.7 Вкладка **Characteristics**

4. Установка интервалов времени и частоты (вкладка **Parameters**). На данной вкладке задается временной интервал для расчета переходного процесса (панель **Time Vector**), а также интервал частот для расчета частотных характеристик (панель **Frequency Vector**). Внешний вид вкладки **Parameters** показан на рис. 11.8.

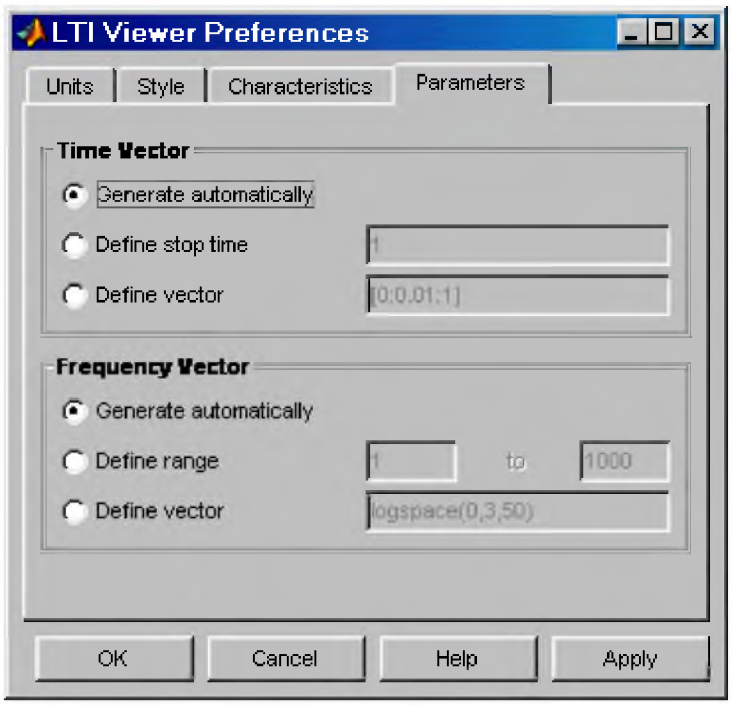


Рис. 11.8. Вкладка **Parameters**

Векторы времени и частоты можно вычислять в автоматическом режиме **(Generate automatically)**, ввести конкретное значение для времени окончания расчета (**Define stop time**) или диапазон значений по частоте (**Define range**), либо задать непосредственно вектор значений времени или частоты (**Define vector**).

11.3. Экспорт модели

Команда **File/Export** позволяет выполнить экспорт модели в виде матриц уравнений пространства состояния в рабочую область **MATLAB** или в файл. При выполнении этой команды открывается окно диалога (см. рис. 11.9) в котором необходимо выбрать вариант экспорта.

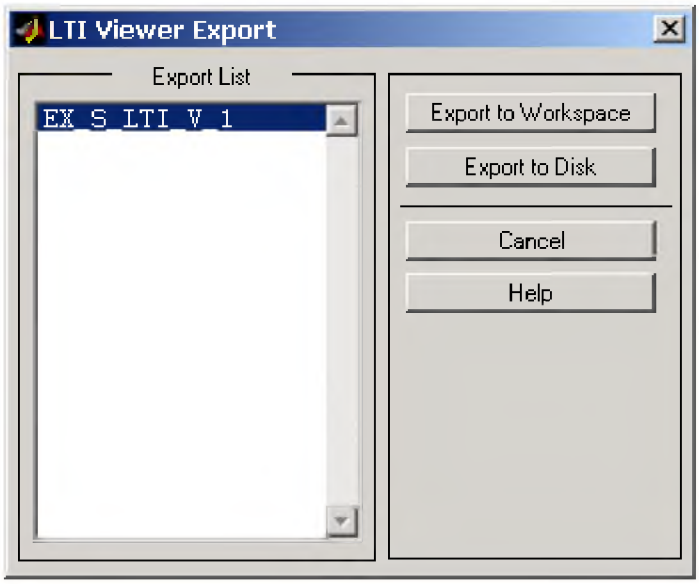


Рис. 11.9. Окно диалога при экспорте модели.

В случае экспорта в рабочую область **MATLAB** в ней появляется структура с именем ***имя_модели*_1** (при последующих операциях экспорта ***имя_модели*_2**, ***имя_модели*_3** и т.д.). Для рассматриваемой модели с именем **EX_S_LTI_V.mdl** имя структуры будет **EX_S_LTI_V_1 .** Для просмотра значений матриц уравнений пространства состояния необходимо ввести в командной строке **MATLAB** имя структуры и нажать клавишу **Enter** на клавиатуре. В командном окне появится распечатка значений матриц:

```
>> EX_S_LTI_V_1 % Ввод с клавиатуры (символы >> не вводятся)

a =
                EX_S_LTI_V/T  EX_S_LTI_V/T  EX_S_LTI_V/T
  EX_S_LTI_V/T      -0.5          -1          -10
  EX_S_LTI_V/T        1            0            0
  EX_S_LTI_V/T        0            1          -10

b =
                Input Point
  EX_S_LTI_V/T          1
  EX_S_LTI_V/T          0
  EX_S_LTI_V/T          0

c =
                EX_S_LTI_V/T  EX_S_LTI_V/T  EX_S_LTI_V/T
  Output Point          0           1           0

d =
                Input Point
  Output Point          0

Continuous-time model.
```

Для работы с матрицами удобно извлечь их из структуры командами вида:

A = ***имя_структуры*.a**
B = ***имя_структуры*.b**
C = ***имя_структуры*.c**
D = ***имя_структуры*.d**

Применительно к рассматриваемой модели эти команды будут выглядеть следующим образом:

```
>> A=EX_S_LTI_V_1.a          % Ввод с клавиатуры (символы >> не вводятся)
A =
  -0.5000   -1.0000  -10.0000
   1.0000         0         0
        0    1.0000  -10.0000

>> B=EX_S_LTI_V_1.b          % Ввод с клавиатуры (символы >> не вводятся)
B =
   1.0000
```

```
     0
     0

>> C=EX_S_LTI_V_1.c          % Ввод с клавиатуры (символы >> не вводятся)
C =
     0    1.0000         0

>> D=EX_S_LTI_V_1.d          % Ввод с клавиатурым (символы >> не вводятся)
D =
     0
```

Сформированные в рабочей области матрицы **A, B, C** и **D** могут использоваться для дальнейших преобразований (создание **LTI**-объектов, переход к передаточным функциям, переход к дискретным моделям и т.д.).

12. Основные команды MATLAB для управления Simulink-моделью

При разработке графического интерфейса пользователя, создании **S**-функций и т.п. задач требующих управления конфигурацией, параметрами и работой **Simulink**-модели допускается использовать специальные команды (функции) языка **MATLAB**. С помощью таких команд можно открывать и закрывать модель, запускать модель на расчет, добавлять и убирать блоки, изменять параметры блоков и осуществлять иные операции с моделью.

12.1. add_block

Назначение: Добавление нового блока в модель

Синтаксис:

add_block('src', 'dest')

Команда добавляет блок, полный путь которого задан параметром **'src'**, в модель в соответствии с путем назначения **'dest'**.

add_block('src', 'dest', 'parameter1', value1, ...)

Команда добавляет блок, полный путь которого задан параметром **'src'**, в модель в соответствии с путем назначения **'dest'** и устанавливает значения параметров блока.

Пример 1:

Команда **add_block('built-in/Gain', 'EX_add_block/Gain')** добавляет в модель **EX_add_block.mdl** усилитель **Gain** из встроенной библиотеки.

Пример 2:

Команда **add_block('EX_add_block / In1', 'EX_add_block/My_Subsystem / In1')**

копирует блок входного порта **In1** из модели **EX_add_block.mdl** в подсистему **My_Subsystem** той же самой модели.

Пример 3:

Команда **add_block('built-in/Constant', 'EX_add_block/Constant','Value','150')** добавляет в модель **EX_add_block.mdl** блок **Constant** и устанавливает параметр **Value** этого блока равным **150**.

12.2. add_line

Назначение: Добавление новой линии связи в модель

Синтаксис:

h = add_line('sys','oport','iport')

Команда добавляет новую линию связи в модели **sys** от выходного порта **oport** ко входному порту **iport**. Параметры **oport** и **iport** задают полные пути блоков.

h = add_line('sys','oport','iport', 'autorouting','on')

Команда аналогична предыдущей. Дополнительный параметр **autorouting** (автоматическая трассировка), значение которого равно **on**, обеспечивает создание линии без пересечения пиктограмм блоков. По “умолчанию” значение этого параметра равно **off**.

h = add_line('sys', points)

Команда добавляет новую линию связи в модели **sys** в соответствии с координатами, заданными матрицей **points**. Началом системы координат окна модели считается левый верхний угол окна.

Пример 1:

Команда **add_line('EX_add_line','Step/1','Sum/2')** добавляет новую линию связи в модели **EX_add_line.mdl** между выходом блока **Step** (блок имеет один выход) и вторым входом блока **Sum**.

Пример 2:

Команда **add_line('EX_add_line','Step1/1','Sum/2','autorouting','on')** добавляет новую линию связи в модели **EX_add_line.mdl** между выходом блока **Step1** и вторым входом блока **Sum**, при включенном режиме автотрассировки.

Пример 3:

Команда **add_line('EX_add_line',[20 55; 40 10; 60 60])** добавляет новую линию связи в модели **EX_add_line.mdl** в соответствии с координатами, заданными матрицей **[20 55;40 10;60 60])** .

12.3. add_param

Назначение: Добавление нового параметра в модель.

Синтаксис:

Команда **add_param('sys', ,value1, ,value2,...)** добавляет в модель **sys** новые параметры **parameter1, parameter2…**и присваивает им значения **value1, ,value2,...** Новые параметры доступны командам **get_param**, **set_param** и ничем не отличаются от стандартных параметров **Simulink**-модели. Имена параметров не чувствительны к регистру символов. Значения параметров чувствительны к регистру символов.

Пример 1:

Команда **add_param('EX_add_param','data','01 december 2002','time','21.00')** добавляет в модель **EX_add_param.mdl** новые параметры **data** и **time** и присваивает им значения **'01 december 2002'** и **'21.00'**, соответственно.

12.4. bdclose

Назначение: Команда закрывает **Simulink**-модель (все модели) без сохранения изменений.

Синтаксис:

bdclose

Команда закрывает активную модель.

bdclose('sys')

Команда закрывает модель **sys**.

bdclose('all')

Команда закрывает все открытые модели.

Пример:

Команда **bdclose('my_model')** закрывает модель **my_model.mdl** .

12.5. bdroot

Назначение: Возвращает имя модели (подсистемы верхнего уровня).

Синтаксис:

bdroot

Команда возвращает имя активной модели.

bdroot('obj')

Команда возвращает имя модели содержащей объект **obj**.

Пример:

Команда **bdroot(gcb)** возвращает имя модели содержащей выделенный в данный момент блок.

12.6. close_system

Назначение: Команда закрывает модель с возможностью сохранения изменений.

Синтаксис:

close_system

Команда закрывает модель или подсистему. Если модель или подсистема были изменены, то на экран будет выведено окно с вопросом о сохранении изменений.

close_system('sys')

Команда закрывает модель или подсистему с указанным именем **sys**.

close_system('sys', saveflag)

Команда закрывает модель или подсистему и, в зависимости, от параметра **saveflag** выполняет или не выполняет сохранение изменений. В случае, если параметр **saveflag** равен **0** изменения не сохраняются. Если же значение данного параметра равно **1**, то внесенные в модель или подсистему изменения сохраняются.

close_system('sys', 'newname')

Команда сохраняет модель **sys** под новым именем **newname**.

close_system('blk')

Команда закрывает окно диалога блока, полный путь которого задан параметром **blk**, или вызывает **CloseFcn** функцию, если данная функция для блока определена.

Пример 1:

Команда **close_system('my_model', 'new_model')** сохраняет модель **my_model.mdl** под новым именем **new_model.mdl** .

Пример 2:

Команда **close_system('my_model', 1)** сохраняет модель **my_model.mdl** со всеми изменениями.

12.7. delete_block

Назначение: Удаление блока из модели.

Синтаксис:

delete_block('blk')

Команда удаляет блок в соответствии с параметром **blk**, задающим полный путь к блоку.

Пример 1:

Команда **delete_block('EX_delete_block/My_Subsystem/In1')** удаляет входной порт **In1** из подсистемы **My_Subsystem** модели **EX_delete_block.mdl** .

12.8. delete_line

Назначение: Удаление линии связи

Синтаксис:

delete_line('sys', 'oport', 'iport')

Команда удаляет линию связи в модели **sys** от выходного порта **oport** ко входному порту **iport**. Параметры **oport** и **iport** задают полные пути блоков.

delete_line('sys', [x y])

Команда удаляет линию связи, которой принадлежит точка с координатами **[x y]**.

Пример 1:

Команда **delete_line('EX_delete_line', 'Step/1', 'Sum/2')** удаляет линию связи в модели **EX_delete_line.mdl** между выходом блока **Step** (блок имеет один выход) и вторым входом блока **Sum**.

Пример 2:

Команда **delete_line('EX_delete_line',[20 55])** удаляет линию связи, которой принадлежит точка с координатами **[20 55]**.

12.9. delete_param

Назначение: Удаление параметра модели, добавленного командой **add_param** .

Синтаксис:

delete_param('sys','parameter1','parameter2',...)

Команда удаляет из модели **sys** параметры **parameter1, parameter2…**, добавленные ранее командой **add_param** .

Пример:

Команда **delete_param('EX_delete_param', 'data', 'time')** удаляет из модели **EX_delete_param.mdl** параметры **data** и **time** , добавленные ранее командой **add_param** .

12.10. gcb

Назначение: Получение пути текущего блока.

Синтаксис:

gcb

Команда возвращает полный путь текущего блока.

gcb('sys')

Команда возвращает полный путь текущего блока в модели **sys**.

Под текущим блоком понимается выделенный в окне модели блок, блок который выполняется в данный момент времени под управлением **S**-функции, блок **callback**-функция которого выполняется в данный момент времени или маскированный блок для которого выполняется функция инициализации.

Команду удобно использовать при получении пути блока для команд **get_param** и **set_param**.

Пример:

Команда **get_param(gcb,'Gain')** для текущего блока **Gain** возвращает значение параметра **Gain**.

12.11. gcs

Назначение: Получение пути текущей модели.

Синтаксис и правила использования команды аналогичны команде **gcb**.

12.12. find_system

Назначение: Поиск моделей (подсистем), блоков, линий, портов и текстовых описаний.

Синтаксис:

find_system (sys, 'c1', cv1, 'c2', cv2,...'p1', v1, 'p2', v2,...)

Команда выполняет поиск моделей (подсистем), блоков, линий, портов и их описаний, полный путь которых задан параметром **sys**, с использованием ограничений, заданных параметрами **c1, c2** и имеющих значения параметров **v1, v2** .

Виды ограничений приведены в Таблице 12.1.

Таблица 12.1

Ограничение	**Значение**	**Описание**
'SearchDepth'	**scalar**	Устанавливает глубину поиска (**0** – только для открытых систем, **1** – для блоков и подсистем верхнего уровня, **2** – для системы верхнего уровня и ее дочерних подсистем, и т.д.) Значение по умолчанию **all** – все уровни.
'LookUnderMasks'	**'none'**	Пропуск маскированных блоков.
	{'graphical'}	Поиск внутри маскированных блоков, не имеющих окон диалога и рабочей области маски. Этот параметр используется “по умолчанию”.
	'functional'	Поиск внутри маскированных блоков, не имеющих окон диалога.
	'all'	Поиск внутри всех маскированных блоков.
'FollowLinks'	**'on'| {'off'}**	Если параметр имеет значение **'on'**, то отслеживаются связи с библиотечными блоками. Значение по умолчанию **'off'** .
'FindAll'	**'on'| {'off'}**	Если параметр имеет значение **'on'**, то поиск распространяется на линии, порты и текстовые описания в пределах текущей модели. Значение по умолчанию **'off'** .
'CaseSensitive'	**{'on'}| 'off'**	Поиск с учетом регистра символов (при поиске строковых параметров). Значение по умолчанию **'on'** .
'RegExp'	**'on'| {'off'}**	Если параметр имеет значение **'on'**, то допускается проводить поиск с использованием шаблонов. Значение по умолчанию **'off'** .

В таблице значения используемых “по умолчанию” параметров приведены в фигурных скобках.

Пример 1:

Команда **find_system** возвращает массив ячеек содержащих имена всех открытых подсистем и блоков.

Пример 2:

Команда **find_system('type', 'block_diagram')** возвращает массив ячеек содержащих имена всех открытых моделей.

Пример 3:

Команда **find_system('my_model','SearchDepth',2,'BlockType','Product')** выполняет поиск блоков умножения **Product** в модели **my_model.mdl** и в ее вложенных подсистемах.

Пример 4:

Команда **find_system('my_model', 'BlockType', 'Constant','Value', '100')** выполняет поиск блоков **Constant** у которых значение параметра **Value** равно **100**.

Для поиска с использованием шаблонов можно применять специальные символы приведенные в таблице Таблица 12.2

Таблица 12.2

Символ	**Описание**
.	Заменяет любой символ. Например, шаблону **'a.'** соответствуют выражения **'aa'**, **'ab'**, **'ac'** и т.п.
*	Заменяет любую последовательность символов (включая пустую). Например, шаблону **'a*'** соответствуют выражения **'a'**, **'ab'**, **'abc'** и т.п. Шаблону **'.*'** соответствует любая строка, в том числе и пустая.
+	Заменяет любое количество предшествующих символов. Например, шаблону **'ab+'** соответствуют выражения **'ab'**, **'abb'**, **'abbb'** и т.п
^	Отмечает начало последовательности символов. Например, шаблону **'^a'** соответствует любая строка начинающаяся на символ **'a'**.
$	Отмечает последний символ строки символов. Например, шаблону **'$a'** соответствует любая строка, оканчивающаяся на символ **'a'**.
\	Предписывает считать следующий символ обычным текстовым символом. Например, шаблон '\\' соответствует строке содержащей символ '\'.

[ ]	Определяет набор символов в выражении поиска. Например, шаблон **'f[oa]r'** соответствует выражениям **'for'** и **'far'**. Символ (-) задает диапазон символов. Например, шаблон **'[a-zA-Z1-9]'** соответствует любому алфавитно-цифровому символу. Символ (^) определяет исключаемые символы при поиске. Например, шаблон **'f[^i]r'** соответствует строкам **'far' and 'for'**, но не соответствует строке **'fir'**.
\w	Задает поиск строк, содержащих только алфавитно-цифровые символы. Например, шаблон **'^\w'** соответствует строке **'mu'**, но не соответствует строке **'&mu'**.
\d	Задает поиск строк, содержащих только цифровые символы. Например, шаблон **'\d+'** задает поиск любого целого числа.
\D	Задает поиск строк, не содержащих цифровые символы (аналог шаблона **[^0-9]**).
\s	Задает пробел в выражении поиска (аналог шаблона **[\t\r\n\f]**).
\S	Исключает пробелы из выражения поиска (аналог шаблона **[^ \t\r\n\f]**).
\<WORD\>	Задает поиск слова (последовательности символов отделенных с обеих сторон пробелами). Например, шаблону **'\<to\>'** соответствует слово **'to',** но не соответствует слово **'today'**.

Пример5:

Команда **find_system('my_model', 'regexp', 'on', 'blocktype', 'port')** задает поиск входных и выходных портов в модели **my_model.mdl**.

12.13. get_param

Назначение: Получение значения параметров модели или блока.

Синтаксис:

get_param('obj', 'parameter')

Команда возвращает значение параметра **parameter**, для объекта, полный путь которого задан выражением **obj**.

Пример 1:

Команда **get_param('EX_get_param/Constant','Value')** определяет значение параметра **Value** блока **Constant** модели **EX_get_param.mdl** .

Пример 2:

Команда **get_param('EX_get_param/Constant', 'ObjectParameters')** определяет все атрибуты блока **Constant** модели **EX_get_param.mdl** .

Пример 3:

Команда **get_param('EX_get_param/Constant', 'DialogParameters')** определяет параметры задаваемые в окне диалога блока **Constant** модели **EX_get_param.mdl** .

Пример 4:

Команда **get_param('EX_get_param', 'MaxStep')** определяет значение параметра **MaxStep** (максимальный шаг расчета) модели **EX_get_param.mdl** .

12.14. new_system

Назначение: Создание новой модели.

Синтаксис:

new_system('sys')

Команда создает новую модель **sys** . При этом окно модели не открывается. Для открытия окна следует использовать команду **open_system('sys')** .

Пример:

Команда **new_system('my_model')** создает модель **my_model.mdl** .

12.15. open_system

Назначение: Команда открывает окно модели, подсистемы, окно диалога блока.

Синтаксис:

open_system('sys')

Команда открывает модель **sys.mdl** .

open_system('blk')

Команда открывает окно диалога блока **blk** модели **sys.mdl** .

open_system('sys/Subsystem','force')

Команда открывает маскированную подсистему **Subsystem** модели **sys.mdl** . Команда аналогична пункту меню **Look Under Mask**.

Пример 1:

Команда **open_system('my_model')** открывает модель **my_model.mdl** .

Пример 2:

Команда **open_system('my_model/Constant')** открывает окно диалога блока **Constant** модели **my_model.mdl** .

Пример 3:

Команда **open_system('my_model/Subsystem')** открывает окно маскированной подсистемы **Subsystem** модели **my_model.mdl** .

12.16. replace_block

Назначение: Команда выполняет замену одного блока на другой.

Синтаксис:

replace_block('sys', 'blk1', 'blk2', 'noprompt')

Команда заменяет все блоки типа **blk1** на блоки **blk2** модели **sys** без запроса на подтверждение операции. Если **blk2** не является библиотечным блоком, то требуется указать полный путь к блоку.

replace_block('sys', 'Parameter', 'value', 'blk', ...)

Команда заменяет все блоки, параметр которых **Parameter** равен **value** на блоки **blk** модели **sys** .

Пример 1:

Команда **replace_block('EX_replace_block', 'Step', 'Inport', 'noprompt')** заменяет в модели **EX_replace_block.mdl** блок **Step** на блок **Inport** без запроса на подтверждение операции.

Пример 2:

Команда **replace_block('EX_replace_block', 'Value','100','Gain', 'noprompt')** заменяет в модели **EX_replace_block.mdl** блоки, параметр которых равен **100** на блоки **Gain** без запроса на подтверждение операции.

12.17. save_system

Назначение: Сохранение файла модели.

Синтаксис:

save_system

Сохранение открытой модели под текущим именем.

save_system('sys')

Сохранение модели **sys** под текущим именем.

save_system('sys', 'newname')

Сохранение модели **sys** под новым именем **newname**.

Пример 1:

Команда **save_system('my_model')** сохраняет модель в файле **my_model.mdl** .

Пример 2:

Команда **save_system('my_model,'new_model')** сохраняет модель в файле **new_model.mdl** .

12.18. set_param

Назначение: Установка параметров модели или блока.

Синтаксис:

set_param('obj', 'parameter1', value1, 'parameter2', value2, ...)

Команда выполняет присваивание новых значений **value1**, **value2…** параметрам **parameter1**, **parameter2…** модели (блока) **obj** . Имена параметров не чувствительны к регистру символов. Значения параметров чувствительны к регистру символов.

Пример 1:

Команда **set_param('EX_set_param', 'Solver', 'ode15s', 'StopTime', '100')** устанавливает метод решения (параметр **Solver**) **ode15s** и время окончания расчета (параметр **StopTime**) **100** для модели **EX_set_param.mdl** .

Пример 2:

Команда **set_param('EX_set_param/Step', 'After', '1.5')** устанавливает параметр **Final Value** блока **Step** модели **EX_set_param.mdl** равным **1.5**.

Пример 3:

Команда **set_param('EX_set_param/Transfer Fcn', 'Numerator', '[5 7 9]','Denominator', '[2 3 0]')** устанавливает параметры блока **Transfer Fcn** , таким образом, чтобы получить передаточную функцию следующего вида:

$$\frac{5s^2+7s+9}{2s^2+3s}$$

Команда может использоваться для изменения параметров модели или блока в процессе расчета. Однако не все параметры блоков могут быть изменены в этом случае. Например, нельзя изменить в процессе расчета размерности входных и выходных портов подсистемы или блока. Параметры блоков библиотеки **Power System Blockset** также нельзя изменять в процессе расчета. Следует иметь в виду еще и то, что иногда название параметра, данное в окне диалога, отличается от фактического названия параметра (имени переменной, которой присваивается значение параметра). Так, например, для блока **Step**, фактическое имя параметра **Initial Value** есть **Before**, а фактическое имя параметра **Final Value** есть **After**. Для выяснения фактических имен параметров можно открыть файл модели в каком-либо текстовом редакторе и просмотреть секцию, в которой описан данный блок. Ниже приведен пример текстового описания блока **Step** в файле модели:

Block {
BlockType Step
Name "Step"
Position [125, 75, 155, 105]
Time "0.1"
Before "10"
After "20"
SampleTime "0"
VectorParams1D on
} .

Из приведенного фрагмента хорошо видно, какие фактические имена имеют параметры данного блока.

12.19. simulink

Назначение: Команда открывает окно библиотеки блоков **simulink**.

Синтаксис:

Simulink

13. Отладчик Simulink моделей

Отладчик **Simulink** является инструментом для поиска и диагностирования ошибок в моделях **Simulink**. Он дает возможность точно определить проблемы, выполняя моделирование постепенно с отображением значений входных и выходных сигналов любого из интересующих блоков модели. **Simulink**-отладчик имеет и графический, и интерфейс пользователя командной строки. Графический интерфейс позволяет наиболее удобно использовать основные возможности отладчика. Интерфейс командной строки дает способ обращаться ко всем возможностям отладчика. Пользователь, как правило, работает с графическим интерфейсом отладчика и обращается к интерфейсу командной строки по мере необходимости.

13.1. Графический интерфейс отладчика Simulink моделей

Запуск графического интерфейса отладчика возможен одним из двух способов:

1. С помощью команды меню **Tools/Debugger** окна **Simulink** модели.
2. С помощью кнопки  панели инструментов окна **Simulink** модели.

После запуска отладчика на экране появится его окно (см. рис. 13.1).

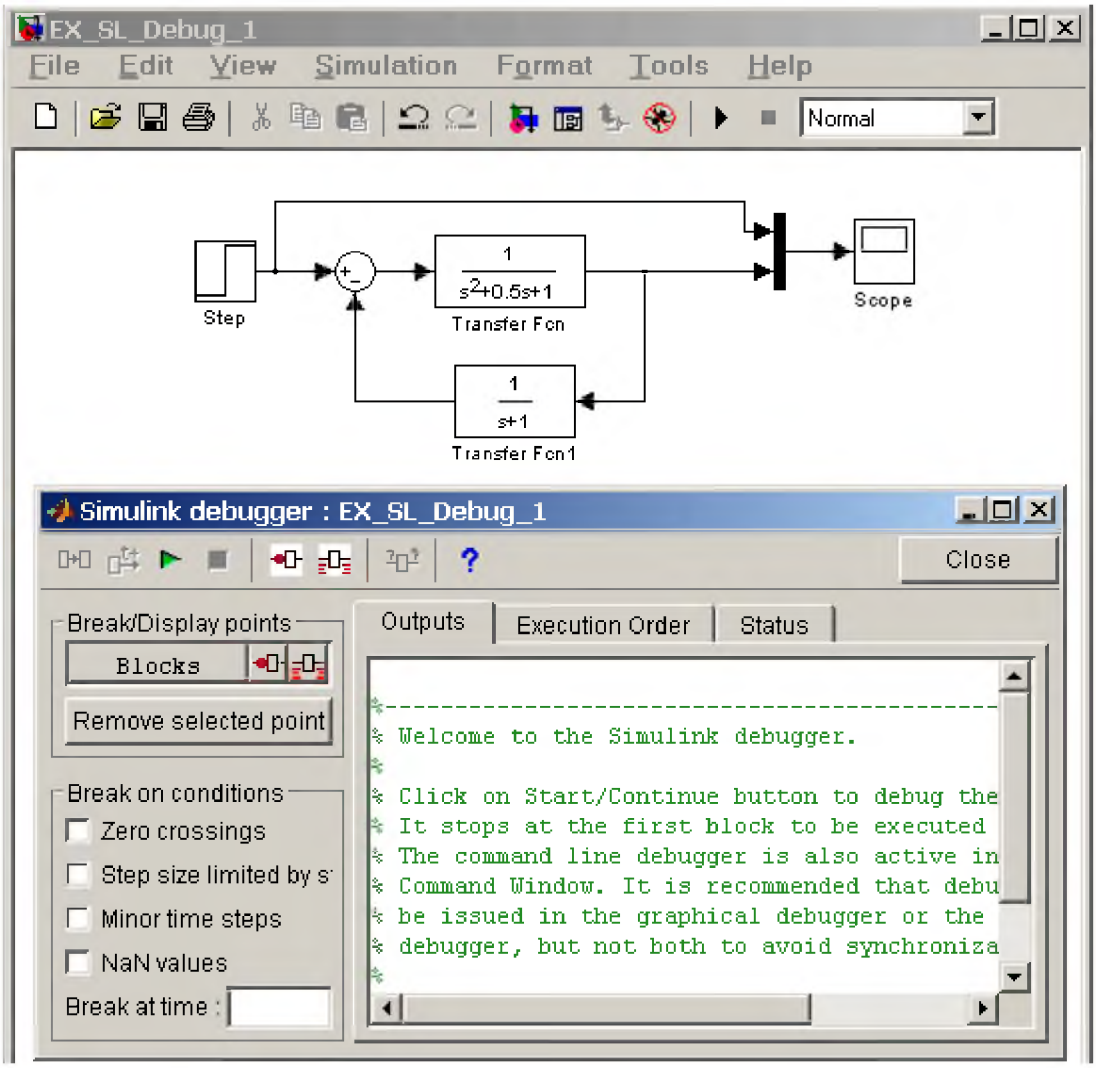


Рис. 13.1 Окно отладчика **Simulink** модели.

Окно отладчика содержит следующие элементы:

- Панель инструментов
- Список контрольных точек **Break/Display points**
- Панель задания точек прерывания по условию **Break on conditions**
- Главное окно отладчика

13.1.1. Панель инструментов

Общий вид панели инструментов показан на рис. 13.2.

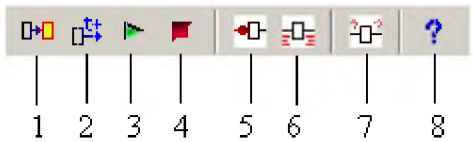


Рис. 13.2 Панель инструментов отладчика

Эта панель имеет следующие кнопки:

1. **Next Block** – Переход к следующему блоку. С помощью данной кнопки осуществляется режим отладки с остановкой моделирования после каждого выполненного блока.
2. **Next Time Step** – Переход к следующему временному шагу. С помощью данной кнопки выполняется режим отладки с остановкой моделирования после каждого выполненного временного шага.
3. **Start/Continue** – Начало/Продолжение отладки. Нажатие данной кнопки после запуска отладчика приводит к началу процесса моделирования и остановке его перед первым исполняемым блоком. Данная кнопка служит также для продолжения процесса отладки при установленных точках прерывания (остановки). Если точки прерывания установлены в модели, то нажатие данной кнопки позволяет продолжить моделирование и, затем, остановить его в заданной точке прерывания. Повторное нажатие кнопки возобновляет процесс моделирования и вызывает остановку на следующей точке прерывания. Если на текущем временном шаге все точки прерывания пройдены, то происходит переход на следующий временной шаг и остановка в первой точке прерывания на новом временном шаге.
4. **Stop** – Остановка отладки.
5. **Break before selected block** – Установка точки прерывания перед выделенным блоком. Точка прерывания не может быть установлена для виртуального блока, т.е. такого блока, чья функция является исключительно графической. На рис. 13.1 таким блоком является мультиплексор **Mux**. Список невиртуальных блоков можно посмотреть на вкладке **Execution Order** окна отладчика или вывести его командой **slist** при работе с отладчиком в режиме командной строки. При попытке установить контрольную точку для виртуального блока отладчик выведет на экран предупреждающее сообщение.

   Для установки точки прерывания достаточно выделить блок в окне модели и, затем, нажать данную кнопку. Название блока немедленно появится в окне **Break/Display points** отладчика, как это показано на рис. 13.3.

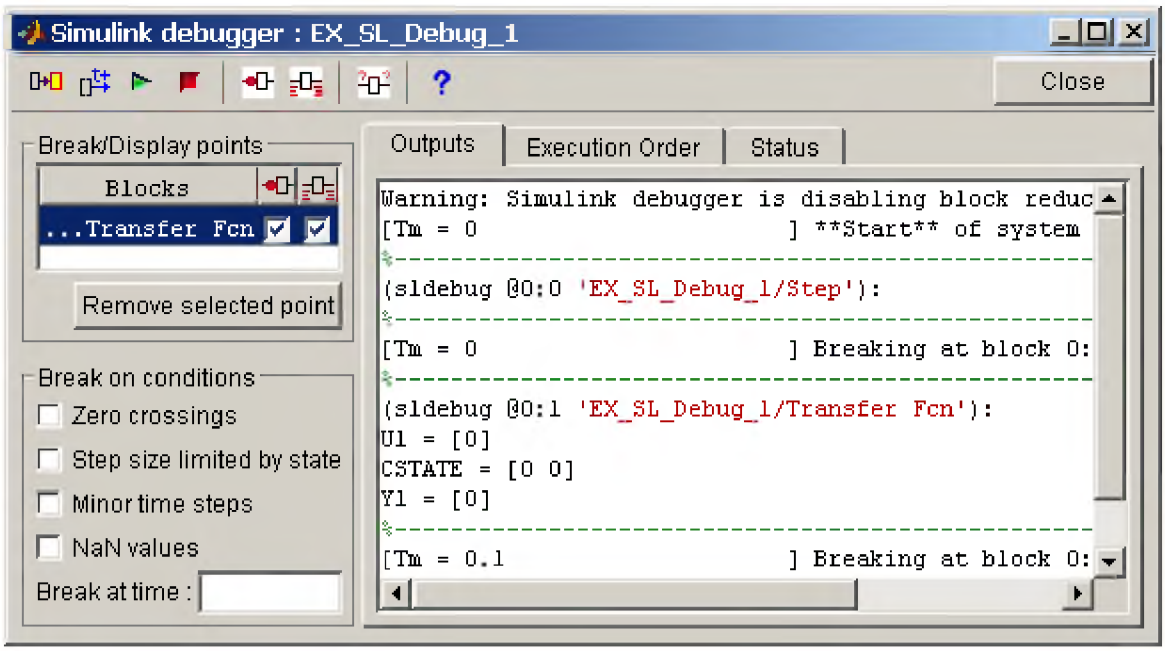


Рис. 13.3 Окно отладчика с установленной точкой прерывания для блока **Transfer Fcn**

6. **Display I/O of selected block with executed** – Отображать значения входных и выходных сигналов. Данная кнопка позволяет установить режим просмотра входных и выходных сигналов какого-либо невиртуального блока. Режим может быть установлен как для блока, для которого установлена точка прерывания, так и для блока для которого контрольная точка не установлена. Значения входных и выходных сигналов отображаются на вкладке **Outputs** окна отладчика. Входные сигналы отображаются идентификаторами **U1, U2, U3** и т.д., а выходные – **Y1, Y2, Y3** и т.д. Дополнительно здесь же отображаются значения переменных состояния – **CSTATE** (см. рис. 13.3).
7. **Display current I/O of selected block** – Показать значения входных и выходных сигналов для выделенного блока. Данной кнопкой удобно пользоваться, когда необходимо просмотреть значения сигналов какого-либо блока в момент остановки. Например, если для модели на рис. 13.1 установлена точка прерывания перед выполнением блока **Transfer Fcn** и остановка моделирования в этой точке произошла, то просмотреть значения сигналов сумматора можно выделив этот блок в окне модели и нажав кнопку отладчика.
8. **Help** – Вызов справки по отладчику.

13.1.2. Список контрольных точек Break/Display points

Окно списка контрольных точек (рис. 13.4) содержит список блоков, для которых установлены контрольные точки (графа **Blocks**), а также свойства этих точек указанные с помощью флажков. Пользователь снимая или устанавливая флажки может изменять свойства контрольной точки, а именно: задавать/убирать точку прерывания на входе блока (графа) или включать/выключать режим отображения значений сигналов блока (графа).

Удалить контрольную точку можно выделив ее в списке, и нажав кнопку **Remove selected point** (убрать выделенную точку).

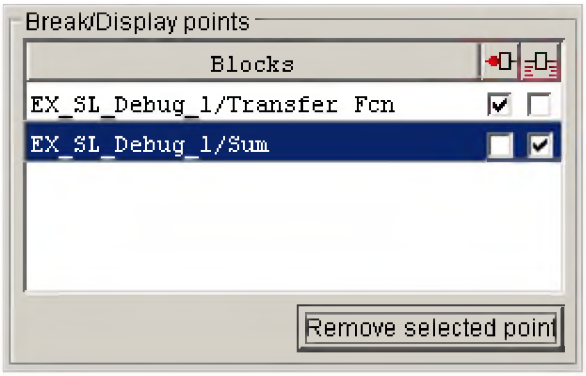


Рис. 13.4 Окно списка контрольных точек **Break/Display points**

13.1.3. Панель задания точек прерывания по условию Break on conditions

Панель (рис. 13.5) содержит список условий, при наступлении которых расчет должен быть остановлен.

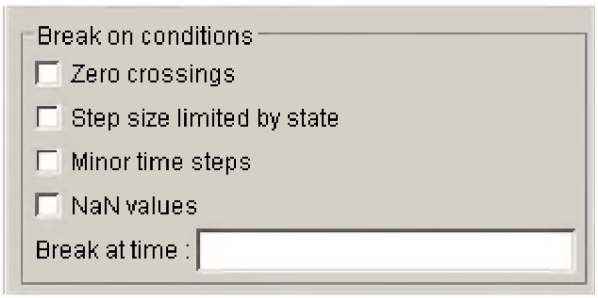


Рис. 13.5 Список условий прерывания расчетов

Список включает в себя:

1. **Zero crossings** – Переход сигнала через нулевой уровень при его скачкообразном изменении. На рис. рис. 13.6 показан пример модели, в которой имеет место такая ситуация. На рисунке хорошо видно, что в момент времени **t = 5 c** происходит скачкообразное изменение сигнала с пересечением нулевого уровня.

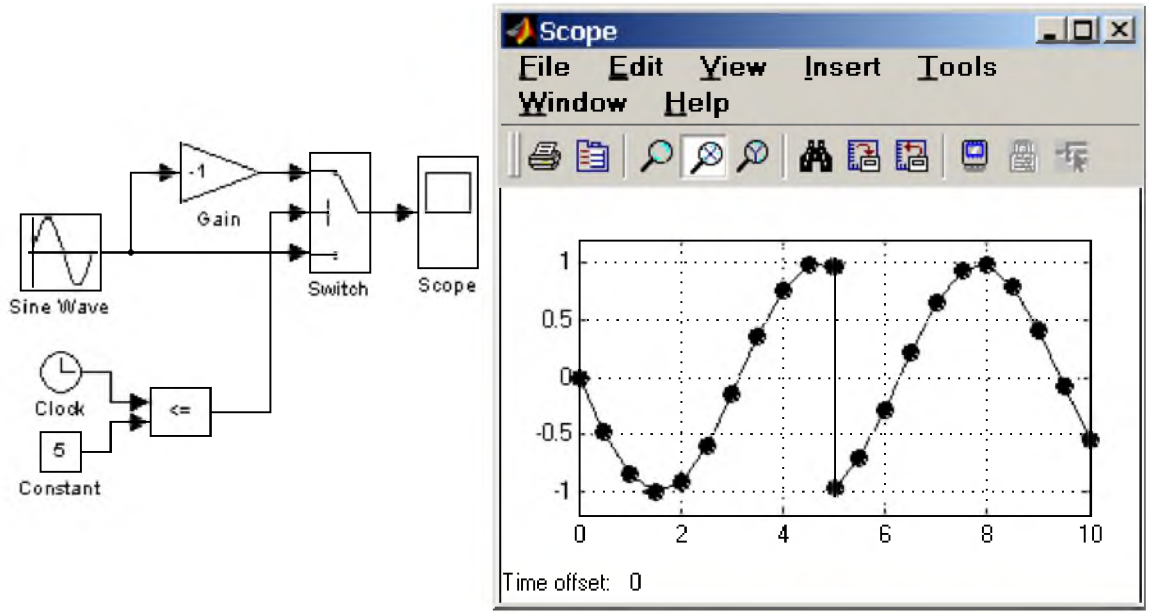


Рис. 13.6 Пример модели с изменением полярности сигнала

2. **Step size limited by state** – Состояние ограничивающее шаг расчета. Опция заставляет отладчик останавливать моделирование, когда модель использует решатель с переменным шагом и решатель сталкивается с состоянием требующим ограничения размера шага расчета. Эта опция полезна при отладке моделей, требующих, как кажется, чрезмерно много расчетных шагов.
3. **Minor time steps –** режим отладки с использованием внутренних (малых) шагов. При выполнении расчетов **Simulink** может уменьшать заданный шаг расчета для достижения нужной точности. Для того, чтобы увидеть и эти малые (внутренние) шаги необходимо установить опцию **Minor time steps**.
4. **NaN values –** Не числовое значение. Расчет будет прерван, когда вычисленное значение бесконечно или лежит вне диапазона значений, которые могут быть представлены компьютером, выполняющим моделирование (слишком малые или слишком большие значения). Эта опция полезна для точного определения вычислительных погрешностей в модели **Simulink**.
5. **Break at time –** Остановка в заданный момент времени. Параметр позволяет задать время до которого модель рассчитывается в обычном режиме. По достижении заданного времени расчет будет остановлен. Далее расчет необходимо возобновить с использованием возможностей отладчика. Данный режим удобен, если ошибка возникает не в начале интервала моделирования, а в какой-либо более поздний момент. В этом случае можно установить время остановки непосредственно перед появлением ошибки, а затем продолжить расчет в пошаговом режиме.

13.1.4. Главное окно отладчика

Главное окно содержит три вкладки:

1. **Outputs** – Отображение результатов работы в режиме отладки. На данной вкладке (рис. 13.7) отображается текущее модельное время **Tm** (или **Ti** для внутренних шагов), индекс контролируемого блока в виде **@s:b**, где **s** – номер модели (подсистемы), **b** – номер блока, а также имя блока с указанием полного пути к блоку. На этой же вкладке выводятся значения входных (**U**) и выходных (**Y**) сигналов блока, если соответствующая опция установлена.

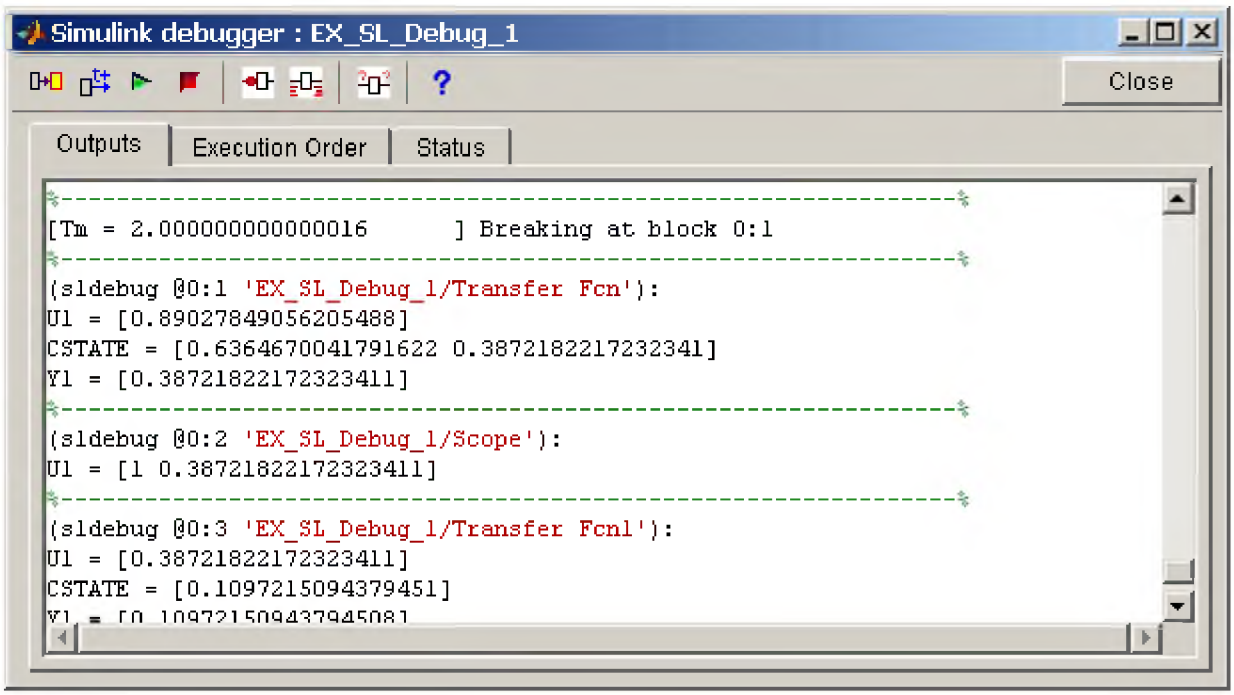


Рис. 13.7 Вкладка **Outputs** главного окна отладчика

2. **Execution Order** – Порядок выполнения. На вкладке отображается список невиртуальных блоков в порядке их выполнения. Блоки, расположенные в начале списка выполняются раньше, чем блоки, расположенные в конце списка. На рис. 13.8 показан пример данной вкладки для модели, изображенной на рис. 13.1.

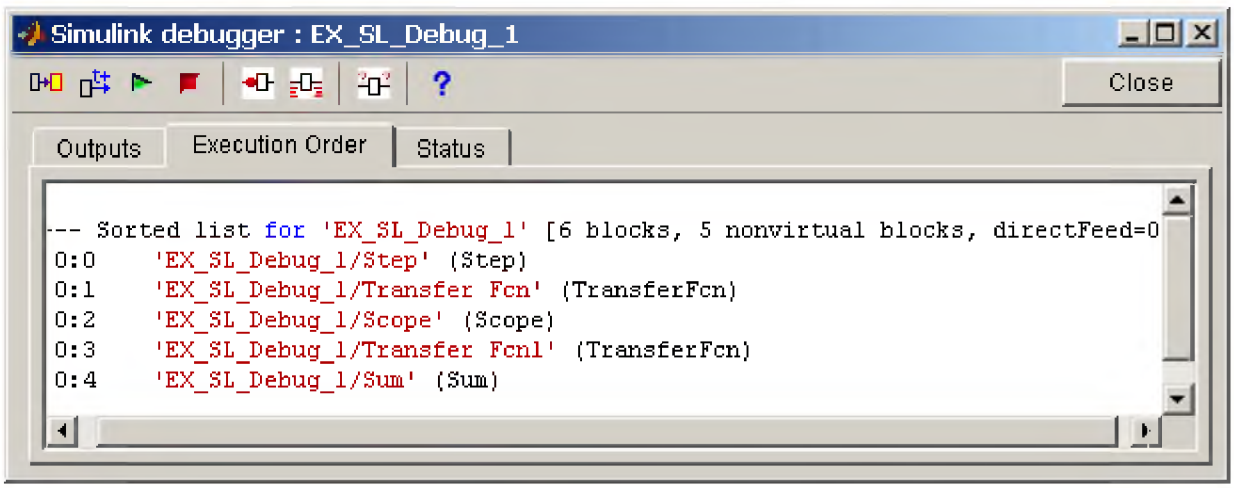


Рис. 13.8 Вкладка **Execution Order** главного окна отладчика

3. **Status** – Статус отладчика. На данной вкладке (рис. 13.9) отображается информация о настройках и текущем состоянии отладчика: значение текущего временного шага, количество точек прерывания, информация об установке точек прерывания по условию и т.п.

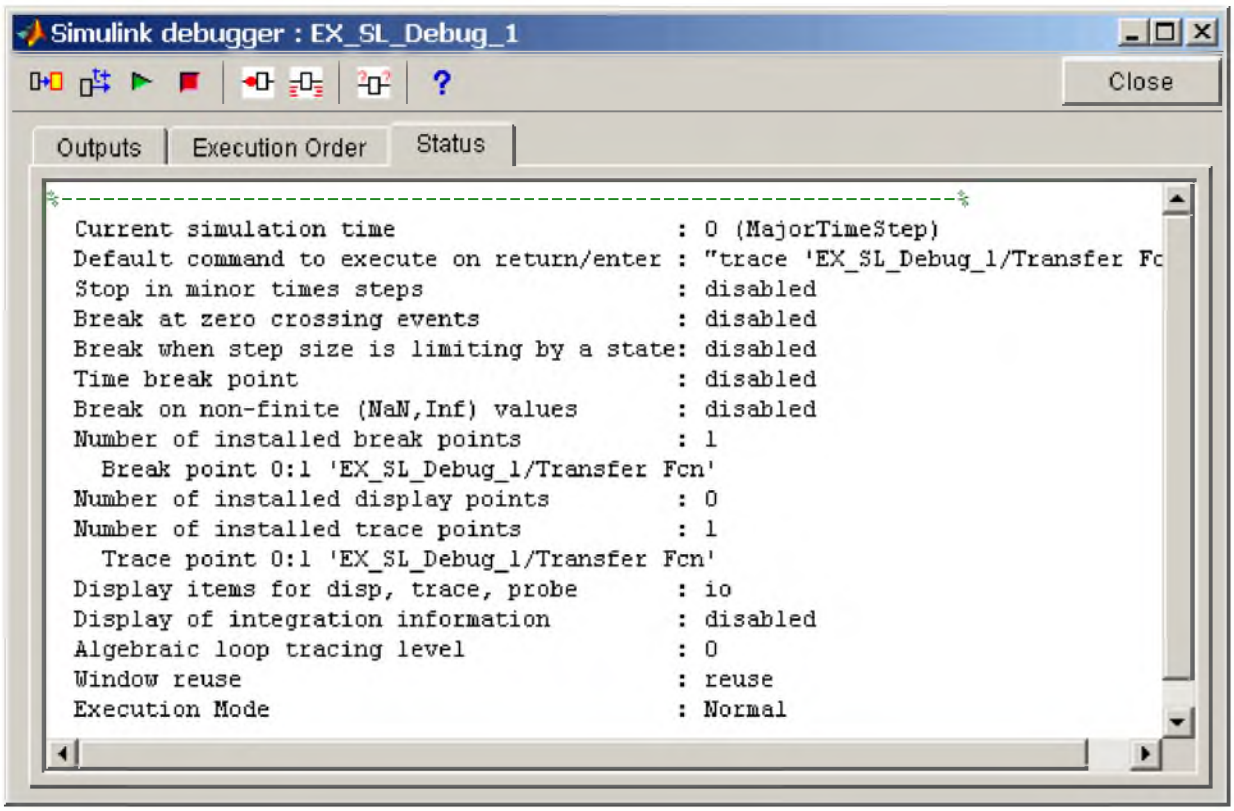


Рис. 13.9 Вкладка **Status** главного окна отладчика

Таким образом, запустив отладчик в графическом режиме, пользователь может провести пошаговую (по блокам или по временным шагам) отладку модели, установив при необходимости нужные контрольные точки.

13.2 Интерфейс командной строки отладчика Simulink моделей

Интерфейс командной строки дает пользователю доступ ко всем возможностям отладчика.

Запуск отладчика в режиме командной строки возможен с помощью команды, задаваемой в рабочем окне **MATLAB**:

sldebug('My_model') , где **My_model** – имя отлаживаемой модели.

Для работы с отладчиком требуется вводить команды в главном окне **MATLAB**. Список команд приведен в Таблице 13.1.

Таблица 13.1

Команда	**Краткая форма**	**Повтор**	**Назначение**
step	**s**	**да**	Переход к следующему блоку

next	**n**	**да**	Переход к следующему временному шагу
disp [s:b | gcb]	**d**	**да**	Показ входных и выходных сигналов блока при остановке
undisp <s:b | gcb>	**und**	**да**	Удаление блока из списка отображаемых
trace <s:b | gcb>	**tr**	**да**	Показ входных и выходных сигналов блока во время выполнения
untrace <s:b | gcb>	**unt**	**да**	Удаление блока из списка трассировки
probe [s:b | gcb]	**p**	**нет**	Показ входных и выходных сигналов указанного блока
break <s:b | gcb>	**b**	**нет**	Вставка точки остановки при входе в блок
bafter <s:b | gcb>	**ba**	**нет**	Вставка точки остановки при выходе из блока
bshow s:b	**bs**	**нет**	Показ указанного с помощью индекса блока
clear <s:b | gcb>	**cl**	**нет**	Удаление точки останова
zcbreak	**zcb**	**нет**	Прерывание при обнаружении скачкообразного перехода сигналом нулевого уровня (непредусмотренное пересечение нуля)
zclist	**zcl**	**нет**	Список блоков дающих непредусмотренное пересечение нуля
xbreak	**x**	**нет**	Прерывание при переменном шаге расчета в состоянии требующем ограничения шага расчета
tbreak [t]	**tb**	**нет**	Установка/удаление остановки в указанный момент времени
nanbreak	**na**	**нет**	Установка/удаление остановки при обнаружении не числового (**NaN, Inf**) значения
continue	**c**	**да**	Продолжение моделирования
run	**r**	**нет**	Окончание режима отладки и продолжение расчета в обычном режиме
stop	**sto**	**нет**	Остановка моделирования
quit	**q**	**нет**	Прерывание моделирования

status [all]	**stat**	**нет**	Показ параметров отладчика
states	**state**	**нет**	Показ текущих значений переменных состояния
systems	**sys**	**нет**	Показ списка невиртуальных подсистем
slist	**sli**	**нет**	Список невиртуальных блоков
minor	**m**	**нет**	Режим отладки с использованием внутренних (малых) шагов
ishow	**i**	**нет**	Включение/выключение режима показа информации об интегрирующих блоках
emode	**e**	**нет**	Вывод информации о текущем режиме моделирования (обычный или ускоренный)
probe level {all} | io		**нет**	Установить уровень подробности показа сигналов блоков (все либо только входные и выходные)
atrace level	**at**	**нет**	Установка уровня отображения информации при трассировке алгебраических контуров (**0** – ничего, **4** - все)
ashow <gcb | s:b>	**as**	**нет**	Показ алгебраического контура, содержащего указанный блок
ashow s#n	**as**	**нет**	Показ алгебраического контура с номером **n** в подсистеме (модели) **s**
ashow clear	**as**	**нет**	Отменить показ алгебраического контура

Часть команд приведенных в таблице требуют указания индекса блока (см. п. 13.1.4). При использовании таких команд вместо имени блока можно указывать команду **gcb** (получить путь текущего блока), предварительно выделив нужный блок в окне модели.

Пример командного окна **MATLAB** в процессе отладки модели показан на рис. 13.10.

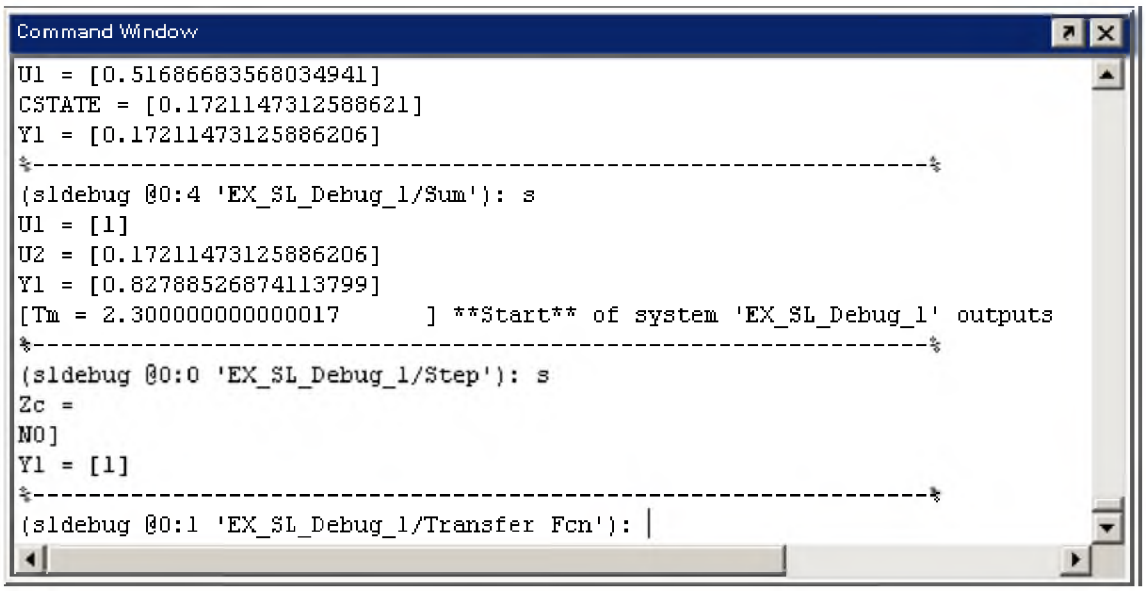


Рис. 13.10. Командное окно **MATLAB** в процессе отладки модели

14. Повышение скорости и точности расчетов

На точность и скорость расчета модели в **Simulink** можно воздействовать многими способами, включая структуру модели и ее параметры. Решающие модули **Simulink** работают точно и эффективно и с параметрами заданными для них “по умолчанию”. Однако для некоторых моделей можно добиться лучших результатов по скорости и точности, если задать более точно параметры решателя дифференциальных уравнений. Также, если предполагаемое поведение модели известно, то можно используя эту информацию повысить скорость и точность расчетов.

14.1. Повышение скорости расчета

Малая скорость моделирования может иметь много причин. Среди них можно выделить основные:

- Модель содержит блок **MATLAB Fcn**. При использовании блока **MATLAB Fcn** в модели **Simulink** на каждом расчетном шаге обращается к интерпретатору языка **MATLAB** для выполнения расчетов в данном блоке. Вместо блока **MATLAB Fcn ,** если это возможно, следует использовать блоки **Fcn** или **Math Function**.
- Модель включает **S**-функцию, написанную на языке **MATLAB**. В этом случае также происходит обращение к интерпретатору языка **MATLAB** на каждом расчетном шаге. Вместо **MATLAB S**-функции более предпочтительным было бы использование **S**-функций написанных на языках **C** или **Fortran** и откомпилированных в исполняемый машинный код в виде динамической библиотеки.
- Модель включает блок памяти **Memory**. Использование блока памяти заставляет решающие модули с переменным порядком (**ode15s** и **ode113**) выполнять снижение порядка до первого на каждом расчетном шаге.
- Максимальный размер шага (**Max step size**) слишком мал. Если этот параметр был изменен, то следует попробовать выполнить моделирование снова, установив этот параметр равным **auto**.
- Задана слишком высокая точность расчетов. Обычно значение абсолютной погрешности (**Relative tolerance**) заданное равным **0.1%** достаточно для большинства расчетов. При слишком малых значениях этого параметра шаг расчета может оказаться также достаточно малым, что приведет к замедлению расчетов.

- Задан слишком большой интервал расчета по времени. Как правило, при моделировании динамических систем переходные процессы представляют больший интерес, нежели установившийся режим. По достижении установившегося режима расчет можно прекратить, поскольку далее никаких изменений в состоянии модели не будет. Желательно заранее оценить предполагаемое время расчета исходя из знаний о моделируемом объекте.
- Модель может оказаться жесткой, а используемый решатель не предназначен для моделирования жестких систем. Следует попробовать использовать методы **ode15s** или **ode23tb** и сравнить время расчета при решении этими методами.
- В модели используются блоки, шаг дискретизации которых (**Sample time**) не является кратным. В этом случае **Simulink** уменьшает шаг расчета до такого значения, чтобы он был кратен шагу дискретизации каждого блока. Например, если шаг дискретизации одного блок равен **0.5**, а другого – **0.7**, то **Simulink** установит максимальное значение шага расчета равное **0.1** .
- Модель содержит алгебраический контур. Алгебраические контуры рассчитываются в **Simulink** с помощью итерационной процедуры на каждом шаге расчета, что замедляет общее время расчета.
- Модель имеет блок **Random Number**, который передает свой выходной сигнал на вход интегратора (блок **Integrator)**. Предпочтительнее использовать блок **Band-Limited White Noise block** из библиотеки **Sources**.
- Модель включает большое число блоков **Scope**. Блоки требуют значительного объема памяти для хранения отображаемых данных, что может привести к использованию компьютером виртуальной (дисковой) памяти и существенному замедлению расчетов.
- В блоках **Scope** параметр **Limit data points to last** задан значительно меньшим, чем фактическое число расчетных шагов (либо флажок этого параметра снят). В этом случае, при превышении числом шагов значения параметра **Limit data points to last**, для отображения каждой новой расчетной точки будет выполняться процедура выделения памяти, что существенно замедляет скорость расчета. Рекомендуется заранее установить параметр **Limit data points to last** большим, чем фактическое число расчетных шагов. Имеет смысл также задать параметр **Decimation** (прореживание) большим **1**, чтобы сократить число хранимых блоком **Scope** данных.

Скорость расчета можно также повысить в несколько раз, используя ускоренный (**Accelerator**) режим расчета. Это можно сделать с помощью меню **Tools** или панели инструментов. В ускоренном режиме расчета предварительно проводится трансляция модели в исполнительный код (**dll**-файл), а затем уже проводится сам расчет. Некоторые дополнительные затраты времени на трансляцию с лихвой окупаются ускорением расчета модели. Однако при изменении структуры модели процедура трансляции будет повторена. К сожалению, ускоренный режим расчета не может быть использован в моделях имеющих алгебраические контуры.

Существенный выигрыш по времени может дать использование дискретных моделей вместо непрерывных.

Наиболее существенным же с точки зрения скорости вычислений может оказаться правильный выбор уровня детализации модели. К примеру, если выполняется моделирование системы электроснабжения города, вряд ли стоит моделировать каждый потребитель электрической энергии: электрический двигатель, чайник, сварочный аппарат и т.п. Вполне достаточным будет создание обобщенных моделей электрических потребителей на уровне заводского цеха, жилого дома, трамвайного парка и т.п.

14.2. Повышение точности расчета

Чтобы проверить достаточно ли точно выполняется моделирование, следует провести сравнительные расчеты с разными значениями параметра **Relative tolerance** (относительная погрешность). К примеру,

можно провести расчет с заданным “по умолчанию” значением этого параметра – **1e-3** и с меньшим (**1e-4**) значением. Если результаты расчетов отличаются незначительно, то можно полагать, что найденное решение является верным. Если решения значительно отличаются в начальной стадии, то следует задать в явном виде достаточно малый начальный шаг расчета (**Initial step size**).

Если решение оказывается неустойчивым, то это может быть вызвано следующими причинами:

- Моделируемая система сама является неустойчивой.
- Используется метод **ode15s**. Следует ограничить порядок величиной **2** или использовать метод **ode23s**.

Если решение кажется не точным:

- Следует задать в явном виде параметр **Absolute tolerance** (абсолютная погрешность) и выполнить ряд расчетов, уменьшая величину этого параметра.
- Если уменьшение абсолютной погрешности точность расчетов не улучшается, следует уменьшить относительную погрешность (что приведет уменьшению шага расчета) либо в явном виде задавать достаточно малую величину максимального шага расчета.

15. Обзор набора инструментов Simulink Performance Tools

Simulink Performance Tools включает четыре приложения, которые расширяют возможности **Simulink** и существенно увеличивают производительность программы. Использование этих инструментов может значительно повысить скорость процесса моделирования. Пользователь получает инструмент для сравнения разных вариантов модели, а также для быстрого тестирования модели.

Набор инструментов содержит:

- **Simulink Accelerator**, ускоряющий моделирование, благодаря использованию скомпилированного кода вместо того, чтобы запускать модель в режиме интерпретатора.
- **Simulink Model Profiler**, собирающий данные о производительности в ходе выполнения модели и генерирующий отчет со списком информации о времени выполнения для каждой составляющей в детальной и обобщенной формах.
- **Simulink Model Coverage**, помогающий улучшить модель и определить области повышенного риска в моделях **Simulink** и **Stateflow**. Приложение генерирует детальный **HTML** отчет, показывающий какие блоки, состояния и условия были выполнены в ходе имитации.
- **Simulink Model Differencing**, позволяющий сравнить две модели **Simulink** и генерирующий графическое изображение различий.

15.1. Simulink Accelerator

Simulink Accelerator использует технологию генерации кода и пользовательский компилятор языка **C** для создания выполняемого файла (**dll**-файла), который заменяет интерактивный код, обычно используемый программой **Simulink** (в состав программы **Simulink** входит собственный компилятор **lcc**).

Simulink Accelerator обеспечивает:

- Имитацию приблизительно в **2** – **10** раз более быструю, чем в нормальном (не ускоренном режиме). Степень повышения производительности связана с размером и сложностью модели.

Увеличение размера или сложности обычно приводит к увеличению степени производительности.

- Тесную интеграцию с наборами блоков **Simulink**. **Simulink Accelerator** полностью совместим с наборами блоков **Simulink**. В частности, с **Fixed-Point Blockset**, **Power System Blockset** и **DSP Blockset**.
- Поддержку отладчика **Simulin**k. Процесс отладки больших и сложных моделей значительно ускоряется.
- Доступ из программ. Контроль за выполнением модели может быть осуществлен из командной строки **MATLAB** или из **m**-файлов.

Для перехода в ускоренный режим расчета необходимо в меню **Simulation** выбрать пункт **Accelerator**. После запуска модели на расчет будет произведена компиляция модели и выполнен расчет. При повторных запусках, если структура модели не менялась, компиляция выполнятся не будет, а будет сразу производиться расчет. При изменении параметров блоков повторная компиляция также не производится. Для возврата в обычный режим расчета следует меню **Simulation** выбрать пункт **Normal**.

При использовании ускоренного режима расчета следует иметь в виду, что модели, имеющие замкнутые алгебраические контуры, не могут быть рассчитаны в этом режиме.

15.2. Simulink Model Profiling

Simulink Model Profiling собирает данные о производительности в процессе выполнения модели и затем генерирует отчет, называемый профилем имитации на основании собранных данных. Этот отчет состоит из двух **HTML**-файлов: обобщающий файл и детальный файл. Обобщающий файл аккумулирует временную информацию и выводит ее в список, упорядоченный по временам выполнения для каждого метода. Детальный файл показывает, как много времени использует **Simulink** выполняя каждый метод, требующийся для моделирования, включая производные и основные методы.

Для выполнения профилирования необходимо выполнить команду **Profiler** в меню **Tools** и запустить модель на расчет. По завершении расчета будет открыт файл отчета в окне справочной системы. Гиперссылки в отчете позволяют просмотреть детальную информацию для каждого метода. В результате пользователь может легко локализовать области модели, которые требуют наибольшего времени выполнения и быстро определить, где необходимо сконцентрировать усилия по оптимизации. На рис. 15.1 приведен пример модели и фрагмент отчета профилирования для нее.

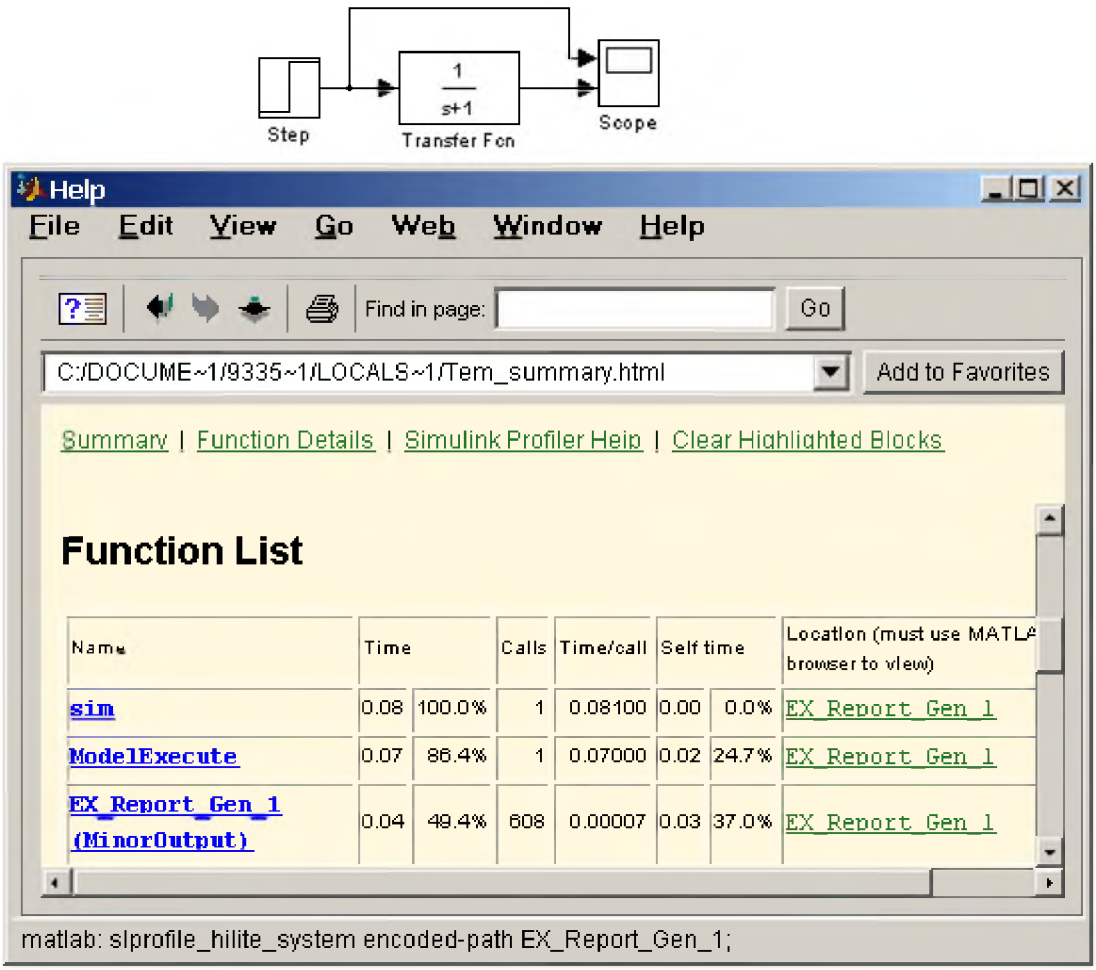


Рис. 15.1 Пример модели и отчета профилирования.

15.3. Simulink Model Coverage

При разработке больших моделей, имеющих сложную логику переключения путей по которым распространяются сигналы, пользователь может столкнуться с проблемой тестирования модели. В ходе тестирования пользователь обычно пытается разработать тест, который охватывал бы все возможные пути, чтобы быть уверенным, что модель полностью проверена. **Simulink Model Coverage** помогает проверить эффективность подобных проверочных тестов. Используя **Simulink Model Coverage**, пользователь может интерпретировать поведение модели внутри индивидуальных блоков **Simulink** и объектов **Stateflow**, определить степень выполнения имитации (за счет вычисления количества выполнений каждого из блоков), а также идентифицировать избыточность или недостаточность частей модели. Для определения полноты тестирования модели возможна комбинация данных, полученных из разных имитаций.

Simulink Model Coverage обеспечивает:

- Охват блоков **Simulink** и объектов **Stateflow**.
- Генерацию **HTML** документа, который представляет собой полный отчет по выполнению частей модели.
- Сохранение и загрузку данных охвата между сеансами имитации.

- Поддержку интерфейса для ввода команд, который автоматизирует выполнение имитаций и сбор данных.

Simulink Model Coverage позволит получить необходимый уровень тестирования разработки и определить количество тестов, необходимых для полной проверки. Анализ набора тестов в ходе разработки существенно уменьшает риск дефектов конструкции на поздних стадиях создания модели.

Для использования **Simulink Model Coverage** необходимо задать параметры отчета с помощью пункта **Coverage Setting** меню **Tools**. После выполнения моделирования будет открыт файл отчета в окне справочной системы.

На рис. 15.2 приведена схема модели и отчет, полученный я помощью **Simulink Model Coverage**. Из рисунка видно, что при данных параметрах схемы выполняется тестирования только 50% модели. Для полной проверки модели необходимо, чтобы сигнал, подаваемый на управляющий вход ключа менял свою полярность.

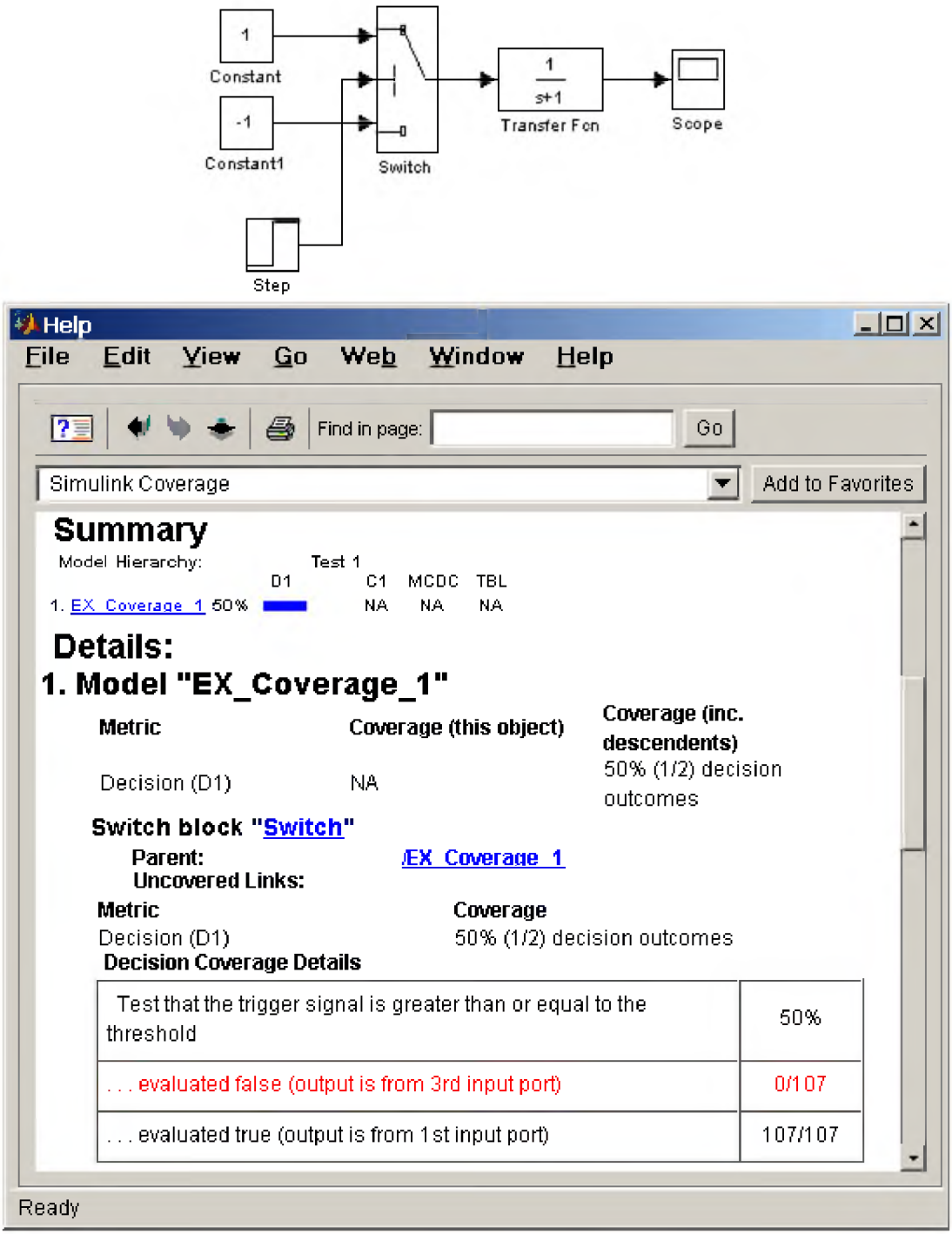


Рис. 15.2 Пример модели и отчета, полученного с помощью

Simulink Model Coverage

15.4. Simulink Model Differencing

Simulink Model Differencing сравнивает две **Simulink**-модели и генерирует графическое изображение различий между ними. На данном изображении выделяются одинаковые блоки моделей, имеющие различные атрибуты (красным цветом) и блоки, которые присутствуют только в одной из двух моделей

(синим цветом). Пользователь может настроить изображение, чтобы просмотреть только блоки с графическими различиями, только блоки с неграфическими отличиями или блоки с любыми отличиями. Для выполнения сравнения моделей необходимо выполнить команду **Model differences\Merge/Compare two models** из меню **Tools** окна первой модели и в процессе диалога выбрать файл второй модели. Возможно также выполнить сравнение текущего состояния модели и ее последней записи на диске.

На рис. 15.3 показан пример сравнения моделей.

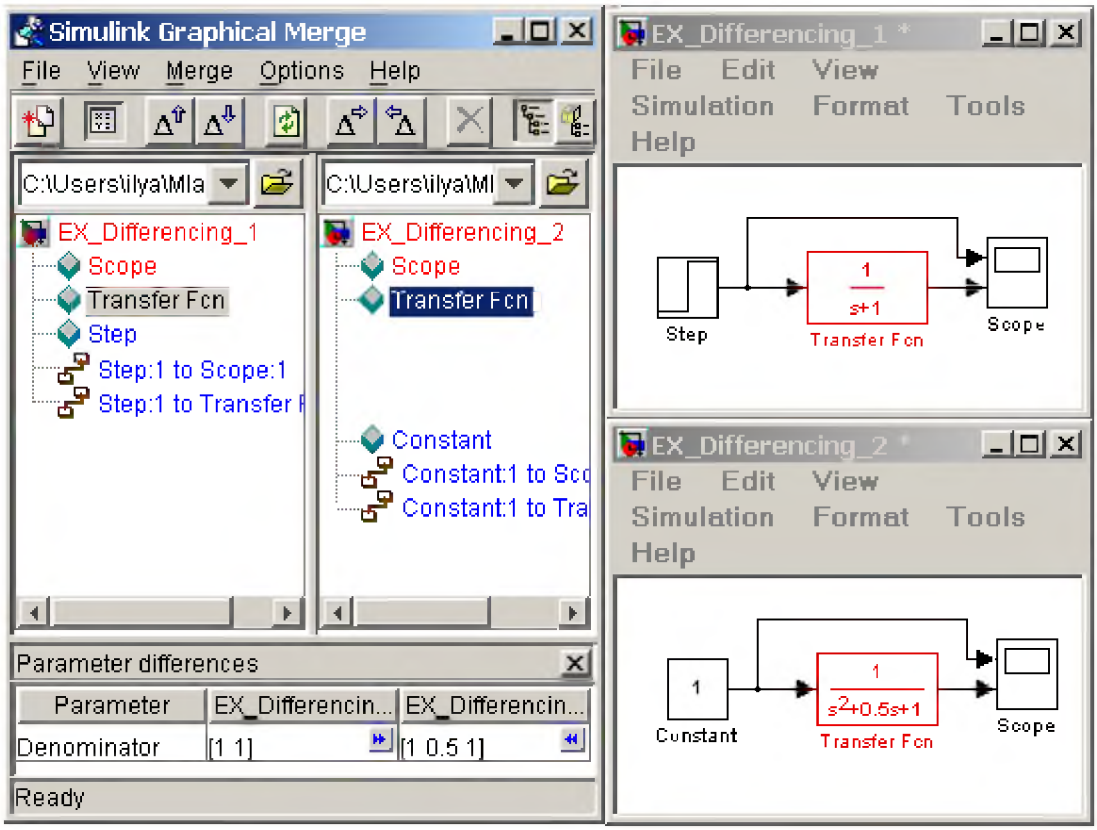


Рис. 15.3. Сравнение моделей с помощью **Simulink Model Differencing**

16. Simulink-функции

16.1. Введение

Simulink-функции (**S-функции, S-functions**) являются описанием блока на одном из языков программирования: **MATLAB, C, C++, Ada**, или **Fortran**. Набор стандартных блоков **Simulink**, достаточно обширен, однако в практике моделирования встречаются ситуации, когда нужного блока нет, либо структурное моделирование делает модель слишком сложной. В этом случае необходимо использовать технологию **S-функций** для создания нужного блока. С помощью языков программирования пользователь может создать описание сколь угодно сложного блока и подключить его к **Simulink**-модели, при этом с точки зрения взаимодействия пользователя с моделью, блок на основе **S-функции** ничем не отличается от стандартного библиотечного блока **Simulink**. Создаваемые блоки могут быть непрерывными, дискретными или гибридными. **S-функции**, созданные на **C, C++, Ada** или **Fortran** компилируются в исполняемые (***.dll**) файлы, за счет чего обеспечивается повышенная скорость выполнения таких блоков. Такие **S-функции** обладают еще и дополнительными возможностями, которые включают работу с разными типами данных (целыми, действительными и

комплексными числами различной степени точности), использование матриц в качестве входных и выходных переменных (**MATLAB S-функции** могут оперировать только векторами в качестве входных и выходных переменных), а также больший набор внутренних функций (**callback-методов**).

Чаще всего **S-функции** используются при создании новых библиотечных блоков, блоков, обеспечивающих взаимодействие **Simulink** с аппаратными средствами компьютера, при создании блоков на основе математических уравнений, блоков реализующих анимационные возможности **MATLAB**, а также при подключении к модели **Simulink** существующего программного кода языков высокого уровня.

16.1. Блок **S-function**

S-функция подключается к модели **Simulink** с помощью библиотечного блока **S-function** (библиотека **Functions&Tables**). На рис. 16.1 показано окно модели с блоком **S-function** и его окно диалога.

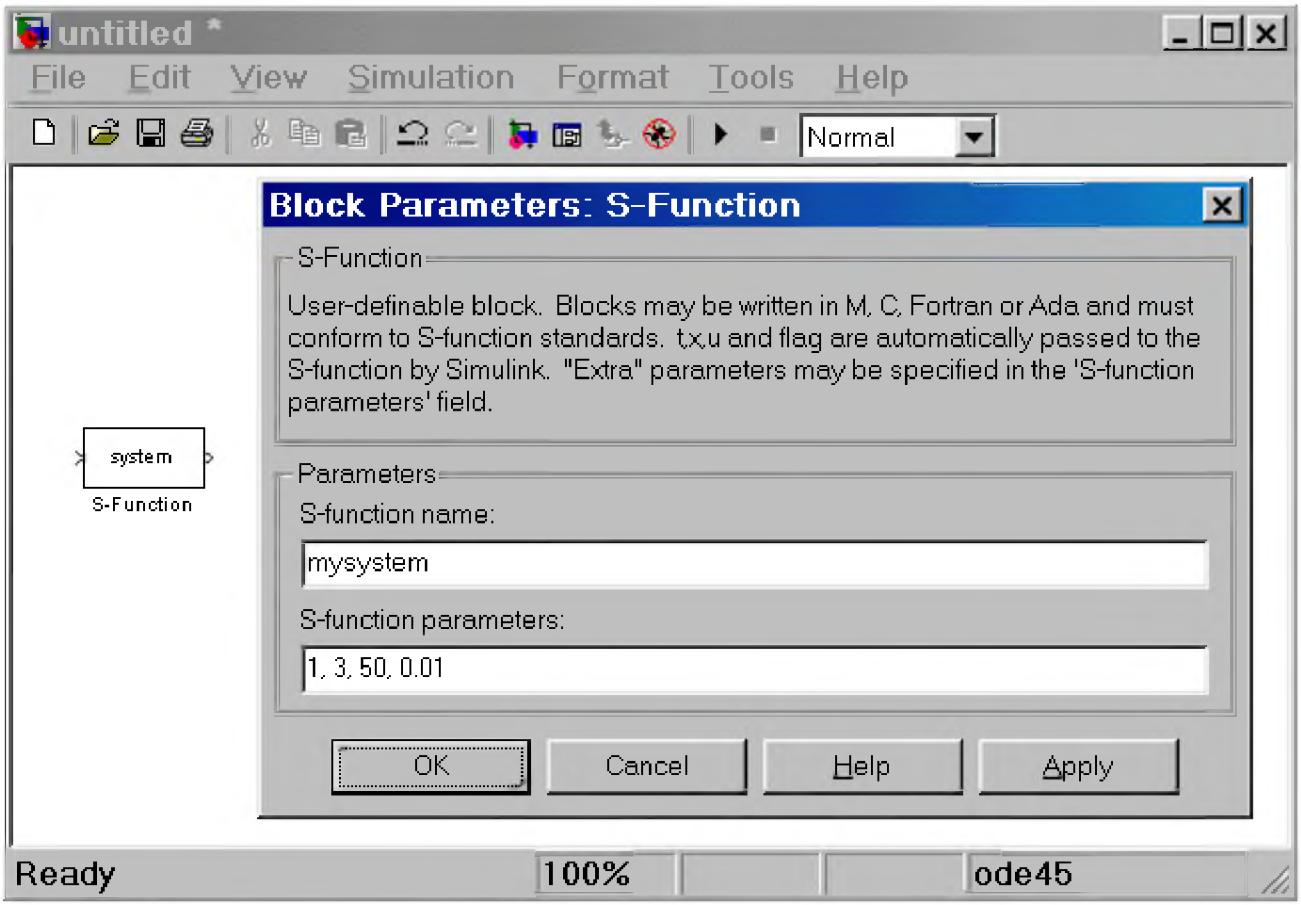


Рис. 16.1 Блок **S-function** и его окно диалога

Параметрами блока являются:

- **S-function name** – Имя **S-функции**. Имя **S-функции** не должно совпадать с именем модели (**mdl**-файла).
- **S-function parameters** – Параметры **S-функции**, передаваемые в нее через окно диалога. Параметры записываются в окне диалога в том же порядке, что и в заголовке **S-функции**.

16.2. Математическое описание S-функции

Simulink-блок однозначно описывается наборами входных переменных **u**, переменных состояния **x** и выходных переменных **y** (рис. 16.2).

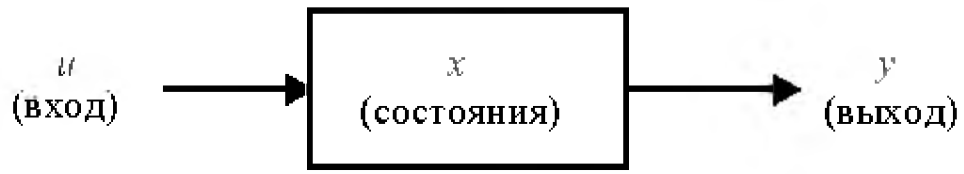


Рис. 16.2 Общий вид **Simulink**-блока

В математической форме блок можно описать в общем виде следующей системой уравнений:

$y = f_0(t, x, u)$ **(выходы)**

$\dot{x}_c = f_d(t, x, u)$ **(производные непрерывных переменных состояния)**

$x_{d_{k+1}} = f_u(t, x, u)$ **(дискретные переменные состояния)**

где $x = x_c + x_d$

16.3. Этапы моделирования

Процесс расчета модели выполняется **Simulink** в несколько этапов. На первом этапе выполняется инициализация модели: подключение библиотечных блоков к модели, определение размерностей сигналов, типов данных, величин шагов модельного времени, оценка параметров блоков, а также определяется порядок выполнения блоков и выполняется выделение памяти для проведения расчета. Затем **Simulink** начинает выполнять цикл моделирования. На каждом цикле моделирования (временном шаге) происходит расчет блоков в порядке, определенном на этапе инициализации. Для каждого блока, **Simulink** вызывает функции, которые вычисляют переменные состояния блока **x**, производные переменных состояния, и выходы **y** в течение текущего шага модельного времени. Этот процесс продолжается, пока моделирование не будет завершено. На рис. 16.3 показана диаграмма, иллюстрирующая этот процесс.

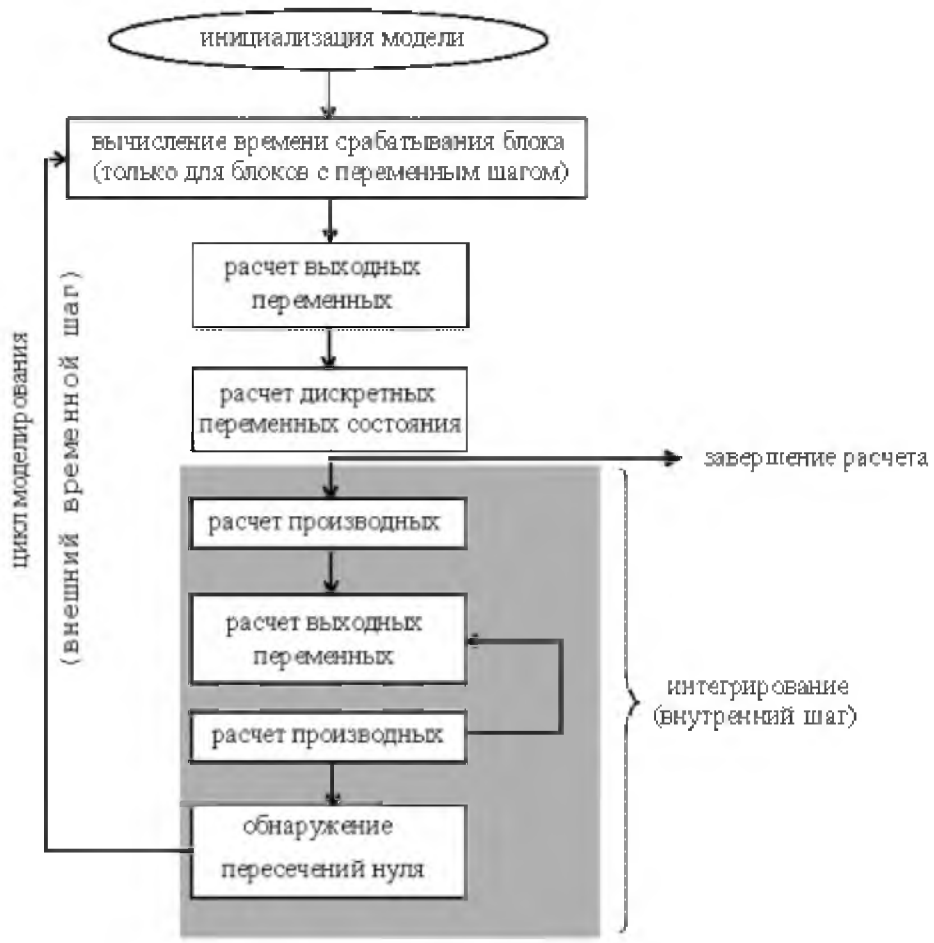


Рис. 16.3 Процесс моделирования

16.4. Callback-методы S-функции

Каждая задача при вызове **S-функции** в процессе моде моделирования решается с помощью специальной внутренней функцией (**callback-метода**). В **MATLAB S-функции** используются следующие методы:

1. **mdlInitializesizes –** Инициализация. До начала первого цикла моделирования, **Simulink** инициализирует **S-функцию**. В течение этого этапа **Simulink**:
   - Инициализирует структуру с именем **SimStruct**, содержащую информацию о **S-функции**.
   - Устанавливает количество и размерность входных и выходных портов.
   - Устанавливает шаг модельного времени для блока.
   - Выделяет память для хранения переменных и устанавливает размерность массивов.
2. **mdlGetTimeOfNextVarHit** – Вычисление времени следующего срабатывания блока (для блоков с дискретным переменным шагом расчета).
3. **mdlOutputs** – Вычисление значений выходных сигналов на внешнем шаге моделирования. На этом этапе рассчитанные выходные сигналы блока передаются на его выходные порты.
4. **mdlUpdate** – Расчет дискретных переменных состояния на внешнем шаге моделирования. Дискретные переменные состояния сохраняют свое значение до следующего цикла моделирования.
5. **mdlDerivatives** – Расчет производных переменных состояния.
6. **mdlTerminate** – Завершение работы **S-функции**.

Если **S-функция** содержит непрерывные переменные состояния, **Simulink** вызывает **callback-методы mdlDerivatives** и **mdlOutputs** для расчета производных переменных состояния и выходных переменных на внутренних шагах моделирования.

Вызов каждого из методов **Simulink** задает с помощью переменной **flag**, являющейся входным параметром **S-функции**.

16.5. Основные понятия S-функции

Для того, чтобы создать **S-функцию** правильно необходимо определить основные понятия, используемые в технологии создания **S-функций**. К эти понятиям относятся:

- **Direct feedthrough** – Прямой проход. Проход входных сигналов на выход. Прямой проход реализуется в **S-функций**, если в выражениях для выходных переменных присутствуют входные переменные, либо при расчете времени следующего срабатывания блока также используются входные переменные. Установка правильного значения параметра **Direct feedthrough** очень важна, поскольку именно с помощь него **Simulink** определяет наличие в модели замкнутых алгебраических контуров.
- **Dynamically sized inputs** – Динамическая размерность входов. **S-функция** может быть написана таким образом, чтобы обеспечить произвольную размерность векторов входных и выходных переменных, а также векторов состояния непрерывной и(или) дискретной части системы. В этом случае фактическая размерность переменных определяется в самом начале процесса моделирования и устанавливается равной размерности входных сигналов. Чтобы задать динамическую размерность какой-либо переменной, нужно задать значение размерности для этой переменной равное **-1** (минус один) в соответствующем поле структуры **sizes** (см. приведенный ниже шаблон **S-функции**).
- **Setting sample times and offsets -** Установка шагов модельного времени и смещений. **S-функция** может задавать время срабатывания достаточно гибко. **Simulink** обеспечивает следующие варианты задания шага модельного времени:

1. **Continuous sample time** – Непрерывное модельное время. Задается для систем имеющих непрерывные переменные состояния. Для этого типа **S-функций** выходные переменные вычисляются на внутреннем шаге моделирования.
2. **•Continuous but fixed in minor time step sample time** – Непрерывное модельное время с фиксированным шагом во внутреннем цикле. Задается для **S-функций**, выходные переменные которых должны изменяться только в соответствии с внешним шагом моделирования, но должны быть неизменными на внутреннем.
3. **Discrete sample time** – Дискретное модельное время. Задается для дискретной системы (дискретной части системы). Пользователь должен задать шаг модельного времени **sample time** и смещение (задержку) **offset**, чтобы определить моменты времени, в которые **Simulink** должен вызвать на выполнение данный блок. Величина смещения не может превышать величину шага модельного времени. Время срабатывания блока определяется выражением: **TimeHit = (n * sample_time) + offset** , где **n** – целое число шагов расчета. Если задано дискретное модельное время, то **Simulink** обращается к методам **mdlUpdate** и **mdlOutputs** на каждом внешнем шаге моделирования.
4. **Variable sample time** – Дискретный переменный шаг расчета. Модельное время дискретное, но интервалы времени между срабатываниями блока могут быть разными. В начале каждого шага моделирования **S-функция** должна определить значение времени следующего срабатывания. Для этого используется **mdlGetTimeOfNextVarHit** метод.

5. **Inherited sample time** – Наследуемый шаг расчета. В некоторых случаях работа блока не зависит от выбора варианта задания шага модельного времени. На пример, для блока **Gain** не имеет значения, какой шаг модельного времени реализован – блок выполняет усиление входного сигнала для любого варианта **sample time**. В подобных случаях параметр **sample time** может быть унаследован от предыдущего или последующего блока, либо от блока, имеющего наименьший шаг расчета.

16.6. Создание S-функций на языке MATLAB

Наиболее простой и быстрый путь создать **S-функцию** – это написать ее на языке **MATLAB** с использованием файла-шаблона. И хотя создание **S-функций** на языке **MATLAB** имеет некоторые ограничения (например, **MATLAB S-функция** может иметь только по одному входному и выходному порту, а также передаваемые и принимаемые данные через эти порты могут быть только скалярами и векторами типа **double**), этот способ является наилучшим с точки зрения изучения механизма работы **S-функции**.

Ниже приводится шаблон **S-функции** с авторским переводом комментариев. Оригинальный файл шаблона **sfuntmpl.m** находится в папке **...\toolbox\simulink\blocks** .

Шаблон **MATLAB S-функции**:

```
function [sys,x0,str,ts] = sfuntmpl(t,x,u,flag)
%
% SFUNTMPL - Базовый шаблон для создания MATLAB S-функции.
%   С помощью MATLAB S-функции пользователь может задать систему
%   обыкновенных дифференциальных уравнений (ODE), уравнения дискретной
%   системы, и(или) любой алгоритм, описывающий работу Simulink-блока.
%
%   Базовая форма синтаксиса S-функции выглядит следующим образом:
%       [sys,x0,str,ts] = sfunc(t,x,u,flag,p1,...,pn)
%
%      Параметры S-функции:
%
%  t    -   Текущее время
%  x    -   Вектор переменных состояния системы
%  u    -   Вектор входных сигналов
%  flag -   Флаг - целое число, определяющее какая функция внутри S-функции
%           выполняется при вызове.
%  p1,...,pn - Параметры S-функции, задаваемые в окне диалога
%              блока "S-function".
%
%   Результат, возвращаемый (вычисляемый) S-функцией в момент времени t
%   зависит от значения переменной flag, значения вектора состояния системы x
%   и текущего значения вектора входного сигнала u.
%
%   Результаты работы S-функции в зависимости от значения переменной flag
% приведены в таблице:
%  ____________________________________________________________________
% |      |            |                            |                   |
% | flag | РЕЗУЛЬТАТ  |  ВЫПОЛНяЕМАя ФУНКЦИя      |     ОПИСАНИЕ     |
% |      |            |    (callback-метод)        |                   |
% |______|____________|____________________________|___________________|
% |  0   | [sizes,x0, | mdlInitializesizes         |  Инициализация:   |
% |      |  str,ts]   |                            | начальных условий,|
% |      |            |                            | вектора модельного|
% |      |            |                            | времени, размерности|
% |  1   |    dx      | mdlDerivatives             |  Расчет значений  |
```

```
% |      |           |                          | вектора x состояния системы.|
% |  2   |    ds     | mdlUpdate                |  Расчет значений вектора    |
% |      |           |                          | состояний   x   дискретной  |
% |      |           |                          | системы: sys = x(n+1).      |
% |  3   |    y      | mdlOutputs               |  Расчет значений выходного  |
% |      |           |                          | вектора sys .               |
% |  4   |   tnext   | mdlGetTimeOfNextVarHit   |  Расчет значения времени для|
% |      |           |                          | следующей расчетной точки   |
% |      |           |                          | дискретной части системы.   |
% |      |           |                          | Используется при  расчете   |
% |      |           |                          | дискретной или гибридной    |
% |      |           |                          | системы с переменным шагом. |
% |  5   |           |                          | Зарезервировано для будущего|
% |      |           |                          | использования.              |
% |  9   |    []     | mdlTerminate             |  Завершение расчета         |
% |______|___________|__________________________|_____________________________|
%
%
%   Параметры блока "S-function"  p1,...,pn передаются в S-функцию при любом
% значении переменной flag.
%
%   При вызове S-функции со значением переменной  flag = 0 ею рассчитываются
%   следующие величины:
%
% sys(1) - Число непрерывных переменных состояния.
% sys(2) - Число дискретных  переменных состояния.
% sys(3) - Число выходных переменных.
% sys(4) - Число входных переменных (размерность вектора u).
%          Любой из первых четырех элементов в sys может быть задан
%          равным -1 (минус один), что означает динамически задаваемую
%          размерность соответствующих переменных. Фактическая размерность
%          при вызове S-функции с другими значениями переменной flag
%          будет равна размерности входного вектора u.
%          Например, при sys(3) = -1, размерность выходного вектора будет
%          задана равной размерности входного вектора.
% sys(5) - Значение зарезервировано для будущего использования.
% sys(6) - Значение, равное 1, соответствует прохождению входного сигнала
%          на выход. Значение, равное 0, определяет, что входной сигнал не
%          используется для вычисления выходного сигнала в функции
%          mdlOutputs, вызываемой при значении переменной flag = 3.
% sys(7) - Размерность вектора модельного времени (количество строк в
%          матрице ts).
%
%
% x0     - Задание вектора начальных значений переменных состояния.
%          Если переменных состояния нет - значение параметра задается
%          равным [] .
%
% str    - Зарезервированный параметр. Значение параметра задается
%          равным [] .
%
% ts     - Матрица, размерностью m-на-2, задающая модельное время
%          и смещение (period и offset), где m - число строк в матрице ts.
%
%     Например:
%
%     ts = [0   0,      % Шаг модельного времени для непрерывной
%                       % части системы.
%           0   1,      % Фиксированный шаг расчета для непрерывной
%                       %  части системы.
%        period offset, % Фиксированный шаг модельного времени для
```

```
%                       % дискретной части системы,
%                       % где - period > 0 & offset < PERIOD.
%          -2   0];     % Переменный шаг модельного времени для
%                         дискретной части системы. При вызове
%                         S-функции со значением переменной flag = 4
%                         выполняется расчет следующей точки по времени.
%
%        Если в матрице ts заданы несколько значений шага модельного времени,
%     то его значения должны располагаться в порядке возрастания.
%     В том случае, если задано более одного значения шага требуется проверка
%     времени срабатывания блока в явном виде:
%        abs(round((T-OFFSET)/PERIOD) - (T-OFFSET)/PERIOD)
%     Обычно задаваемая погрешность при проверке равна 1e-8. Она зависит от
%     величин шага модельного времени и времени окончания расчета.
%     Можно также задать, чтобы значение временного шага передавалось в
%     блок "S-function" из предшествующего блока модели. Для функций,
%     изменяющихся внутри основного временного шага должно быть задано
%     sys(7) = 1 и ts = [-1 0] .
%        Для функций, не изменяющихся внутри основного временного шага
%     должно быть задано sys(7) = 1 и ts = [-1 1] .
%
%   Copyright 1990-2001 The MathWorks, Inc.
%   $Revision: 1.17 $
%   Авторский перевод комментариев: Черных И.В.
%
%   Нижележащие строки показывают базовую структуру S-функции:
%
switch flag, % В зависимости от значения переменной flag происходит
             % вызов того или иного метода::

%===============%
% Инициализация %
%===============%
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;

%====================%
% Расчет производных %
%====================%
  case 1,
    sys=mdlDerivatives(t,x,u);

%============================================================%
% Расчет значений вектора состояний дискретной части системы %
%============================================================%
  case 2,
    sys=mdlUpdate(t,x,u);

%======================================================================%
% Расчет значений вектора выходных сигналов непрерывной части системы %
%======================================================================%
  case 3,
    sys=mdlOutputs(t,x,u);

%============================================================%
% Расчет значения времени для следующей расчетной точки дискретной %
% части системы                                              %
%============================================================%
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);
```

```
%====================%
% Завершение расчета %
%====================%
  case 9,
    sys=mdlTerminate(t,x,u);

%=====================================%
% Неизвестное значение переменной flag %
%=====================================%
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);

end
% Окончание функции sfuntmpl
%=====================================================================%
% mdlInitializeSizes                                                  %
% Функция инициализации                                              %
% Расчет начальных условий, значений вектора модельного времени,      %
% размерности матриц                                                  %
%=====================================================================%
function [sys,x0,str,ts]=mdlInitializeSizes
% Приведенные ниже значения параметров даны в качестве примера и не отражают
% реально задаваемых значений.

sizes = simsizes; % Первый вызов функции simsizes создает структуру sizes
sizes.NumContStates  = 0; % Число непрерывных переменных состояния
sizes.NumDiscStates  = 0; % Число дискретных  переменных состояния
sizes.NumOutputs     = 0; % Число выходных переменных (размерность выходного
                          %  вектора)
sizes.NumInputs      = 0; % Число входных переменных (размерность вектора u)
sizes.DirFeedthrough = 1; % Параметр, задающий проход входного сигнала на
                          % выход.
                          % Этот параметр нужно задавать равным 1, в том
                          % случае, если входной сигнал прямо или
                          % опосредованно(например, через логическое
                          % выражение или алгебраическую операцию)
                          % проходит на выход системы (иными словами: если
                          % входной сигнал u, входит в выражения задаваемые в
                          % функции mdlOutputs) или используется метод
                          % mdlGetTimeOfNextVarHit.
                          % При описании системы в уравнениях пространства
                          % состояний этот параметр следует задать равным 1,
                          % если матрица D (матрица обхода) не пустая и
                          % равным нулю, в противном случае.
sizes.NumSampleTimes = 1; % Размерность вектора модельного времени.
                          % Минимальное значение параметра = 1
                          % (одна строка в матрице ts).

sys = simsizes(sizes);    % Второй вызов функции simsizes. Данные о
                          % размерностях передаются в Simulink.

x0  = [];     % Задание вектора начальных значений переменных состояния
              % (начальных условий).

str = [];     % Задание параметра str, как пустой матрицы. Параметр
              % заразервирован для будущего использования.

ts  = [0 0]; % Матрица из двух колонок, задающая шаг модельного времени
              % и смещение.
% Окончание mdlInitializeSizes
```

```
%=============================================================================%
% mdlDerivatives                                                              %
% Функция для расчета значений производных вектора состояния непрерывной %
% части системы                                                               %
%=============================================================================%
function sys=mdlDerivatives(t,x,u)
sys = [];
% Окончание mdlDerivatives

%=============================================================================% %
mdlUpdate %
% Функция для расчета значений вектора выходных сигналов непрерывной части % % системы %
%=============================================================================%
function sys=mdlUpdate(t,x,u)
sys = [];
% Окончание mdlUpdate

%=============================================================================%
% mdlOutputs                                                                  %
% Функция для расчета значений вектора выходных сигналов непрерывной части  %
% системы                                                                     %
%=============================================================================%
function sys=mdlOutputs(t,x,u)
sys = [];
% Окончание mdlOutputs

%=============================================================================%
% mdlGetTimeOfNextVarHit                                                      %
% Расчет значения времени для следующей расчетной точки дискретной части     %
% системы.                                                                    %
% Функция рассчитывает время (абсолютное значение), по достижении которого   %
% значения дискретной части системы передаютсяв Simulink-модель.             %
% Функция используется только в случае моделирования дискретной части        %
% системы с переменным шагом (variable discrete-time sample time). В этом    %
% случае параметр ts функции mdlInitializeSizes должен быть задан как [-2 0]%
%=============================================================================%
function sys=mdlGetTimeOfNextVarHit(t,x,u)
sampleTime = 1;  % Пример: время срабатывания блока увеличивается
                 % на 1 секунду.
sys = t + sampleTime;
% Окончание mdlGetTimeOfNextVarHit

%=======================================%
% mdlTerminate                          %
% Функция, выполняющая завершение расчета %
%=======================================%
function sys=mdlTerminate(t,x,u)

sys = [];% Окончание mdlTerminate
```

16.7. Примеры S-функций языке MATLAB

16.7.1. Простейшая S-функция

Одним из самых простых примеров **S-функций** поставляемых с пакетом **MATLAB** является функция **timestwo** (файл **timestwo.m**). Данная **S-функция** выполняет умножение входного сигнала на коэффициент **2**. Ниже приведен текст этой **S-функции**.

```
function [sys,x0,str,ts] = timestwo(t,x,u,flag)
%
%   TIMESTWO - Пример S-функции. Выходной сигнал равен входному,
%   умноженному на 2:
%      y = 2 * u;
%
%   Шаблон для создания S-функции - файл sfuntmpl.m .
%
%   Copyright 1990-2001 The MathWorks, Inc.
%   $Revision: 1.6 $
%   Авторский перевод комментариев: Черных И.В.
%


switch flag, % В зависимости от значения переменной flag происходит
             % вызов того или иного метода:

%===============%
% Инициализация %
%===============%
  case 0
    [sys,x0,str,ts]=mdlInitializeSizes;

%==============================================%
% Расчет значений вектора выходных сигналов %
%==============================================%

  case 3
    sys=mdlOutputs(t,x,u);

%========================================%
% Неиспользуемые значения переменной flag %
%========================================%

  % В примере не используются методы для завершения работы S-функции,
  % нет непрерывных и дискретных переменных состояния,
  % поэтому значения переменной flag = 1, 2, 4, 9 не используются.
  % Результатом S-функции в этом случае является пустая матрица.
  case { 1, 2, 4, 9 }
    sys=[];

%====================================%
% Неизвестное значение переменной flag %
%====================================%
otherwise
    error(['Unhandled flag = ',num2str(flag)]);

end

% Окончание функции timestwo


%=============================================================%
% mdlInitializeSizes                                          %
% Функция инициализации                                       %
% Расчет начальных условий, значений вектора шагов модельного %
```

```
% времени, размерности матриц                                    %
%=================================================================%
%
function [sys,x0,str,ts] = mdlInitializeSizes()

sizes = simsizes;
sizes.NumContStates  = 0;   % Число непрерывных переменных состояния.
sizes.NumDiscStates  = 0;   % Число дискретных  переменных состояния.
sizes.NumOutputs     = -1;  % Число выходных переменных (размерность выходного вектора).
                            % Динамическая размерность выходного вектора.
sizes.NumInputs      = -1;  % Число входных переменных (размерность входного
                            % вектора).
                            % Динамическая размерность входного вектора.
sizes.DirFeedthrough = 1;   % Прямой проход. Есть проход входного сигнала
                            % на выход.
sizes.NumSampleTimes = 1;   % Размерность вектора шагов модельного времени.

sys = simsizes(sizes);
str = [];                   % Параметр заразервирован для будущего
                            % использования.
x0  = [];                   % Задание вектора начальных значений переменных
                            % состояния.
                            % Переменных сомтояния нет, поэтому значение
                            % параметра - пустая матрица.
ts  = [-1 0];               % Матрица из двух колонок, задающая шаг
                            % модельного времени и смещение.
                            % Шаг наследуется из предшествующего блока.

% Окончание mdlInitializeSizes

%
%=================================================================%
% mdlOutputs                                                      %
% Функция для расчета значений вектора выходных сигналов          %
%=================================================================%
%
function sys = mdlOutputs(t,x,u)

sys = u * 2; % Выходной сигнал блока есть входной сигнал, умноженный на
             % коэффициент 2.

% Окончание mdlOutputs
```

Пример модели с **S-функцией timestwo** приведен на рис.16.4.

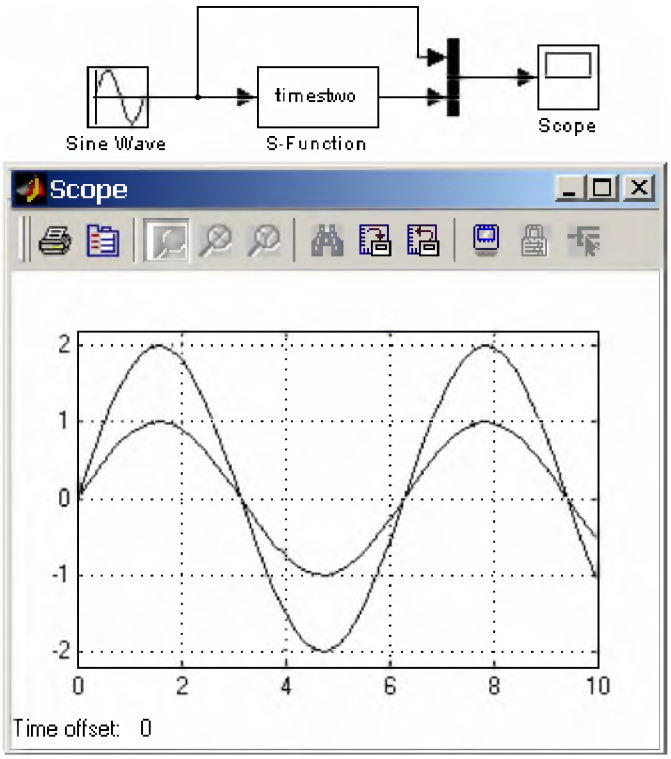


Рис. 16.4 Модель с **S-функцией timestwo**

16.7.2. Модель непрерывной системы

Модель непрерывной системы, описываемой уравнениями пространства состояния дана в файле **csfunc.m** . Данная **S-функция** моделирует непрерывную систему с двумя входами, двумя выходами и двумя переменными состояния. Параметры модели (значения матриц **A,B,C,D**) задаются в теле **S-функции** и передаются в **callback**-методы через их заголовки в качестве дополнительных параметров. Ниже приведен текст этой **S-функции**.

S-функция csfunc:

```
function [sys,x0,str,ts] = csfunc(t,x,u,flag)
%  CSFUNC Пример S-функции. Спомощью уравнений пространства состояния
%  моделируется непрерывная система:
%
%      x' = Ax + Bu
%      y  = Cx + Du
%
%  Значения матриц передаются в callback-методы через их заголовки
%  в качестве дополнительных параметров
%
%
%   Шаблон для создания S-функции - файл sfuntmpl.m .
%
%   Copyright 1990-2001 The MathWorks, Inc.
%   $Revision: 1.8 $
%   Авторский перевод комментариев: Черных И.В.

%   Задание матриц:
```

```
A=[-0.09   -0.01
    1         0];      % Матрица системы.

B=[ 1   -7
    0   -2];           % Матрица входа.

C=[ 0    2
    1   -5];           % Матрица выхода.

D=[-3    0
    1    0];           % Матрица обхода.

switch flag,  % В зависимости от значения переменной flag происходит
              % вызов того или иного метода:

%===============%
% Инициализация %
%===============%
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes(A,B,C,D);

%====================%
% Расчет производных %
%====================%
  case 1,
    sys=mdlDerivatives(t,x,u,A,B,C,D);

%=========================================%
% Расчет значений вектора выходных сигналов %
%=========================================%
  case 3,
    sys=mdlOutputs(t,x,u,A,B,C,D);

%=========================================%
% Неиспользуемые значения переменной flag %
%=========================================%

  % В примере не используются методы для завершения работы S-функции,
  % нет дискретных переменных состояния,
  % поэтому значения переменной flag = 2, 4, 9 не используются.
  % Результатом S-функции в этом случае является пустая матрица.

  case { 2, 4, 9 }
    sys=[];


%====================================%
% Неизвестное значение переменной flag %
%====================================%
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);

end
% Окончание csfunc

%
%=============================================================%
% mdlInitializeSizes                                          %
% Функция инициализации                                       %
% Расчет начальных условий, значений вектора модельного времени,%
% размерности матриц                                          %
```

```
%=============================================================%
%
function [sys,x0,str,ts]=mdlInitializeSizes(A,B,C,D)

sizes = simsizes;
sizes.NumContStates  = 2; % Число непрерывных переменных состояния.
sizes.NumDiscStates  = 0; % Число дискретных  переменных состояния.
sizes.NumOutputs     = 2; % Число выходных переменных (размерность выходного
                          % вектора).
sizes.NumInputs      = 2; % Число входных переменных (размерность входного
                          % вектора).
sizes.DirFeedthrough = 1; % Прямой проход. Есть проход входного сигнала
                          % на выход.
                          % (матрица D не пустая).
sizes.NumSampleTimes = 1; % Размерность вектора шагов модельного времени.

sys = simsizes(sizes);
x0  = zeros(2,1);         % Задание вектора начальных значений переменных
                          % состояния. Начальные условия нулевые.
str = [];                 % Параметр заразервирован для будущего
                          % использования.
ts  = [0 0];              % Матрица из двух колонок, задающая шаг модельного
                          % времени и смещение.

% Окончание mdlInitializeSizes
%
%=============================================================================%
% mdlDerivatives                                                              %
% Функция для расчета значений производных вектора состояния непрерывной     %
% части системы                                                               %
%=============================================================================%
%
function sys=mdlDerivatives(t,x,u,A,B,C,D)

sys = A*x + B*u;

% Окончание mdlDerivatives
%
%=============================================================%
% mdlOutputs                                                  %
% Функция для расчета значений вектора выходных сигналов      %
%=============================================================%
function sys=mdlOutputs(t,x,u,A,B,C,D)

sys = C*x + D*u;

% Окончание mdlOutputs
```

Пример модели с **S-функцией csfunc** приведен на рис.16.5.

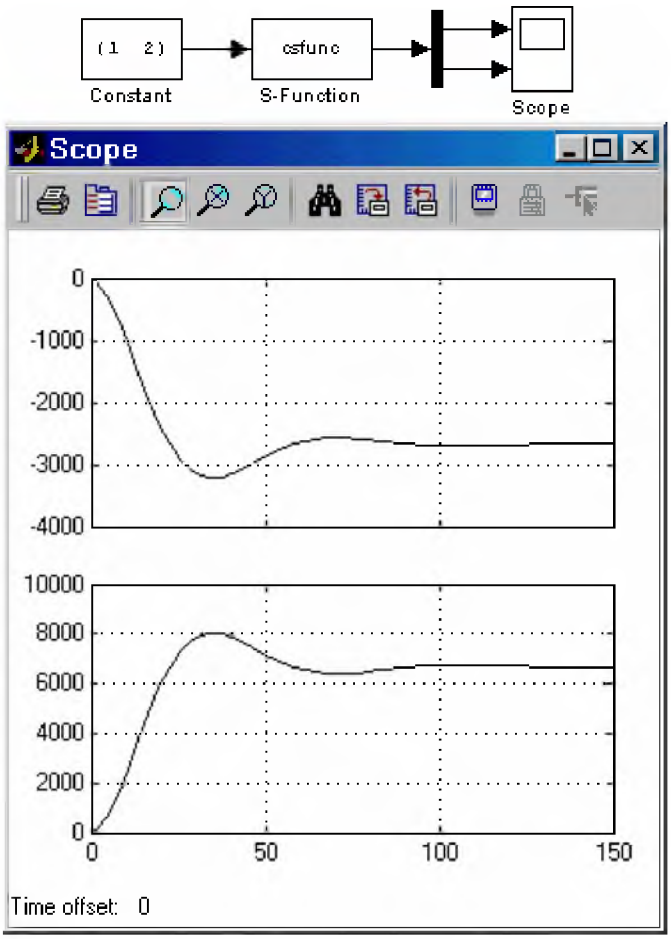


Рис. 16.5 Модель с **S-функцией csfunc**

16.7.3. Модель дискретнойной системы

Модель дискретной системы, описываемой уравнениями пространства состояния, дана в файле **dsfunc.m** . Данная **S-функция** моделирует дискретную систему с двумя входами, двумя выходами и двумя переменными состояния. Параметры модели (значения матриц **A, B, C, D**) задаются в теле **S-функции** и передаются в **callback-методы** через их заголовки в качестве дополнительных параметров. Ниже приведен текст этой **S-функции**.

S-функция dsfunc:

```
function [sys,x0,str,ts] = dsfunc(t,x,u,flag)
%  DSFUNC Пример S-функции. С помощью уравнений пространства состояния
%  моделируется дискретная система:
%      x(n+1) = Ax(n) + Bu(n)
%      y(n)   = Cx(n) + Du(n)
%
%  Значения матриц передаются в callback-методы через их заголовки
%  в качестве дополнительных параметров
%
%   Шаблон для создания S-функции - файл sfuntmpl.m .
%
%   Copyright 1990-2001 The MathWorks, Inc.
%   $Revision: 1.8 $
```

```
%   Авторский перевод комментариев: Черных И.В.

%   Задание матриц:
A = [0.9135 0.1594
    -0.7971 0.5947];   % Матрица системы.

B = [0.05189 0
     0.4782 0];        % Матрица входа.

C = [0 1
     1 0];             % Матрица выхода.

D = [0.01  0
     0    -0.02];      % Матрица обхода.

switch flag,     % В зависимости от значения переменной flag происходит
                 % вызов того или иного метода:

%===============%
% Инициализация %
%===============%
  case 0,
    [sys,x0,str,ts] = mdlInitializeSizes(A,B,C,D);

%=============================================================%
% Расчет значений вектора состояний дискретной части системы %
%=============================================================%
  case 2,
    sys = mdlUpdate(t,x,u,A,B,C,D);

%=======================================================================%
% Расчет значений вектора выходных сигналов непрерывной части системы %
%=======================================================================%
  case 3,
    sys = mdlOutputs(t,x,u,A,C,D);


%=========================================%
% Неиспользуемые значения переменной flag %
%=========================================%

  % В примере не используются методы для завершения работы S-функции,
  % нет непрерывных переменных состояния,
  % поэтому значения переменной flag = 1, 4, 9 не используются.
  % Результатом S-функции в этом случае является пустая матрица.

  case { 1, 4, 9 }
    sys=[];


%====================================%
% Неизвестное значение переменной flag %
%====================================%
  otherwise
    error(['unhandled flag = ',num2str(flag)]);
end

% Окончание dsfunc

%
%=============================================================%
```

```
% mdlInitializeSizes                                                  %
% Функция инициализации                                              %
% Расчет начальных условий, значений вектора шагов модельного       %
% времени, размерности матриц                                        %
%====================================================================%
%
function [sys,x0,str,ts] = mdlInitializeSizes(A,B,C,D)

sizes = simsizes;
sizes.NumContStates  = 0;          % Число непрерывных переменных состояния.
sizes.NumDiscStates  = size(A,1); % Число дискретных  переменных состояния.
sizes.NumOutputs     = size(D,1); % Число выходных переменных (размерность
                                  % выходного вектора).
sizes.NumInputs      = size(D,2); % Число входных переменных (размерность
                                  % входного вектора).
sizes.DirFeedthrough = 1;         % Прямой проход. Есть проход входного
                                  % сигнала на выход (матрица D не пустая).
sizes.NumSampleTimes = 1;         % Размерность вектора  шагов модельного
                                  % времени.

sys = simsizes(sizes);

x0  = zeros(sizes.NumDiscStates,1); % Задание вектора начальных значений
                                    % переменных состояния.
                                    % Начальные условия нулевые
str = [];                           % Параметр заразервирован для будущего
                                    % использования.
ts  = [0.2 0];                      % Матрица из двух колонок, задающая шаг
                                    % модельного времени и смещение.
% Окончание mdlInitializeSizes

%

%=========================================================================%
% mdlUpdate                                                               %
% Функция для расчета значений вектора состояния дискретной части системы %
%=========================================================================%
%
function sys = mdlUpdate(t,x,u,A,B,C,D)
sys = A*x+B*u;

% Окончание mdlUpdate

%=========================================================%
% mdlOutputs                                              %
% Функция для расчета значений вектора выходных сигналов %
%=========================================================%
%
function sys = mdlOutputs(t,x,u,A,C,D)
sys = C*x+D*u;

% Окончание mdlOutputs
```

Пример модели с **S-функцией dsfunc** приведен на рис.16.6.

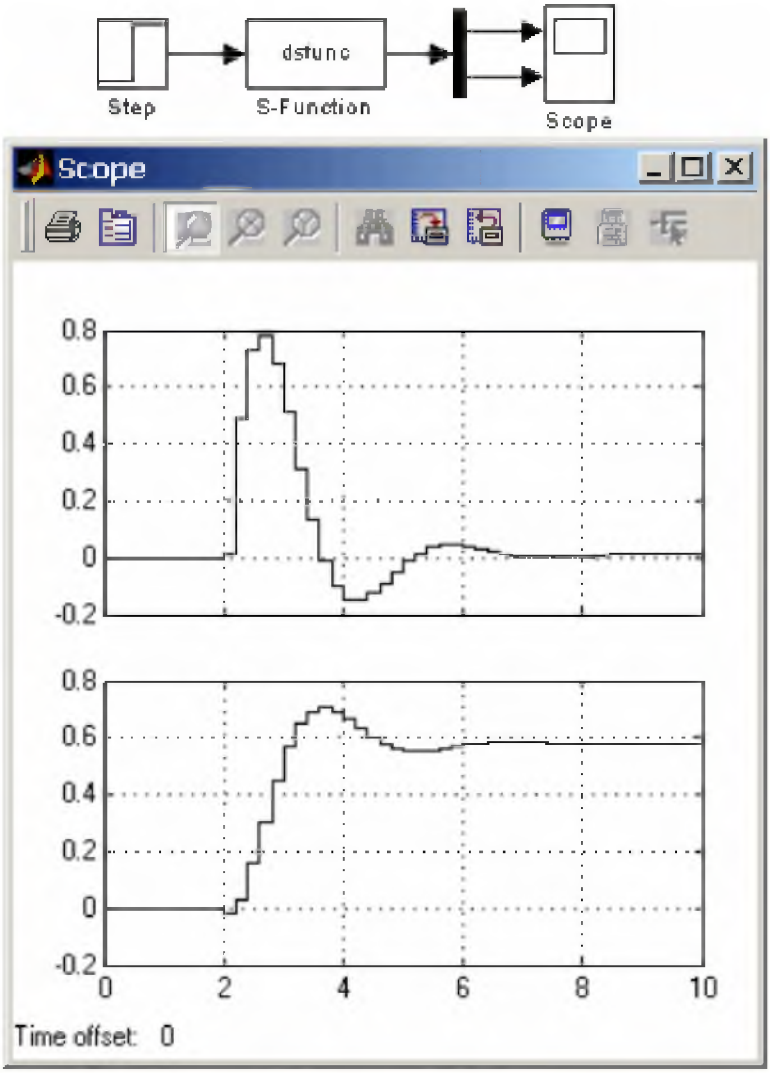


Рис. 16.6 Модель с **S-функцией dsfunc**

16.7.4. Модель гибридной системы

Модель гибридной системы (комбинации непрерывной и дискретной системы) дана в файле **mixedm.m** . Рассматриваемая **S-функция** моделирует систему, состоящую из последовательно включенных интегратора (**1/s**) и блока задержки (**1/z**). Особенность **S-функции** для гибридной системы в том, что вызов **callback-методов** для расчета дискретной части системы выполняется в те же моменты времени, что и для непрерывной ее части, поэтому пользователь, при написании **S-функции**, должен предусмотреть проверку правильности вызова **callback-методов** рассчитывающих дискретную часть системы.
Ниже приведен текст этой **S-функции**.

S-функция dsfunc:

```
function [sys,x0,str,ts] = mixedm(t,x,u,flag)
% MIXEDM Пример S-функции. S-функция моделирует систему, состоящую из
% последовательно включенных интегратора (1/s) и блока задержки (1/z).
%
%   Шаблон для создания S-функции - файл sfuntmpl.m .
```

```
%
%   Copyright 1990-2001 The MathWorks, Inc.
%   $Revision: 1.27 $
%   Авторский перевод комментариев: Черных И.В.

% Шаг модельного времени и смещение для дискретной части системы:
 dperiod = 1;
 doffset = 0;

switch flag      % В зависимости от значения переменной flag происходит
                 % вызов того или иного метода:


%===============%
% Инициализация %
%===============%
  case 0
    [sys,x0,str,ts]=mdlInitializeSizes(dperiod,doffset);

%====================%
% Расчет производных %
%====================%
  case 1
    sys=mdlDerivatives(t,x,u);

%=============================================================%
% Расчет значений вектора состояний дискретной части системы %
%=============================================================%
  case 2,
    sys=mdlUpdate(t,x,u,dperiod,doffset);

%==========================================%
% Расчет значений вектора выходных сигналов %
%==========================================%
  case 3
    sys=mdlOutputs(t,x,u,doffset,dperiod);

%====================%
% Завершение расчета %
%====================%
  case 9
    sys = [];

%====================================%
% Неизвестное значение переменной flag %
%====================================%
  otherwise
    error(['unhandled flag = ',num2str(flag)]);

end

% Окончание mixedm

%=============================================================%
% mdlInitializeSizes                                          %
% Функция инициализации                                       %
% Расчет начальных условий, значений вектора модельного времени,%
% размерности матриц                                          %
%=============================================================%
function [sys,x0,str,ts]=mdlInitializeSizes(dperiod,doffset)
```

```
sizes = simsizes;
sizes.NumContStates  = 1;   % Число непрерывных переменных состояния.
sizes.NumDiscStates  = 1;   % Число дискретных  переменных состояния.
sizes.NumOutputs     = 1;   % Число выходных переменных (размерность выходного
                            % вектора).
sizes.NumInputs      = 1;   % Число входных переменных (размерность входного
                            % вектора).
sizes.DirFeedthrough = 0;   % Прямой проход. Прохода входного сигнала на выход
                            % нет.
sizes.NumSampleTimes = 2;   % Размерность вектора  шагов модельного времени.
                            % Шаги модельного времени задаются для непрерывной
                            % и для дискретной частей системы.
sys = simsizes(sizes);
x0  = zeros(2,1);           % Задание вектора начальных значений переменных
                            % состояния.
                            % Начальные условия нулевые
str = [];                   % Параметр заразервирован для будущего
                            % использования.
ts  = [0 0];                % Шаг модельного времени для непрерывной части
                            % системы.
       dperiod doffset];    % Шаг модельного времени для дискретной части
                            % системы.

% Окончание mdlInitializeSizes
%
%=============================================================================%
% mdlDerivatives                                                              %
% Функция для расчета значений производных вектора состояния непрерывной     %
% части системы                                                               %
%=============================================================================%
function sys=mdlDerivatives(t,x,u)

sys = u;

% Окончание mdlDerivatives


%=============================================================================%
% mdlUpdate                                                                   %
% Функция для расчета значений вектора состояния дискретной части системы     %
%=============================================================================%
%
function sys=mdlUpdate(t,x,u,dperiod,doffset)

% Расчет значения переменной состояния дискретной части системы
% выполняется в соответствии с дискретным шагом расчета
% (погрешность по времени выбрана равной 1e-15).

if abs(round((t - doffset)/dperiod) - (t - doffset)/dperiod) < 1e-15
  sys = x(1);
else
  sys = [];
end

% Окончание mdlUpdate

%
%=============================================================%
% mdlOutputs                                                  %
% Функция для расчета значений вектора выходных сигналов      %
%=============================================================%
```

```
%
function sys=mdlOutputs(t,x,u,doffset,dperiod)


% Расчет значения выходного сигнала системы
% выполняется в соответствии с дискретным шагом расчета
% (погрешность по времени выбрана равной 1e-15).
% Если условие, заданное оператором if истинно, то выходной сигнал блока
% изменяется. В противном случае выходной сигнал остается равным значению
% на предыдущем шаге.
%
if abs(round((t - doffset)/dperiod) - (t - doffset)/dperiod) < 1e-15
  sys = x(2);
else
  sys = [];
end

% Окончание mdlOutputs
```

Пример модели с **S-функцией mixedm** приведен на рис.16.7.

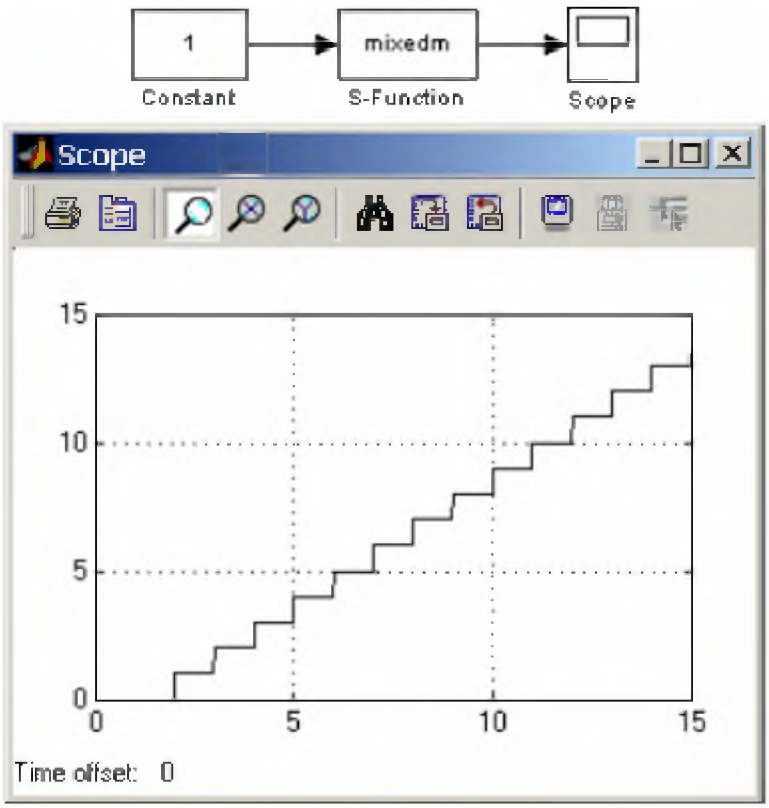


Рис. 16.7 Модель с **S-функцией mixedm**

16.7.5. Модель дискретной системы с переменным шагом расчета

Модель дискретной системы с переменным шагом расчета дана в файле **vsfunc.m**. Рассматриваемая **S-функция** моделирует устройство задержки сигнала. Время задержки определяется величиной входного сигнала. Для вычисления момента времени срабатывания блока используется **callback-метод mdlGetTimeOfNextVarHit**, вызываемый при значении переменой **flag = 4**. Поскольку время срабатывания блока зависит от входного сигнала, то параметр **DirFeedthrough** должен иметь значение

равное **1**. В целом, любой блок, в котором время срабатывания вычисляется по величине входного сигнала, должен иметь **DirFeedthrough = 1**.

Ниже приведен текст этой **S-функции**.

S-функция dsfunc:

```
function [sys,x0,str,ts] = vsfunc(t,x,u,flag)
% VSFUNC Пример S-функции. S-функция моделирует систему, с переменным шагом
% расчета (устройство задержки сигнала).
% Время задержки определяется величиной сигнала. поступающего на второй вход:
%
%     dt      = u(2)
%     y(t+dt) = u(t)
%
%   Смотри также SFUNTMPL, CSFUNC, DSFUNC.

%   Copyright 1990-2001 The MathWorks, Inc.
%   $Revision: 1.9 $
%   Авторский перевод комментариев: Черных И.В.
%
%
switch flag,     % В зависимости от значения переменной flag происходит
                 % вызов того или иного метода:


%===============%
% Инициализация %
%===============%
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;

%=============================================================%
% Расчет значений вектора состояний дискретной части системы %
%=============================================================%
  case 2,
    sys=mdlUpdate(t,x,u);

%=========================================%
% Расчет значений вектора выходных сигналов %
%=========================================%
  case 3,
    sys=mdlOutputs(t,x,u);

%=================================================================%
% Расчет значения времени для следующей расчетной точки дискретной %
% части системы                                                   %
%=================================================================%
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);

%====================%
% Завершение расчета %
%====================%
  case 9,
    sys=mdlTerminate(t,x,u);

%=====================================%
% Неиспользуемые значения переменной flag %
%=====================================%
```

```
  % В примере нет непрерывных переменных состояния,
  % поэтому значение переменной flag = 1 не используется.
  % Результатом S-функции в этом случае является пустая матрица.
  case 1,
    sys = [];

%====================================%
% Неизвестное значение переменной flag %
%====================================%
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);

end

% Окончание vsfunc

%
%=============================================================%
% mdlInitializeSizes                                          %
% Функция инициализации                                       %
% Расчет начальных условий, значений вектора шагов модельного %
% времени, размерности матриц                                 %
%=============================================================%
%
function [sys,x0,str,ts]=mdlInitializeSizes
%
sizes = simsizes;

sizes.NumContStates  = 0;  % Число непрерывных переменных состояния.
sizes.NumDiscStates  = 1;  % Число дискретных  переменных состояния.
sizes.NumOutputs     = 1;  % Число выходных переменных (размерность выходного
%                          вектора).
sizes.NumInputs      = 2;  % Число входных переменных (размерность входного
                           % вектора).
sizes.DirFeedthrough = 1;  % Прямой проход. Входной сигал используется
                           % в методе mdlGetTimeOfNextVarHit .
sizes.NumSampleTimes = 1;  % Размерность вектора  шагов модельного времени.

sys = simsizes(sizes);

x0  = [0];                 % Задание вектора начальных значений переменных
                           % состояния.
                           % Начальные условия нулевые.
str = [];                  % Параметр заразервирован для будущего
                           % использования.

ts  = [-2 0];              % Переменный шаг модельного времени для дискретной
                           % части системы. При вызове S-функции со значением
                           % переменной flag = 4 выполняется расчет следующей
                           % точки по времени.

% Окончание mdlInitializeSizes

%
%=============================================================================%
% mdlUpdate                                                                   %
% Функция для расчета значений вектора состояния дискретной части системы %
%=============================================================================%
%
function sys=mdlUpdate(t,x,u)
```

```
sys = u(1);

% Окончание mdlUpdate

%
%=============================================================%
% mdlOutputs                                                  %
% Функция для расчета значений вектора выходных сигналов      %
%=============================================================%
%
function sys=mdlOutputs(t,x,u)

sys = x(1);

% Окончание mdlOutputs

%
%=============================================================================%
% mdlGetTimeOfNextVarHit                                                      %
% Расчет значения времени для следующей расчетной точки дискретной части     %
% системы.                                                                    %
% Функция рассчитывает время (абсолютное значение), по достижении которого   %
% значения дискретной части системы передаютсяв Simulink-модель.             %
% Функция используется только в случае моделирования дискретной части        %
% системы с переменным шагом (variable discrete-time sample time). В этом    %
% случае параметр ts функции mdlInitializeSizes должен быть задан как [-2 0]%
%=============================================================================%
%
function sys=mdlGetTimeOfNextVarHit(t,x,u)

sys = t + u(2);

% Окончание mdlGetTimeOfNextVarHit

%
%=========================================%
% mdlTerminate                            %
% Функция, выполняющая завершение расчета %
%=========================================%
%
function sys=mdlTerminate(t,x,u)

sys = [];

% Окончание mdlTerminate
```

Пример модели с **S-функцией** vsfunc приведен на рис.16.8. В примере время задержки сигнала увеличивается с 0.0005 с до 0.0015 с.

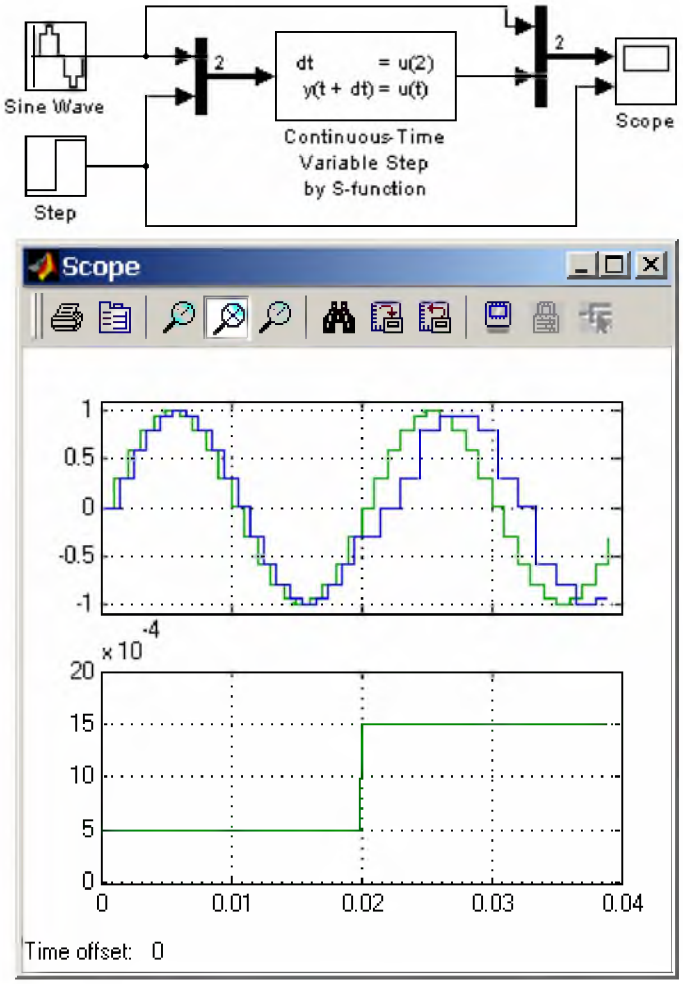


Рис. 16.8 Модель с **S-функцией vsfunc**