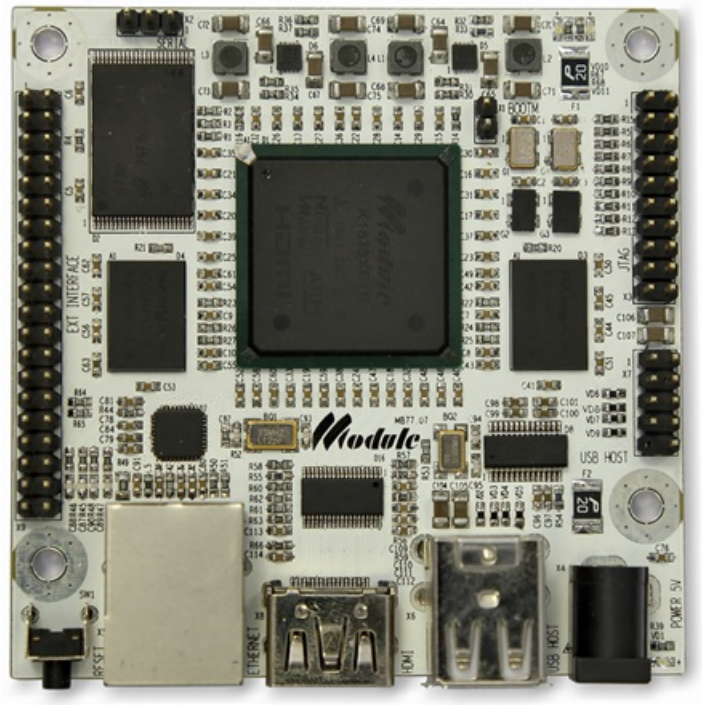
Пакет целевой поддержки NeuroMatrix для Embedded Coder

Документация пользователя



Оглавление

Пакет целевой поддержки NeuroMatrix

Пакет расширения для среды разработки компании НТЦ Модуль, обеспечивающий поддержку модельно-ориентированного проектирования на основе платформы MATLAB/Simulink.

Данное расширение (в дальнейшем "пакет поддержки" или "система") обеспечивает поддержку модельно-ориентированного проектирования на основе платформы MATLAB/Simulink при использовании целевых вычислителей (цифровых сигнальных процессоров) компании НТЦ Модуль. Полное наименование системы: **"Пакет поддержки целевой платформы MB77.07 на базе СБИС К1879ХБ1Я"**.

Содержание

Назначение и цели создания системы

Назначение системы

Автоматизация процесса быстрого прототипирования встраиваемого ПО с использованием отладочной платы с микропроцессором MB77.07 на базе СБИС К1879ХБ1Я (в дальнейшем "целевой вычислитель") компании НТЦ Модуль и с применением инструментов модельно-ориентированного проектирования MATLAB/Simulink и Embedded Coder компании MathWorks.

Цели создания системы

- Смещение мероприятий написания кода и верификации на более ранние этапы процесса разработки (стадии проектирования и задания спецификации) на стороне клиентов НТЦ Модуль.
- Смещение фокуса с написания низкоуровневого кода С на разработку алгоритмов в среде модельно-ориентированного проектирования и графического программирования целевого вычислителя.
- Автоматизация процедуры генерации и верификации кода на целевом вычислителе.
- Обеспечение возможностей автоматизированного проведения специализированных процедур верификации алгоритмов на целевом вычислителе, в частности для удовлетворения требований авиационных и военных стандартов для ПО повышенной надежности (DO-178/КТ-178, ГОСТ Р 51904)

Возможности

- Интеграция с набором инструментов для построения кода
- Загрузка и запуск исполняемого файла на целевой системе
- Независимое выполнение на целевой системе (интеграция с планировщиком ОС Linux)
- Верификация в режиме Процессор-в-контуре (PIL)
- Профилирование времени выполнения в режиме PIL
- Генерация оптимизированного для целевой системы кода
- Специализированные блоки Simulink
- Установщик пакета поддержки целевой системы
- Документация для пакета поддержки целевой системы

Установка

Начало работы

Примеры

Независимое выполнение модели

Режим симуляции "Процессор-в-контуре"

Технология Code Replacement

Блоки Simulink

Установка пакета поддержки

В данном разделе описывается процесс установки пакета поддержки.

Содержание

Предварительные требования

Перед установкой пакета поддержки убедитесь, что у вас установлен поддерживаемый хост-компилятор, выполнив в командном окне MATLAB команду:

```
mex -setup
```

Должно появиться сообщение: MEX configured to use **'Название компилятора'** for C language compilation. Для получения дополнительной информации о поддерживаемых компиляторах обратитесь по ссылке: http://www.mathworks.com/support/compilers/current_release/

Требования к инструментам

Пакет целевой поддержки процессора NeuroMatrix разрабатывается и тестируется с использованием следующих версий инструментов. Работоспособность пакета при использовании других версий любого инструмента не может гарантироваться.

- Операционная система: Microsoft Windows 7, 64-битная версия.
- Версия MATLAB: R2015b.
- Необходимые тулбоксы: MATLAB, Simulink, Embedded Coder, Fixed-Point Designer, DSP System Toolbox
- Версия инструментальных средств сборки кода для процессора ARM: Поставляется с установщиком.
- Версия инструментальных средств сборки кода для процессора NMC: Поставляется с установщиком.
- Версия библиотеки NMPP для работы с ядром NeuroMatrix: Поставляется с установщиком.
- Версия библиотеки AURA для работы с ядром NeuroMatrix: Поставляется с установщиком.
- Версия прошивки Linux процессора ARM: Поставляется с установщиком.
- Целевой вычислитель: микрокомпьютер МВ77.07 на базе СБИС К1879ХБ1Я (http://www.module.ru/catalog/micro/micro_pc/).

Установка пакета поддержки

Система обеспечивает автоматическую установку или обновление пакета поддержки путем запуска пользователем скрипта MATLAB, выполняющего мероприятия установки. Этот скрипт называется **installNeuroMatrix** и находится в корне пакета поддержки.

Для запуска установки:

- Запустите MATLAB
- В MATLAB перейдите в директорию, содержащую скрипт **installNeuroMatrix**
- Запустите скрипт **installNeuroMatrix**, щелкнув по нему правой кнопкой мышки и выбрав "Run".

После запуска скрипт выполняет следующие действия:

- Автоматически скачивает с сайта НТЦ Модуль и устанавливает (распаковывает) инструментальные средства сборки кода для ARM и NMC, библиотеки NMPP и AURA для работы с NMC, а также актуальную версию прошивки

для NeuroMatrix.

- У пользователя запрашивается путь к директории, куда следует установить инструменты поддержки NeuroMatrix
- У пользователя запрашивается информация о целевом вычислителе (IP адрес для режима PIL)
- Требуемые директории добавляются в путь поиска MATLAB и сохраняются для следующей сессии MATLAB
- Сохраняются все настройки между сессиями MATLAB
- Пакет поддержки регистрируется в MATLAB
- Компилируются S-функции для дополнительных блоков Simulink
- Пользователю выводятся ссылки на документацию и демонстрационные примеры

Ссылки на скачивание инструментальных средств и библиотеки поддержки, используемые установщиком, предоставляются НТЦ Модуль. Если установщику не удалось скачать требуемые файлы, попробуйте снова или обратитесь в группу поддержки. Скрипт по установке осуществляет базовую валидацию введенных пользователем данных, чтобы снизить вероятность ошибок пользователя при установке.

Обновление прошивки NeuroMatrix

Прежде, чем иметь возможность пользоваться функционалом пакета целевой поддержки, нужно обновить прошивку NeuroMatrix. Прошивка представляет собой образ дистрибутива Linux, предоставляемый НТЦ Модуль.

Указанная прошивка автоматически скачивается установщиком пакета поддержки с сайта НТЦ Модуль. Однако, загрузка прошивки на плату осуществляется пользователем самостоятельно. Это связано с тем, что этот процесс требует ручного вмешательства пользователя (в частности, перестановки перемычек на плате), а также занимает продолжительное время. Это однократный процесс, через который необходимо пройти один раз после установки пакета поддержки.

Установка актуальной версии прошивки

- Перейти в директорию, содержащую скачанную прошивку. Путь к директории можно узнать, выполнив команду:

```
getpref('neuromatrix','NMFWRoot')
```

- Директория содержит упакованный архив, который называется, например, fw-debian-jessie-4.x-ubifs-13112015.tgz. Следует распаковать этот архив - он содержит образ (прошивку) и инструменты для записи прошивки в микроконтроллер.

- Следует обратиться к файлу *README.windows.txt* или *README.linux.txt* , который находится в директории с прошивкой, для получения инструкций по установке.

- После установки прошивки и загрузки платы требуется установить все deb-пакеты, которые находятся в директории с архивом прошивки (называются *nmc-utils-*.deb*). Чтобы установить пакеты, следует скопировать их на микрокомпьютер (например, при помощи сторонней утилиты WinSCP), а затем через ssh (например, при помощи сторонней утилиты putty) выполнить команду "dpkg -i nmc-utils-*.deb" на микрокомпьютере.

Документация пользователя

Документация пользователя для пакета целевой поддержки NeuroMatrix доступна непосредственно из MATLAB.

Чтобы открыть документацию, следует нажать кнопку Help в панели инструментов MATLAB (или нажать F1), а затем выбрать раздел Supplemental Software -> Пакет целевой поддержки NeuroMatrix.

Начало работы с NeuroMatrix

Начало работы с пакетом поддержки процессора NeuroMatrix в Embedded Coder.

Пакет поддержки процессора NeuroMatrix в Embedded Coder обеспечивает автоматизацию процесса быстрого прототипирования алгоритмов (цифровой обработки сигналов, управления и общего назначения) путем **автоматической генерации** кода C из графических моделей алгоритмов, **сборки кода C** и **загрузки** его на процессор ARM целевого вычислителя. **Оптимизированные функции**, использующие библиотечные функции векторного сопроцессора NeuroMatrix, вызываются из кода C и выполняются на векторном сопроцессоре. Поддержка вызова векторных функций NeuroMatrix со стороны процессора ARM (т.е. есть кода C) обеспечивается библиотекой векторных функций, разработанных компанией НТЦ Модуль. Интеграция со средой Simulink позволяет быстро тестировать разработанные алгоритмы на целевом вычислителе, при этом обеспечивая **двусторонний обмен данным**и между моделью Simulink и исполняемым объектным кодом для формирования тестовых воздействий и анализа результатов.

Содержание

Возможности пакета поддержки процессора NeuroMatrix в Embedded Coder

Автоматическая генерация кода

Пакет поддержки процессора NeuroMatrix в Embedded Coder обеспечивает поддержку автоматической генерации кода C из моделей Simulink для использования на целевом вычислителе. Настройка параметров модели под свойства целевого вычислителя (такие, как размеры типов данных целевого вычислителя, выбор решателя Simulink для генерации кода) осуществляется автоматически при выборе соответствующего целевого процессора в настройках модели.

Автоматическое построение кода

Пакет поддержки процессора NeuroMatrix в Embedded Coder обеспечивает автоматическое построение (компиляцию и линковку) сгенерированного кода.

Пользователь имеет возможность указать собственную реализацию функции main, а по умолчанию автоматически генерируется функция main для обеспечения возможности создания независимого от Simulink исполняемого файла и запуска его на целевом вычислителе.

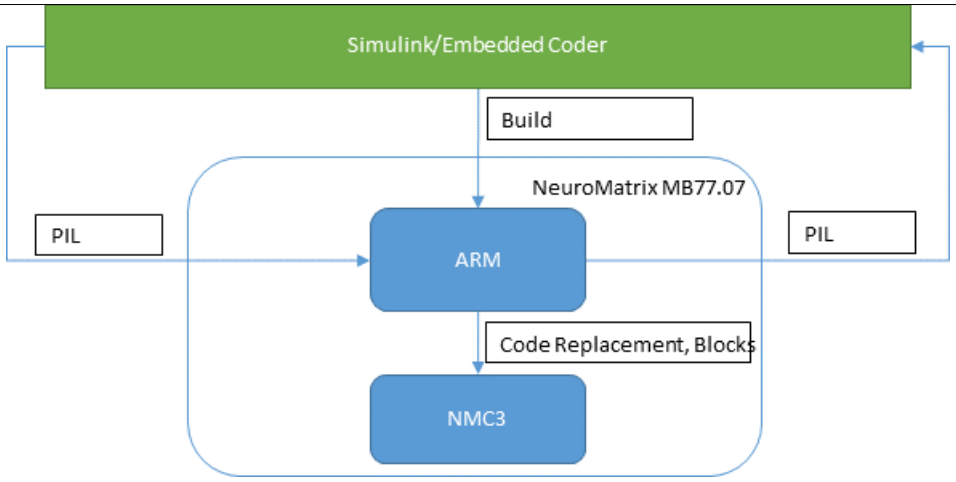
Пакет поддержки обеспечивает автоматическую загрузку исполняемого объектного кода (приложения) на целевой вычислитель и запуск его выполнения. Автоматическая генерация, построение и загрузка кода C осуществляются для процессора ARM1176 архитектуры v6.

Пакет поддержки использует инструментальные средства поддержки (компилятор, линковщик, прошивку Linux) для процессора ARM, предоставляемые компанией НТЦ Модуль.

Мероприятия построения и загрузки кода осуществляются без необходимости ручного вмешательство пользователя, в полностью автоматическом режиме, после нажатия кнопки Deploy to Hardware в модели.

Блок-схема системы

На диаграмме ниже приводится блок-схема системы. Показаны основные составляющие системы и взаимодействие между ними.



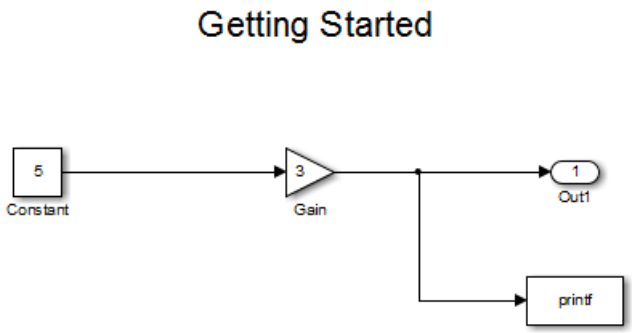
Пример "Hello World"

В этом примере показано, как использовать пакет целевой поддержки процессора NeuroMatrix в Embedded Coder для запуска моделей Simulink® на процессоре NeuroMatrix MB77.07.

Требования

Если вы раньше не работали с Simulink, мы рекомендуем пройти Интерактивный вводный курс по Simulink. Если вы раньше не работали с Embedded Coder, посетите страницу продукта Embedded Coder для получения обзорной информации и примеров.

```
open_system('nmdemo_gettingstarted');
```

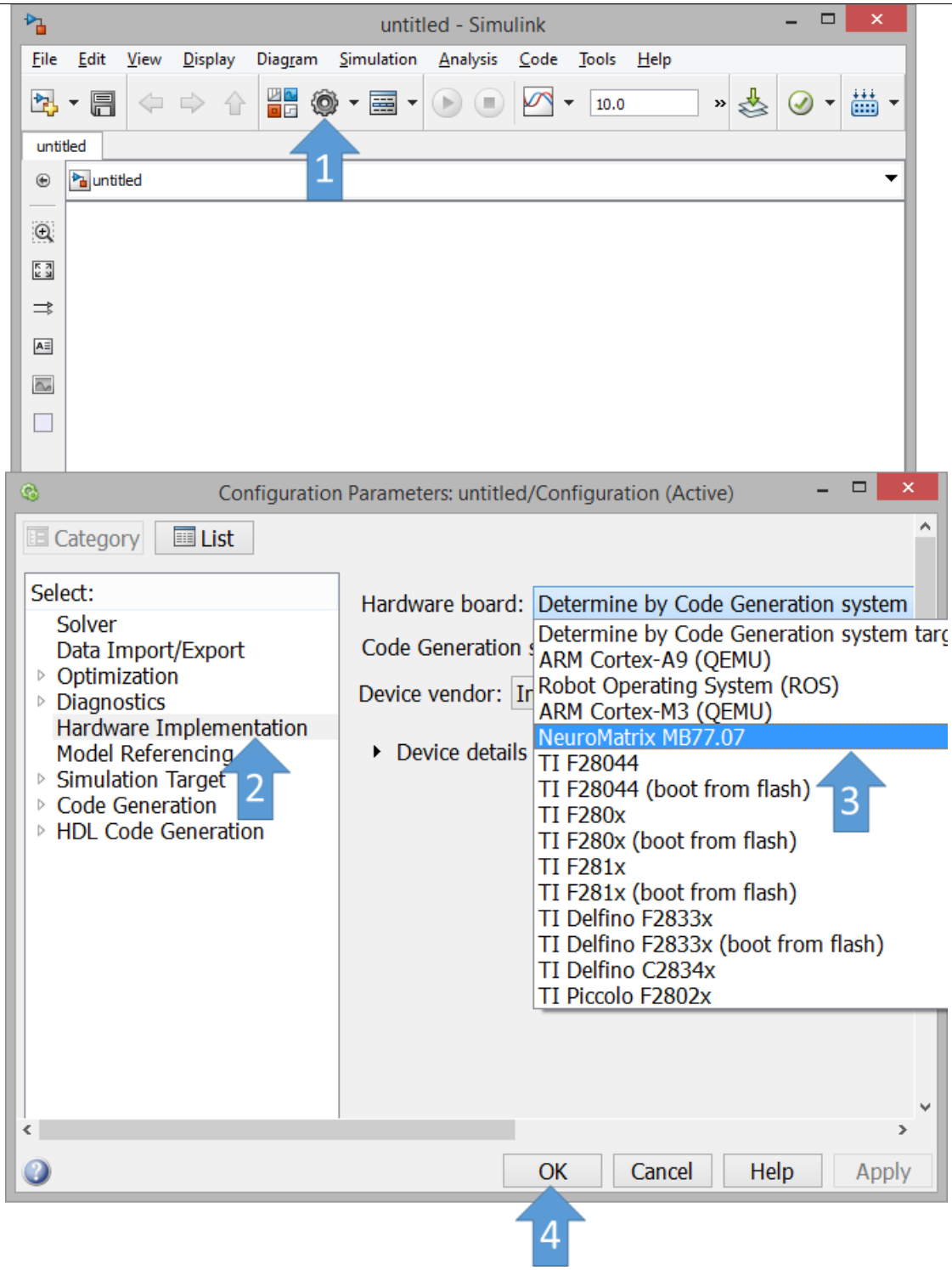


Настройка модели для генерации кода

На этом шаге вы создадите простую модель, которая будет запущена на процессоре NeuroMatrix.

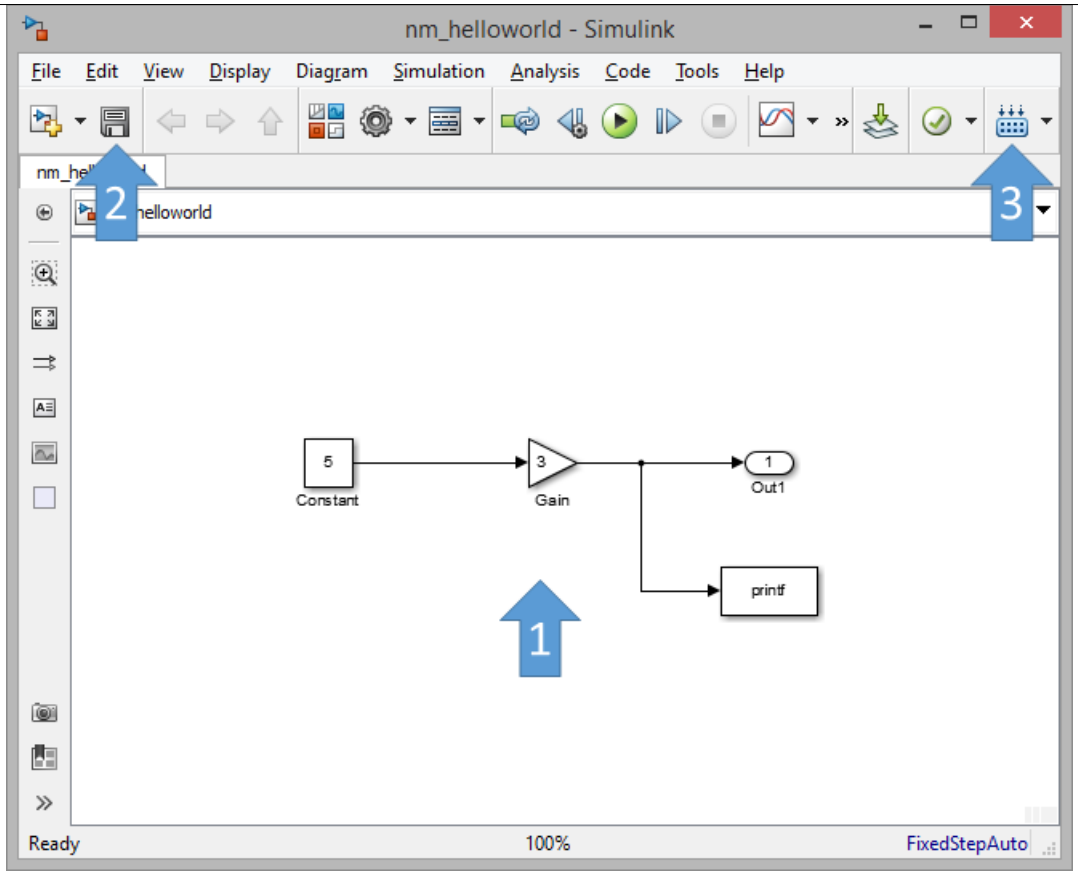
1. В панели инструментов MATLAB, выберите HOME > New > Simulink Model.

2. Чтобы настроить модель для использования пакета целевой поддержки NeuroMatrix, следуйте инструкциям ниже:



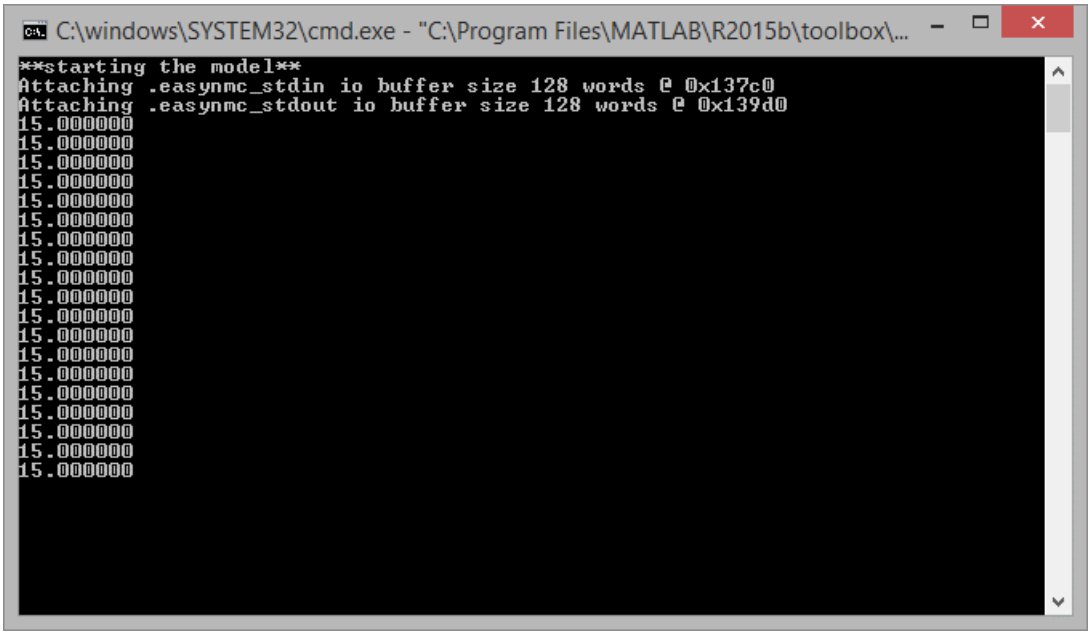
3. Откройте библиотеку блоков Simulink. После открытия библиотеки блоков Simluink, в браузере блоков Simulink (Simulink Library Browser) щелкните по вкладке Simulink > Commonly Used Blocks. Перетащите блок **Constant**, блок **Gain** и блок **Out1** в вашу модель. Также в браузере блоков Simulink щелкните по вкладке NeuroMatrix Blocks > Linux Utilities. Перетащите блок **printf** в вашу модель.

4. Подключите эти блоки, как показано на рисунке ниже, задайте значения блокам **Constant** и **Gain** и сохраните модель. Сгенерируйте и запустите код на процессоре NeuroMatrix нажав на кнопку **Deploy to Hardware**:



Сгенерированный код компилируется и линкуется на хост-компьютере используя набор инструментов для NeuroMatrix. Затем исполняемый файл загружается и запускается на процессоре NeuroMatrix.

Появляется командное окно, сообщающее, что исполняемый файл работает на процессоре NeuroMatrix. Блок printf выводит значение сигнала на стандартный вывод Linux на каждом шаге расчета модели:



4. Закройте это командное окно. Модель завершит выполнение на процессоре NeuroMatrix.

5. Для взаимодействия с кодом, работающим на процессоре NeuroMatrix, обратитесь к разделу документации Верификация и валидация кода в режиме Процессор-в-контуре.

Заключение

В этом примере был показан рабочий процесс для настройки и запуска модели Simulink на процессоре NeuroMatrix.

Теперь можно закрыть модель Simulink:

```
close_system('nmdemo_gettingstarted', 0);
```

Независимое выполнение модели

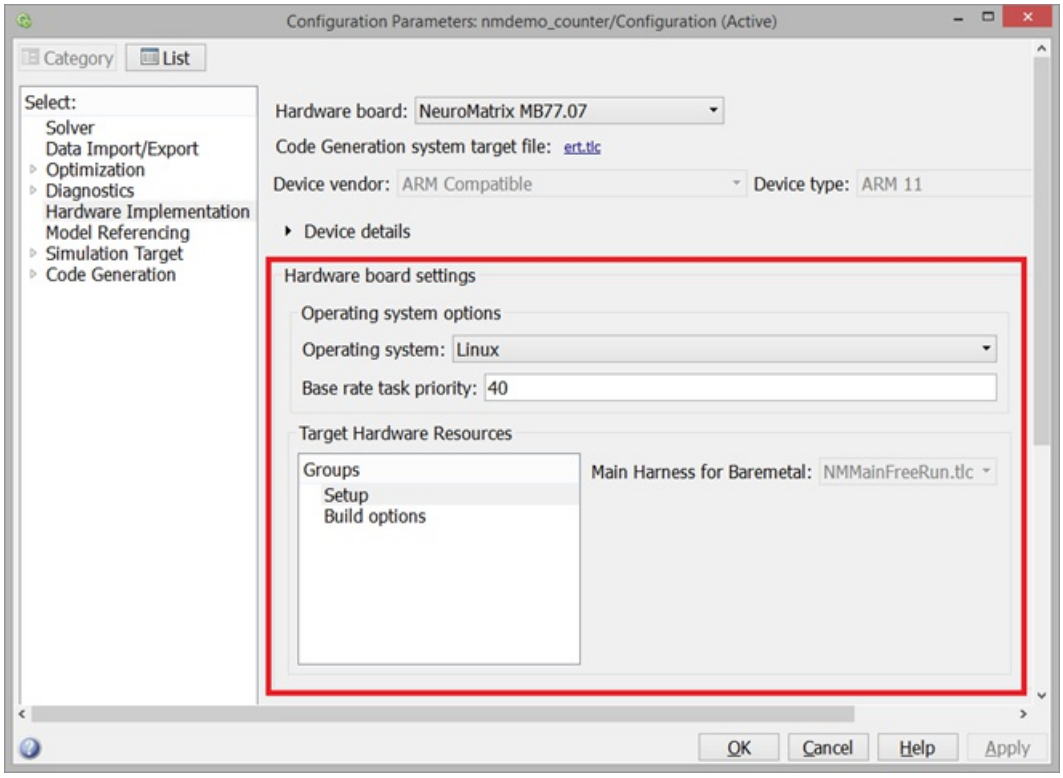
Под независимым выполнением модели подразумевается автономная, независимая от Simulink, работа сгенерированного и скомпилированного кода на целевом процессоре под управлением операционной системы Linux.

Независимое выполнение модели достигается за счет автоматической генерации обвязки, содержащей функцию *main()*. Эта обвязка управляет вызовом автоматически сгенерированной функции *model_step()*, обеспечивая работу сгенерированного кода в реальном времени.

Содержание

Настройки модели для выполнения на целевом процессоре

После выбора в настройках модели целевого процессора NeuroMatrix MB77.07, появляется панель Hardware Board Settings:



- **Панель Operating System Options**

В панели Operating System Options можно выбрать настройку Operating system как Linux или Baremetal.

Опция **Linux:** предоставляет возможность использовать обвязку для функции main(), которая генерируется автоматически при помощи Embedded Coder. Настройка Base rate task priority позволяет задать приоритет выполняемой программы. Данная обвязка поддерживает все режимы работы решателя Simulink.

Опция **Baremetal:** предоставляет возможность пользователю указать свой собственный файл настроек для Embedded Coder (TLC файл), описывающий, как генерировать обвязку для функции *main()*. После выбора этой опции пользователю становится доступна настройка Main Harness for Baremetal в группе Setup панели Target Hardware Resources.

- **Панель Target Hardware Resources**

Группа настроек Setup

Вместе с данным пакетом поддержки поставляются два TLC файла в качестве примера того, как можно настроить под себя генерацию обвязки функции *main()*.

NMMainFreeRun.tlc: последовательные вызовы функции *model_step()* в бесконечном цикле *while*.

NMMainPolling.tlc: вызовы функции *model_step()* с опросом системного таймера.

Пользователь имеет возможность разработать свой собственный TLC файл и поместить его в директорию tlc пакета поддержки. После этого пользовательский файл TLC появится в списке (ниспадающем меню) настройки Main Harness for Baremetal.

Дополнительная информация о создании собственных TLC файлов доступна в документации Embedded Coder, а также в разделе Примеры независимого выполнения модели.

Группа настроек Build options

В группе настроек Build options доступны следующие опции для Build action: Build или Build, load and run.

Опция **Build**: только собрать сгенерированный исходный код.

Опция **Build, load and run**: собрать сгенерированный исходный код, загрузить на целевой процессор и запустить.

Примеры независимого выполнения модели

Примеры независимого выполнения модели

Примеры независимого выполнения модели

В данном разделе приведены примеры моделей, которые могут быть запущены на целевом процессоре и работать автономно. Перед тем, как изучать эти примеры, рекомендуется ознакомиться с разделом документации Независимое выполнение модели.

Содержание

Простой пример

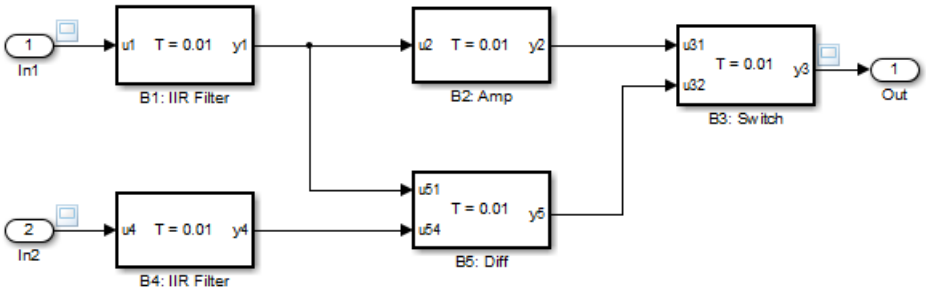
Простой пример (Hello World) независимого выполнения модели представлен в разделе документации Начало работы с NeuroMatrix.

Одночастотная, однозадачная модель

В этом примере рассматривается одночастотная, однозадачная модель (singlerate, singletasking) Simulink.

- Откройте модель nmdemo_ss. В модели представлены два БИХ-фильтра, между которыми осуществляется переключение.

```
open_system('nmdemo_ss');
```



- Выберите опцию ***Display -> Sample Time -> All***, чтобы увидеть частоты дискретизации в модели. Как вы видите, все блоки в модели работают на одной частоте дискретизации и шаг расчета равен 0.01 с.

- Зайдите в настройки модели, в раздел ***Solver***. В меню ***Additional Options*** настройка ***Tasking mode for periodic sample times*** выставлена в значение ***SingleTasking***. Поскольку в модели только одна частота, это аналогично тому, как если бы мы выставили значение этой опции в ***Auto*** (значение по умолчанию). Дополнительную информацию можно получить в документации, щелкнув правой кнопкой по тексту "Tasking mode for periodic sample times" и выбрав в меню "What's This?".

- Сгенерируйте код из модели, нажав кнопку Build.

- В открывшемся отчете по генерации кода, откройте подотчет ***Code Interface Report***. Обратите внимание, что в коде есть функция *nmdemo_ss_step*, которая вызывается периодически, с шагом 0.01 с.

- Исследуйте автоматически сгенерированный код для функции *nmdemo_ss_step* в файле *nmdemo_ss.c*. Обратите внимание, что все блоки модели вычисляются последовательно, один за другим, в порядке, рассчитанном в модели Simulink.

- Исследуйте автоматически сгенерированную функцию *main* в файле *ert_main.c* . Обратите внимание, как из

функции *baseRateTask* вызывается функция *nmdemo_ss_step* . Функция *baseRateTask* в свою очередь привязывается к отдельному потоку в исполняемом приложении. Дополнительные детали можно увидеть в файле linuxinitialize.c, в функции *myRTOSInit*.

- После загрузки этой модели на целевой процессор, скомпилированное приложение будет выполняться в одном потоке, с частотой дискретизации, заданной в настройках модели.

Теперь можно закрыть модель Simulink:

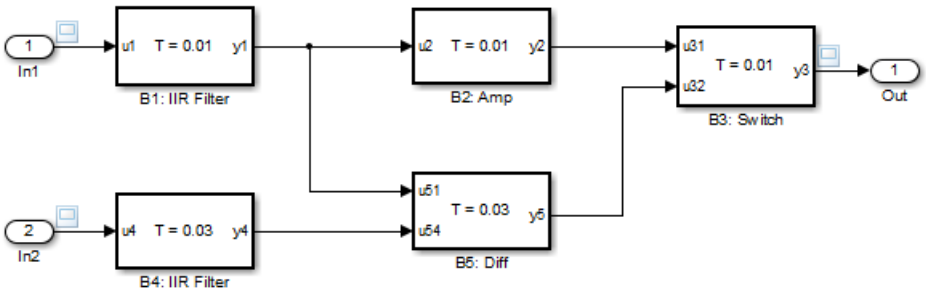
```
close_system('nmdemo_ss', 0);
```

Многочастотная, однозадачная модель

В этом примере рассматривается многочастотная, однозадачная модель (multiraterate, singletasking) Simulink.

- Откройте модель nmdemo_ms. В модели представлены два БИХ-фильтра, между которыми осуществляется переключение.

```
open_system('nmdemo_ms');
```



- Выберите опцию ***Display -> Sample Time -> All***, чтобы увидеть частоты дискретизации в модели. Как вы видите, в модели есть две частоты дискретизации, которые отмечаются разными цветами в модели. В модели используются шаги расчета 0.01 с и 0.03 с.

- Зайдите в настройки модели, в раздел ***Solver***. В меню ***Additional Options*** настройка ***Tasking mode for periodic sample times*** выставлена в значение ***SingleTasking***. Это сделано потому, что мы хотим выполнять сгенерированный код в рамках одной задачи - несмотря на то, что модель содержит несколько частот. Дополнительную информацию можно получить в документации, щелкнув правой кнопкой по тексту "Tasking mode for periodic sample times" и выбрав в меню "What's This?".

- Сгенерируйте код из модели, нажав кнопку Build.

- В открывшемся отчете по генерации кода, откройте подотчет ***Code Interface Report***. Обратите внимание, что в коде есть функция *nmdemo_ms_step*, которая вызывается периодически, с шагом 0.01 с.

- Исследуйте автоматически сгенерированный код для функции *nmdemo_ms_step* в файле *nmdemo_ms.c*. Обратите внимание, что часть блоков (которые считаются с более медленным шагом 0.03 с) рассчитываются по условию *nmdemo_ms_M->Timing.TaskCounters.TID[1] == 0*. По сути, эта часть кода будет активна на каждом третьем вызове функции *nmdemo_ms_step*. Таким образом, Embedded Coder автоматически сгенерировал планировщик для вызова более медленных блоков модели с указанной частотой дискретизации. Обратите внимание на функцию *rate_scheduler* в этом же файле, в которой поддерживается значение счетчика для всех более медленных частот дискретизации модели.

- Исследуйте автоматически сгенерированную функцию *main* в файле *ert_main.c* . Обратите внимание, как из функции *baseRateTask* вызывается функция *nmdemo_ms_step*. Функция *baseRateTask* в свою очередь привязывается к отдельному потоку в исполняемом приложении. Дополнительные детали можно увидеть в файле linuxinitialize.c, в функции *myRTOSInit*. По сути, в автоматически сгенерированной функции *main* ничего не изменилось относительно примера с одночастотной, однозадачной моделью. Всё планирование вызовов подчастот (subrates) осуществляется в файле *nmdemo_ms.c*.

- После загрузки этой модели на целевой процессор, скомпилированное многочастотное приложение будет выполняться в одном потоке, с частотой дискретизации, заданной в настройках модели.

Теперь можно закрыть модель Simulink:

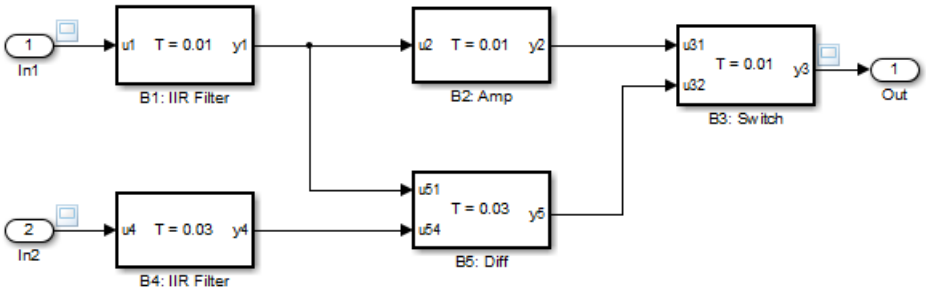
```
close_system('nmdemo_ms', 0);
```

Многочастотная, многозадачная модель

В этом примере рассматривается многочастотная, многозадачная модель (multiraterate, multitasking) Simulink.

- Откройте модель nmdemo_mm. В модели представлены два БИХ-фильтра, между которыми осуществляется переключение.

```
open_system('nmdemo_mm');
```



- Выберите опцию ***Display -> Sample Time -> All***, чтобы увидеть частоты дискретизации в модели. Как вы видите, в модели есть две частоты дискретизации, которые отмечаются разными цветами в модели. В модели используются шаги расчета 0.01 с и 0.03 с.

- Зайдите в настройки модели, в раздел ***Solver***. В меню ***Additional Options*** настройка ***Tasking mode for periodic sample times*** выставлена в значение ***MultiTasking***. Это сделано потому, что мы хотим выполнять сгенерированный код в рамках нескольких отдельных задач (потоков). Дополнительно, в настройках решателя выставлена опция ***Automatically handle rate transition for data transfer***, которая используется для автоматической синхронизации данных (буферов) между потоками. Дополнительную информацию можно получить в документации, щелкнув правой кнопкой по тексту интересующей вас опции и выбрав в меню "What's This?".

- Сгенерируйте код из модели, нажав кнопку Build.

- В открывшемся отчете по генерации кода, откройте подотчет ***Code Interface Report***. Обратите внимание, что в коде есть две периодических функции, *nmdemo_ms_step0*, которая считается с шагом 0.01 с и *nmdemo_ms_step1*, которая считается с шагом 0.03 с.

- Исследуйте автоматически сгенерированный код для функции *nmdemo_mm_step* в файле *nmdemo_mm.c*. Обратите внимание, что это лишь функция-обвязка, которая принимает на вход идентификатор задачи (tid) и вызывает соответствующую функцию *nmdemo_ms_step0* или *nmdemo_ms_step1*. Обратите также внимание на дополнительный код в функции *nmdemo_ms_step0*, который управляет буферами для передачи данных между быстрой и медленной частями модели. Таким образом, Embedded Coder автоматически сгенерировал планировщик с вытесняющей многозадачностью (rate-monotonic scheduler). Обратите внимание на функцию *rate_monotonic_scheduler* в этом же файле, в которой поддерживается значение счетчика для более медленных частот дискретизации модели, который используется для синхронизации буфера обмена.

- Исследуйте автоматически сгенерированную функцию *main* в файле *ert_main.c* . Обратите внимание, как из функции *baseRateTask* вызывается функция *nmdemo_mm_step(0)*, соответствующая самой быстрой (базовой) частоте. Функция *baseRateTask* в свою очередь привязывается к отдельному потоку в исполняемом приложении. Дополнительно в коде сгенерирована функция *subrateTask*, которая привязана к отдельному потоку для каждой подчастоты (subrate) и вызывает соответствующую функцию *nmdemo_mm_step(subRateId)*. Дополнительные детали можно увидеть в файле linuxinitialize.c, в функции *myRTOSInit*.

- После загрузки этой модели на целевой процессор, скомпилированное многочастотное приложение будет выполняться в нескольких потоках, с частотой дискретизации для каждого потока, заданной в настройках модели.

- Чтобы наглядно увидеть, как создаются и выполняются задачи при работе приложения, выберите в настройках модели, во вкладке **Code Generation**, опцию **Build Configuration**, как **Debug**. В таком случае при компиляции кода будет активен макрос *MW_RTOS_DEBUG* и в командном окне выполняемого приложения вы увидите дополнительную отладочную информацию о том, как создаются и выполняются потоки.

Теперь можно закрыть модель Simulink:

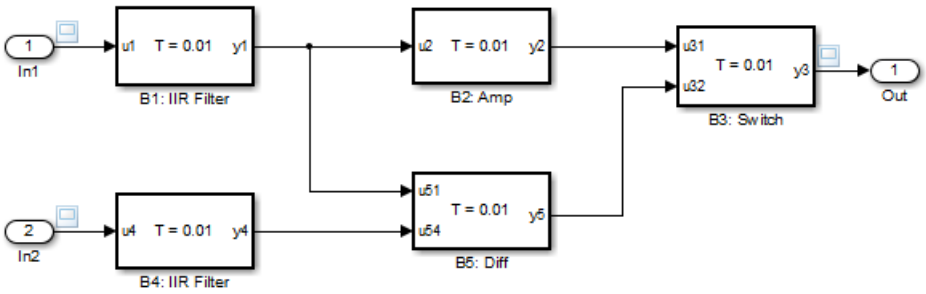
```
close_system('nmdemo_mm', 0);
```

Использование собственной функции main

В случае, если вы хотите иметь полный контроль над тем, как должна выглядеть автоматически сгенерированная функция *main*, вы можете написать TLC файл, в котором будет реализован ваш собственный планировщик. Дополнительная информация приводится в разделе документации Независимое выполнение модели, в подразделе "Группа настроек Setup".

- Откройте модель nmdemo_custommain. В модели представлены два БИХ-фильтра, между которыми осуществляется переключение.

```
open_system('nmdemo_custommain');
```



- В настройках модели, во вкладке **Hardware Implementation**, опция **Operating system** выбрана как **Baremetal**. В группе настроек **Setup**, опция **Main Harness for Baremetal** выбрана как **NMMainPolling.tlc**.

- Исследуйте файл NMMainPolling.tlc. В нем вы видите описание функции *main* на языке TLC, понятном для Embedded Coder. Внутри функции *main* осуществляются вызовы функции *model_step()* с опросом системного таймера (polling).

- Сгенерируйте код из модели, нажав кнопку Build.

- В открывшемся отчете по генерации кода, исследуйте файл *ert_main.c*. Обратите внимание на то, что сгенерированный код соответствует описанию в TLC файле.

- Запустите модель на целевом процессоре. Убедитесь, что используется ваша собственная функция *main*, наблюдая сообщения в командном окне выполняемого приложения.

Теперь можно закрыть модель Simulink:

```
close_system('nmdemo_custommain', 0);
```

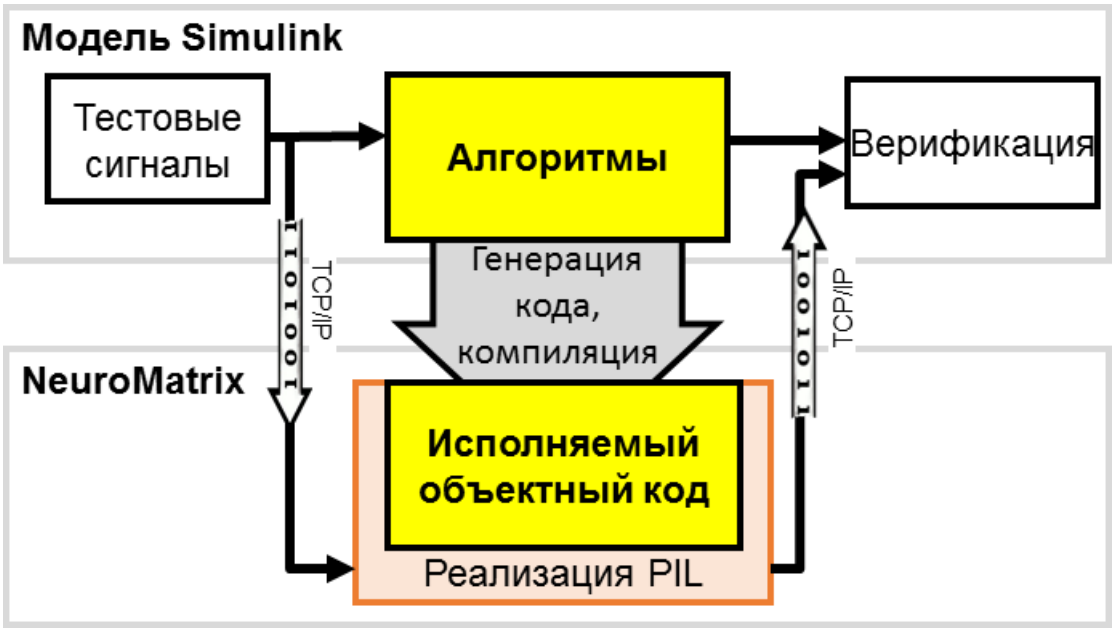
Заключение

В данных примерах были показаны различные варианты запуска моделей Simulink, использующих разные типы решателей, на целевом процессоре в автономном режиме.

Режим симуляции "Процессор-в-контуре"

Пакет поддержки процессора NeuroMatrix обеспечивает возможность верификации исполняемого объектного кода в режиме симуляции **"Процессор-в-контуре"** (Processor-in-the-Loop, PIL).

Режим PIL используется для верификации численной эквивалентности между работой сгенерированного кода и оригинальной модели Simulink. Кроме того, режим PIL может применяться для проверки работы сгенерированного кода в оригинальном окружении Simulink – т.е. с теми же входными данными, внешними моделями и возможностями анализа и визуализации выходных сигналов алгоритма:



Дополнительная информация о режиме PIL доступна в документации к Embedded Coder.

Пакет поддержки процессора NeuroMatrix использует интерфейс TCP/IP для обмена данными в режиме PIL между моделью Simulink ("хост") и целевым вычислителем ARM ("таргет"). Для обмена данными посредством TCP/IP между хостом и таргетом используется стандартная библиотека сетевых сокетов, входящая в состав ядра Linux. Пакет поддержки процессора NeuroMatrix обеспечивает возможность профилирования (измерения времени выполнения) кода в режиме PIL. Профилирование осуществляется с использованием функций таймера для ARM, предоставляемых ядром Linux. Измеряется только время выполнения функции на целевом процессоре, исключая время на обмен данными между хостом и таргетом.

Содержание

Примеры верификации в режиме "Процессор-в-контуре"

Примеры верификации в режиме "Процессор-в-контуре"

Примеры верификации в режиме "Процессор-в-контуре"

В этих примерах показывается, как использовать режим симуляции Процессор-в-контуре (Processor-in-the-Loop, PIL) для верификации и валидации кода, работающего на процессоре NeuroMatrix.

Содержание

Введение

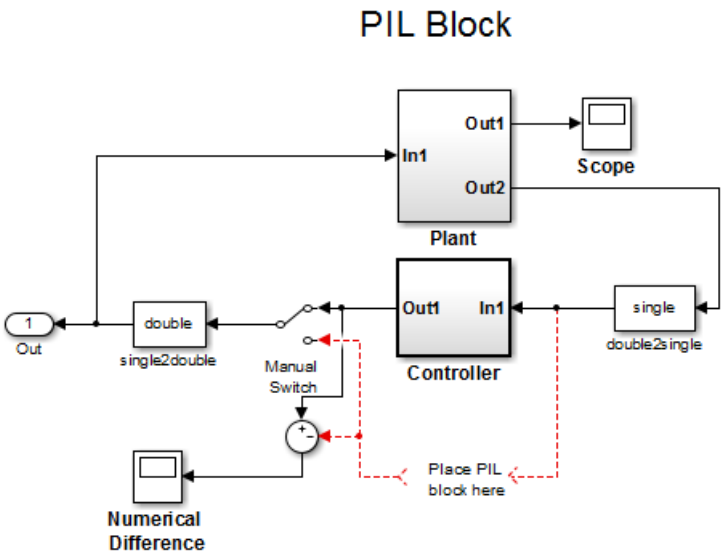
В этих примерах вы узнаете, как настроить модель Simulink для симуляции в режиме Процессор-в-контуре (Processor-in-the-Loop, PIL). В режиме PIL, сгенерированный код работает на процессоре NeuroMatrix. Результаты симуляции PIL передаются в Simulink для верификации численной эквивалентности результатов симуляции модели и работы сгенерированного и скомпилированного кода. Дополнительно, режим PIL позволяет осуществлять профилирование кода (измерение времени выполнения). Верификация в режиме PIL является важной частью процесса проектирования, обеспечивающей корректное поведение работы развернутого кода относительно проекта.

Требования

Мы рекомендуем изучить пример Начало работы с NeuroMatrix.

Пример 1 - Верификация сгенерированного кода с использованием блока PIL

```
open_system('nmdemo_pil_block');
```



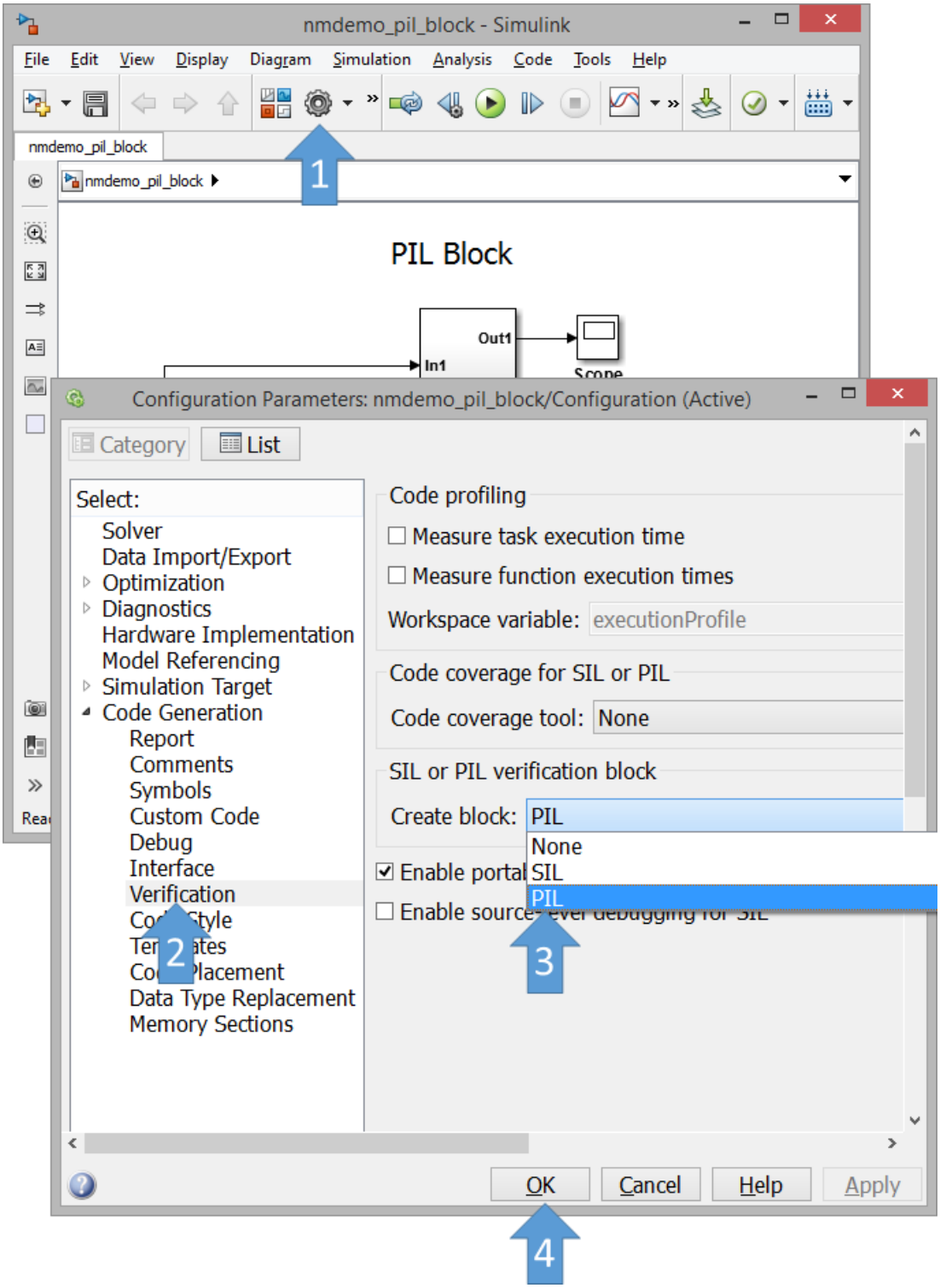
В этом примере показывается, как автоматически сгенерированный блок PIL может использоваться для верификации. Используя такой подход:

- Вы можете верифицировать код, сгенерированный для подсистемы
- Вы должны использовать модель с тестовой обвязкой для передачи тестовых воздействий (входных данных для кода)
- Вы должны заменить оригинальную подсистему автоматически сгенерированным блоком PIL. Следует быть

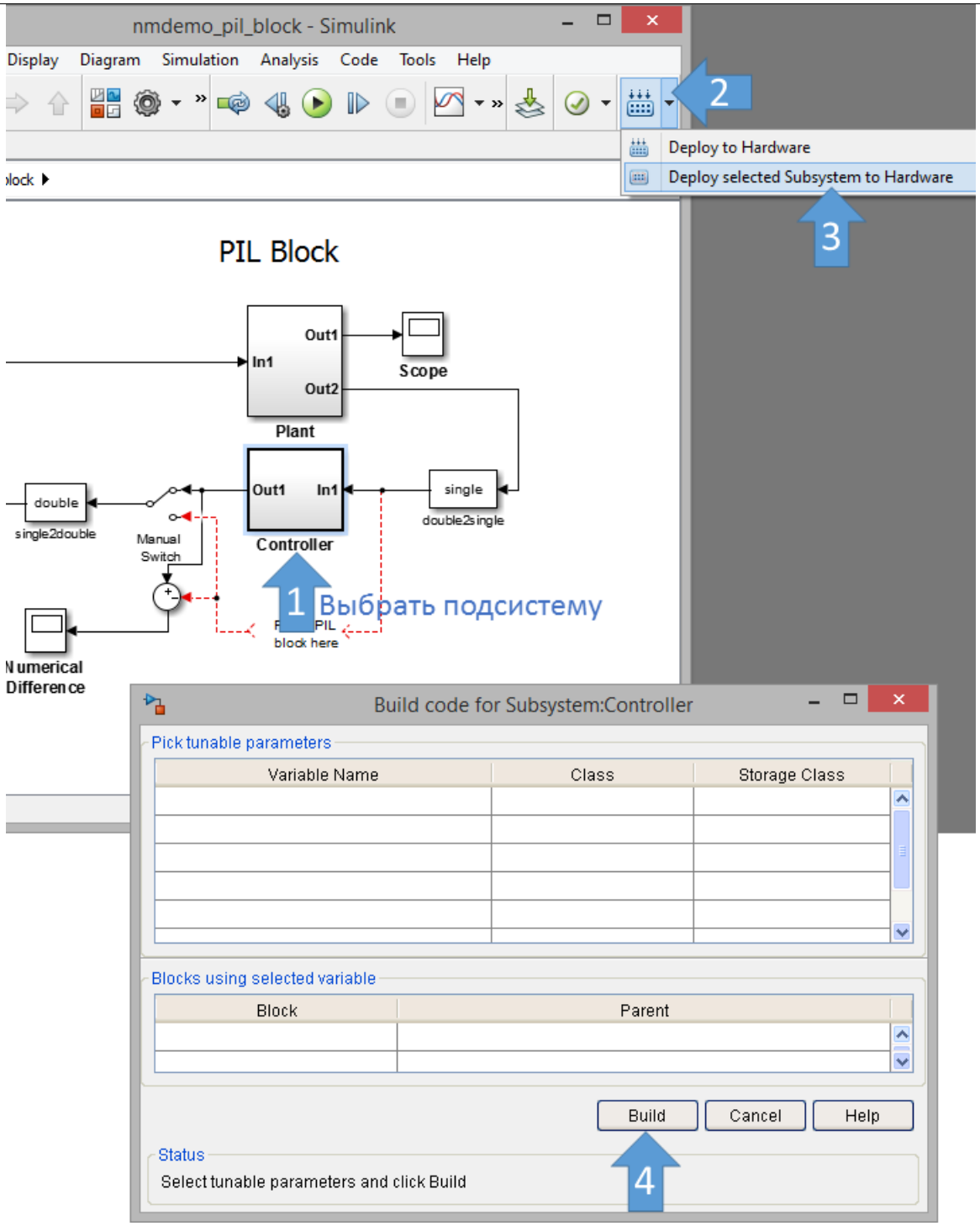
осторожным и не сохранять модель, если вы хотите сохранить оригинальную подсистему

1. Откройте модель с блоком PIL. Эта модель настроена на использование целевой поддержки для **NeuroMatrix**. Цель этого задания заключается в создании блока PIL из подсистемы **Controller**, и запуска его на процессоре NeuroMatrix.

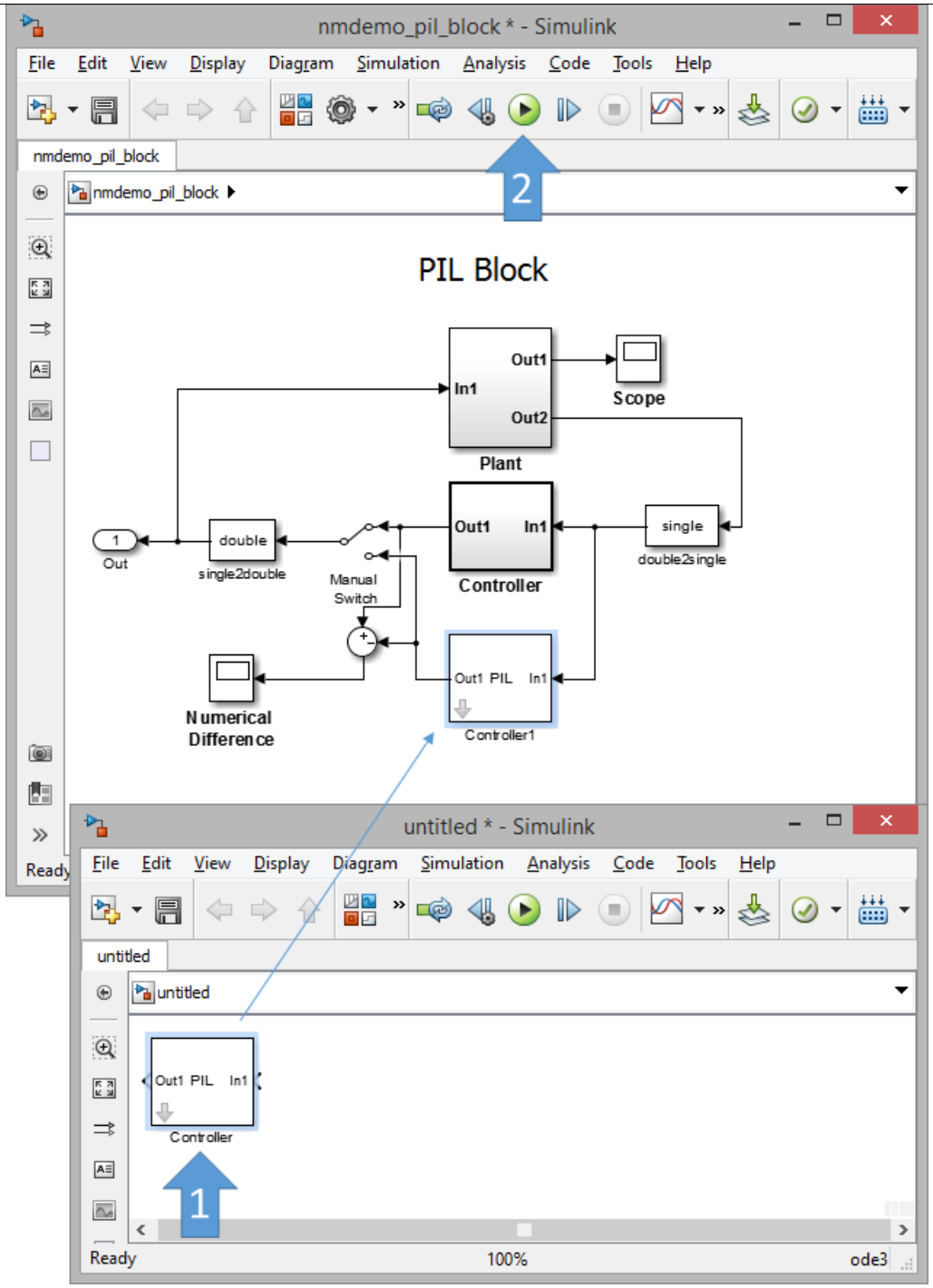
2. Активируйте опцию PIL следуя инструкциям:



3. Создайте блок PIL для подсистемы **Controller**, следуя инструкциям:



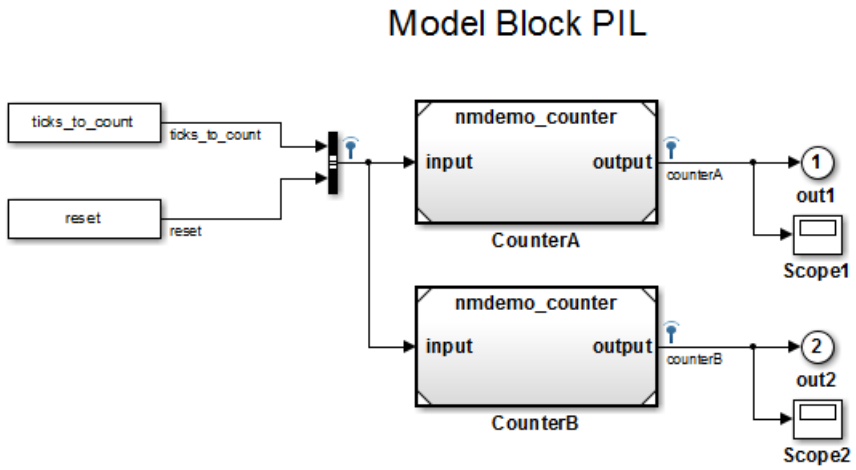
4. Запустите симуляцию PIL:



5. После запуска симуляции модели, код будет автоматически запущен на процессоре NeuroMatrix. Вы можете переключаться между оригинальной подсистемой и блоком PIL, дважды щелкнув по блоку **Manual Switch**. Щелкните по блоку **Numerical Differences**, чтобы увидеть численное отличие между симулируемой подсистемой **Controller** и блоком PIL, работающим на процессоре NeuroMatrix.

Пример 2 - Верификация сгенерированного кода в режиме PIL с использованием блока Model

```
open_system('nmdemo_model_pil_block');
```

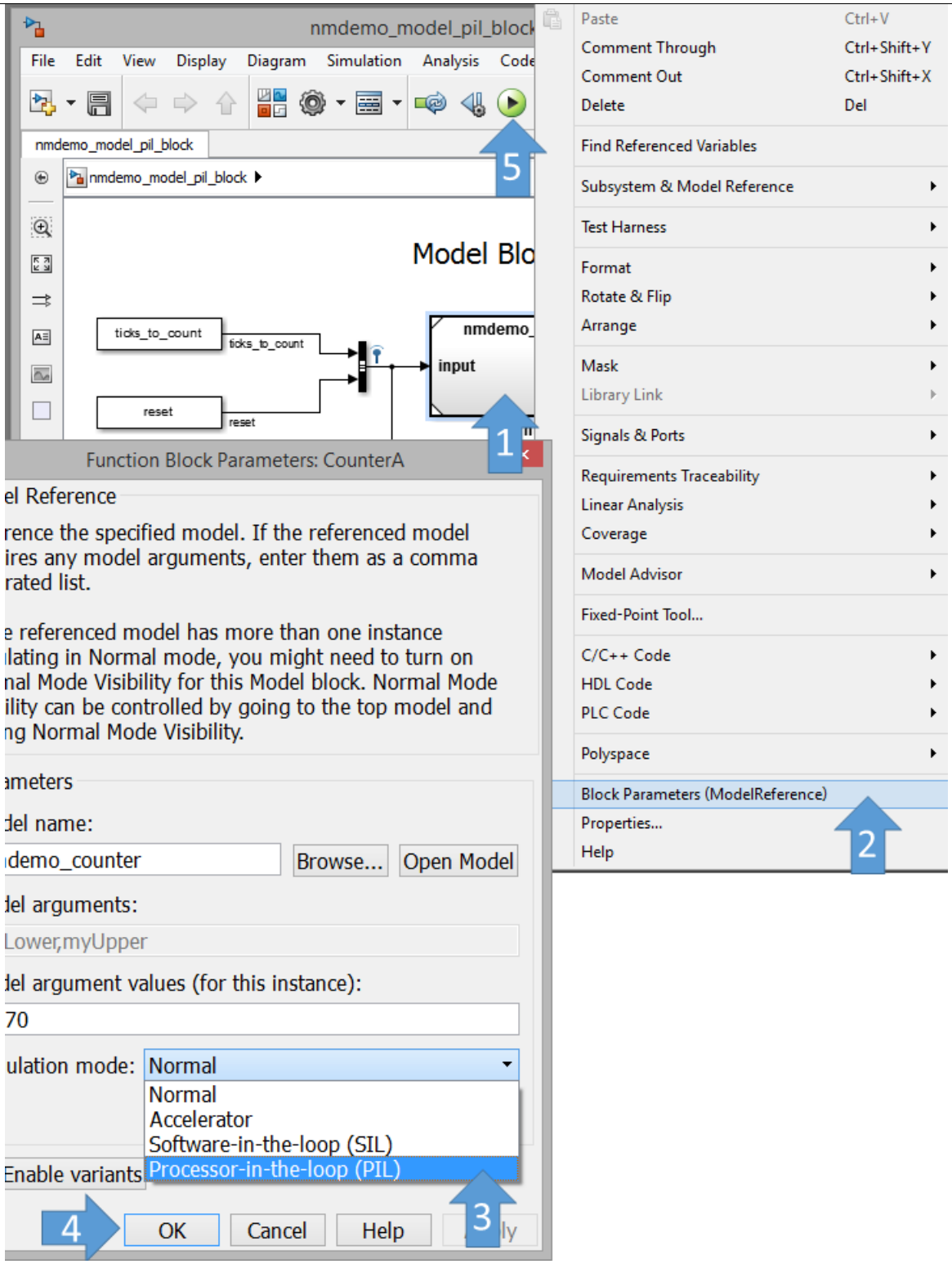


В этом примере показано, как осуществлять верификацию автоматически сгенерированного кода для модели-ссылки (model reference) при помощи симуляции в режиме PIL. С использованием такого подхода:

- Вы можете верифицировать код, сгенерированный для модели-ссылки
- Вы должны использовать модель с тестовой обвязкой для передачи тестовых воздействий (входных данных для кода)
- Вы можете легко переключать режим симуляции блока Model между обычным режимом симуляции и PIL

1. Откройте модель с блоком Model для PIL. Эта модель настроена на использование целевой поддержки для **NeuroMatrix**. В модели содержатся два блока Model, которые оба указывают на одну и ту же модель-ссылку. Вы настроите один из блоков Model для симуляции в режиме PIL, а другой для симуляции в обычном режиме.

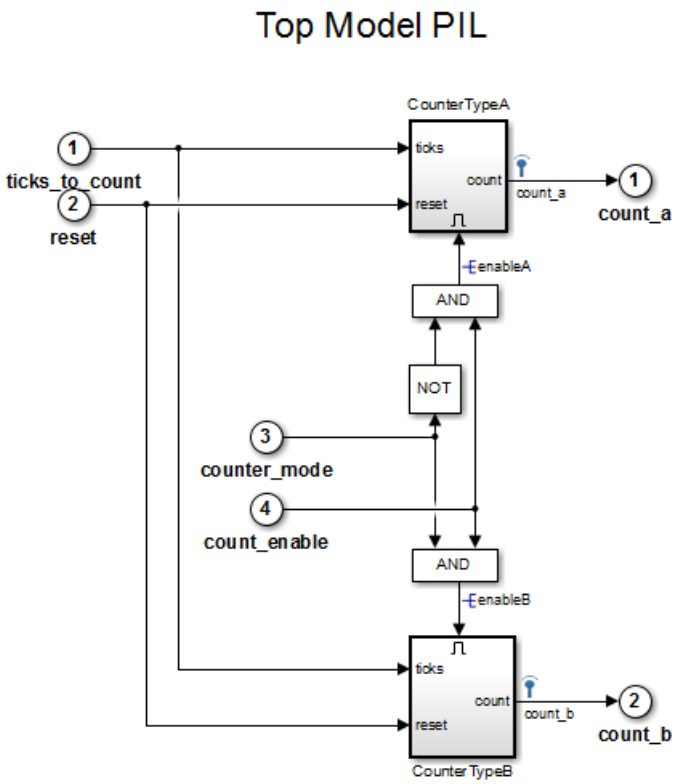
2. Настройте и запустите блок Model с именем **CounterA** в режиме симуляции PIL, следуя инструкциям:



3. После запуска симуляции модели, блок **Scope1** отображает выходной результат симуляции блока **CounterA**, запущенного на процессоре NeuroMatrix в режиме PIL, а блок **Scope2** отображает выходы блока **CounterB**, работающего в обычном режиме симуляции.

Пример 3 - Верификация сгенерированного кода в режиме PIL для модели верхнего уровня

```
open_system('nmdemo_top_model_pil');
```

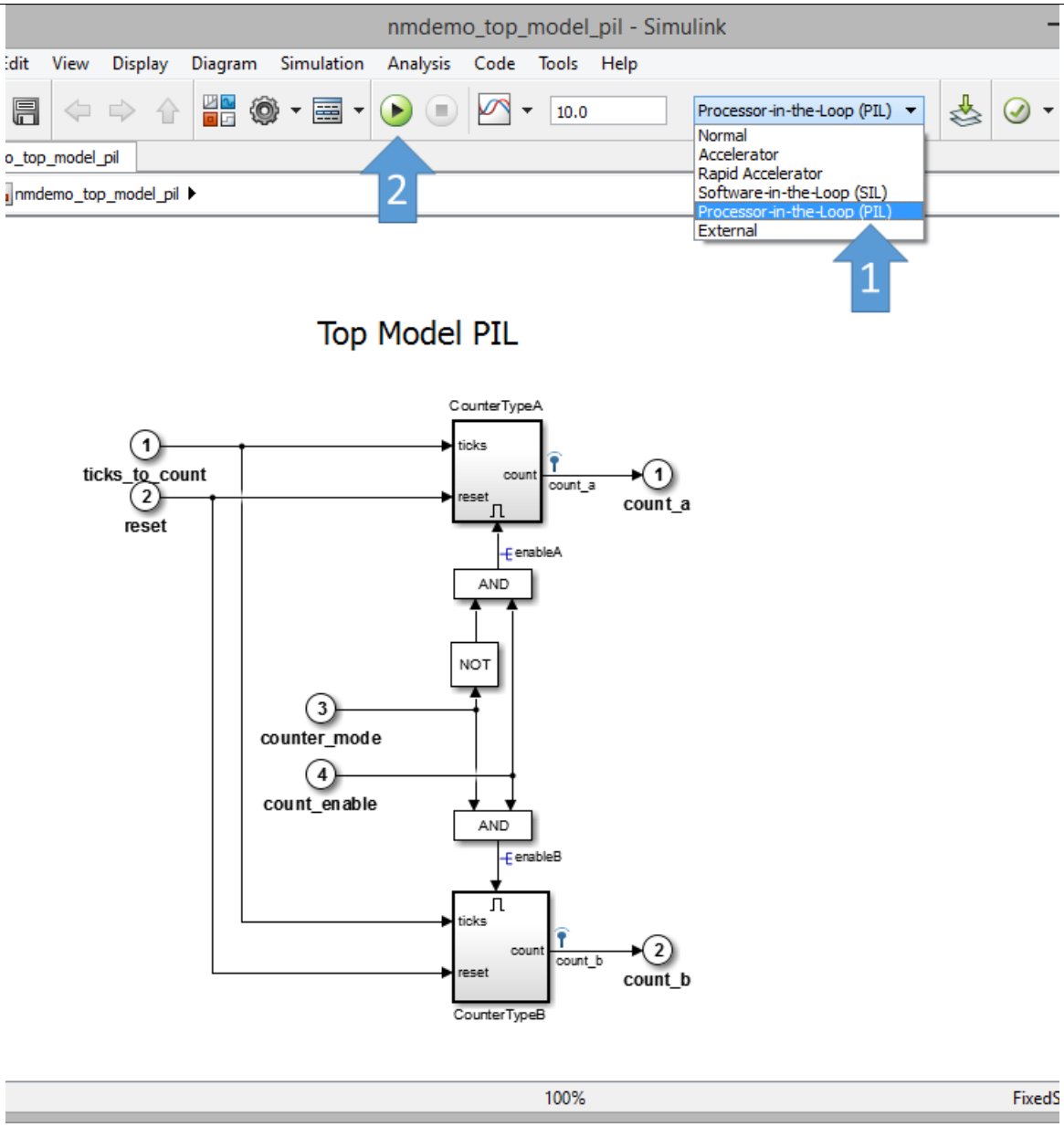


В этом примере показано, как осуществлять верификацию автоматически сгенерированного кода для модели верхнего уровня при помощи PIL симуляции. С использованием такого подхода:

- Вы можете верифицировать код, сгенерированный для модели верхнего уровня
- Вы должны настроить модель для загрузки тестовых воздействий (входных данных) из рабочего пространства MATLAB
- Вы можете легко переключаться между симуляцией всей модели в обычном режиме и в режиме PIL

1. Откройте модель верхнего уровня для PIL. Эта модель настроена на использование целевой поддержки для **NeuroMatrix**. Вы запустите эту модель в режиме симуляции PIL.

2. Запустите всю модель в режиме симуляции PIL следуя инструкциям:



3. По окончании симуляции PIL, в рабочем пространстве создается переменная **logsOut**. В этой переменной **logsOut** содержатся результаты симуляции PIL. Вы можете обратиться к записанным данным для сигналов **count_a** и **count_b** используя следующие команды:

```
count_a = get(logsOut,'count_a');
count_a.Values.Data

count_b = get(logsOut,'count_b');
count_b.Values.Data
```

Пример 4 - Профилирование времени выполнения в режиме PIL

В этом примере показано, как осуществлять профилирование (измерение времени выполнения) автоматически сгенерированного кода во время симуляции в режиме PIL. С использованием такого подхода:

- Вы можете измерять время выполнения кода в режиме PIL
- Вы можете осуществлять профилирование кода для всех режимов симуляции PIL (блок PIL, блок Model, модель верхнего уровня)
- Вы можете осуществлять визуализацию времени выполнения функций в автоматически сгенерированном коде

1. Откройте модель с блоком Model для PIL. Эта модель настроена на использование целевой поддержки для **NeuroMatrix**. В этом примере вы будете использовать режим симуляции PIL для блока Model, но для других режимов PIL настройка аналогична.

2. Настройте блок Model с именем **CounterA** для режима симуляции PIL, следуя инструкциям из Примера 2.

3. Настройте модель для сбора информации о времени выполнения во время симуляции PIL:

- Щелкните **Model Configuration Parameters** на панели инструментов модели, выберите раздел настроек **Code Generation**, а затем пункт **Verification**.
- Выставьте галочку **Measure task execution time** ("Измерять время выполнения"). Эта опция позволяет осуществлять профилирование времени выполнения для каждой частоты в модели. Переменная с именем **executionProfile** в рабочем пространстве MATLAB будет содержать данные профилирования. Нажмите OK.

4. Запустите симуляцию. По окончании симуляции, используйте следующие команды, чтобы подробно изучить результаты профилирования:

```
executionProfile.report

executionProfile.timeline
```

Заключение

В этих примерах было дано введение в рабочий процесс для верификации кода с использованием возможностей PIL симуляции.

Теперь можно закрыть модели Simulink:

```
close_system('nmdemo_pil_block', 0);
close_system('nmdemo_model_pil_block', 0);
close_system('nmdemo_top_model_pil', 0);
```

Технология Code Replacement

Пакет целевой поддержки обеспечивает автоматическую генерацию оптимизированного кода для векторного сопроцессора NeuroMatrix для поддерживаемых блоков Simulink. В основе этой возможности лежит технология "Подстановка кода" (**Code Replacement Library**, CRL). Дополнительная информация о CRL доступна в документации к Embedded Coder. Поддерживаемые разрядности в блоках Simulink соответствуют поддерживаемым разрядностям функций NeuroMatrix. По имени функции NeuroMatrix можно определить, какие типы данных входных аргументов поддерживаются. Например, *8s* - знаковый 8-битный тип данных, т.е. int8. *8u* - беззнаковый 8-битный тип данных. *8s16s* - означает, что первый вход (аргумент) 8-битный, а второй вход 16-битный. *64sc* - это комплексный тип данных для 64-битного знакового числа. Обратите внимание, что размерность соответствующего сигнала должна быть кратной 64 битам. Т.е., если вы работаете с 8-битным типом данных, размерность обрабатываемого массива должна быть равна как минимум 8, для 16-битных типов - 4 и т.д.

Список поддерживаемых блоков Simulink для CRL и соответствующих функций NeuroMatrix:

Имя стандартного блока Simulink	**Имя функции NeuroMatrix**	**Примечание**
Abs	nmppsAbs_8s nmppsAbs_16s nmppsAbs_32s nmppsAbs_64s	Вычисление абсолютных значений для элементов вектора
Add	nmppsAddC_8s nmppsAddC_16s nmppsAddC_32s nmppsAddC_64s	Добавление к вектору константы
Add	nmppsAdd_8s nmppsAdd_16s nmppsAdd_32s nmppsAdd_64s	Сложение двух векторов
Subtract	nmppsSubC_8s nmppsSubC_16s nmppsSubC_32s nmppsSubC_64s nmppsSubCRev_8s nmppsSubCRev_16s nmppsSubCRev_32s nmppsSubCRev_64s	Вычитание константы из вектора или вектора из константы
Subtract	nmppsSub_8s nmppsSub_16s nmppsSub_32s nmppsSub_64s	Вычитание двух векторов
Shift Arithmetic	nmppsRShiftC_8s nmppsRShiftC_16s nmppsRShiftC_32s nmppsRShiftC_64s nmppsRShiftC_8u nmppsRShiftC_16u nmppsRShiftC_32u nmppsRShiftC_64u	Операция арифметического сдвига вправо
Product	nmppsMulC_8s nmppsMulC_16s nmppsMulC_32s nmppsMulC_64s nmppsMulC_8s16s nmppsMulC_16s32s nmppsMulC_32s64s	Умножение вектора на константу
Product	nmppmMul_mm_8s8s nmppmMul_mm_8s16s	Умножение матрицы на матрицу

	nmppmMul_mm_8s32s nmppmMul_mm_8s64s nmppmMul_mm_16s16s nmppmMul_mm_16s32s nmppmMul_mm_16s64s nmppmMul_mm_32s32s nmppmMul_mm_32s64s	
Product	nmppmMul_mv_8s64s nmppmMul_mv_16s64s nmppmMul_mv_32s64s	Умножение матрицы на вектор
Compare to Constant	nmppsCmpLtC_8s8u nmppsCmpLtC_16s8u nmppsCmpLtC_32s8u	Сравнение элементов массива на признак "меньше константы"
Compare to Constant	nmppsCmpNeC_8s8u nmppsCmpNeC_16s8u nmppsCmpNeC_32s8u nmppsCmpNeC_64s8u	Сравнение элементов массива на признак "неравенство константе"
Relational Operator	nmppsCmpLt_8s8um nmppsCmpLt_16s8um nmppsCmpLt_32s8um	Поэлементное сравнение элементов двух векторов на признак "меньше"
Relational Operator	nmppsCmpNe_8s8um nmppsCmpNe_16s8um nmppsCmpNe_32s8um nmppsCmpNe_64s8um	Поэлементное сравнение элементов двух векторов на признак "неравенство"

Содержание

Примеры использования Code Replacement

Примеры использования Code Replacement

Примеры использования Code Replacement

В данном разделе приведены примеры использования Code Replacement.

Исследуйте отчет по генерации кода (и особенно подотчет Code Replacement) для получения информации о подстановках кода, которые имели место в данной модели.

Содержание

Вычисление абсолютных значений для элементов вектора:

nmdemo_crl_abs_top

Добавление к вектору константы и сложение двух векторов:

nmdemo_crl_add_top

Вычитание константы из вектора или вектора из константы и вычитание двух векторов:

nmdemo_crl_sub_top

Операция арифметического сдвига вправо:

nmdemo_crl_shift_top

Умножение вектора на константу:

nmdemo_crl_mulc_top

Умножение матрицы на матрицу и матрицы на вектор:

nmdemo_crl_mulm_top

Сравнение элементов массива на признаки "меньше константы", "неравенство константе":

nmdemo_crl_cmpc_top

Поэлементное сравнение элементов двух векторов на признаки "меньше", "неравенство":

nmdemo_crl_cmp_top

Специализированные блоки Simulink

Пакет целевой поддержки предоставляет дополнительные, специализированные блоки Simulink для симуляции и генерации оптимизированного кода для векторного сопроцессора NeuroMatrix. Блоки Simulink должны применяться тогда, когда соответствующий блок Simulink не поддерживает CRL, или, когда для библиотечной функции NeuroMatrix нет соответствующего блока Simulink. Блоки NeuroMatrix находятся в браузере библиотек Simulink, в разделе "NeuroMatrix Blocks". Список дополнительных блоков Simulink и соответствующих функций NeuroMatrix:

Имя дополнительного блока Simulink	**Имя функции NeuroMatrix**	**Примечание**
FFT	nmppsFFT256Fwd nmppsFFT512Fwd nmppsFFT1024Fwd	Быстрое преобразование Фурье
IFFT	nmppsFFT256Inv nmppsFFT512Inv nmppsFFT1024Inv	Обратное преобразование Фурье
ANDC	nmppsAndC_8u nmppsAndC_16u nmppsAndC_32u nmppsAndC_64u	Функция логического "И" между вектором и константой
AND	nmppsAnd_8u nmppsAnd_16u nmppsAnd_32u nmppsAnd_64u	Функция логического "И" между двумя векторами
ORC	nmppsOrC_8u nmppsOrC_16u nmppsOrC_32u nmppsOrC_64u	Функция логического "ИЛИ" между вектором и константой
OR	nmppsOr_8u nmppsOr_16u nmppsOr_32u nmppsOr_64u	Функция логического "ИЛИ" между двумя векторами
XORC	nmppsXorC_8u nmppsXorC_16u nmppsXorC_32u nmppsXorC_64u	Функция логического "Исключающего ИЛИ" между вектором и константой
XOR	nmppsXor_8u nmppsXor_16u nmppsXor_32u nmppsXor_64u	Функция логического "Исключающего ИЛИ" между двумя векторами
NOT	nmppsNot_8u nmppsNot_16u nmppsNot_32u nmppsNot_64u	Функция логического "НЕ" над элементами вектора
SUM	nmppsSum_8s nmppsSum_16s nmppsSum_32s nmppsSum_64s	Возвращает сумму всех элементов вектора
DOT	nmppsDotProd_8s8sm nmppsDotProd_8s16sm nmppsDotProd_8s32sm nmppsDotProd_8s64s nmppsDotProd_16s16sm nmppsDotProd_16s32sm nmppsDotProd_16s64s nmppsDotProd_32s32sm nmppsDotProd_32s64s	Скалярное произведение двух векторов

	nmppsDotProd_64s64s	
MAX	nmppsMax_8s nmppsMax_16s nmppsMax_32s	Поиск значения максимального элемента вектора
MIN	nmppsMin_8s nmppsMin_16s nmppsMin_32s	Поиск значения минимального элемента вектора

Кроме того, поддерживаются специализированные блоки Simulink, предоставляющие доступ к периферийным устройствам, доступным на плате и функциям ОС Linux:

Имя дополнительного блока Simulink	**Примечание**
printf	Печать значения на стандартный вывод Linux

Содержание

Примеры использования блоков Simulink

Примеры использования блоков Simulink

Примеры использования блоков Simulink

В данном разделе приведены примеры использования блоков Simulink из библиотеки NeuroMatrix Blocks.

Содержание

Быстрое преобразование Фурье:

nmdemo_blocks_fft_top

Обратное преобразование Фурье:

nmdemo_blocks_ifft_top

Поиск значения минимального или максимального элемента вектора:

nmdemo_blocks_minmax_top

Функция логического "И" между двумя векторами:

nmdemo_blocks_and_top

Функция логического "И" между вектором и константой:

nmdemo_blocks_andc_top

Функция логического "ИЛИ" между двумя векторами:

nmdemo_blocks_or_top

Функция логического "ИЛИ" между вектором и константой:

nmdemo_blocks_orc_top

Функция логического "Исключающего ИЛИ" между двумя векторами:

nmdemo_blocks_xor_top

Функция логического "Исключающего ИЛИ" между вектором и константой:

nmdemo_blocks_xorc_top

Нахождение суммы всех элементов вектора:

nmdemo_blocks_sum_top

Функция логического "НЕ" над элементами вектора:

nmdemo_blocks_not_top

Нахождение скалярного произведения двух векторов:

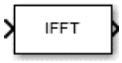
nmdemo_blocks_dot_top

Печать значения сигнала на стандартный вывод Linux:

nmdemo_linux_printf

FFT, IFFT

Быстрое преобразование Фурье. Обратное преобразование Фурье.



Содержание

Библиотека

Signals

Описание

Блок FFT вычисляет быстрое преобразование Фурье (БПФ) входного сигнала.

Блок IFFT вычисляет обратное преобразование Фурье входного сигнала.

Поддерживаемые типы данных

Блоки FFT и IFFT поддерживают 32-битные знаковые типы данных.

Поддерживаемые длины БПФ: 256, 512 и 1024 элементов.

Параметры и диалоговое окно

Блок не содержит дополнительных параметров.

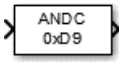
Примеры

Использование блока FFT: nmdemo_blocks_fft_top

Использование блока IFFT: nmdemo_blocks_ifft_top

ANDC

Функция логического "И" между вектором и константой



Содержание

Библиотека

Bitwise

Описание

Блок ANDC вычисляет логическое "И" между вектором и константой.

Поддерживаемые типы данных

Блок ANDC поддерживает 8, 16, 32 и 64-битные беззнаковые типы данных.

Длина (т.е. количество элементов) входного вектора в битах должна быть кратна 64 битам.

Параметры и диалоговое окно

Блок содержит следующие параметры:

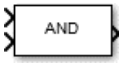
- **Битовая маска** - константа, представляющая собой битовую маску, которая применяется ко входному вектору.

Примеры

Использование блока ANDC: nmdemo_blocks_andc_top

AND

Функция логического "И" между двумя векторами



Содержание

Библиотека

Bitwise

Описание

Блок AND вычисляет логическое "И" между двумя векторами.

Поддерживаемые типы данных

Блок AND поддерживает 8, 16, 32 и 64-битные беззнаковые типы данных.

Длина (т.е. количество элементов) входного вектора в битах должна быть кратна 64 битам.

Параметры и диалоговое окно

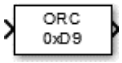
Блок не содержит дополнительных параметров.

Примеры

Использование блока AND: nmdemo_blocks_and_top

ORC

Функция логического "ИЛИ" между вектором и константой



Содержание

Библиотека

Bitwise

Описание

Блок ORC вычисляет логическое "ИЛИ" между вектором и константой.

Поддерживаемые типы данных

Блок ORC поддерживает 8, 16, 32 и 64-битные беззнаковые типы данных.

Длина (т.е. количество элементов) входного вектора в битах должна быть кратна 64 битам.

Параметры и диалоговое окно

Блок содержит следующие параметры:

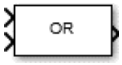
- **Битовая маска** - константа, представляющая собой битовую маску, которая применяется ко входному вектору.

Примеры

Использование блока ORC: nmdemo_blocks_orc_top

OR

Функция логического "ИЛИ" между двумя векторами



Содержание

Библиотека

Bitwise

Описание

Блок OR вычисляет логическое "ИЛИ" между двумя векторами.

Поддерживаемые типы данных

Блок OR поддерживает 8, 16, 32 и 64-битные беззнаковые типы данных.

Длина (т.е. количество элементов) входного вектора в битах должна быть кратна 64 битам.

Параметры и диалоговое окно

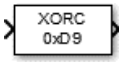
Блок не содержит дополнительных параметров.

Примеры

Использование блока OR: nmdemo_blocks_or_top

XORC

Функция логического "Исключающего ИЛИ" между вектором и константой



Содержание

Библиотека

Bitwise

Описание

Блок XORC вычисляет логическое "Исключающее ИЛИ" между вектором и константой.

Поддерживаемые типы данных

Блок XORC поддерживает 8, 16, 32 и 64-битные беззнаковые типы данных.

Длина (т.е. количество элементов) входного вектора в битах должна быть кратна 64 битам.

Параметры и диалоговое окно

Блок содержит следующие параметры:

- **Битовая маска** - константа, представляющая собой битовую маску, которая применяется ко входному вектору.

Примеры

Использование блока XORC: nmdemo_blocks_xorc_top

XOR

Функция логического "Исключающего ИЛИ" между двумя векторами

Содержание

Библиотека

Bitwise

Описание

Блок XOR вычисляет логическое "Исключающее ИЛИ" между двумя векторами.

Поддерживаемые типы данных

Блок XOR поддерживает 8, 16, 32 и 64-битные беззнаковые типы данных.

Длина (т.е. количество элементов) входного вектора в битах должна быть кратна 64 битам.

Параметры и диалоговое окно

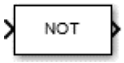
Блок не содержит дополнительных параметров.

Примеры

Использование блока XOR: nmdemo_blocks_xor_top

NOT

Функция логического "НЕ" над элементами вектора



Содержание

Библиотека

Bitwise

Описание

Блок NOT вычисляет логическое "НЕ" над элементами вектора.

Поддерживаемые типы данных

Блок NOT поддерживает 8, 16, 32 и 64-битные беззнаковые типы данных.

Длина (т.е. количество элементов) входного вектора в битах должна быть кратна 64 битам.

Параметры и диалоговое окно

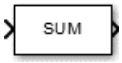
Блок не содержит дополнительных параметров.

Примеры

Использование блока NOT: nmdemo_blocks_not_top

SUM

Возвращает сумму всех элементов вектора



Содержание

Библиотека

Arithmetics

Описание

Блок SUM возвращает сумму всех элементов входного вектора.

Поддерживаемые типы данных

Блок SUM поддерживают 8, 16, 32 и 64-битные знаковые типы данных.

Длина (т.е. количество элементов) входного вектора в битах должна быть кратна 64 битам.

Выходной сигнал имеет тип int32, если входной сигнал типа int8. В остальных случаях выходной сигнал имеет тип int64.

Параметры и диалоговое окно

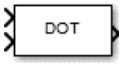
Блок не содержит дополнительных параметров.

Примеры

Использование блока SUM: nmdemo_blocks_sum_top

DOT

Скалярное произведение двух векторов



Содержание

Библиотека

Arithmetics

Описание

Блок DOT вычисляет скалярное произведение двух векторов.

Поддерживаемые типы данных

Блок DOT поддерживает следующие комбинации типов данных для первого и второго входов:

8s8s, 8s16s, 8s32s, 8s64s, 16s16s, 16s32s, 16s64s, 32s32s, 32s64s, 64s64s.

Длина (т.е. количество элементов) как первого, так и второго входа в битах должна быть кратна 64 битам.

Параметры и диалоговое окно

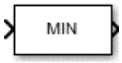
Блок не содержит дополнительных параметров.

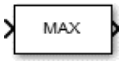
Примеры

Использование блока DOT: nmdemo_blocks_dot_top

MIN, MAX

Поиск значения минимального/максимального элемента вектора





Содержание

Библиотека

Arithmetics

Описание

Блок MIN возвращает значение минимального элемента вектора.

Блок MAX возвращает значение максимального элемента вектора.

Поддерживаемые типы данных

Блоки MIN и MAX поддерживают 8, 16 и 32-битные знаковые типы данных.

Длина (т.е. количество элементов) входного вектора в битах должна быть кратна 64 битам.

Параметры и диалоговое окно

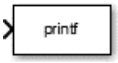
Блок не содержит дополнительных параметров.

Примеры

Использование блоков MIN и MAX: nmdemo_blocks_minmax_top

printf

Печать значения на стандартный вывод Linux



Содержание

Библиотека

Linux Utilities

Описание

Блок printf печатает значение входного сигнала на стандартный вывод Linux.

Поддерживаемые типы данных

Блок printf поддерживает следующие скалярные типы данных:

- int8/uint8
- int16/uint16
- int32/uint32
- int64/uint64
- double
- single

Параметры и диалоговое окно

Блок не содержит дополнительных параметров.

Примеры

Использование блока printf: nmdemo_linux_printf

Примеры и демонстрации

В данном разделе приводятся ссылки на примеры и демонстрации работы с процессором NeuroMatrix при помощи пакета поддержки.

Содержание

Начало работы с NeuroMatrix

Начало работы с NeuroMatrix

Примеры независимого выполнения модели

Примеры независимого выполнения модели

Примеры верификации в режиме Процессор-в-контуре

Примеры верификации в режиме Процессор-в-контуре

Примеры использования Code Replacement

Примеры использования Code Replacement

Примеры использования блоков Simulink

Примеры использования блоков Simulink