The High-Performance Alternative

UEISim User Manual 5.1.0

February 2023 Edition

United Electronic Industries

The High-Performance Alternative

Table of contents

United Electronic Industries

The High-Performance Alternative

The High-Performance Alternative

United Electronic Industries

The High-Performance Alternative

1. Introduction

UEISim turns a PowerDNx Ethernet data acquisition module into a target on which you can run Simulink models and read/write physical I/Os.

The UEISim host software uses the Simulink add-on “Simulink Coder” to convert your Simulink model to C code and then cross-compiles it into an executable that runs directly on the UEISim hardware.

You can access most analog, digital, counter timer, serial, CAN, ARINC-429 and MIL-1553 I/O cards installed on your PowerDNx module from your Simulink model.

You can experiment with control system design, signal processing, data acquisition and similar tasks directly from the Simulink environment using its powerful block library without the need to use any additional tool.

2. Software Installation

The UEISim software runs on a Windows PC.

2.1. Pre-requisites

Before installing the UEISim software make sure that the following software is installed on your computer:

- Matlab
- Matlab Coder
- Simulink
- Simulink Coder or Embedded Coder (for SoloX/ARM CPU only)

2.1.1. Compatibility

UEISIM 2.x:

- runs on PowerPC UEIPAC <= 2.5.x
- compatible with matlab <= r2013b

UEISIM 3.x:

- runs on any PowerPC UEIPAC 3.x and UEIPAC >= 2.6.x
- compatible with Matlab <= R2017a

UEISIM 4.x:

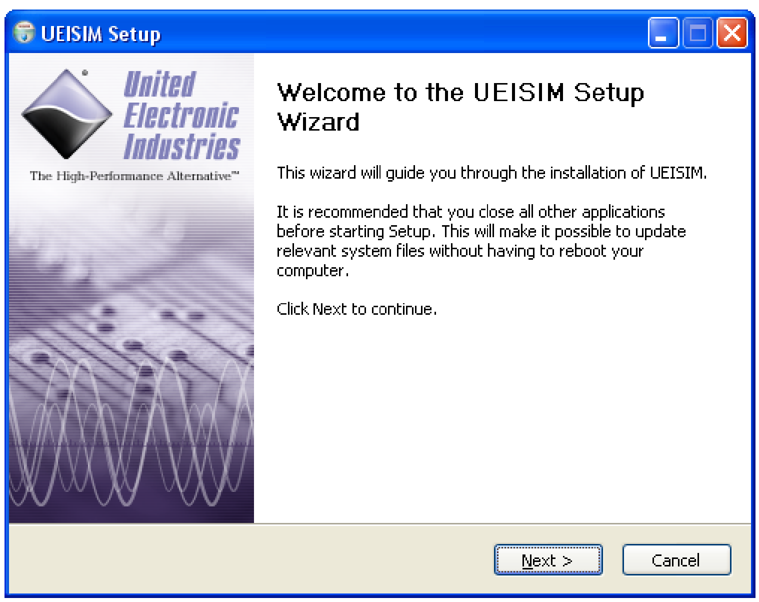
- runs on any PowerPC UEIPAC 4.x
- compatible with Matlab <= R2017b

UEISIM 5.x:

- runs on any PowerPC UEIPAC 4.x and UEIPAC 5.x
- runs on ARM/SoloX UEIPAC 5.x
- runs on ARM64/ZYNQ UEIPAC 5.x
- compatible with Matlab <= R2023a

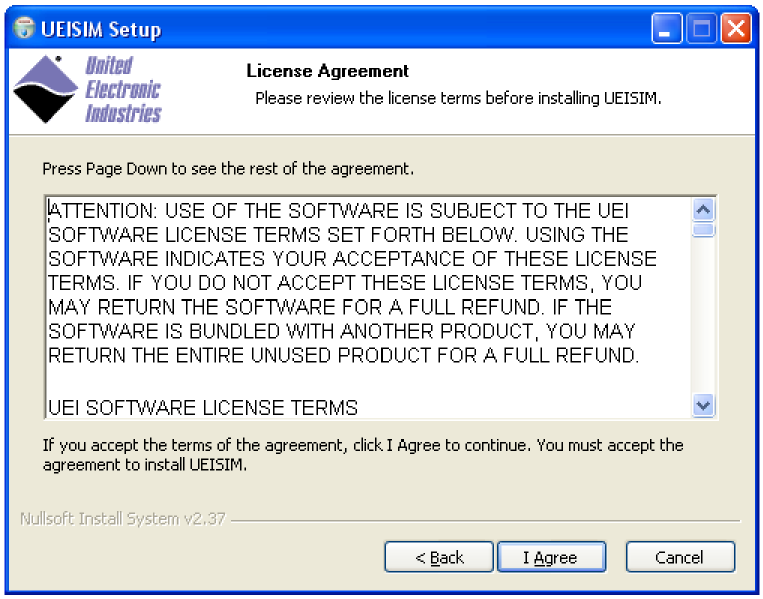
2.2. Install UEISim Software for Windows

Insert the UEISIM Software CDROM in your CD drive. If the installer doesn't start automatically (it depends on whether autorun is enabled or disabled on your PC) run the ueisim_installer.exe program on the CD-ROM.

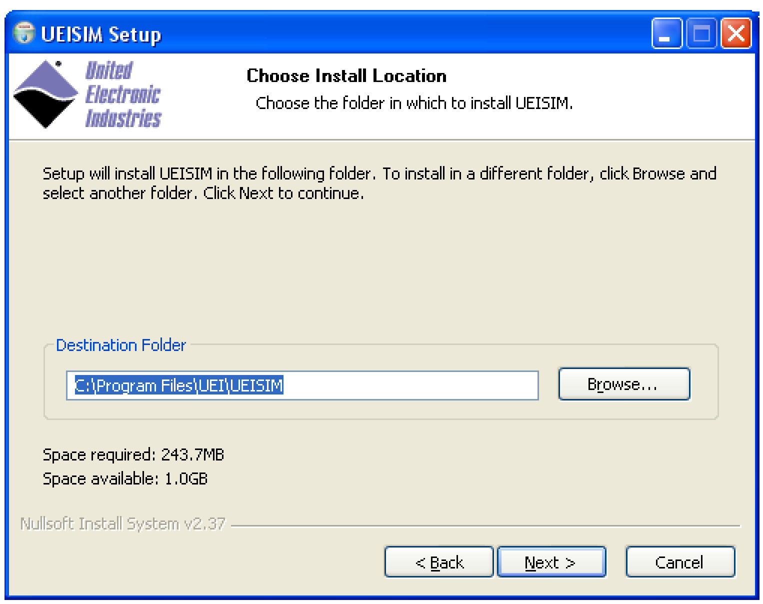


Click on Next to move to the next wizard page.

The High-Performance Alternative

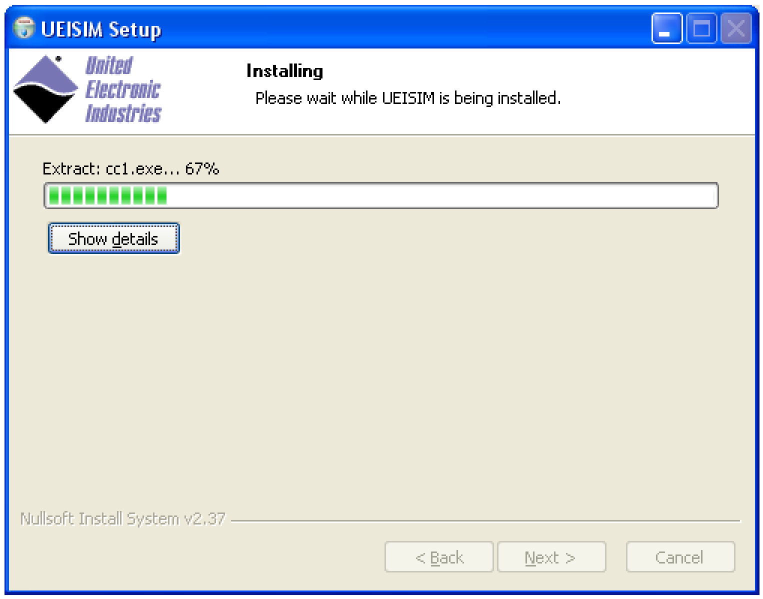


Read the license agreement and click on "I Agree" if you accept the terms of the agreement.

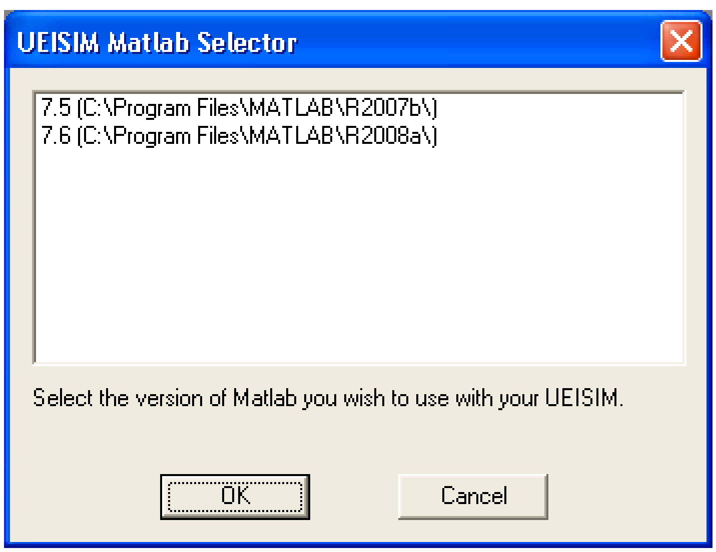


Select the location on your hard drive where you wish to install the software then click "Install". You need to have at least 250MB of free space.

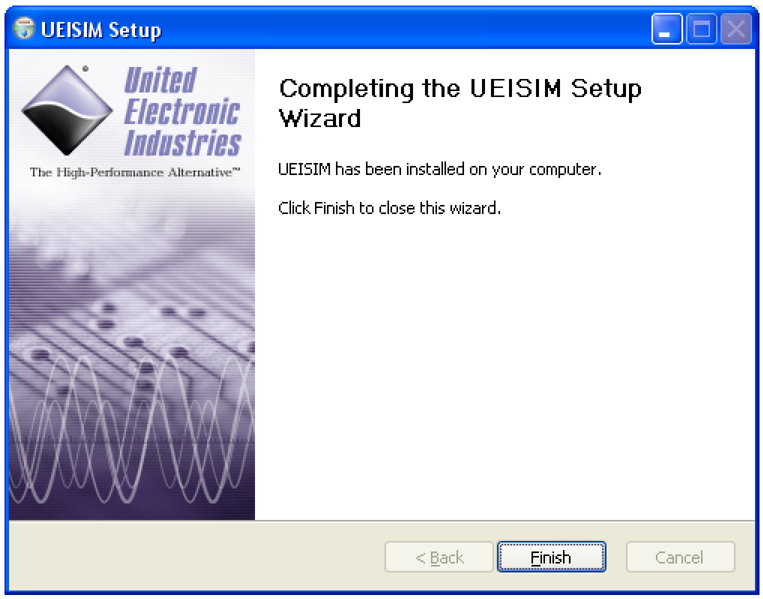
The High-Performance Alternative



Once the files are installed, the "UEISIM Matlab Selector" applet will pop-up, letting you select which version of Matlab/Simulink you wish to use with your UEISIM.



After the installation is done, you can run that applet again if you want to configure another version of Matlab/Simulink to work with your UEISIM.
You can run the "UEISIM Matlab selector" using the shortcut in the Start/Programs/UEI/UEISIM menu.



Once all the files are installed, click on "Finish" to exit the installer.

2.3. Installing UEISim in Matlab's environment

Open Matlab and change directory to the location where UEISIM's support files are installed

```
>> cd('c:\ProgramData\UEI\UEISIM\simulink_libraries')
```

Then run the **ueisim_install** command:

```
>> ueisim_install
```

2.3.1. Compiling s-functions

We no longer ship pre-compiled UEISIM s-functions due to compatibility issues across different versions of matlab.
Type the command **mex –setup** to make sure that you have a C compiler configured for Matlab.
Type the command **makemex** to build the UEISIM s-functions.

2.3.2. Installing support for ARM CPU based UEISIM

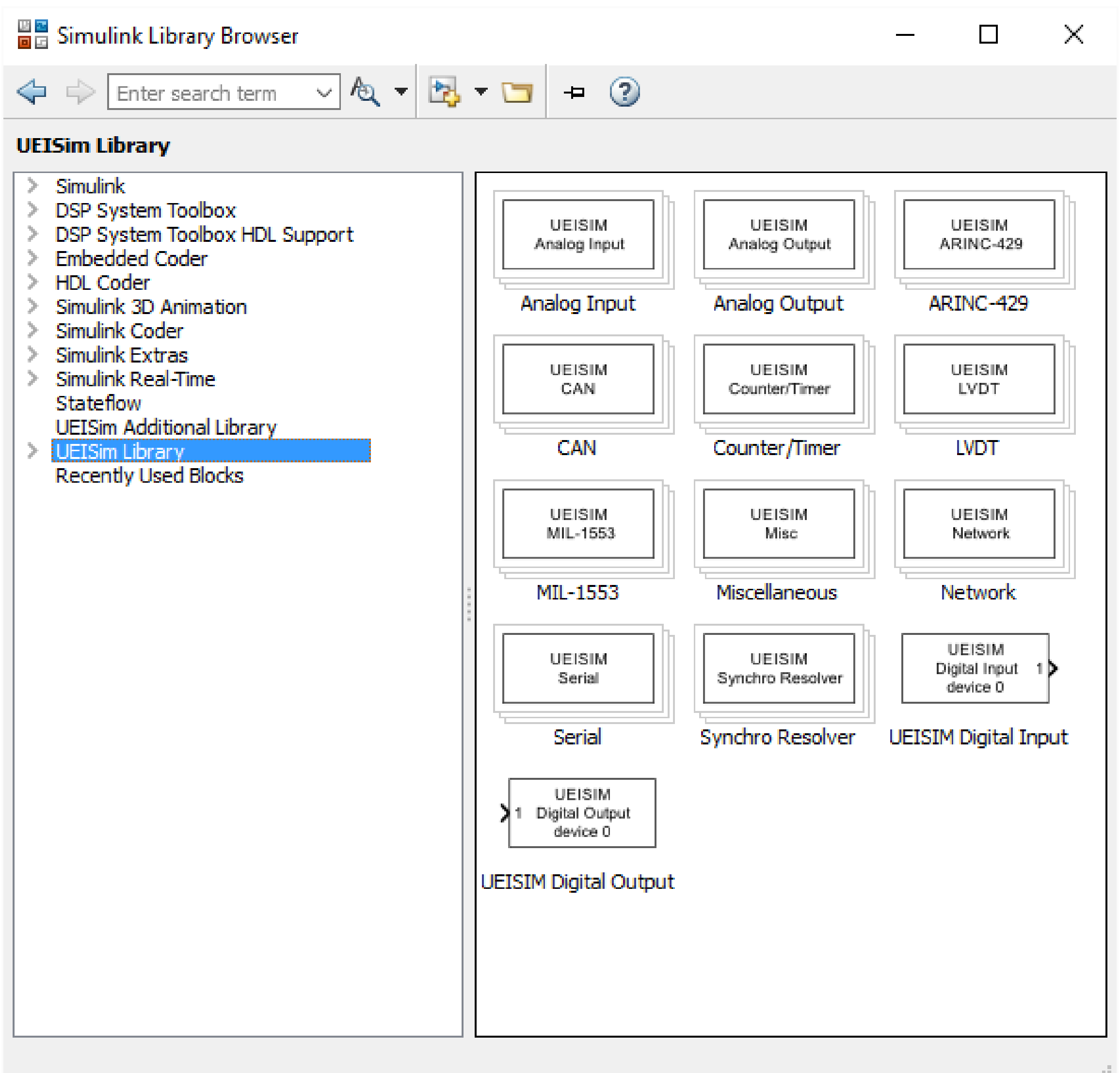
The UESIM installer includes all the tools required to build a model that runs a PowerPC CPU based UEISIMs.

The tools for ARM CPU based UEISIMs are not included. You need to install the UEIPAC ARM SDK and then let Matlab know the location of the SDK with **setpref** command (make sure you use forward slash instead of back slash):

```
>> setpref('ueisim_arm', 'UEIPAC_SDK_PATH',
'e:/cygwin/home/frederic/uei/ueipac-arm-5.0.0_4.11.2.76')
```

2.4. Verifying installation

Open the Simulink library browser and verify that the **UEISIM Library** is present.



United Electronic Industries

The High-Performance Alternative

UEISIM library is shipped as a **.mdl** file to stay compatible with older versions of Matlab.
Starting with r2014a, Matlab will save the UEISIM library as **.slx** (which is the new XML based simulink file format used to save models and libraries).

3. Configuring the UEISim

3.1. PowerPC CPU

The IP address must be configured using the serial port.

3.1.1. Connecting the serial port console

Connect the serial cable to the serial port on the UEISIM cube and the serial port on your PC.

You will need a serial communication program:

- Windows: ucon, MTTTY, putty.
- Linux: minicom or cu (part of the uucp package).

The PowerDNA I/O module uses the serial port settings: 57600 bits/s, 8 data bits, 1 stop bit and no parity. Run your serial terminal program and configure the serial communication settings accordingly.

Connect the DC output of the power supply (24VDC) to the "Power In" connector on the PowerDNA cube and connect the AC input on the power supply to an AC power source.

You should see the following message on your screen:

```
U-Boot 1.1.4 (Jan 10 2006 - 19:20:03)

CPU:   MPC5200 v1.2 at 396 MHz
       Bus 132 MHz, IPB 66 MHz, PCI 33 MHz

Board: UEI PowerDNA MPC5200 Layer
I2C:   85 kHz, ready
DRAM:  128 MB
Reserving 349k for U-Boot at: 07fa8000
FLASH:  4 MB
In:    serial
Out:   serial
Err:   serial
Net:   FEC ETHERNET
```

The High-Performance Alternative

```
Type "run flash_nfs" to mount root filesystem over NFS

Hit any key to stop autoboot:  5

## Booting image at ffc10000 ...
   Image Name:   Linux-2.6.16.1
   Created:      2006-11-10  16:07:06 UTC
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    917636 Bytes = 896.1 kB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
   Uncompressing Kernel Image ... OK
id mach(): done
...
< lots of kernel messages >
...
BusyBox v1.2.2 (2006.11.03-19:16+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

~ #
```

You can now navigate the file system and enter standard Linux commands such as ls, ps, cd…

3.1.2. Configuring the IP address

Your UEISIM cube is configured at the factory with the IP address 192.168.100.2 to be part of a private network.

You can change the IP address for the current session using the command:

```
setip <new IP address>
```

3.1.3. File system

The UEISIM file system contains the libraries, executables and configuration files needed to make the system functional.

By default, the file system is stored on the SD card inserted on the front panel of the UEISIM.

The file system can alternatively be located in a RAM drive loaded from the FLASH memory or loaded from a remote server using the NFS protocol.

The standard UEISIM file system is read/write to ease the configuration and allow uploading of model files during the development phase.

The High-Performance Alternative

Once a model is stable, it is recommended to convert the file system to read-only mode to render the UEISIM file system resilient against un-scheduled shutdowns.

3.1.3.1. Booting the SD card with system partition read-only

The procedure below converts the standard UEISIM file system to a read only one.

1. Edit /etc/fstab as below to mount a RAM disk at /var (ram disk maximum size is set to 2MBytes):

```
/dev/sdcard1    /           ext3    defaults,noatime    1   1
none            /proc       proc    defaults            0   0
none            /sys        sysfs   defaults            0   0
none            /dev/pts    devpts  defaults            0   0
tmpfs           /var        tmpfs   defaults,size=2M    0   0
```

2. Create a new script /etc/varsetup.sh with the content below. It setups the folders needed in /var and maps a few writable folders at /tmp, /mnt and /home

```
mkdir /var/tmp
mkdir /var/log
mkdir /var/lib
mkdir /var/lib/misc
mkdir /var/spool
mkdir /var/spool/cron
mkdir /var/spool/cron/crontabs
mkdir /var/run
mkdir /var/lock
mkdir /var/mnt
mkdir /var/home

mount --bind /var/tmp /tmp
mount --bind /var/mnt /mnt
mount --bind /var/home /home
```

3. Edit /etc/inittab as below to execute varsetup.sh

```
# Mount all filesystem listed in /etc/fstab
::sysinit:/bin/mount -a

# Create and mount non-persistent folders
::sysinit:/etc/varsetup.sh

# Configure local network interface
::sysinit:/sbin/ifconfig lo 127.0.0.1 up
```

```
::sysinit:/sbin/route add -net 127.0.0.0 netmask 255.0.0.0 lo

# run rc scripts
::sysinit:/etc/rcS

# Start a shell on the console
ttyS0::respawn:-/bin/sh

# unmount root file system when shutting-down
::shutdown:/bin/umount -a -r
```

4. Create symbolic links to files stored in /etc that need to be kept writeable.

```
ln –s /var/resolv.conf /etc/resolv.conf
ln –s /var/layers.xml /etc/layers.xml
```

5. Connect the console serial port, power-up the UEISIM and press a key to enter U-Boot. Type the following commands to load the root file system read-only:

```
setenv bootargs console=ttyS0,57600 root=62:1 ro
saveenv
reset
```

3.1.3.2. Restoring or creating a new SD card on a Linux PC

Restoring or initializing a new SD card can be done on a Linux PC (real or virtual).

1. Locate the SD card image file rfs-x.y.z.tgz on your UEISIM CDROM as well as the script containing the sequence of commands to partition, format and initialize a new SD card.
2. Connect the SD card via a USB adapter (or directly if your computer has a built-in reader).
3. Type the command ***dmesg*** to find out what device node is associated with the SD card. (Linux kernel outputs messages when it detects a new removable drive)
4. Assuming that /dev/sdb is the SD card device node, type ***./createsdcard.sh /dev/sdb rfs-x.y.z.tgz*** to partition, format and copy files to the card.

3.1.3.3. Restoring the SD card on the UEISIM itself

Boot the UEISIM from a RAM disk instead of the SD card (follow instructions detailed in chapter 3.3.4 below).

1. Set the IP address:
   `setip <IP address of the UEISIM>`

2. Format the SD card:
   ```
   mke2fs -j /dev/sdcard1
   ```
3. Mount the SD card:
   ```
   mount /dev/sdcard1 /mnt
   ```
4. Transfer the root file system image to the UEIPAC from a Linux or Windows PC:
   ```
   scp rfs-x.y.z.tgz root@<IP address of UEISIM>:/mnt
   ```
5. Un-compress the image:
   ```
   gunzip /mnt/rfs-x.y.z.tgz
   tar xvf /mnt/rfs-x.y.z.tar
   mv /mnt/rfs/* /mnt
   sync
   ```

3.1.3.4. Booting from a RAM drive (no SD card needed)

Booting from a RAM disk is faster than any other method. However the RAM disk size is limited to 16Mbytes and any data written to the RAM disk is lost when the system shuts down or reboot.

The RAM disk can only fit in the flash memory of the UEIPAC models based on the 8347 CPU (UEIPAC-1G or UEIPAC-R).

3.1.3.4.1. Customize the RAM drive image

Customizing the RAM drive image is necessary to add your model and tweak the startup script if you wish to start the model automatically.

This can only be done on a Linux PC. You might need to install the uboot mkimage utility.
For example under Ubuntu or Debian:

```
$sudo apt-get install uboot-mkimage
```

1. Extract compressed RAM disk image from uImage file. The following command converts the file **uRamdisk-x.y.z** to **ramdisk.gz**

   ```
   $ dd if=uRamdisk-x.y.z bs=64 skip=1 of=ramdisk.gz
   21876+1 records in
   21876+1 records out
   ```

2. Un-compress RAM disk image

   ```
   $ gunzip -v ramdisk.gz
   ```

The High-Performance Alternative

```
ramdisk.gz:      66.6% -- replaced with ramdisk
```

3. Mount RAM disk image

```
$ mount -o loop -t ext2 ramdisk /mnt
```

Now you can add, remove, or modify files in the `/mnt` directory. Once you are done, you can re-pack the RAM disk into a U-Boot image:

1. Un-mount RAM disk image:

```
$ umount /mnt
```

2. Compress RAM disk image

```
$ gzip -v9 ramdisk
ramdisk:         66.6% -- replaced with ramdisk.gz
```

3. Create new U-Boot image

```
$ mkimage -T ramdisk -C gzip -n 'My UEISIM RAM disk' -d
ramdisk.gz new-uRamdisk-x.y.z
Image Name:   UEIPAC RAM disk
Created:      Wed Apr 11 17:32:41 2012
Image Type:   PowerPC Linux RAMDisk Image (gzip compressed)
Data Size:    2425561 Bytes = 2368.71 kB = 2.31 MB
Load Address: 0x00000000
Entry Point:  0x00000000
```

3.1.3.4.2. Upload RAM drive image to flash

Uploading the RAM disk image must be done from the boot loader command line using the TFTP protocol. Make sure you have a TFTP server running on your workstation.

Follow the steps below to upload the RAM disk to memory and boot from it

1. Connect a serial cable to your UEISIM and start a serial terminal software with communication settings set to 57600,8,N,1
2. Copy the RAM drive image *uRamdisk-x.y.z* file to the root directory of your TFTP server

United Electronic Industries

The High-Performance Alternative

3. Power-up the UEISIM and press any key to enable the boot loader command line. You should see the prompt '=>'
4. Configure the UEISIM's IP address
   `=> setenv ipaddr <IP address of the UEISIM>`
5. Configure U-Boot to use your host PC as TFTP server:
   `=> setenv serverip <IP address of your host PC>`
6. Upload RAM disk:
   `=> tftp 400000 uRamdisk-x.y.z`
7. Copy the RAM disk to flash:
   `=> erase fe200000 fe7fffff`
   `=> cp.b 400000 fe200000 ${filesize}`
8. Update **bootargs** variable to tell the kernel that its root file system is a RAM disk:
   `=> setenv bootargs console=ttyS0,57600 root=/dev/ram0 rw`
9. Change boot command to unpack the RAM disk in memory before starting the kernel:
   `=> setenv bootcmd bootm fe000000 fe200000`
10. Save environment to make those changes permanent and reset:
   `=> saveenv`
   `=> reset`

3.2. ARM CPU

3.2.1. Configuring the IP address

UEIPAC SoloX is configured at the factory with a static IP address to be part of a private network. NIC1 defaults to 192.168.100.2.
You can change the IP address of NIC1 and NIC2 with the **`ifconfig`** command or make a persistent change by updating the **`<*.network>`** files in **`/etc/systemd/network`**.

The following syntax shows how to update with ifconfig:

```
ifconfig eth0 <NIC1 IP address>
ifconfig eth1 <NIC2 IP address
```

Note that you shouldn't configure both Ethernet ports to be on the same subnet (for example, setting eth0:192.168.100.2 and eth1:192.168.100.3). This will cause errors with the kernel packet routing.

To make changes persistent on a reboot, do the following to configure eth0 (NIC1):

The High-Performance Alternative

1. Edit **/etc/systemd/network/20-wired.network**
   It will have content similar to the following:
   ```
   [Match]
   Name=eth0

   [Network]
   Address=192.168.100.2/24
   ```

2. Change **Address** to your IP address of choice and save.

3. Enter the following to activate the changes immediately:
   ```
   #systemctl restart systemd-networkd
   ```

To configure eth1 (NIC2), do the above steps, but instead edit 21-wired.network, which will list “Name=eth1”.

3.2.2. Configure FTP server

The UEIPAC comes with the vsftpd FTP server. The server is disabled by default. To check whether the service is enabled, enter (the following shows it’s disabled):

```
~# systemctl status vsftpd
● vsftpd.service - Vsftpd ftp daemon
 Loaded: loaded (/lib/systemd/system/vsftpd.service; disabled; vendor
preset:
```

To start the service during your current session, enter the following:

```
~# systemctl start vsftpd
```

To enable the service, so it will always be started when you powerup:

```
~# systemctl enable vsftpd
Created symlink
/etc/systemd/system/multiuser.target.wants/vsftpd.service →
/lib/systemd/system/vsftpd.service.
```

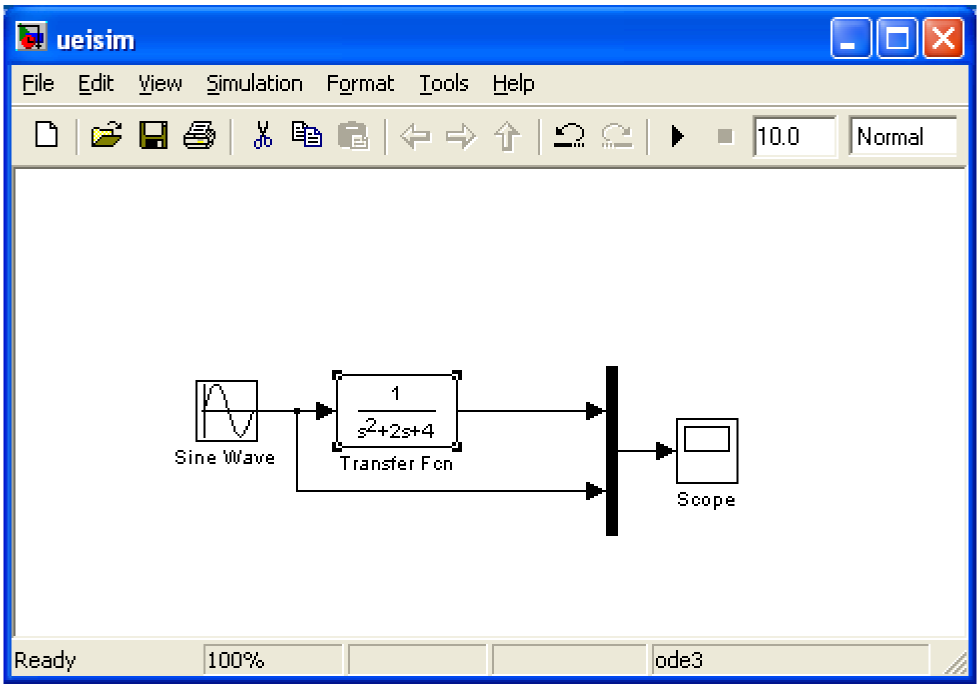
Allow root login. Edit **/etc/vsftpd.ftpusers** and **/etc/vsftpd.user_list** and comment out the “root” name in both files, (i.e. put a # in front of the name ‘root’: #root).
After that, you must restart the vsftpd service by entering the following:

```
~# systemctl restart vsftpd
```

4. Using UEISim add-on from MATLAB/Simulink

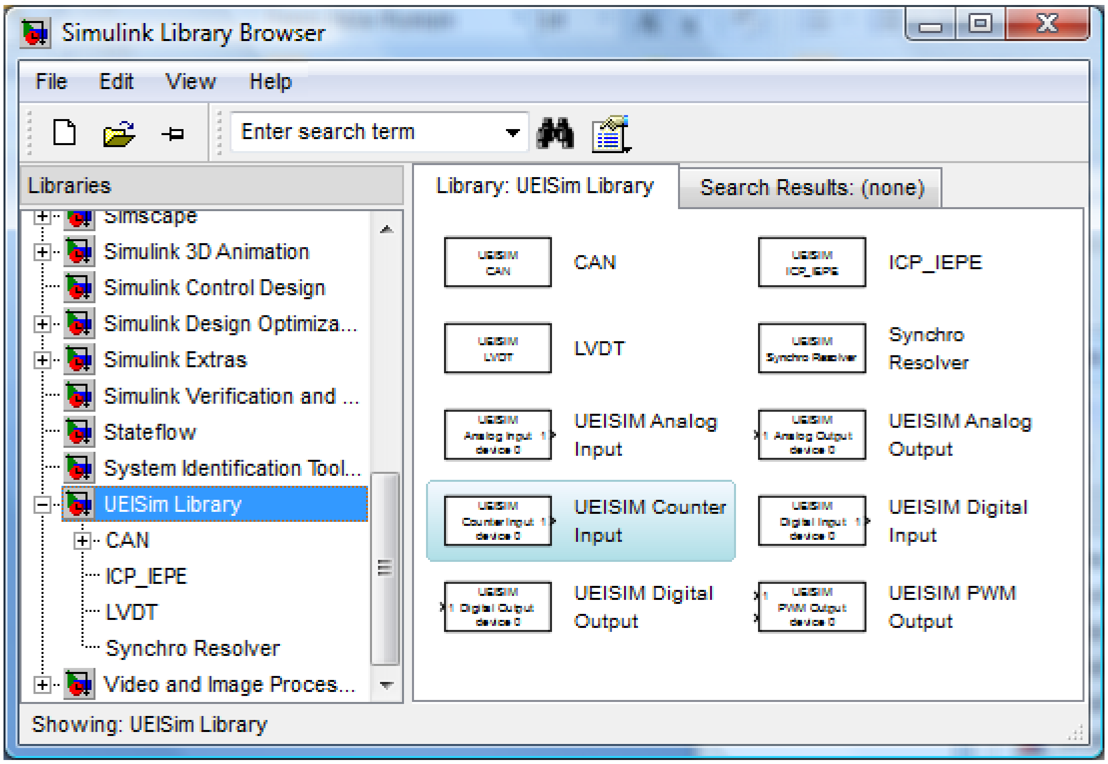
4.1. Convert your model

Let's start with an existing model that process some input signal and view the output on a scope.

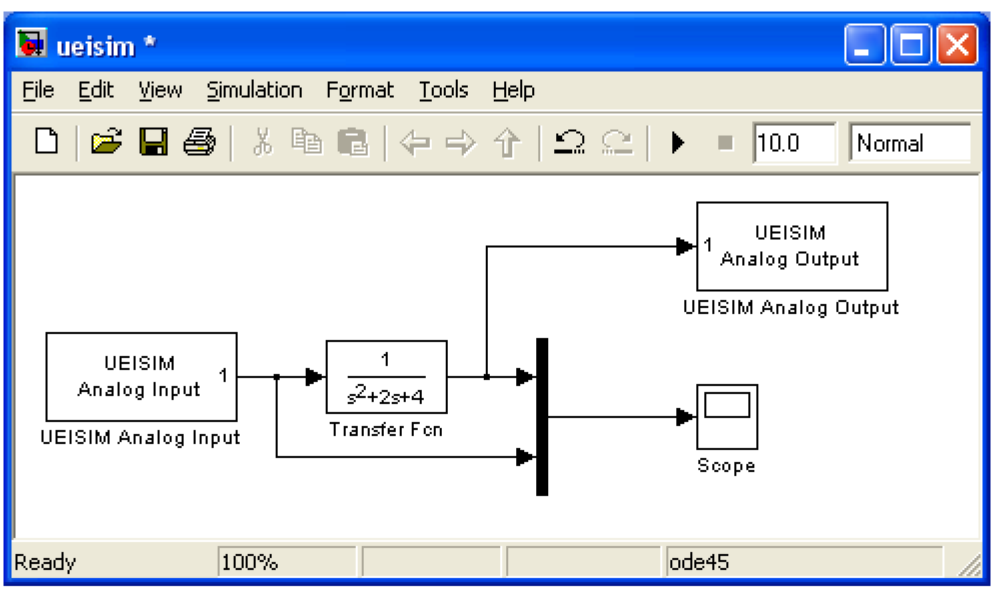


In order to test our model with a real signal, let's use the UEISim analog input and output blocks.

The UEISim I/O blocks are located in the Simulink library:



Replace the input sine wave block with an Analog Input block and add an Analog Output block to generate the result as well as display it on the scope.



United Electronic Industries
The High-Performance Alternative

Double-click on the Analog Input and Output blocks to configure the parameters (see chapter 5 for details on the parameters for each of the UEISIM block).

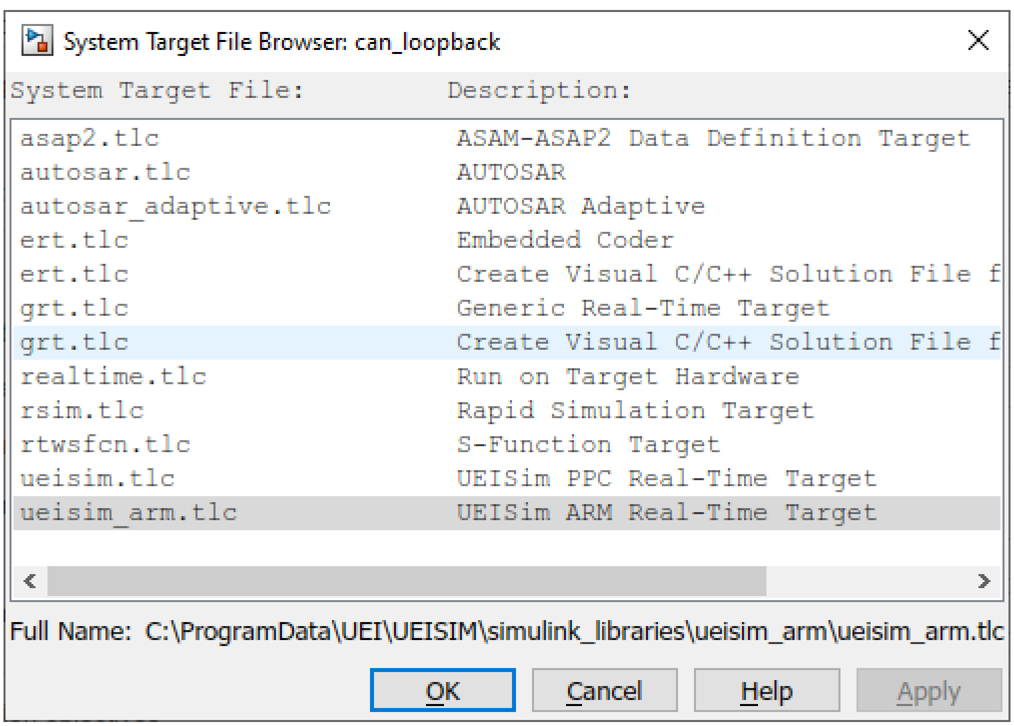
4.2. Create an executable from the model

Select the menu option "Simulation/Configuration Parameters…"

Click on the "Solver" option on the left pane and make sure the solver type is set to "Fixed-step".

If you are running a Matlab version earlier than R2012a, Select the **Real-Time Workshop** option then click on **Browse…** to change the system target file.

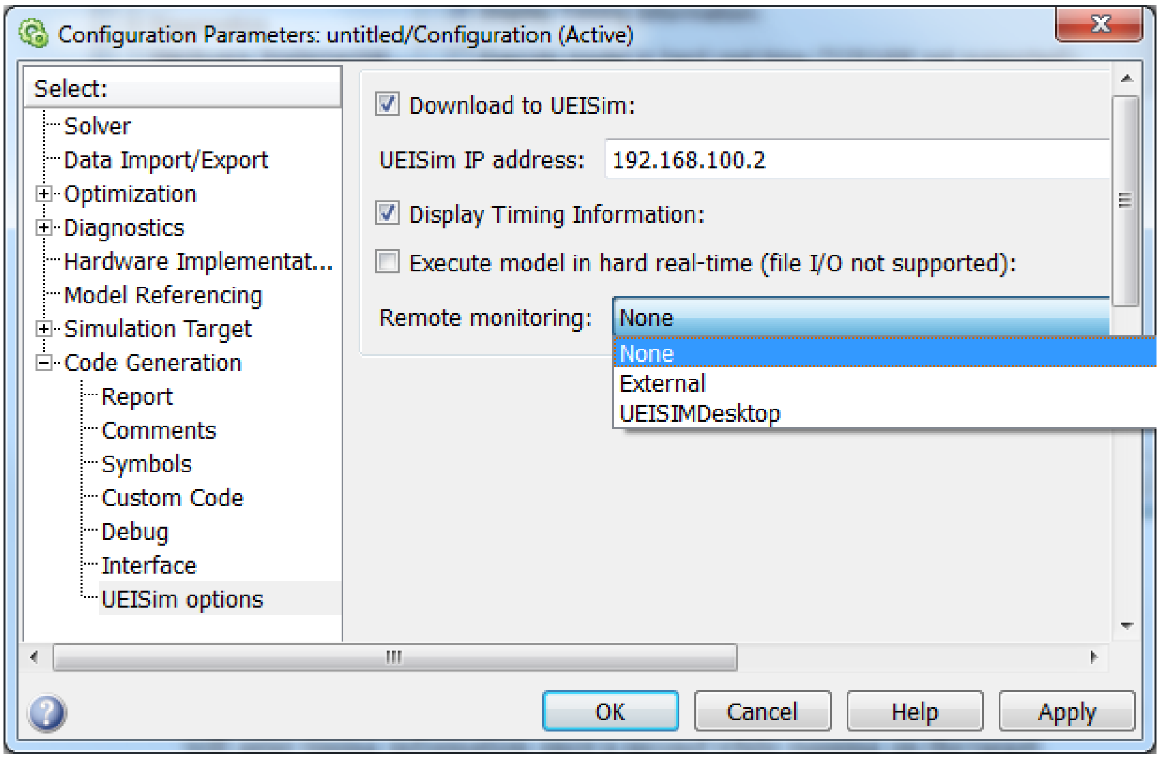
For Matlab R2012a and later, select the **Code Generation** option then on **Browse…** to change the system target file.



Select the **UEISim PPC Real-Time Target** or **UEISim ARM Real-Time Target** and click OK.

The High-Performance Alternative

Select **UEISim options**



- **Download to UEISim**: Check this option to automatically download the simulation executable to the UEISim.
- **UEISim IP address**: Enter the IP address of the UEISim.
- **Display Timing Information**: Turn on timing information output. Your model will print timing information once a second while running on the target.
- **Execute model in hard real-time**: when enabled the model is executed in the context of a Xenomai real-time task. When disabled the model is executed in the context of a high priority Linux process. You cannot use any block doing file I/O (such as “To File”) in hard real-time mode.
- **Remote monitoring**: Select the type of remote monitoring. ‘None’: no monitoring, ‘External’: Use Simulink in external mode, ‘UEISIMDesktop’: Use UEISIMDesktop protocol (more details in section 3.5)

Click on **Real-Time Workshop** (or on **Code Generation**) again and then on **Build**. This will start the code generation and build process.

You should see an output similar to the following in MATLAB's command window:

```
### Generating code into build directory: C:\test\ueisim_ueipac_rtw
### Invoking Target Language Compiler on ueisim.rtw
      tlc
      -r
      C:\test\ueisim.rtw
      e:\uei_svn\software\powerdna\3.3.x\UEIPAC\Simulink_rtw\ueisim.tlc
      -OC:\test\ueisim_ueipac_rtw
      -Ie:\uei_svn\software\powerdna\3.3.x\UEIPAC\Simulink_rtw
      -IC:\test\ueisim_ueipac_rtw\tlc
      -IC:\Program Files\MATLAB\R2007b\rtw\c\tlc\mw
      -IC:\Program Files\MATLAB\R2007b\rtw\c\tlc\lib
      -IC:\Program Files\MATLAB\R2007b\rtw\c\tlc\blocks
      -IC:\Program Files\MATLAB\R2007b\rtw\c\tlc\fixpt
      -IC:\Program Files\MATLAB\R2007b\stateflow\c\tlc
      -aEnforceIntegerDowncast=1
      -aFoldNonRolledExpr=1
      -aInlineInvariantSignals=0
      -aInlineParameters=0
      -aLocalBlockOutputs=1
      -aRollThreshold=5
      -aZeroInternalMemoryAtStartup=1
      -aZeroExternalMemoryAtStartup=1
      -aInitFltsAndDblsToZero=1
      -aGenerateReport=0
      -aGenCodeOnly=0
      -aRTWVerbose=1
      -aIncludeHyperlinkInReport=0
      -aLaunchReport=0
      -aGenerateTraceInfo=0
      -aForceParamTrailComments=0
      -aGenerateComments=1
      -aIgnoreCustomStorageClasses=1
      -aIncHierarchyInIds=0
      -aMaxRTWIdLen=31
      -aShowEliminatedStatements=0
      -aIncDataTypeInIds=0
      -aInsertBlockDesc=0
      -aSimulinkBlockComments=1
      -aInlinedPrmAccess="Literals"
      -aTargetFcnLib="ansi_tfl_table_tmw.mat"
      -aIsPILTarget=0
      -aLogVarNameModifier="rt_"
      -aGenerateFullHeader=1
      -aExtMode=0
      -aExtModeStaticAlloc=0
      -aExtModeTesting=0
```

The High-Performance Alternative

```
        -aExtModeStaticAllocSize=1000000
        -aExtModeTransport=0
        -aRTWCAPISignals=0
        -aRTWCAPIParams=0
        -aGenerateASAP2=0
        -aDownloadToUEIPAC=1
        -aUEIPACIPAddress="192.168.15.200"
        -aGenerateTraceInfo=0
        -p10000


### Loading TLC function libraries

.....
### Initial pass through model to cache user defined code
.
### Caching model source code
.............................
### Writing header file ueisim_types.h
.
### Writing header file ueisim.h
### Writing source file ueisim.c
### Writing header file ueisim_private.h
.
### Writing header file rtmodel.h
### Writing source file ueisim_data.c
### Writing header file rt_nonfinite.h
### Writing source file rt_nonfinite.c
.
### TLC code generation complete.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
### Evaluating PostCodeGenCommand specified in the model
Adding e:\uei_svn\software\powerdna\3326E1~1.X\UEIPAC\SIMULI~1 to
source and include paths
.
### Processing Template Makefile:
e:\uei_svn\software\powerdna\3.3.x\UEIPAC\Simulink_rtw\ueipac.tmf
### ueisim.mk which is generated from
e:\uei_svn\software\powerdna\3.3.x\UEIPAC\Simulink_rtw\ueipac.tmf is up
to date
### Building ueisim: .\ueisim.bat

<lots of compiler output>

Created executable: ueisim
Downloading ../ueisim to UEIPAC at 192.168.15.200
Downloaded: ueisim
>>
```

The High-Performance Alternative

The simulation executable is now ready to be executed.
The executable is located in the /tmp directory on PowerPC based UEISIMs or in /home/root on ARM based UEISIMs.

4.3. Running the simulation

4.3.1. From the command line

Log on the UEISim using the serial port console, Telnet or SSH and run the simulation executable in the **/tmp** folder:

```
/tmp # ./ueisim
StepSize: 0.010000 s
Model: 201 Option: 100
Model: 308 Option: 1
Model: 207 Option: 1
Model: 205 Option: 1
Model: 404 Option: 1

** starting the model **
```

4.3.2. Using the UEISIM desktop API

UEISim Software comes with an API to remotely control the simulation. The API can be used from C, C++, C# or VB.NET.

The UEISIM desktop API can start/stop a simulation, read signal and parameter values as well as timing statistics. It can also write tunable parameter values.

The API is documented in more details in the manual **UEISIM Desktop User Manual**.

4.4. Tuning step size and sample time

The sample time parameter in the various I/O blocks determines the maximum amount of work your model can perform within one step.

To get an idea of your model "load", you can enable the option "Display Timing Information" in the "UEISIm Options" configuration panel.

The model will display timing information once a second while running:

```
**May run forever. Model stop time set to infinity.**

Step completed its work in 0.000085 s, remains 0.000915 s
Min. TET=0.000083, max. TET=0.000148, avg. TET=0.000085
Simulated time 1.000000 s, real time 0.999156 s
```

The High-Performance Alternative

```
Step completed its work in 0.000085 s, remains 0.000915 s
Min. TET=0.000082, max. TET=0.000148, avg. TET=0.000085
Simulated time 2.001000 s, real time 2.000157 s

Step completed its work in 0.000091 s, remains 0.000909 s
Min. TET=0.000082, max. TET=0.000148, avg. TET=0.000085
Simulated time 3.002000 s, real time 3.001146 s

Step completed its work in 0.000085 s, remains 0.000915 s
Min. TET=0.000082, max. TET=0.000148, avg. TET=0.000085
Simulated time 4.003000 s, real time 4.002159 s
^C
Executed 4047 iterations in 4.047741 s (999.816935 updates per sec.)
```

In the output above, the model is running at 1kHz, each step is taking 85us to do its work out of an allocated 1000us.

The TET values are minimum, maximum and average **task execution time**.

Simulated time is the expected simulation time. **Real time** is the measured simulation time while running on the target.

If **real time** exceeds **simulated time**, you are doing too much work in your model. The CPU can't execute the task within the allocated time.

4.5. Remote monitoring

4.5.1. Remote monitoring with UEISIM desktop

UEISIM desktop protocol allows you to remotely monitor a simulation running on the UEISim. You can monitor the simulation using a generic application, a web browser or a custom application developed in C/C++, C# or VB.NET.

Select the menu option **Simulation/Configuration Parameters**….

Click on the **option Code Generation** then on **UEISim options**.

Verify that the UEISIM IP address is correct

Change the Remote monitoring setting to **UEISimDesktop**.

Click on OK and re-build the model.

The High-Performance Alternative

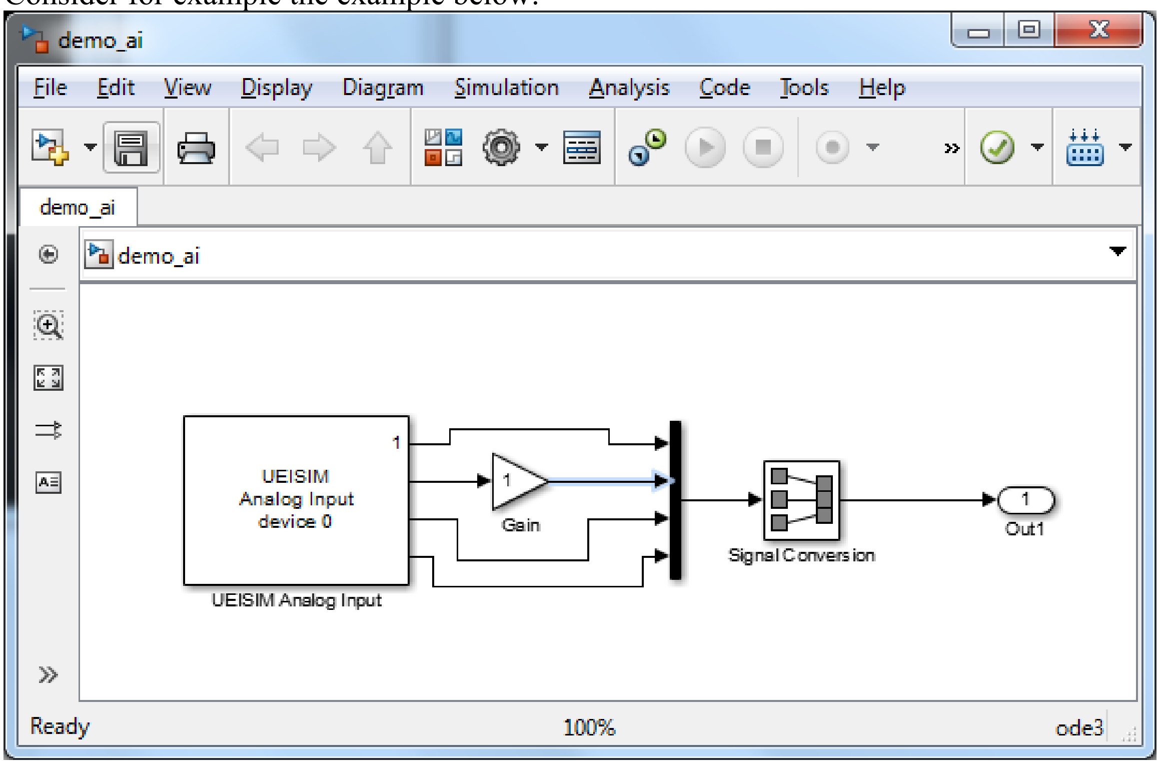
Logon the UEISim and start the simulation. UEISimDesktop protocol uses the TCP/IP port 2345 by default. You can change the port with the command line option '-port'.

```
/tmp # ./ueisim -port 1234
```

You can now run the generic client (or a client you built using the UEISIM target API)

Use the following URL in the generic client "tcp://192.168.100.2:1234"

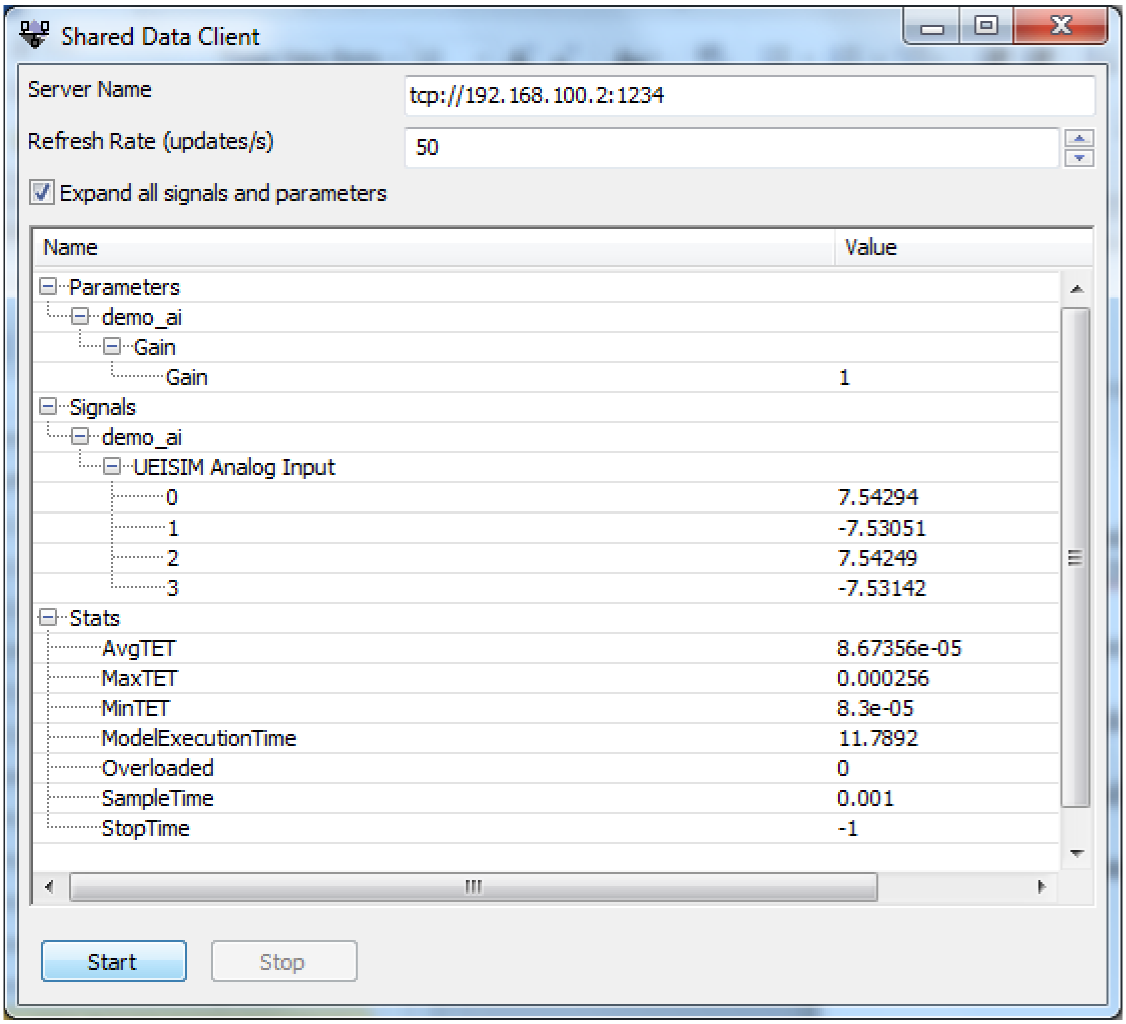
Consider for example the example below:



Here is what this model signals and parameters look like in the generic client:

United Electronic Industries

The High-Performance Alternative



The signals available are the 4 outputs of **UEISIM Analog Input**.

The only tunable parameter is the **Gain** parameter of the **Gain** block (You can not change any of the UEISIM block parameters during simulation)

The UEISIM desktop protocol also makes timing statistics available:

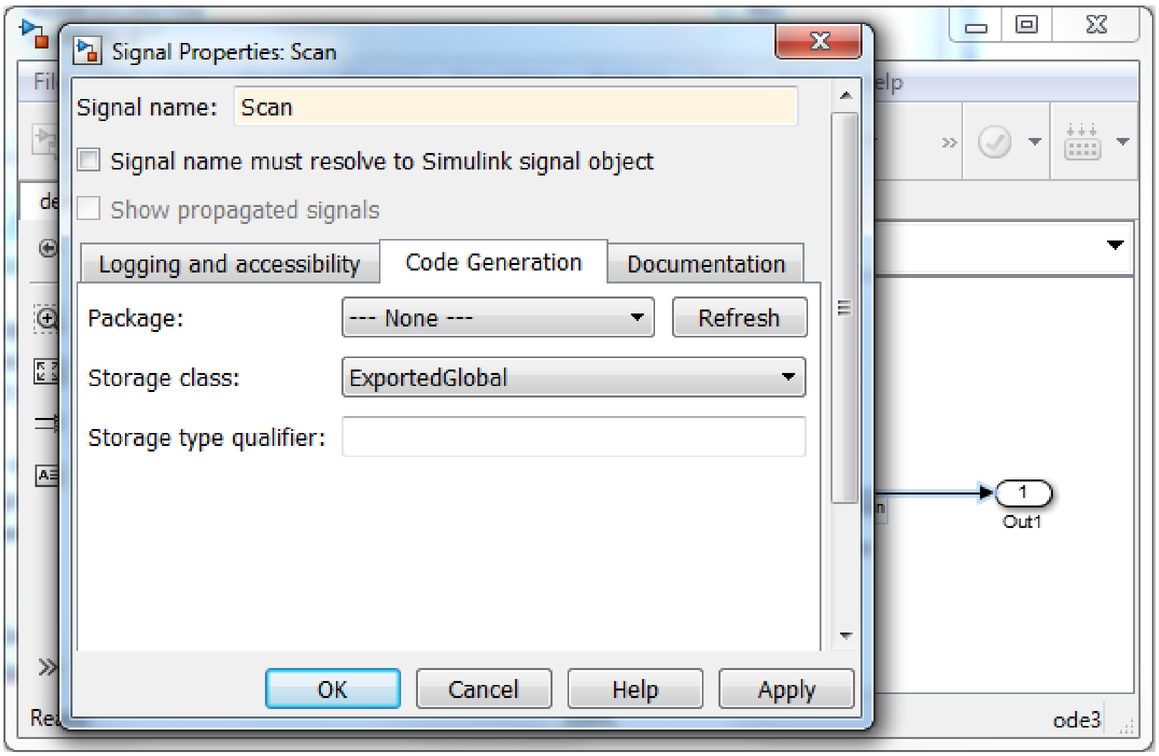
- AvgTET: average task execution in seconds
- MaxTET: maximum task execution time in seconds
- MinTET: minimum task execution time in seconds
- ModelExecutionTime: Number of seconds since simulation started
- Overloaded: 1 is max task execution time ever becomes greater than the sample time. 0 otherwise

United Electronic Industries

The High-Performance Alternative

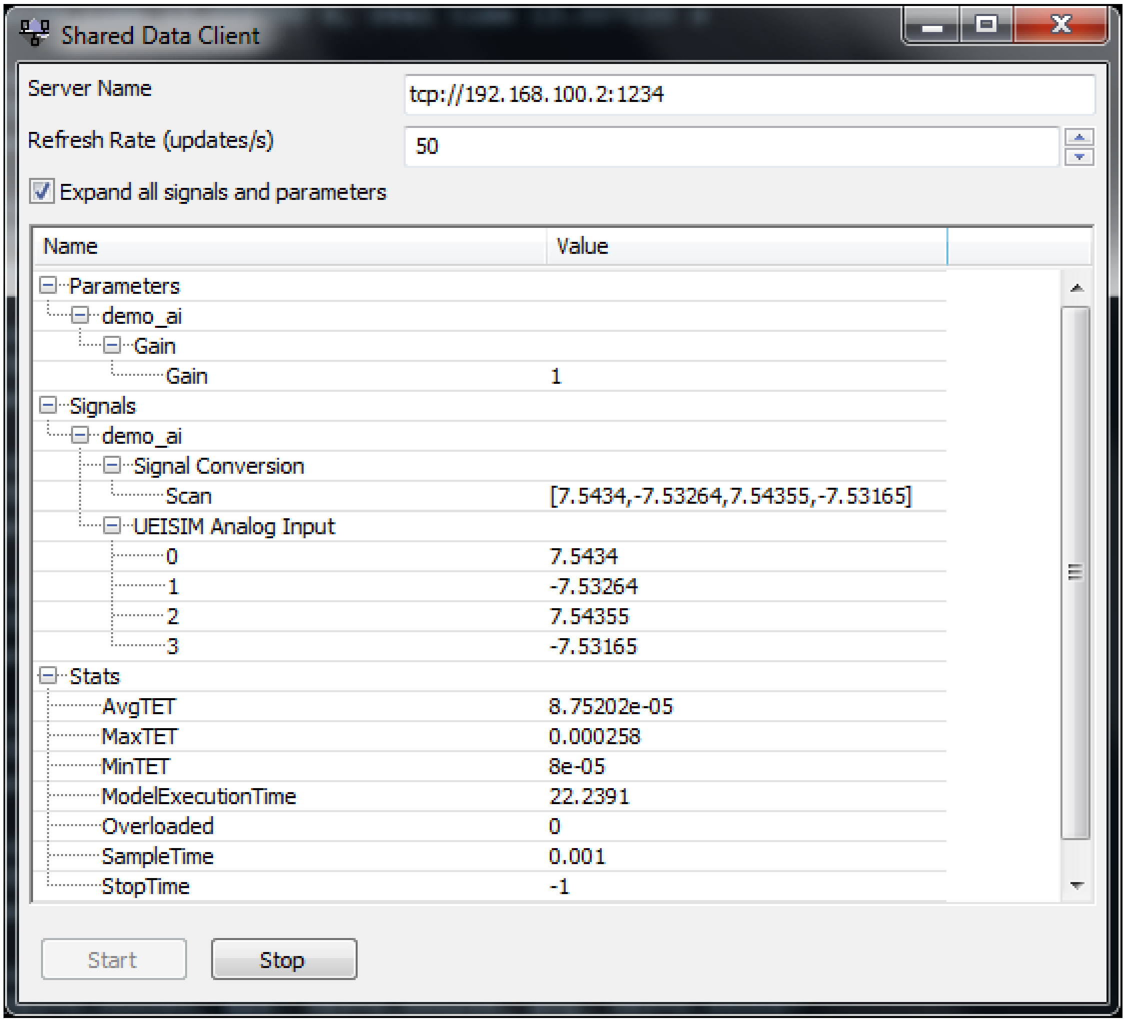
- SampleTime: The simulation base sample time in seconds
- StopTime: The simulation duration in seconds (-1 for inifinite)

Other signals must be exported to be able to monitor them remotely. For example to export the signal out of the **Signal Conversion** block, right-click on the signal wire and select **Properties**

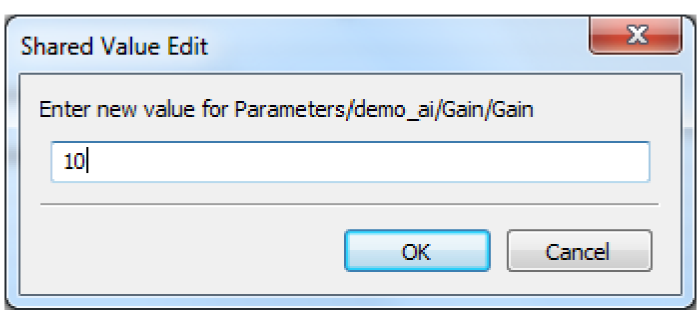


Give the signal a name ("Scan") and click on the **Code Generation** tab. Set **Storage class** to **ExportedGlobal** to export the signal.

After the model is rebuilt and executed the client show the new **Scan** signal (which is a vector of 4 values in this case)

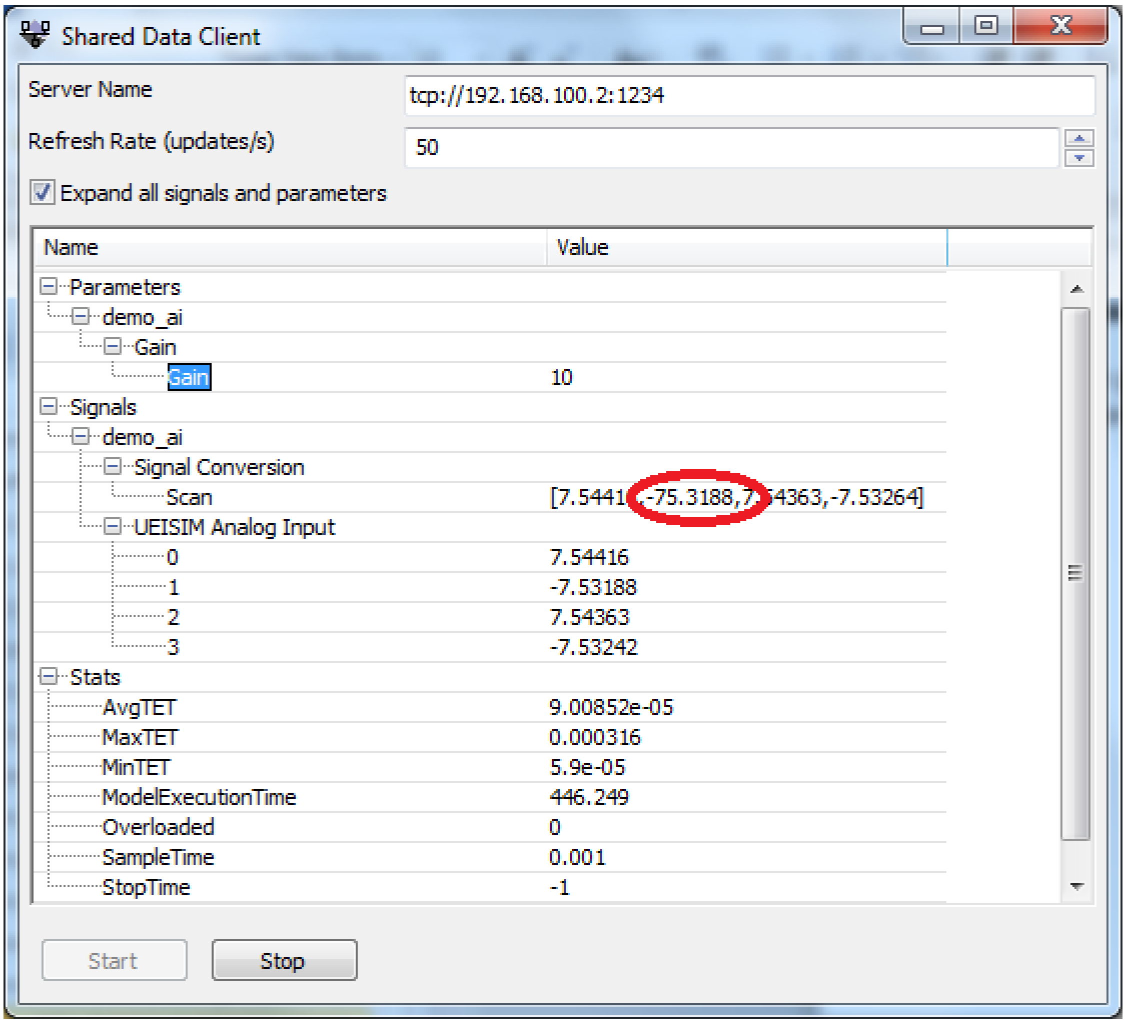


The generic client can change tunable parameters. Double Click on **Gain** and set a new value:



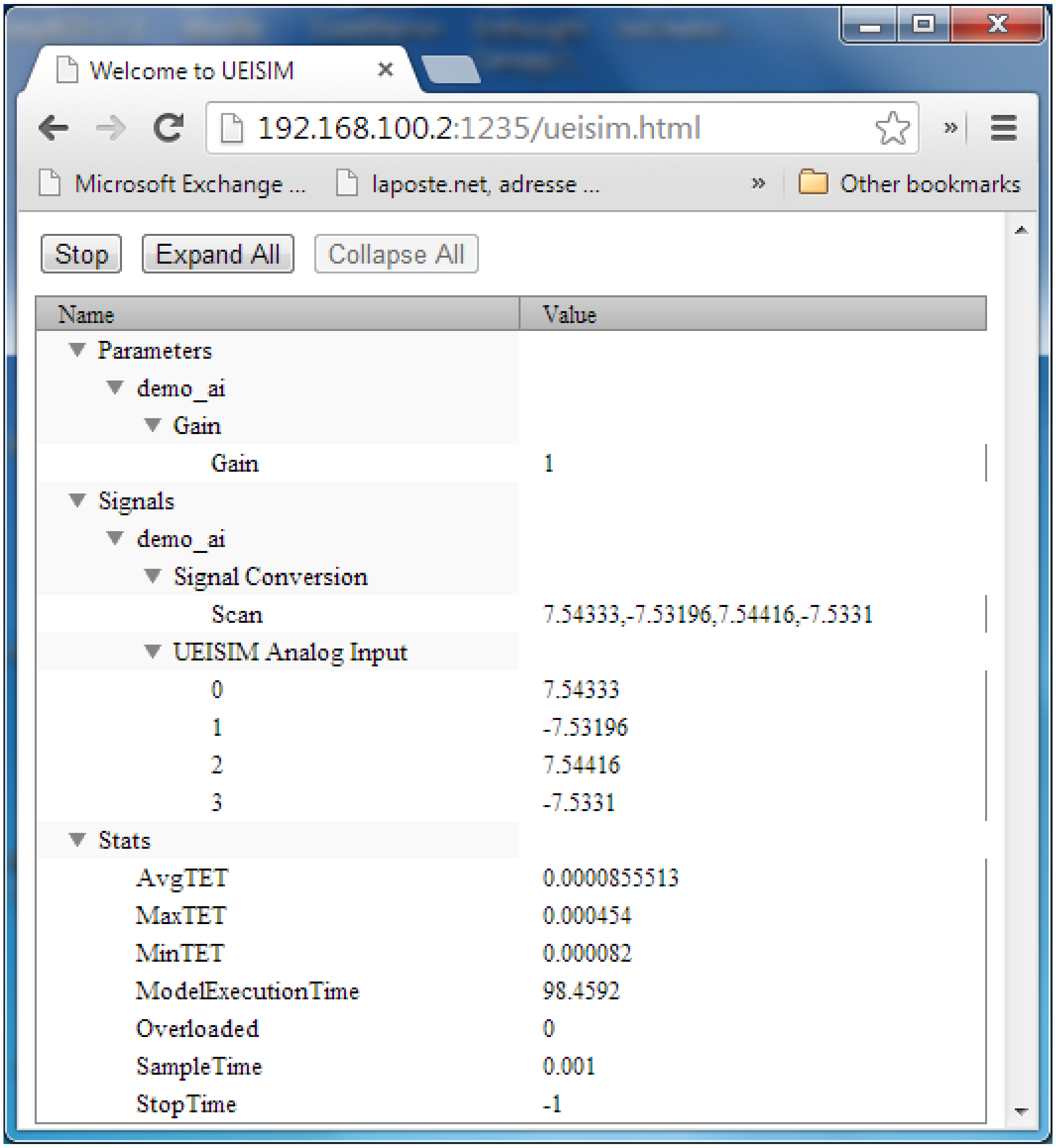
The High-Performance Alternative

We can immediately see the effect of changing the gain, the second channel out of the **UEISIM Analog Input** block is now multiplied by 10.



The simulation can also be monitored from a web browser. The built-in web server uses the client's port incremented by 1.

For example if you start the simulation with **/tmp/ueisim_demo –port 1234**, you can monitor the parameter and signals from the URL **http://192.168.100.2:1235/ueisim.html**



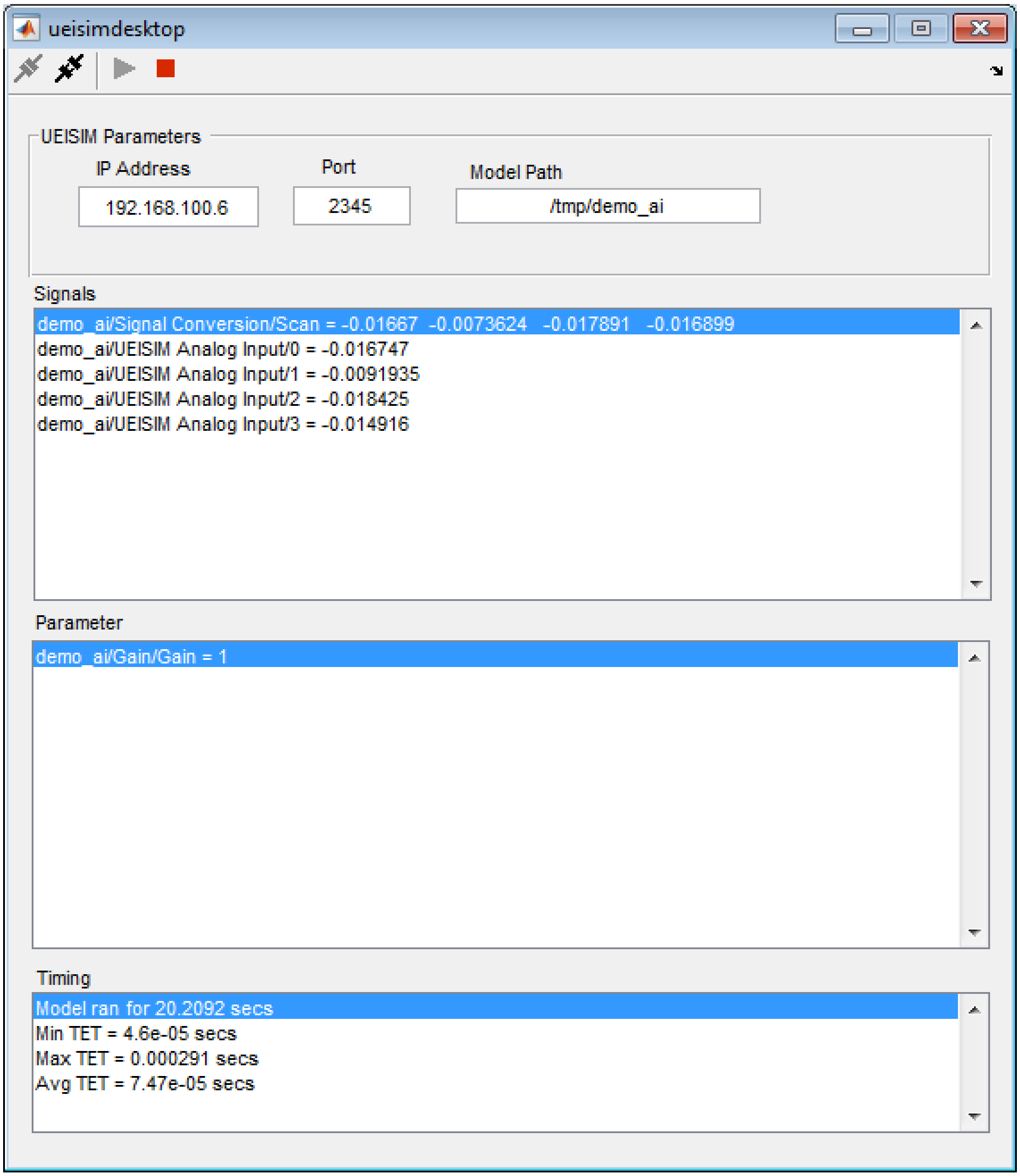
4.5.1.1. UEISIM Desktop Target API

UEISIM Desktop C/C++, .NET and LabVIEW APIs are documents in the UEISIM Desktop User Manual.

The UEISIM Desktop .NET API can be called from matlab scripts and applications.

UEISIM Desktop comes with a built-in Matlab application that allows you to remotely control and monitor a model.
Type **ueisimdesktop** at the matlab command prompt to start this application

The High-Performance Alternative



Configure the UEISIM IP address, port and the path of the model to work with.

The High-Performance Alternative

Then use one of the four buttons in the toolbar to Connect/Disconnect the UEISIM and Start/Stop the model.

4.5.2. Remote monitoring with Simulink in external mode

Simulink's external mode allows you to remotely monitor a simulation running on the UEISim from the Simulink application running on your host OC.

Select the menu option **Simulation/Configuration Parameters…**.

Click on the option **Code Generation** then on **UEISim options**.

Verify that the UEISIM IP address is correct

Change the Remote monitoring setting to **External.**

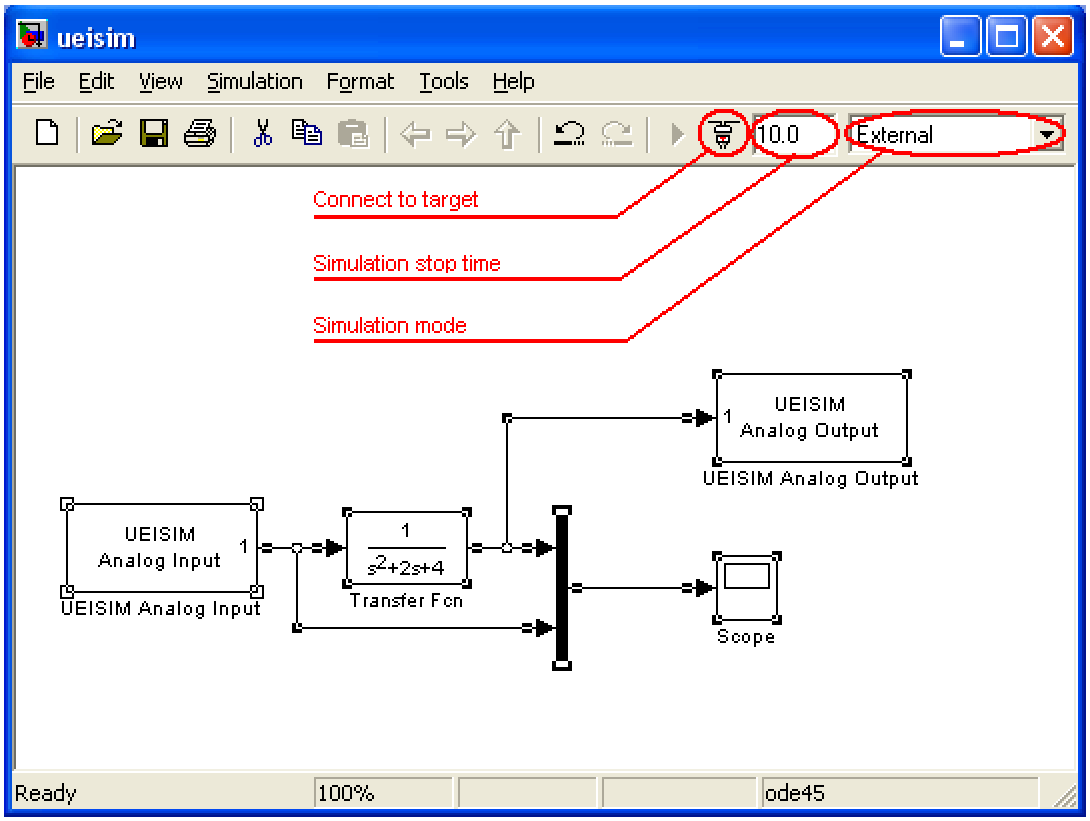
Click on OK and re-build the model.

Logon the UEISim and start the simulation with the command line option '-w'.

```
/tmp # ./ueisim -w
```

This option tells the model to wait for commands received over the network before starting execution.

United Electronic Industries
The High-Performance Alternative



Set the Simulation stop-time to "inf" if you wish to run the simulation continuously.

In your model window, change the **simulation mode** from **normal** to **external** using the toolbar combo-box.

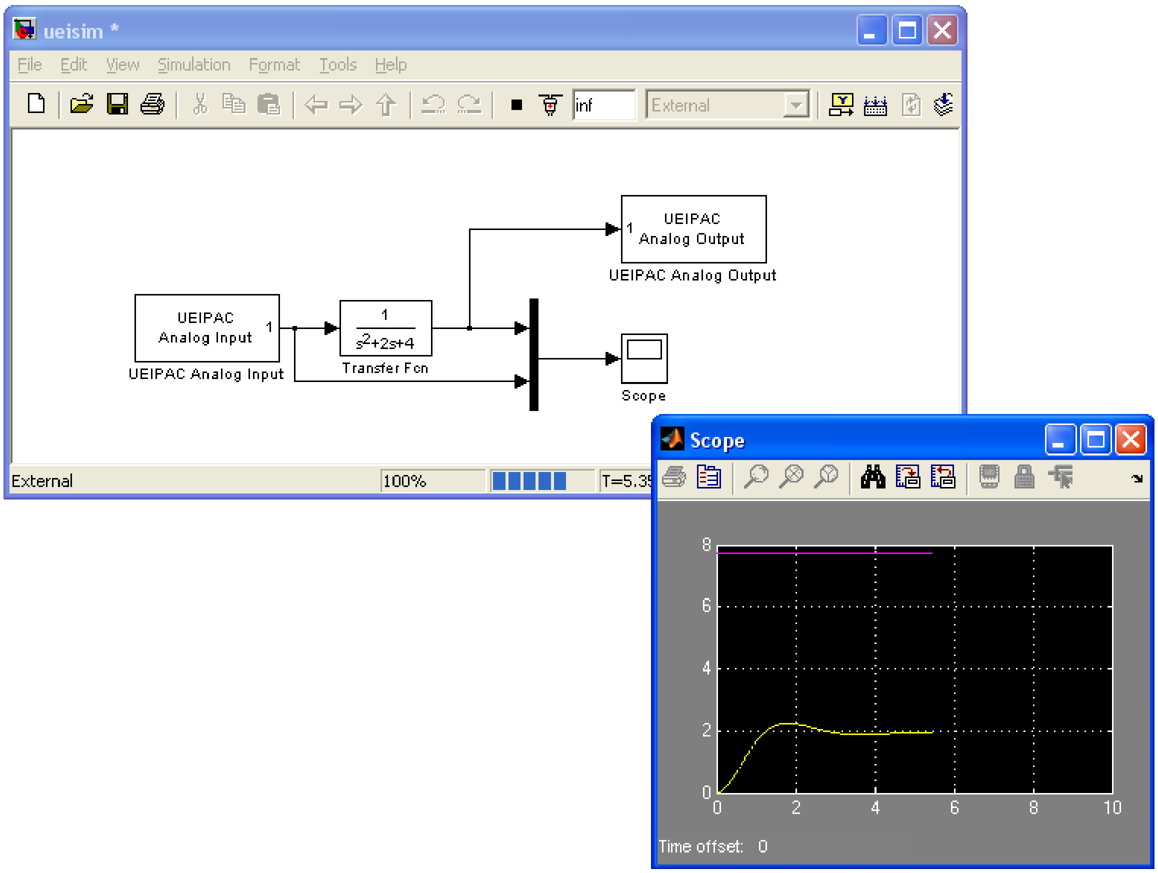
Click on the **Connect to target** button.

After a few seconds, you will be notified that the connection is established when the **Start real-time code** button becomes enabled and the word **External** appears in the status bar.

Click on the **Start real-time code** button to start the simulation.
Double-click on the scope to view the acquired signal as well as the result of the transfer function.

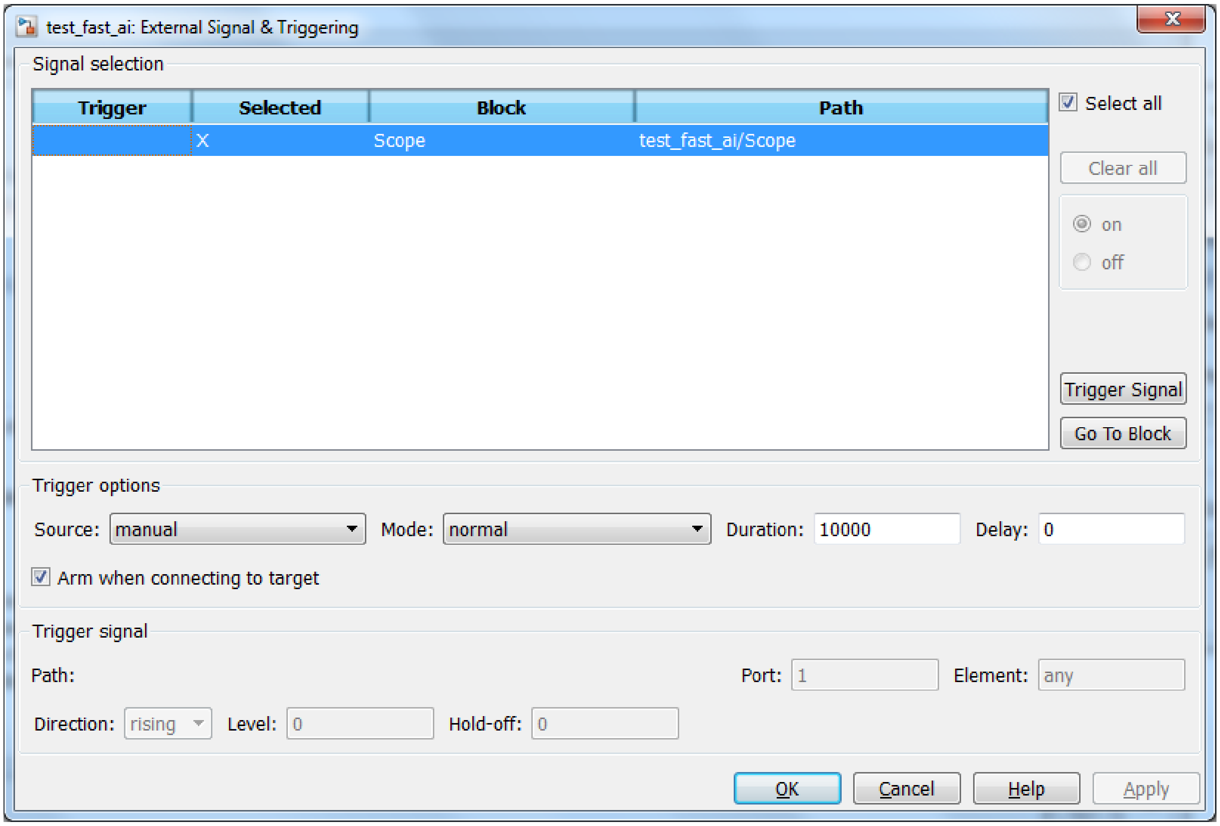
United Electronic Industries

The High-Performance Alternative



You can use the scope block to visualize any signal while the model is executing. Scope only displays 1000 samples per signal, to change the scope's maximum signal duration:

- Select the menu option Code/External Mode Control Panel
- Click on the "Signal & Triggering…" button
- Change the duration field in the "Trigger options"

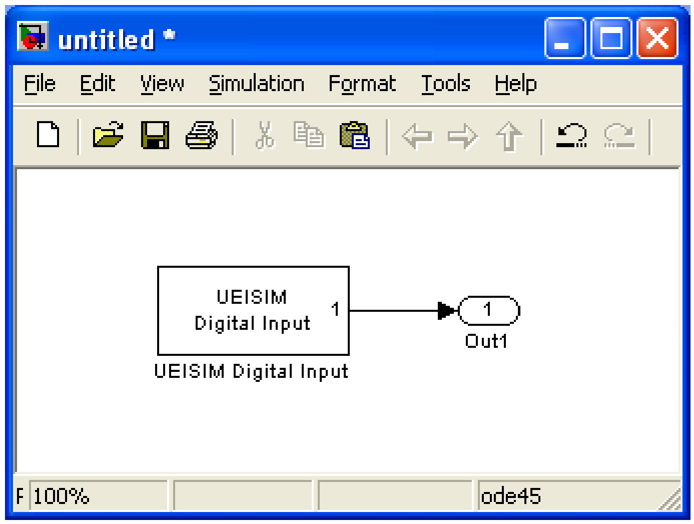


4.6. Logging Data to file

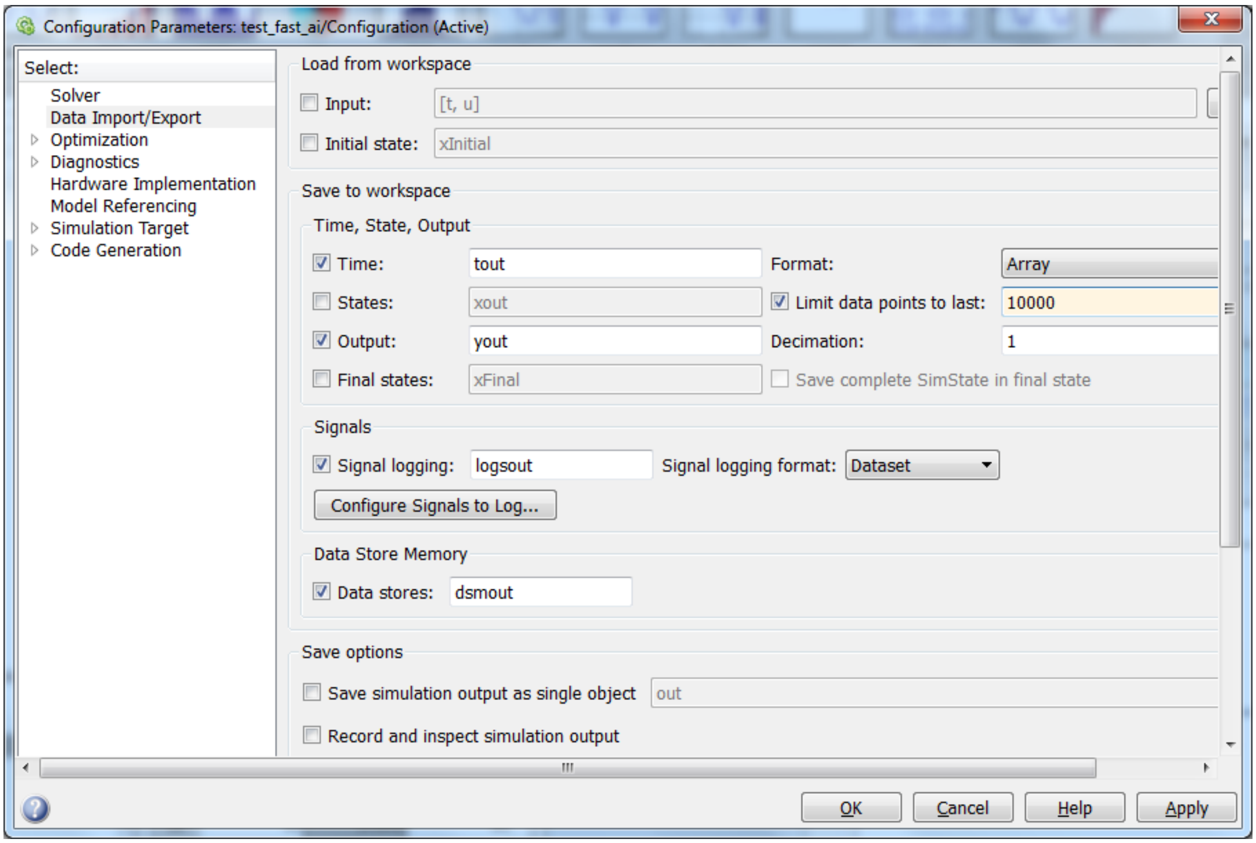
A Matlab MAT data file is automatically created when the model is executed on the UEISIM. By default it only contains one column of data representing the time of each step.

Use the "Out" block to add a column of data to the MAT file. The example below acquires digital inputs and writes them to the MAT file:

The High-Performance Alternative



Simulink uses a circular buffer in RAM to store the most recent values. The default size for the circular buffer is 1000. You change this value in the **Data Import/Export** configuration dialog.
The maximum size depends on the number of signals logged and the memory requirements of the model so that it can all fit in the UEISIM RAM.



The circular buffer containing the latest data points is written to file at the end of the simulation run. The model prints a notification message if the circular buffer wrapped (the simulation ran more steps that the buffer can hold)

```
Executed 52093 iterations in 26.046967 s (1999.964142 updates per
sec.)
*** Log variable tout has wrapped 4 times
    using a circular buffer of size 12000
*** Log variable yout has wrapped 4 times
    using a circular buffer of size 12000
** created test_fast_ai.mat **
```

To look at the content of the MAT file, download the file from the UEISIM (using FTP or SCP) and open it with Matlab.

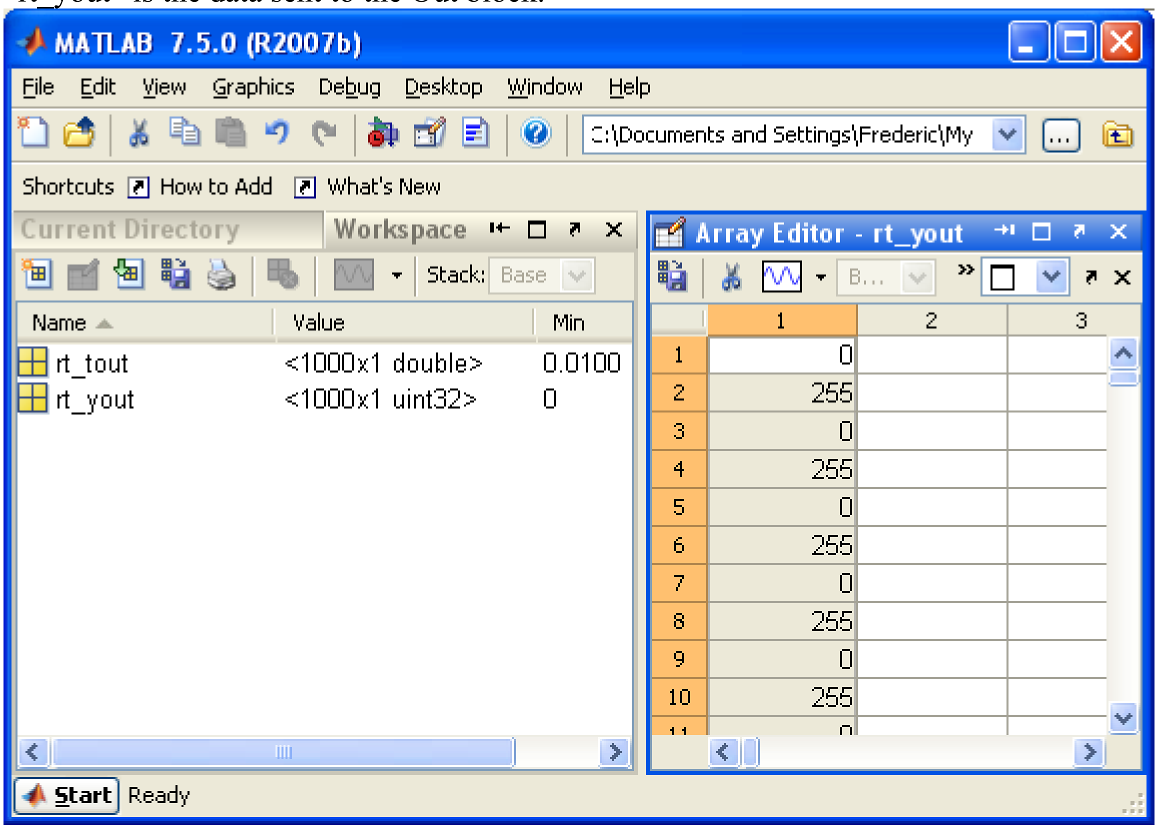
You can download the file directly from Matlab's command line with the following commands:

```
f=ftp('192.168.100.2','root', 'root')
```

The High-Performance Alternative

```
cd(f,'tmp')
binary(f)
mget(f,'untitled.mat')
```

"rt_tout" is the time of each step
"rt_yout" is the data sent to the Out block.



4.7. Running a simulation automatically after boot

Edit the file /etc/rc.local and add an entry for any number of programs that you want to run after the UEISIM complete its power-up sequence.
In the example below, the /etc/rc.local file is modified to run the program "ueisim" at boot time.

```
#!/bin/sh
#
# rc.local
#
# This script is executed at the end of the boot sequence.
# Make sure that the script will "exit 0" on success or any other
```

The High-Performance Alternative

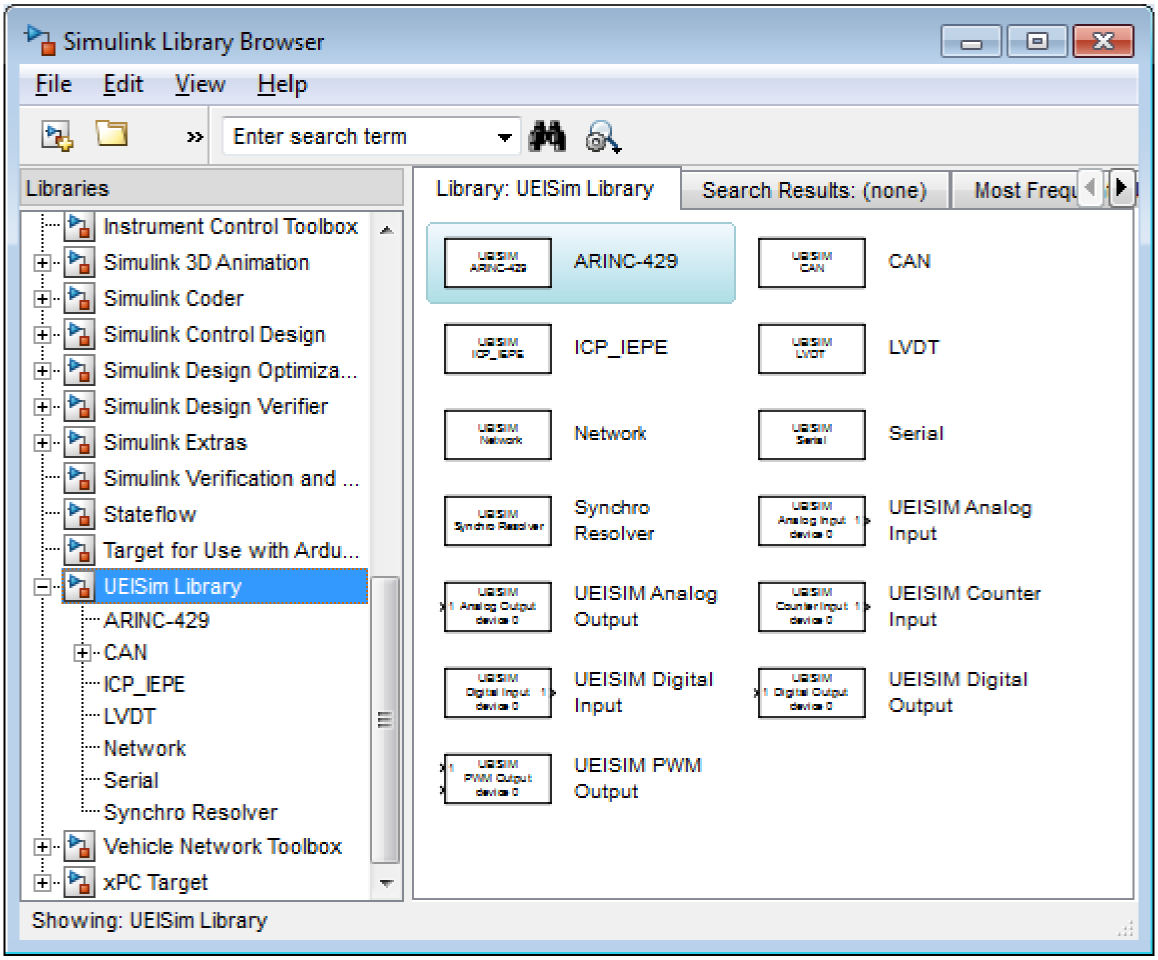
```
# value on error.
#

listlayers > /etc/layers.xml
sync
devtbl

# start Sample201
/tmp/ueisim &

exit 0
```

Note that “ueisim” is executed in the background (‘&’ prefix). To stop “ueisim” you must send the SIGINT signal with the following command (It is equivalent to typing CTRL+C on the console if “ueisim” was running in the foreground):

```
killall -SIGINT ueisim
```

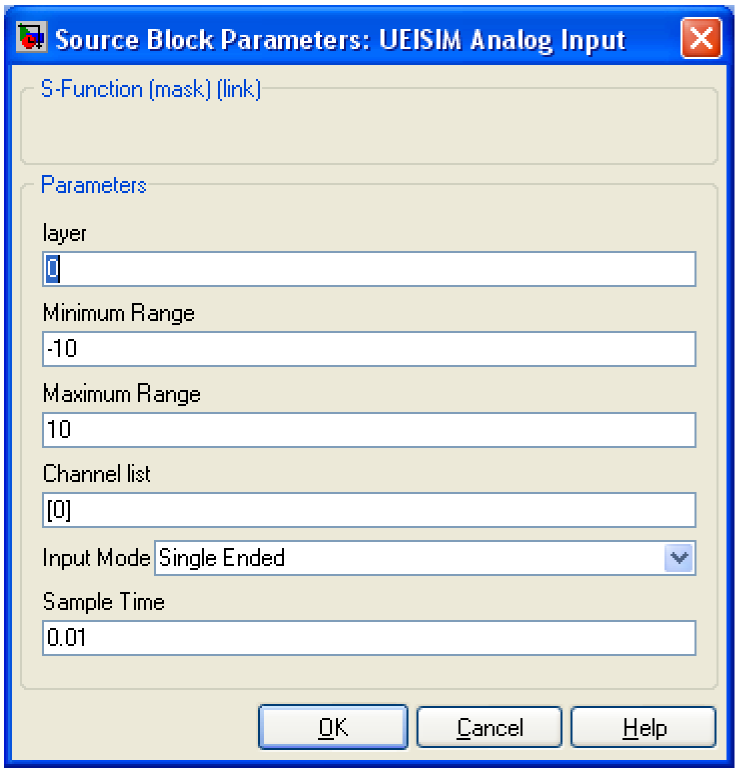
United Electronic Industries
The High-Performance Alternative

5. UEISIM Blockset



5.1. *Analog Input block*

The Analog Input block acquires voltages from the channels specified in the channel list. Each channel measurement is available as a separate output signal.
The data type is double; unit is volts.



- **layer**: The Id of the analog input layer associated with this block. (layer Ids start at 0 with the top layer)
- **Minimum Range**: The minimum voltage expected at the input of each channel
- **Maximum Range**: The maximum voltage expected at the input of each channel
- **Channel list**: Array of channels to acquire from
- **Input Mode**: Single Ended or Differential
- **Sample Time**: The rate at which the block executes during simulation (it also sets the hardware ADC clock).

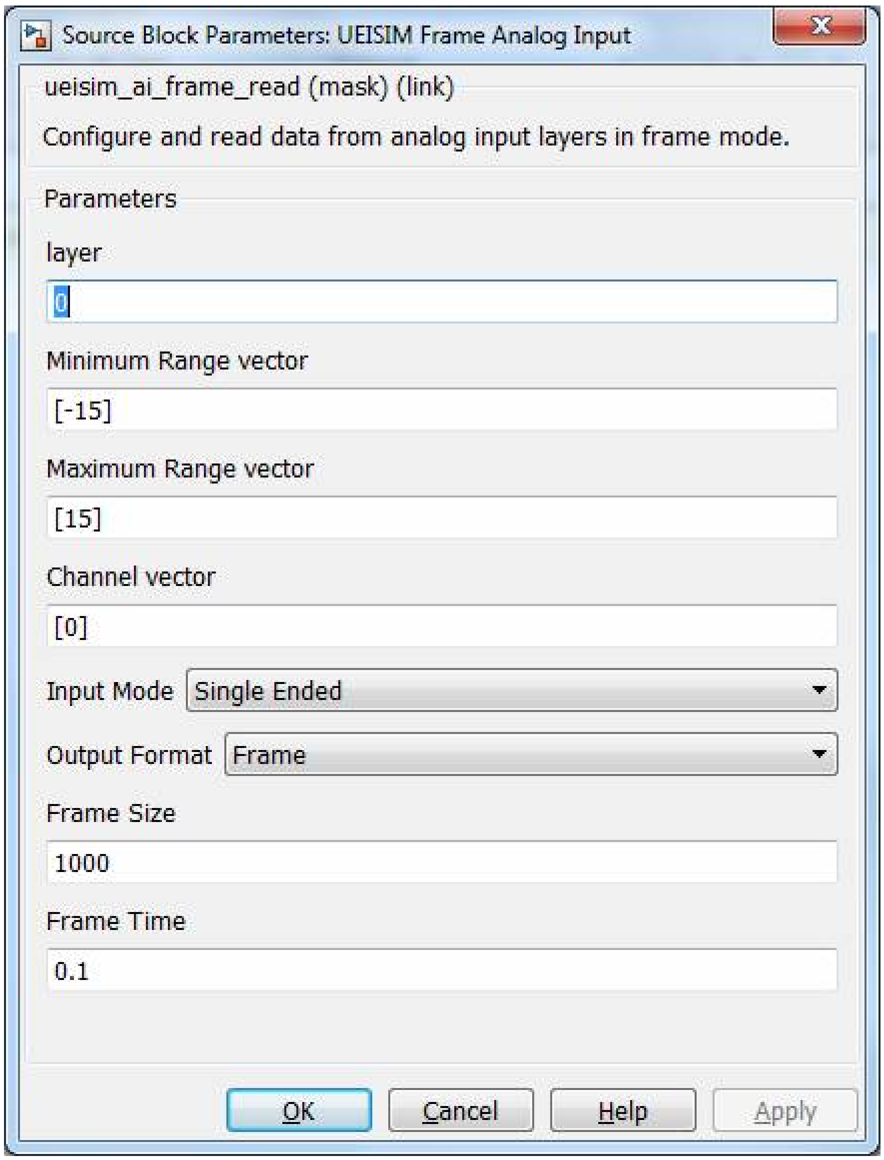
5.2. *Frame Analog Input block*

The Frame Analog Input block acquires voltages from the channels specified in the channel list and returns multiple samples at each simulation step.

Each channel data is available on a separate output. Acquired data can be formatted as frame data or non-frame data (a 1D vector)

The data type is double; unit is volts.

The High-Performance Alternative



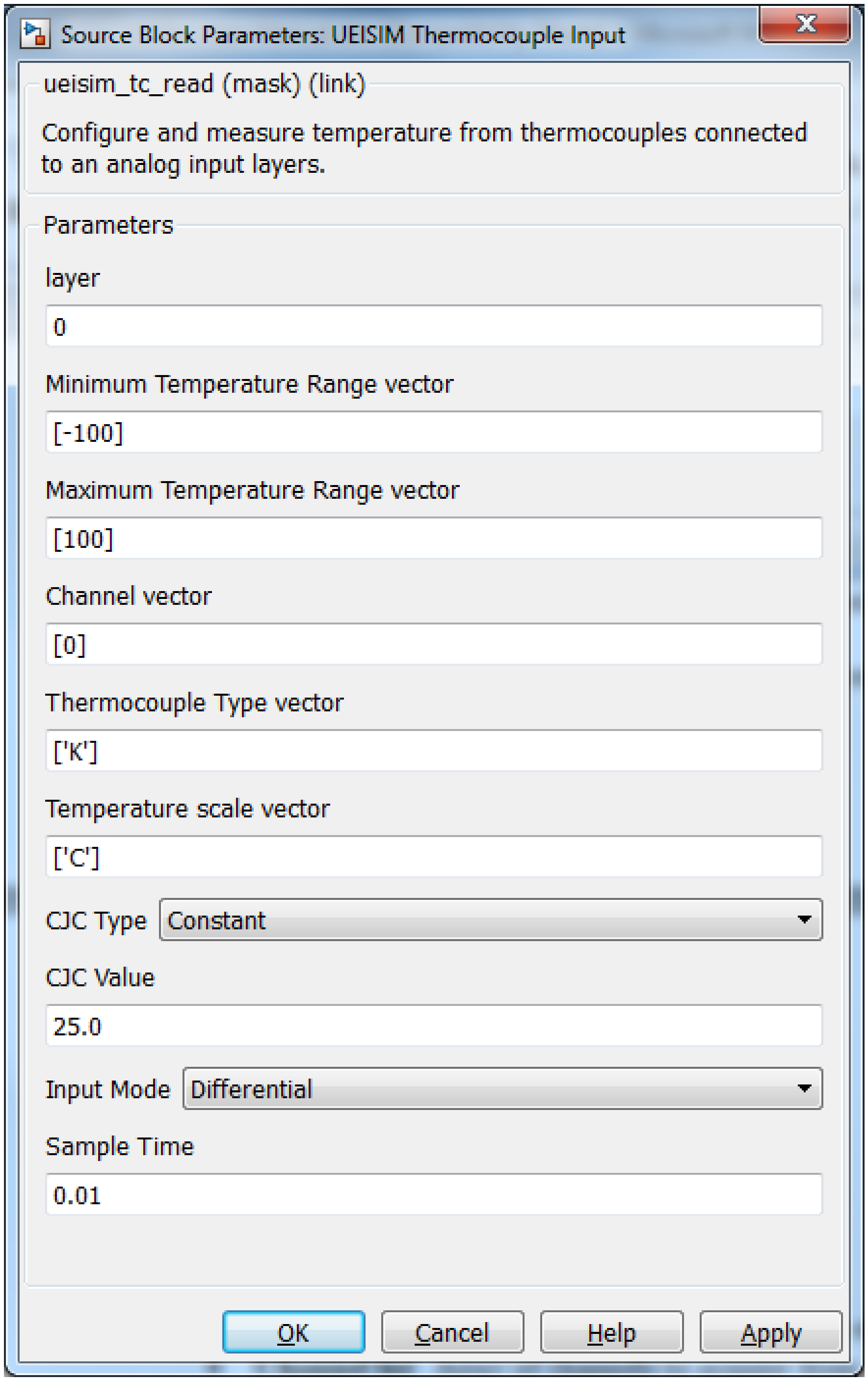
- **layer**: The Id of the analog input layer associated with this block. (layer Ids start at 0 with the top layer)
- **Minimum Range Vector**: A vector containing the e minimum voltage expected at the input of each channel
- **Maximum Range Vector**: A vector containing the maximum voltage expected at the input of each channel
- **Channel vector**: Array of channels to acquire from

The High-Performance Alternative

- **Input Mode**: Single Ended or Differential
- **Output Format**: Set the format used to store the samples in the block output signal(s). Frame or Vector.
- **Frame Size**: The number of samples per channel stored in each frame.
- **Frame Time**: The rate at which the block executes during simulation. The ADC scan period is set to FrameTime/FrameSize.

5.3. Thermocouple Input block

The Thermocouple Input block acquires data from the channels specified in the channel list. Each temperature measurement is available as a separate output.
The data type is double; unit is same as the temperature scale specified in the block parameters.

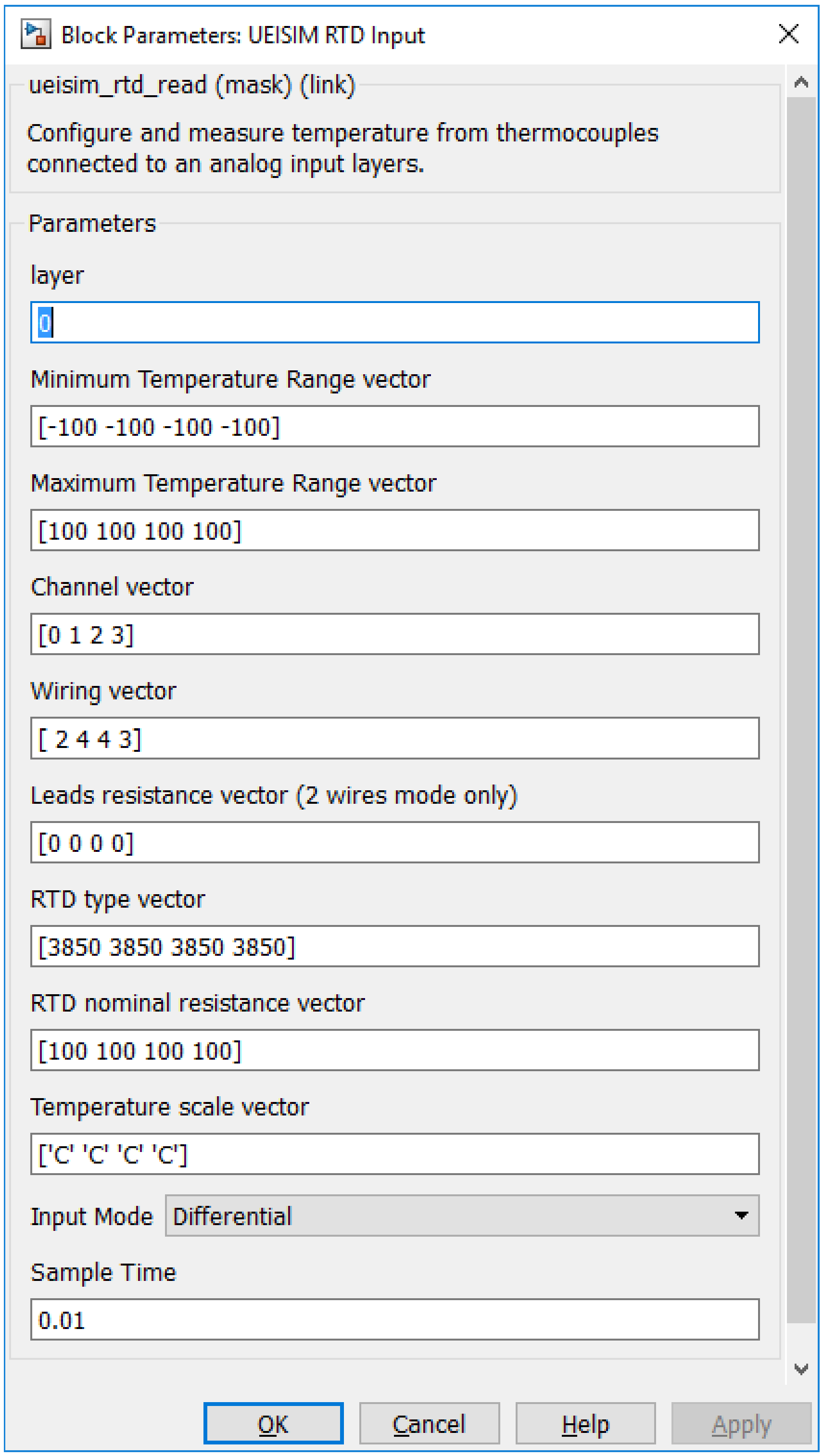


- **layer**: The Id of the analog input layer associated with this block. (layer Ids start at 0 with the top layer)

- **Minimum Range Vector**: The minimum temperature expected at the input of each channel
- **Maximum Range Vector**: The maximum temperature expected at the input of each channel
- **Channel Vector**: Array of channels to acquire from
- **Thermocouple Type Vector**: The type of thermocouple connected to each channel. Supported types are E, J, K, R, S, T, B, N, C
- **Temperature Scale Vector**: The temperature scale for each channel. 'C' for Celsius, 'F' for Fahrenheit, 'K' for Kelvin and 'R' for Rankin.
- **CJC Type**: The type of cold-junction compensation. It can be 'Built-in' or 'Constant'.
- **CJC Value**: The temperature constant used when CJC type is set to 'Constant'
- **Input Mode**: Single Ended or Differential
- **Sample Time**: The rate at which the block executes during simulation (it also sets the hardware ADC clock).

5.4. RTD Input block

The RTD Input block acquires temperatures measured by RTD sensors.. Each temperature measurement is available as a separate output.
The data type is double.
The unit is specified by the **temperature scale** in the block parameters if RTD type is other than 0.
If RTD type is set to zero, the block returns the measured resistance in **Ohms**.


Block Parameters: UEISIM RTD Input
ueisim_rtd_read (mask) (link)
Configure and measure temperature from thermocouples connected to an analog input layers.
Parameters
layer
0
Minimum Temperature Range vector
[-100 -100 -100 -100]
Maximum Temperature Range vector
[100 100 100 100]
Channel vector
[0 1 2 3]
Wiring vector
[2 4 4 3]
Leads resistance vector (2 wires mode only)
[0 0 0 0]
RTD type vector
[3850 3850 3850 3850]
RTD nominal resistance vector
[100 100 100 100]
Temperature scale vector
['C' 'C' 'C' 'C']
Input Mode
Differential
Sample Time
0.01
OK
Cancel
Help
Apply

The High-Performance Alternative

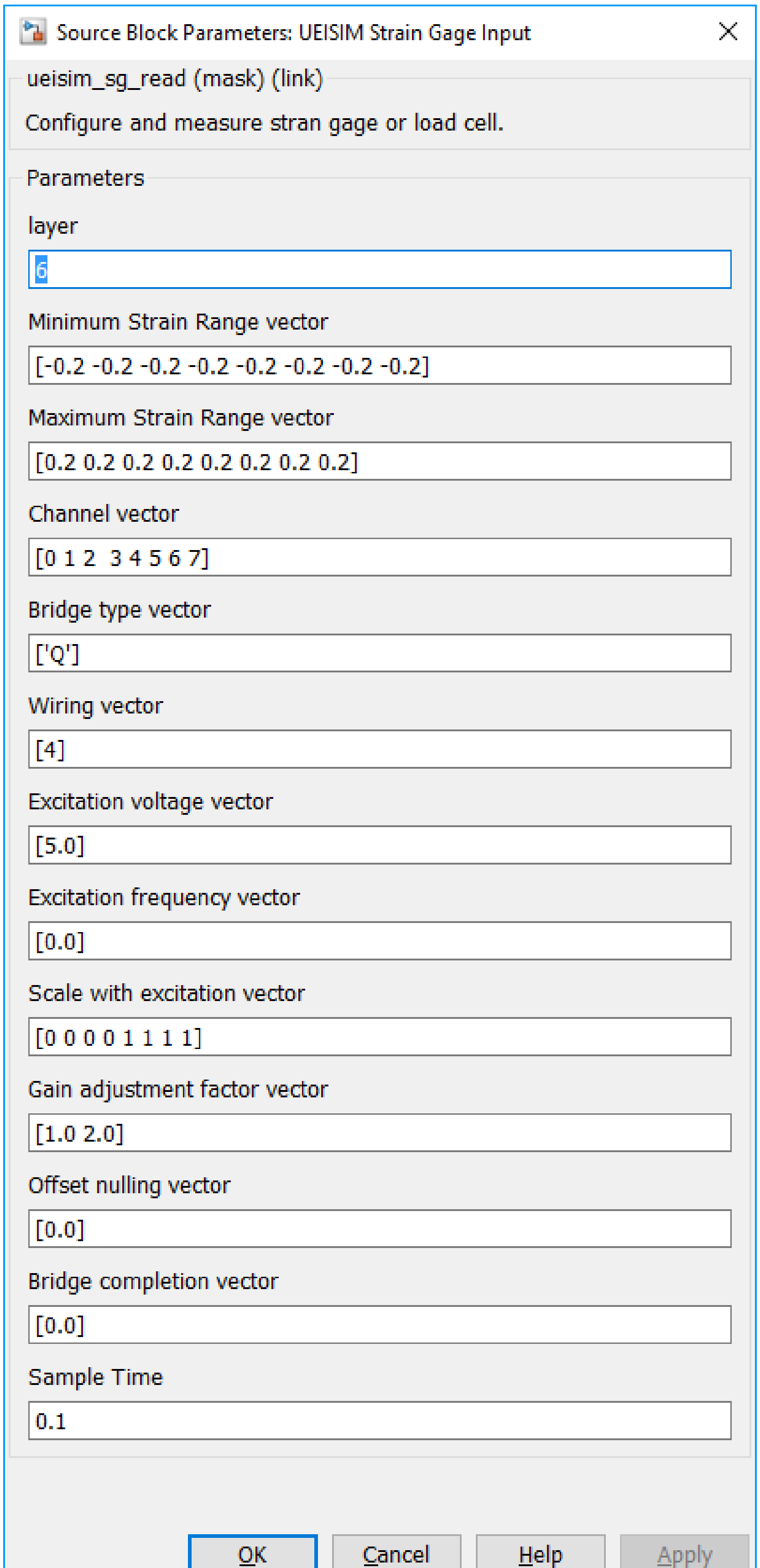
- **layer**: The Id of the analog input layer associated with this block. (layer Ids start at 0 with the top layer)
- **Minimum Range Vector**: The minimum temperature expected at the input of each channel
- **Maximum Range Vector**: The maximum temperature expected at the input of each channel
- **Channel Vector**: Array of channels to acquire from
- **Wiring Vector**: The number of wires used to connect the RTD. Possible values are 2, 3 or 4.
- **Leads Resistance Vector**: The lead resistance in Ohms when connecting RTDs with two wires.
- **RTD Type Vector**: The type of RTD sensor connected to each channel. RTD sensors are specified using the "alpha" constant also known as the temperature coefficient of resistance. Possible values are:
  3750 Low-cost Platinum RTD. a=0.00375 A=3.81E–3 B=-6.02E-7 C=-6.0E-12
  3850 IEC-751 European standard Platinum RTD. a=0.00385 A=3.9083E-3 B=-5.775E-7 C=-4.183E-12
  3902 US Industrial standard Platinum RTD. a=0.003902 A=3.96E-3 B=-5.93E-7 C=-4.3E-12
  3911 ASTM 1137 American standard Platinum RTD. a=0.003911 A=3.9692E-3 B=–5.8495E-7 C=–4.233E-12
  3916 JISC-1604 Japanese standard Platinum RTD. a=0.003916 A=3.9739E-3 B=–5.870E-7 C=–4.4E-12
  3920 Old American standard Platinum RTD. a=0.00392 A=3.9787E-3 B=–5.8686E-7 C=–4.167E-12
  3926 ITS-90 standard Platinum RTD. a=0.003926 A=3.9848E-3 B=–5.870E-7 C=–4.0E-12
  3928 ITS-90 standard Platinum RTD. a=0.003928 A=3.9888E-3 B=–5.915E-7 C=–3.85E-12
  0 Measure the resistance without converting to temperature.
- **RTD Nominal Resistance Vector**: The RTD nominal resistance at 0 deg C.
- **Temperature Scale**: The temperature scale for each channel. ‘C’ for Celsius, ‘F’ for Fahrenheit, ‘K’ for Kelvin and ‘R’ for Rankin.
- **Input Mode**: Single Ended or Differential
- **Sample Time**: The rate at which the block executes during simulation (it also sets the hardware ADC clock).

The High-Performance Alternative

5.5. Strain gage Input block

The Strain Gage Input block acquires voltages measured by strain gages and load cell sensors. Those sensors require a voltage excitation which is configurable in the block property dialog.
Each measurement is available as a separate output. Actual excitation voltage measurements are also available as a separate output. So a SG block configured with N channels will display 2xN outputs. The N first outputs are the measurement and N next outputs are the excitation voltage measurements.
The data type is double; unit is V when **scale with excitation** is set to 0 and mV/V when it is set to 1 in the block parameter dialog.

The High-Performance Alternative



The High-Performance Alternative

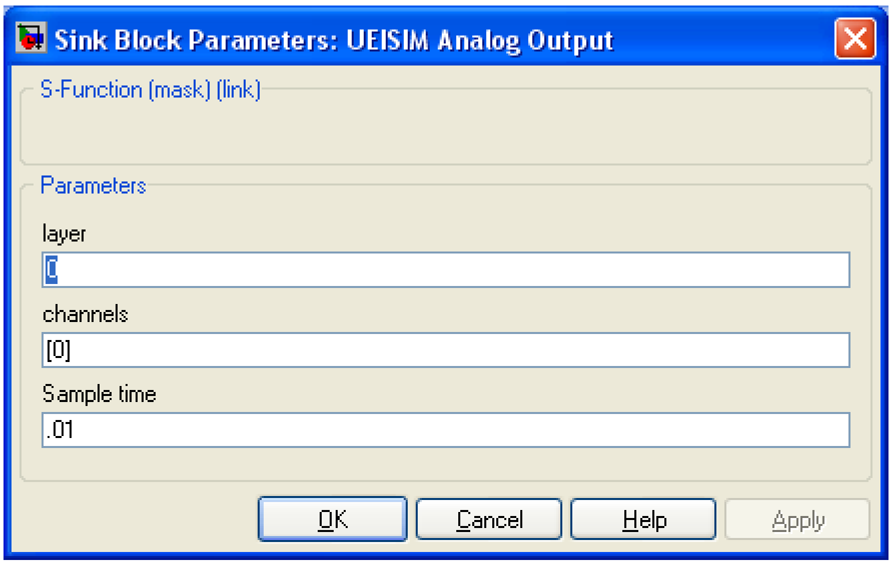
- **layer**: The Id of the analog input layer associated with this block. (layer Ids start at 0 with the top layer)
- **Minimum Range Vector**: The minimum voltage measurement expected at the input of each channel
- **Maximum Range Vector**: The maximum voltage measurement expected at the input of each channel
- **Channel Vector**: Array of channels to acquire from
- **Bridge Type Vector**: The type of bridge used to connect the strain gage to each channel. 'Q' for Quarter-bridge, 'H' for Half-bridge and 'F' for full bridge (use 'F' for load cells)
- **Wiring Vector**: The number of wires used to connect the sensor. Possible values are 4 or 6.
- **Excitation Voltage Vector**: The excitation voltage used to power the sensor.
- **Excitation Frequency Vector**: The excitation frequency used to power sensor that require AC excitation.
- **Scale with Excitation Vector**: 0 disables scaling and measurements are returned in volts, 1 enables scaling and measurements are returned in mV/V (measurement in mV divided by measured excitation)
- **Gain Adjustment Factor Vector**: The GAF is applied to measurements, its value is measured during a shunt calibration procedure (UEISIM is not capable of doing shunt calibration, you need to use a separate program to obtain the GAF)
- **Offset Nulling Vector**: Set the offset nulling setting used to program the nulling circuitry. With Offset nulling enabled, a nulling circuit adds an adjustable DC voltage to the output of the amplifier making sure that the bridge output is 0V when no strain is applied. Set it to 0.0 to automatically perform offset nulling next time the session is started. Make sure no strain is applied on the bridge before nulling the offset. (feature is disabled in this version)
- **Bridge completion Vector**: Set the bridge completion setting used to program the bridge completion circuitry. Set it to 0.0 to automatically perform bridge completion when the model is started. Make sure no strain is applied on the bridge. (feature is disabled in this version)
- **Sample Time**: The rate at which the block executes during simulation (it also sets the hardware ADC clock).

5.6. *Analog Output block*

The Analog Output block updates the voltage generated by the channels specified in the channel list. Each channel update is specified as a separate input.

United Electronic Industries
The High-Performance Alternative

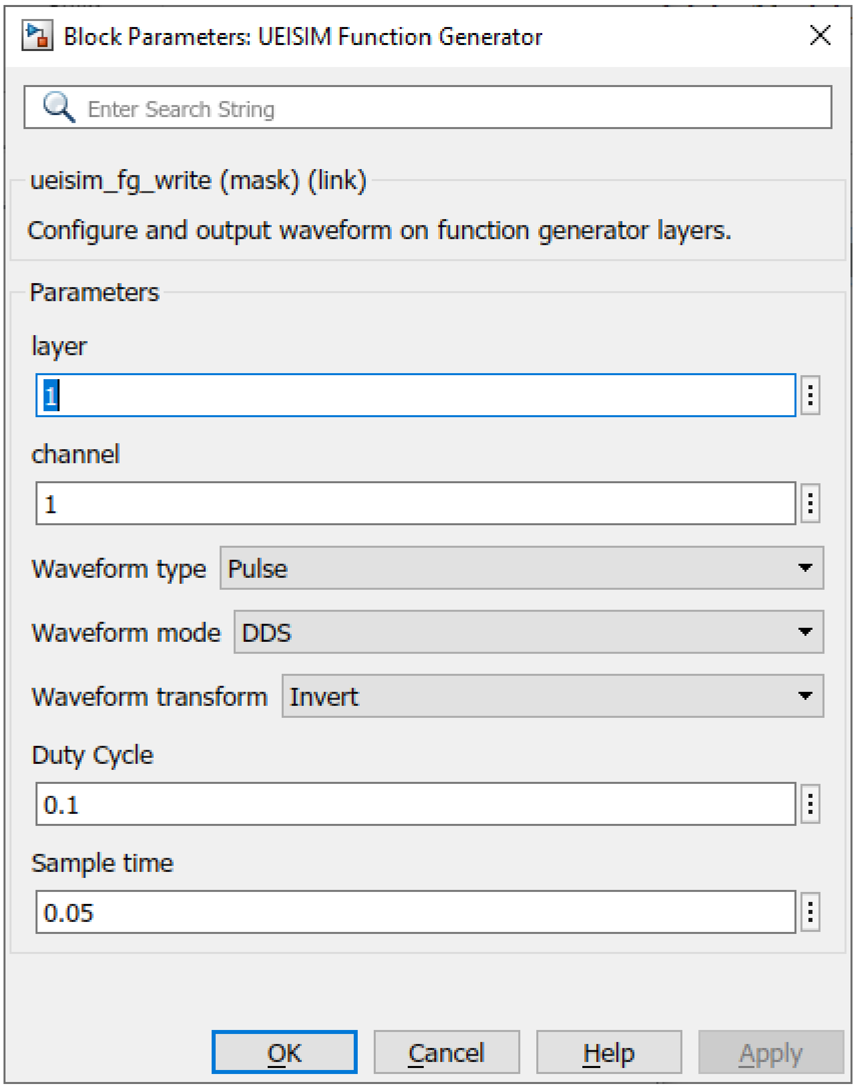
The data type is double; unit is volts.



- **layer**: The Id of the analog output layer associated with this block (layer Ids start at 0 with the top layer)
- **Channels**: Array of channels to generate to
- **Sample Time**: The rate at which the block executes during simulation (it also sets the hardware DAC clock).

5.7. Function Generator block

The Function generator block is designed to control a function generator such as the AO-364. Use one block per channel. You can modulate the waveform frequency, amplitude, offset and phase through the block's four inputs.



- **layer**: The Id of the function generator layer associated with this block (layer Ids start at 0 with the top layer)
- **Channel**: The output channel controlled by this block.
- **Waveform type**: The shape of the waveform (**sine**|**square**|**triangle**|**sawtooth**)
- **Waveform mode**: DDS offers most precise frequency (within 0.1Hz), PLL offers less harmonics and less jitter.
- **Waveform transform**: transform applied to the waveform (**Mirror**|**Invert**|**Both**)
- **Duty cycle**: The duty cycle as a value between 0 and 1. (only have an effect on pulse waveform)
- **Sample Time**: The rate at which the block executes during simulation

The High-Performance Alternative

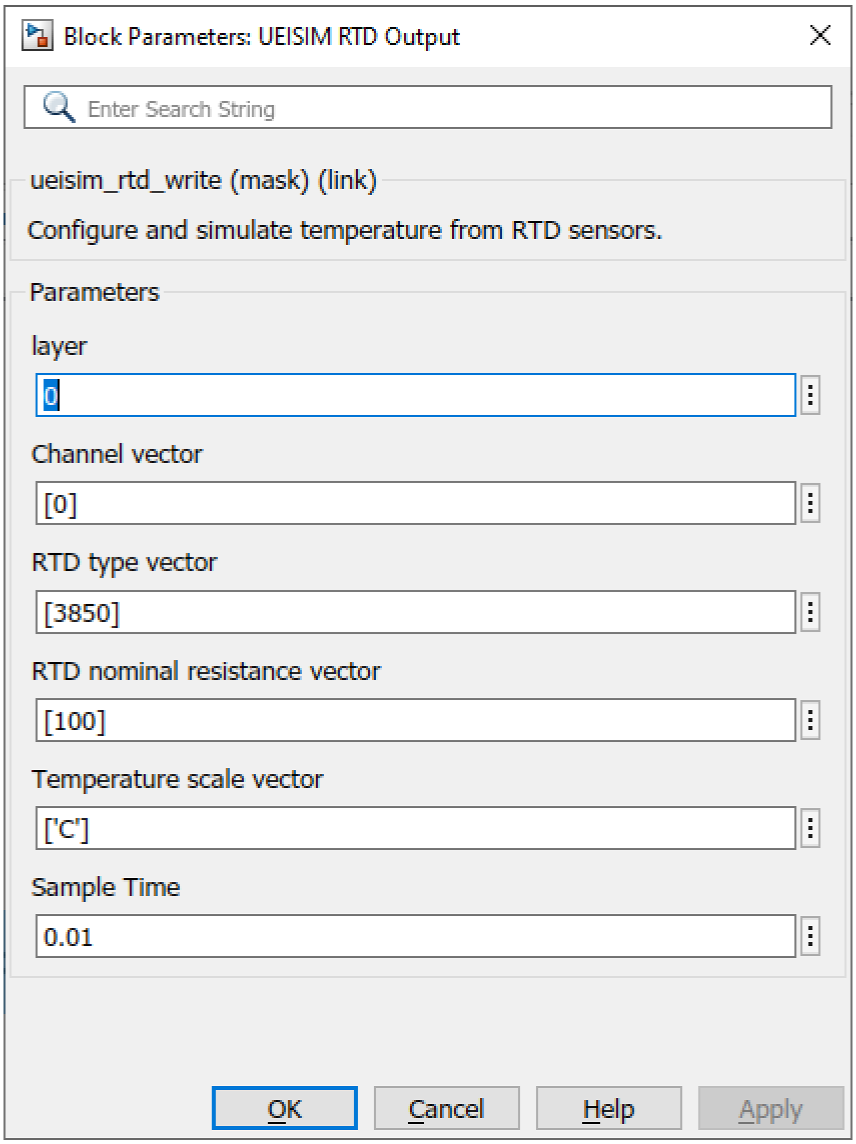
5.8. RTD/Resistance Simulation block

This block is designed to work with a resistance output I/O layer such as the RTD-388. The block updates the resistance at the output of each channels specified in the channel list.

Each channel resistance value (or RTD temperature) is specified as a separate input.

When RTD type is other than zero, the unit is the temperature unit specified by the **temperature scale** parameter.

When RTD type is set to zero, the unit is the resistance in **Ohms**.



- **layer**: The Id of the RTD output layer associated with this block. (layer Ids start at 0 with the top layer)
- **Channel Vector**: Array of channels to output to
- **Wiring Vector**: The number of wires used to connect the RTD. Possible values are 2, 3 or 4.
- **RTD Type Vector**: The type of RTD sensor connected to each channel. RTD sensors are specified using the "alpha" constant also known as the temperature coefficient of resistance. Possible values are:
  3750 Low-cost Platinum RTD. a=0.00375 A=3.81E–3 B=-6.02E-7 C=-6.0E-12

3850 IEC-751 European standard Platinum RTD. a=0.00385 A=3.9083E-3 B=-5.775E-7 C=-4.183E-12
3902 US Industrial standard Platinum RTD. a=0.003902 A=3.96E-3 B=-5.93E-7 C=-4.3E-12
3911 ASTM 1137 American standard Platinum RTD. a=0.003911 A=3.9692E-3 B=–5.8495E-7 C=–4.233E-12
3916 JISC-1604 Japanese standard Platinum RTD. a=0.003916 A=3.9739E-3 B=–5.870E-7 C=–4.4E-12
3920 Old American standard Platinum RTD. a=0.00392 A=3.9787E-3 B=–5.8686E-7 C=–4.167E-12
3926 ITS-90 standard Platinum RTD. a=0.003926 A=3.9848E-3 B=–5.870E-7 C=–4.0E-12
3928 ITS-90 standard Platinum RTD. a=0.003928 A=3.9888E-3 B=–5.915E-7 C=–3.85E-12
0 Output resistance value.

- **RTD Nominal Resistance Vector**: The RTD nominal resistance at 0 deg C.
- **Temperature Scale**: The temperature scale for each channel. ‘C’ for Celsius, ‘F’ for Fahrenheit, ‘K’ for Kelvin and ‘R’ for Rankin.
- **Sample Time**: The rate at which the block executes during simulation (it also sets the hardware clock).

5.9. Wafeform regeneration block

Waveform regeneration is a mode available on AO-308, AO-332 and AO-333 where the on-board FIFO is preloaded with a Waveform. The FIFO works as a storage area (elements don’t disappear from the FIFO after being sent to the analog outputS).
The analog output channels are continuously updated with data from the FIFO at a pre-determined rate which is independent from the model’s sample rate.
The waveform data size can be smaller than the FIFO but must not exceed the FIFO size (1024 samples on AO-308/332/333)

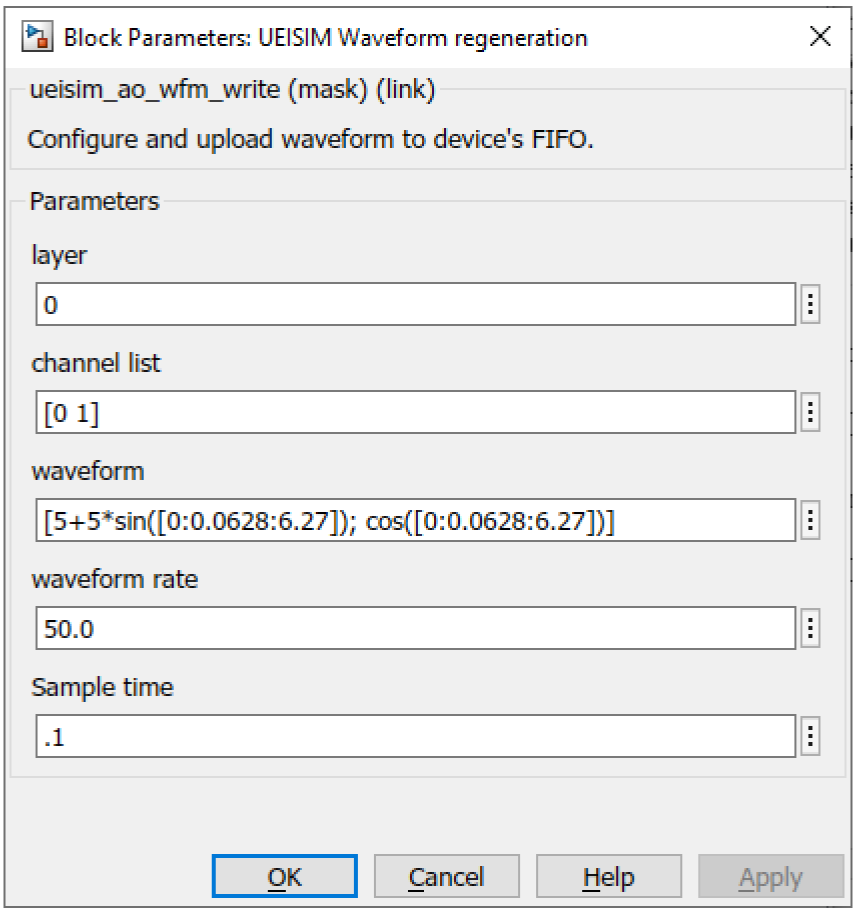
The waveform is specified as one of the blocks parameters and can’t be changed while the model is running. The waveform is represented as a Matlab 2D array where rows correspond to channels and columns to data scans (a group of one sample per channel).

For example, the waveform below is a one cycle sine wave of 100 samples for one channel:

```
[sin([0:0.0628:6.27])]
```

The waveform below contains a sine and a cosine waveform for two analog output channels:
`[sin([0:0.0628:6.27]); cos([0:0.0628:6.27])]`

The waveform below contains three square waves of different frequencies:
[-5 5 -5 5 -5 5 -5 5; -5 -5 5 5 -5 -5 5 5; -5 -5 -5 -5 5 5 5 5]



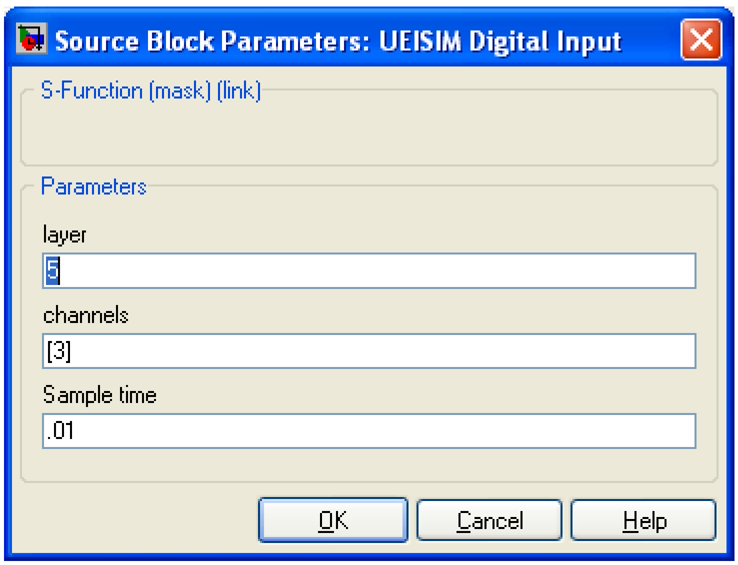
- **layer**: The Id of the analog output layer associated with this block (layer Ids start at 0 with the top layer)
- **Channel**: The list of output channels controlled by this block.
- **Waveform**: The 2D array containing the waveform samples.
- **Waveform rate**: The rate at which the waveforms are generated.
- **Sample Time**: The rate at which the block executes during simulation

Note that this block start the waveform generation as soon as the model is started. Subsequent executions of the block do not have any effect on the hardware or on the model.

The High-Performance Alternative

5.10. Digital Input block

The Digital Input block acquires the digital state of the channels specified in the channel list. Each channel is available as a separate output.
A channel is a group of input lines. The number of input lines contained in each channel depends on the hardware (for example the DIO-405 groups its input lines in one port of twelve lines).
The data type is uint32. Each bit of the value read from a given channel corresponds to the state of one input line.

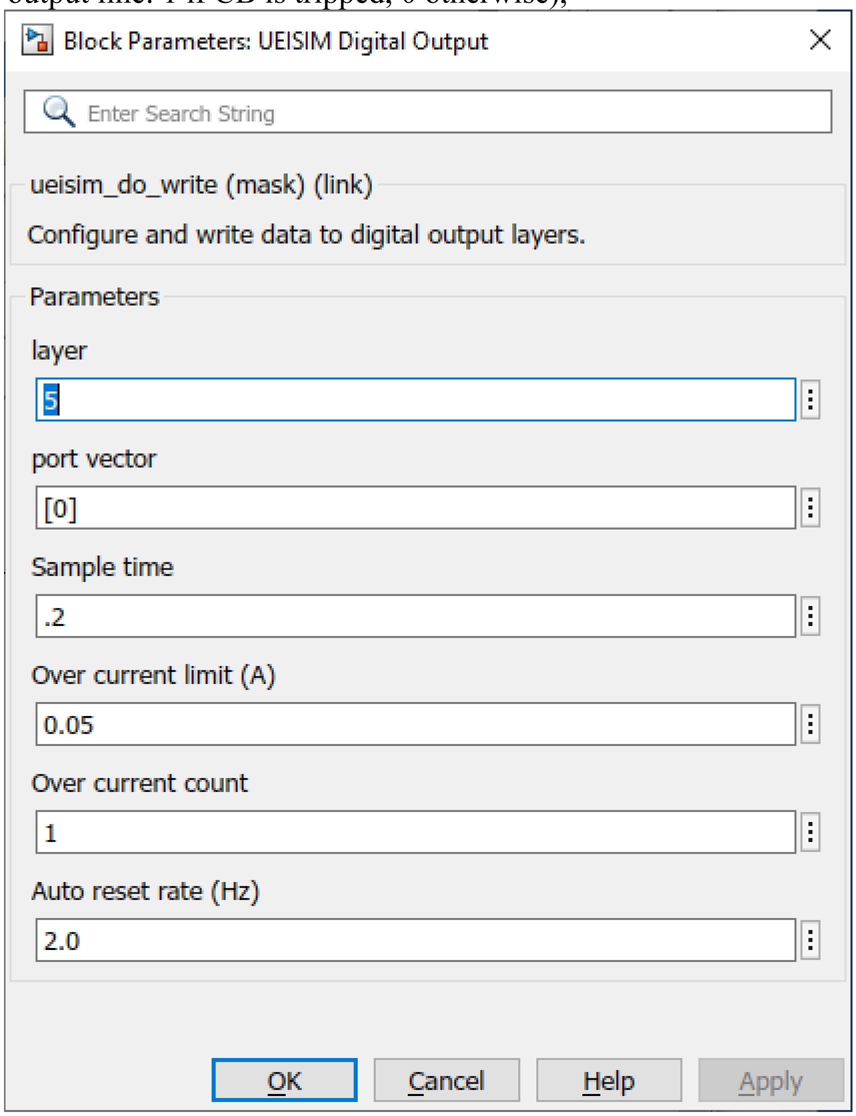


- **layer**: The Id of the digital input layer associated with this block (layer Ids start at 0 with the top layer)
- **Channels**: Array of ports to read from. Input lines are organized into ports (read the manual of your digital layer to find out how many lines there are in each port).
- **Sample Time**: The rate at which the block executes during simulation (it also sets the hardware clock).

5.11. Digital Output block

The Digital Output block updates the digital state of the channels specified in the channel list. Each channel is available as a separate input.
A channel is a group of output lines. The number of output lines contained in each channel depends on the hardware (for example the DIO-405 groups its output lines in one port of twelve lines).

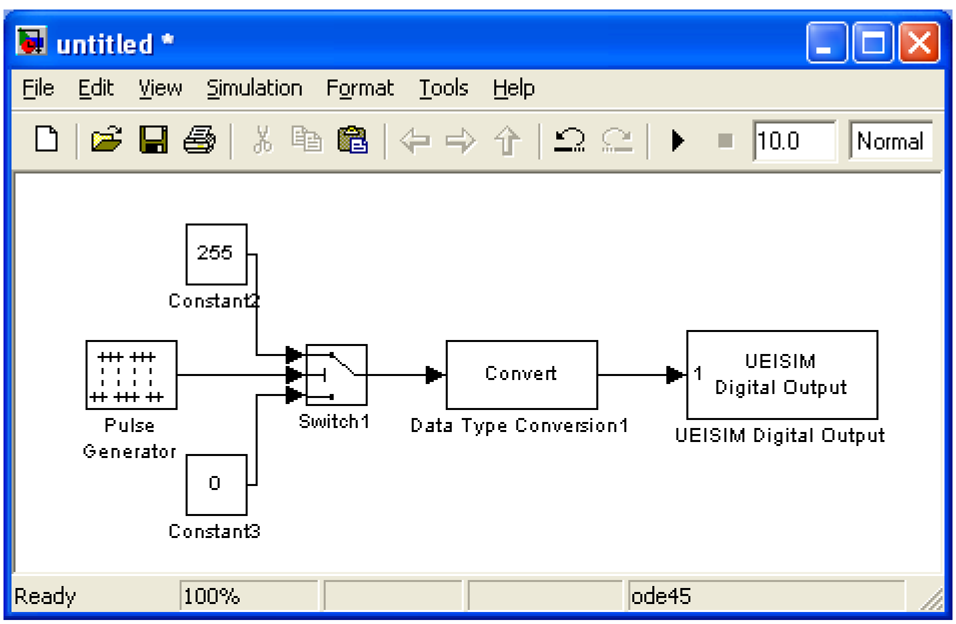
The DO block comes with one input per channel of data type uint32. Each bit of the value written to a given channel corresponds to the state of one output line.
The DO block also comes with one output that contains two values representing the status of the circuit breaks on guardian DO boards. The first value represents the "sticky" state of the circuit breakers (one bit per output line:1 if the CB was tripped since last status read, 0 otherwise), the second value represents the instant state of the circuit breakers (one bit per output line: 1 if CB is tripped, 0 otherwise),



- **layer**: The Id of the digital output layer associated with this block (layer Ids start at 0 with the top layer)

- **Channels**: Array of ports to write to. Input lines are organized into ports (read the manual of your digital layer to find out how many lines there are in each port).
- **Sample Time**: The rate at which the block executes during simulation (it also sets the hardware clock).
- **Over current limit (A)**: The maximum current allowed to flow through an output line before circuit breaker opens (for guardian boards only)
- **Over current count**: The number of current measurements above the limit allowed before circuit breaker opens (for guardian boards only)
- **Auto Reset Rate (Hz)**: The rate at which the board attempts to automatically reset the circuit breakers

The type of the signals connected to the DO block must be “uint32”. You can use Simulink’s “Data Type Conversion block” to convert your signal as shown in the example below:



Note for bi-directional DIO layers:
Some DIO devices come with DIO ports where the direction (input or output is programmable.
For example, the DIO-403 is composed of six 8-bit port.The direction of each port can be input or output.
All port directions are set to input by default. A port direction is changed to output once it is specified in the DO block.

Setting the channel vector to [0 4 5] in the DI block will output 3 uint8 values that contain the states of lines 0-7, 32-39 and 40-47
Setting the channel vector to [1 2 3] in the DO block will require three input values to set the state of lines 8-15,16-23 and 24-31

If the same port is configured in both DI and DO blocks, the port will be output and the DI block will read back the value written by the DO block.

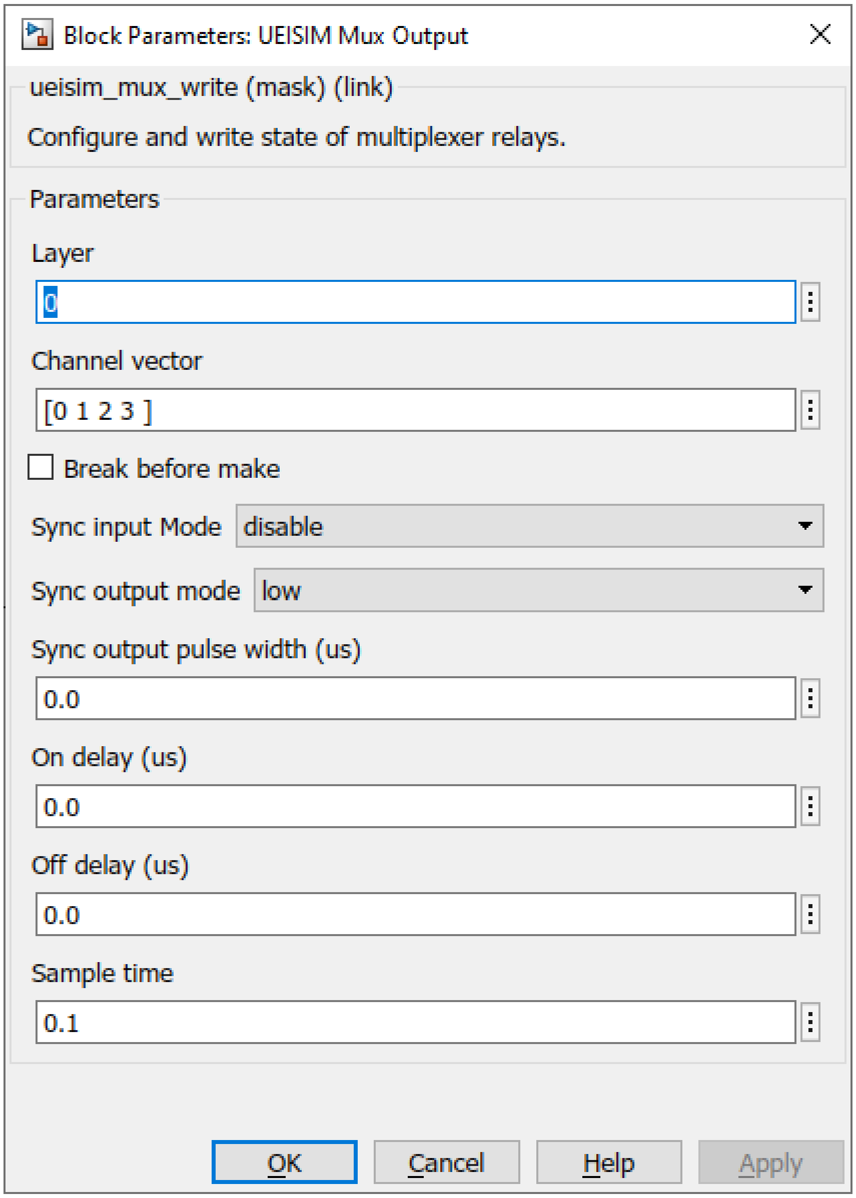
5.12. MUX Output block

The Multiplexer (MUX) Output block controls the state of the relays for each channel specified in the channel list.
A channel is composed of multiple relays (three relays per channel on MUX-414 and MUX-418).
The number of channel depends on the hardware (for example the MUX-414 contains 14 channels).

The MUX block comes with one input per channel of data type uint32. The value of the signal connected to each channel specifies the relay to switch on (only one relay at a time can be on): 0=all relays are off, 1=relay A is on, 2=relay B is on, 3=relay C is on.

The MUX block also comes with four outputs that contains relay status.
The first output represents the state of all A relays (one bit per channel:1 if the relay is on, 0 otherwise), T
The second output represents the state of all B relays.
The third output represents the state of all C relays.
The fourth output represents the low-level status (not detailed here).

The High-Performance Alternative



- **Layer**: The Id of the Mux layer associated with this block (layer Ids start at 0 with the top layer)
- **Channel vector**: Vector of channels to write to.
- **Break before make**: When enabled, the original signal path is opened before the new signal path is closed. It avoids any momentary shorting between two signal sources.

The High-Performance Alternative

- **Sync input mode**: Configure the MUX device to wait for a pulse on a synchronization input before configuring relays.
- **Sync output mode**: Configure the MUX device to emit a pulse on a synchronization output after configuring its relays.
- **Sync output pulse width**: The synchronization output pulse width in microseconds.
- **On delay**: Delays the actual relay closing (in microseconds).
- **Off delay**: Delays the actual relay opening (in microseconds).
- **Sample Time**: The rate at which the block executes during simulation.

The type of the signals connected to the MUX block must be “uint32”. You can use Simulink’s “Data Type Conversion block” to convert your signal.

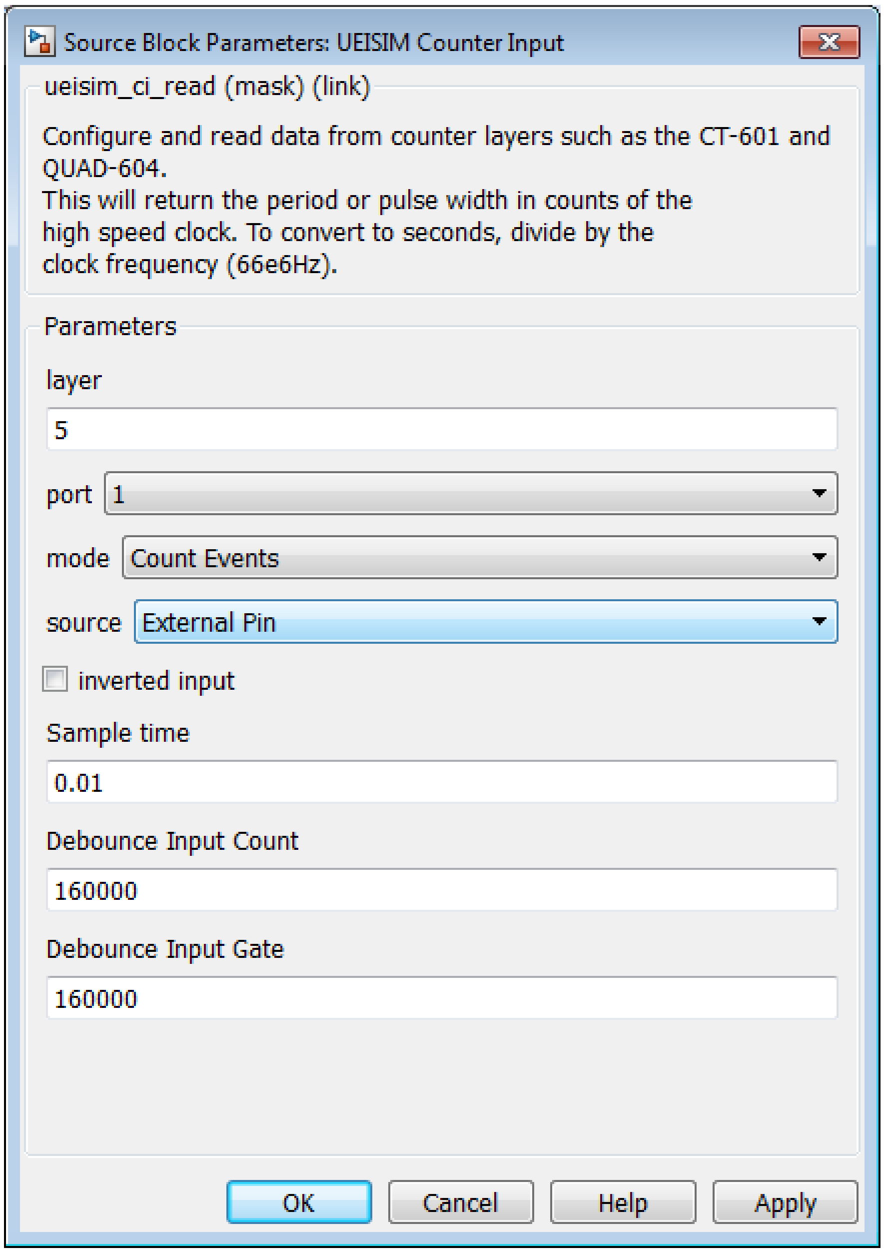
5.13. Counter Input block

The Counter Input block acquires the current count of the specified counter.
Use one instance of this block for each counter you wish to use as input.
The data type is uint32.

The value read depends on the counter operating mode:

- Count Events: Reads the number of rising edges detected on the counter input since the model started
- Pulse Width: The delay between the most recent rising and falling edges detected on the counter input. Delay is returned in 66MHz clock ticks; divide the value by 66000000.0 to convert to seconds.
- Period: The counter input enables two outputs in this mode.
  The first output returns the period (delay between the two most recent rising edges detected on the counter input).
  The second output returns the high state duration (equivalent to pulse width above)
  Period and high pulse width are returned in 66MHz clock ticks; divide the value by 66000000.0 to convert to seconds.
  It possible to average period measurement over multiple periods. Set the “Period Count” parameter to the number of periods minus one.
- Quadrature: Reads the position measured by a quadrature encoder.

The High-Performance Alternative



- **layer**: The Id of the counter input layer associated with this block (layer Ids start at 0 with the top layer)
- **port**: The port to read from.
- **mode**: The operation mode. Possible values are “Count Events”, “Measure Pulse width”, “Measure period” and “Quadrature Encoder”.

The High-Performance Alternative

- **source**: The source of the input signal. Possible values are “Internal Clock” and “External Pin”.
- **gate**: The source of the gate signal. Possible values are “Internal” and “External”.
- **inverted input**: the input signal is inverted when this is checked.
- **Sample Time**: The rate at which the block executes during simulation (it also sets the hardware clock).
- **Period Count**: The number of periods used for one period measurement. The measured period is averaged over (PC+1) periods. Set PC to 0 to measure one period, PC=1 to measure two periods etc…
- **Debounce input count**: the minimum pulse width to accept on counter input. Value is specified in 66Mz ticks. Smaller pulses are rejected.
- **Debounce gate count**: the minimum pulse width to accept on gate input. Value is specified in 66Mz ticks. Smaller pulses are rejected

The type of the signals connected to the CI block input must be “uint32”. You can use Simulink’s “Data Type Conversion block” to convert your signal

5.14. Counter FIFO Input block

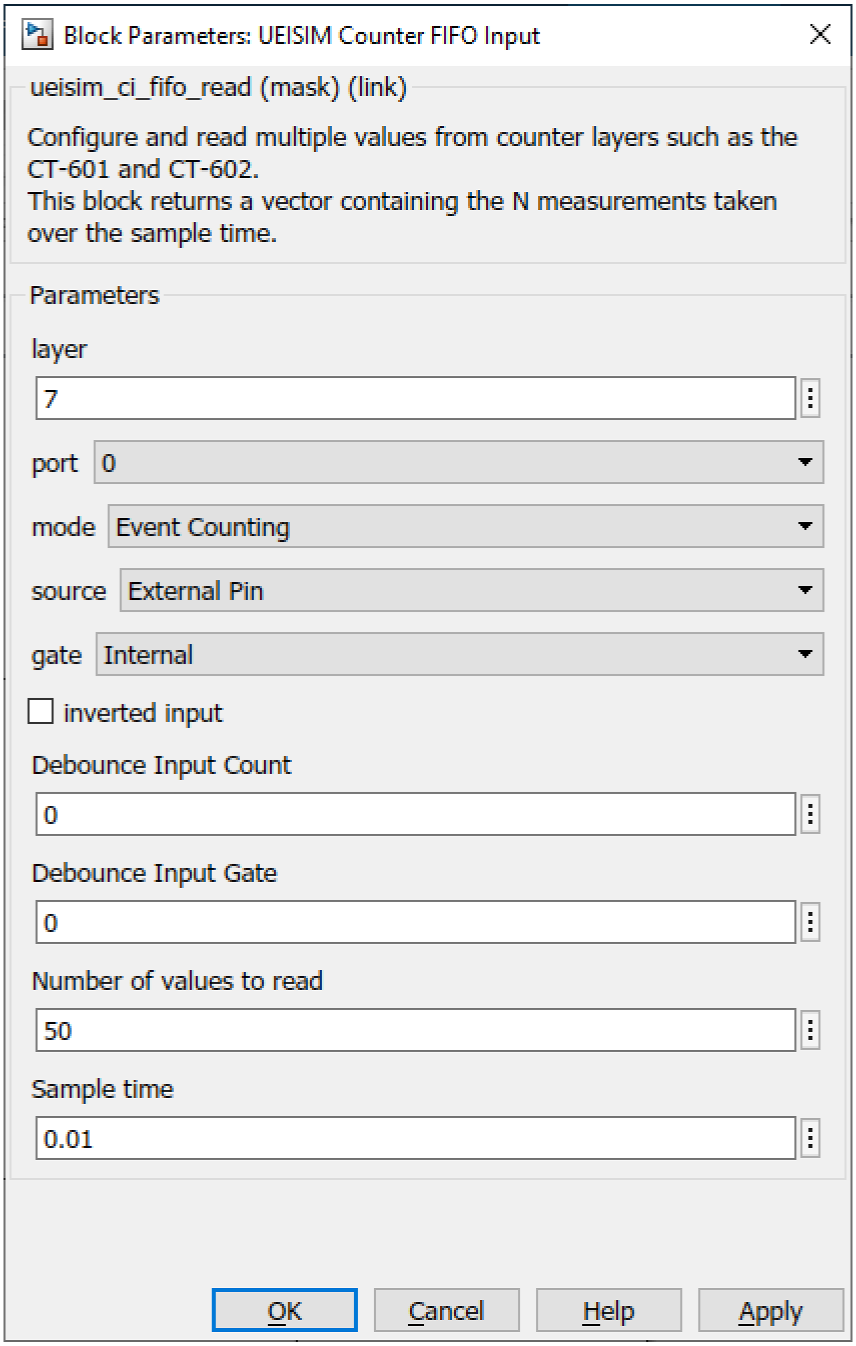
The Counter FIFO input block reads multiple values from the counter’s input FIFO.
The counter pushes values in its FIFO at a rate specified by the block’s sample time and the number of values to read at each time step.
For example with sample time of 0.01s and number of values at 20, the counter will take measurements with an interval of (0.01/20)=0.0005s = 500us.

The value read depends on the counter operating mode:

- Event Counting: Each measurement pushes one value in the FIFO: the number of rising edges detected during the measuring interval.
- TPPM: Each measurement pushes two values in the FIFO: the number of periods counted, the duration of those periods in 66MHz clock ticks

United Electronic Industries

The High-Performance Alternative



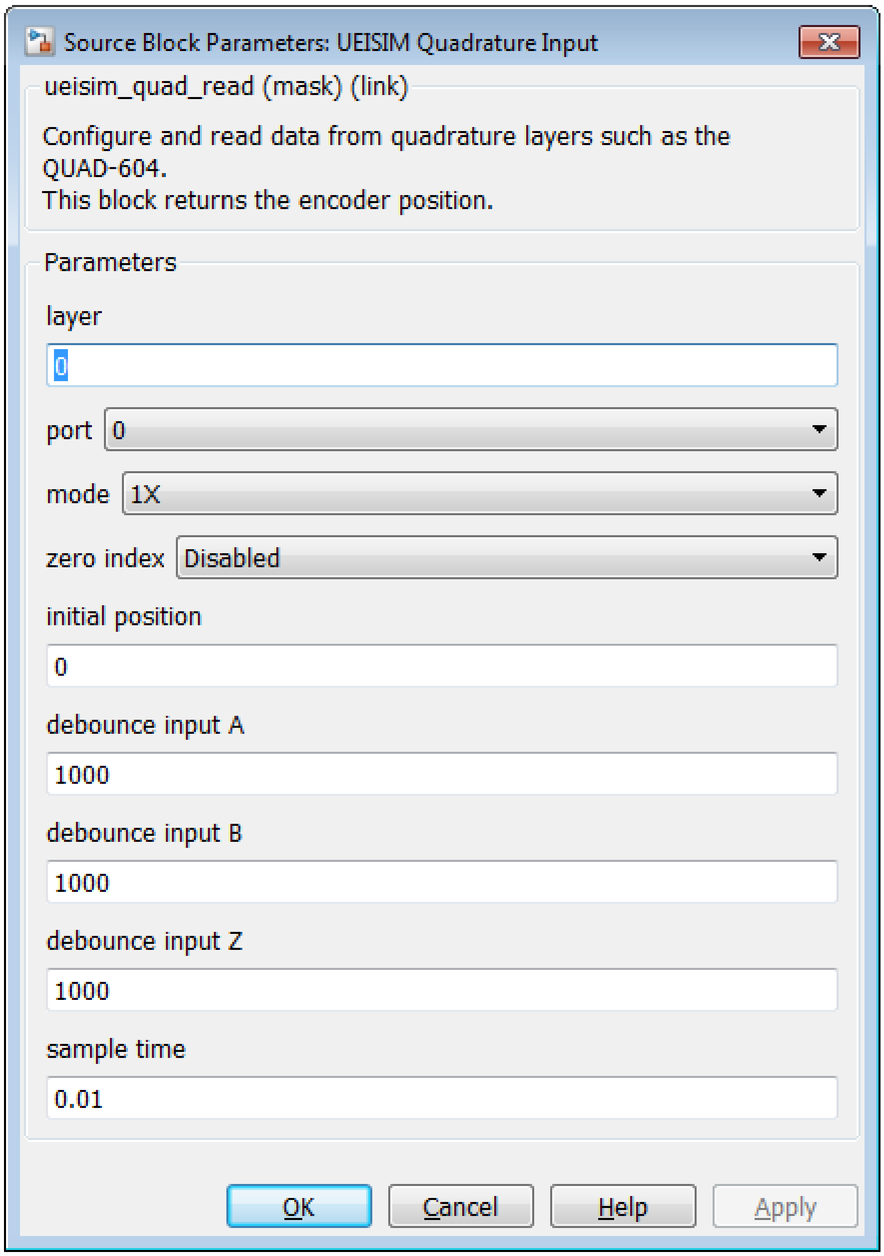
- **layer**: The Id of the counter input layer associated with this block (layer Ids start at 0 with the top layer)
- **port**: The port to read from.

The High-Performance Alternative

- **mode**: The operation mode. Possible values are “Count Events and “TPPM”.
- **source**: The source of the input signal. Possible values are “Internal Clock” and “External Pin”.
- **gate**: The source of the gate signal. Possible values are “Internal” and “External”.
- **inverted input**: the input signal is inverted when this is checked.
- **debounce input count**: the minimum pulse width to accept on counter input. Value is specified in 66Mz ticks. Smaller pulses are rejected.
- **debounce gate count**: the minimum pulse width to accept on gate input. Value is specified in 66Mz ticks. Smaller pulses are rejected
- **number of values:** The number of measurements pushed in the FIFO during a time step.
- **sample Time**: The rate at which the block executes during simulation.

5.15. Quadrature Input block

The quadrature input block is designed to work with devices specialized in quadrature encoder position measurement such as the QUAD-604.
It gives access to additional parameters when compared with the basic counter input block.



- **layer**: The Id of the quadrature input layer associated with this block (layer Ids start at 0 with the top layer)
- **port**: The port to read from.
- **mode**: The mode used to decode position from the quadrature signals. Possible values are 1x (one position per input signal period), 2x (two positions per period) and 4x (four positions per period).

The High-Performance Alternative

- **zero index**: Specifies the states of A, B and Z inputs that will generate a zero index event.
- **initial position**: The initial value of the position measured.
- **debounce input A**: the minimum pulse width to accept on A input. Value is specified in 16.5Mz ticks.
- **debounce input B**: the minimum pulse width to accept on B input. Value is specified in 16.5Mz ticks.
- **debounce input Z**: the minimum pulse width to accept on Z input. Value is specified in 16.5Mz ticks.
- **sample Time**: The rate at which the block executes during simulation (it also sets the hardware clock).

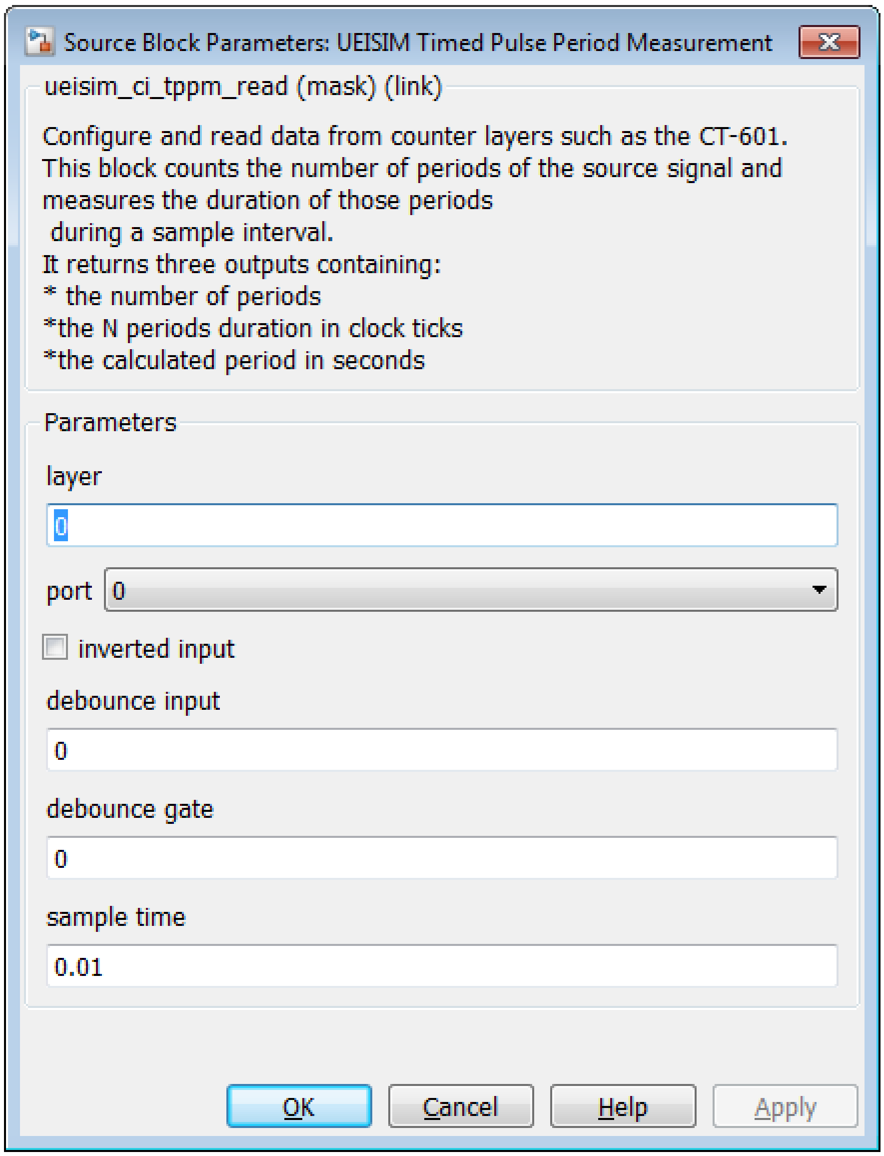
The type of the signals connected to the quadrature input block output must be “uint32”. You can use Simulink’s “Data Type Conversion block” to convert your signal

5.16. Timed Pulse Period Measurement

This block measures periods over a specific duration. It counts the number of periods in the source signal and their duration during a sample interval.
The block outputs three values: The number of periods counted, the duration of those periods in 66MHz clock ticks and the calculated period in seconds.

United Electronic Industries

The High-Performance Alternative



- **layer**: The Id of the counter input layer associated with this block (layer Ids start at 0 with the top layer)
- **port**: The port to read from.
- **inverted input**: the input signal is inverted when this is checked.
- **debounce input**: the minimum pulse width to accept on counter input. Value is specified in 66Mz ticks. Smaller pulses are rejected.
- **debounce gate**: the minimum pulse width to accept on gate input. Value is specified in 66Mz ticks. Smaller pulses are rejected

The High-Performance Alternative

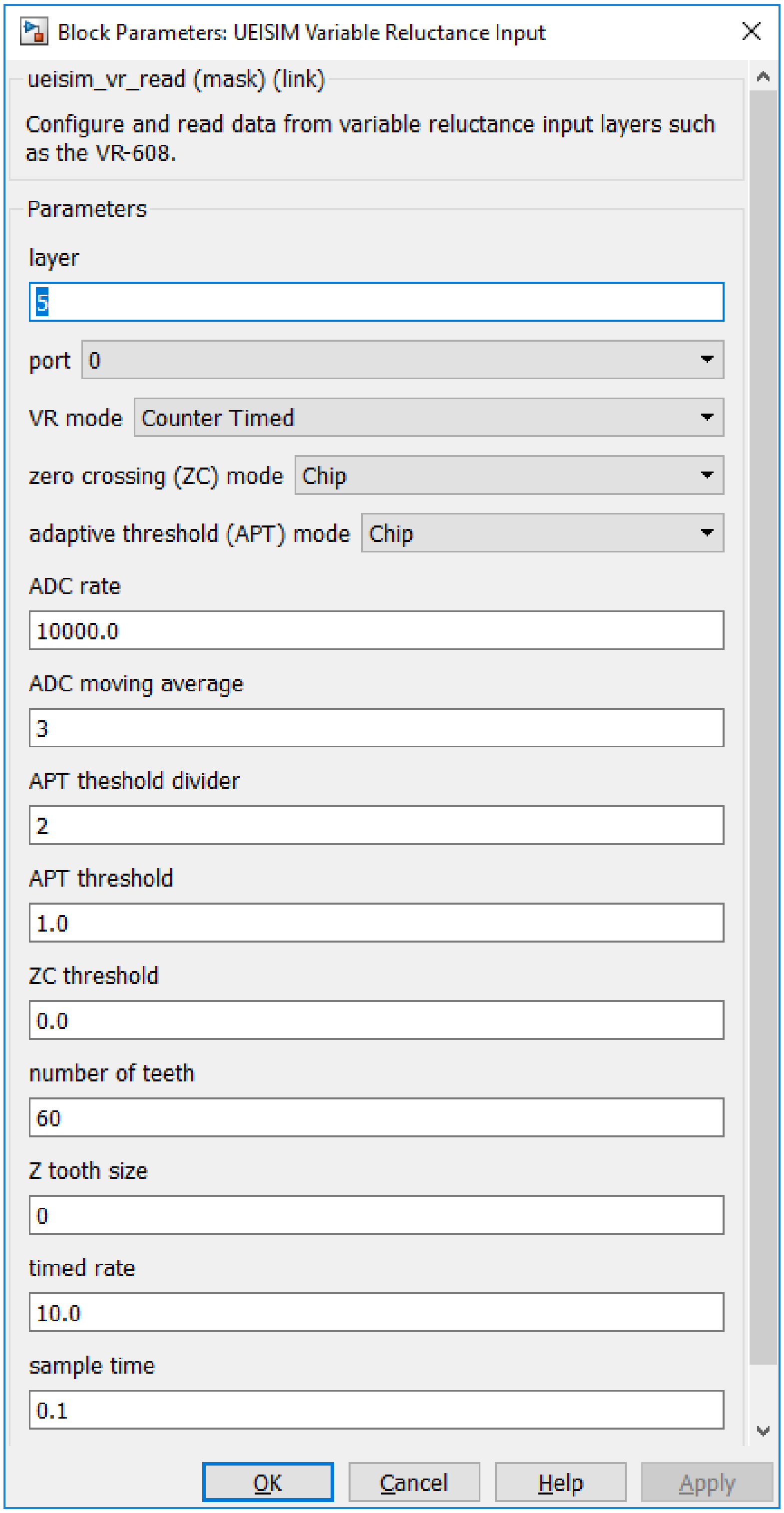
- **Sample Time**: The rate at which the block executes during simulation (it also sets the hardware clock).

The “number of periods” output data type must be “uint32”.
The “duration” output data type must be “uint32”.
The “calculated period” output data type must be “double”.

You can use Simulink’s “Data Type Conversion block” to convert your signal

5.17. Variable Reluctance Measurement

This block measures velocity and position from the signal generated by a variable reluctance sensor.
The block outputs three values: The velocity in RPM, the position (in number of teeth from Z tooth) and the total teeth count since model was started.


Block Parameters: UEISIM Variable Reluctance Input
ueisim_vr_read (mask) (link)
Configure and read data from variable reluctance input layers such as the VR-608.
Parameters
layer
5
port
0
VR mode
Counter Timed
zero crossing (ZC) mode
Chip
adaptive threshold (APT) mode
Chip
ADC rate
10000.0
ADC moving average
3
APT theshold divider
2
APT threshold
1.0
ZC threshold
0.0
number of teeth
60
Z tooth size
0
timed rate
10.0
sample time
0.1
OK
Cancel
Help
Apply

The High-Performance Alternative

- **layer**: The Id of the counter input layer associated with this block (layer Ids start at 0 with the top layer)
- **port**: The port to read from.
- **VrMode**: The mode used to measure velocity, position or direction. The mode can be set to:
  Decoder: Even and Odd channels are used in pair to determine direction and position
  Timed: Count number of teeth detected during a timed interval
  Npulses: Measure the time taken to detect N teeth (Number of teeth needs to be set)
  Zpulse: Measure the number of teeth and the time elapsed between two Z pulses (The Z tooth is usually a gap or a double tooth on the encoder wheel)
- **ZcMode**: Zero crossing finds the point in time where the VR sensor output voltage transitions from positive to negative voltage. This point is when the center of the tooth is lining up with the center of the VR sensor. The zero crossing mode can be set to:
  Chip: The front-end IC will automatically calculate the ZC level
  Logic: The device's FPGA measures the VR sensor signal and calculate the ZC level as (min+max)/2
  Fixed: Use hard-coded ZC level (specified below)
- **APTMode**: APT finds the point in time where the VR sensor output voltage falls below a certain threshold. This point marks the beginning of the gap between two teeth. The APT mode can be set to:
  Chip: The front-end IC will automatically set the AP threshold to 1/3 of the peak input voltage
  Logic: The device's FPGA measures the VR sensor signal and sets the AP threshold to a programmable fraction of the peak input voltage
  Fixed: Use hard-coded AP threshold (specified below)
- **ADCRate**: The rate in Hz at which the VR sensor signal is measured.
- **MovingAverage**: The size of the moving average window applied to the VR sensor signal while it is measured.
- **APTThresholdDivider**: The APT threshold divider is used when APT mode is set to "Logic". It specifies that the AP threshold will be set at a fraction of the peak input voltage. This is a value between 1 and 15: 1=1/2, 2=1/4, 3=1/8 etc...
- **APTThreshold**: The APT threshold is used when APT mode is set to "Fixed".
- **ZCThreshold**: The ZC threshold is used when ZC mode is set to "Fixed".
- **NumberOfTeeth**: The number of teeth on the encoder wheel.

The High-Performance Alternative

- **ZToothSize**: A Z Tooth is usually materialized by one or more missing teeth or one or more fused teeth. This parameter specified the number of fused or missing teeth.
- **TimedRate**: The rate at which teeth are counted when VrMode is set to “timed”.

The “velocity” output data type must be “double”.
The “position” output data type must be “uint32”.
The “teeth count” output data type must be “uint32”.

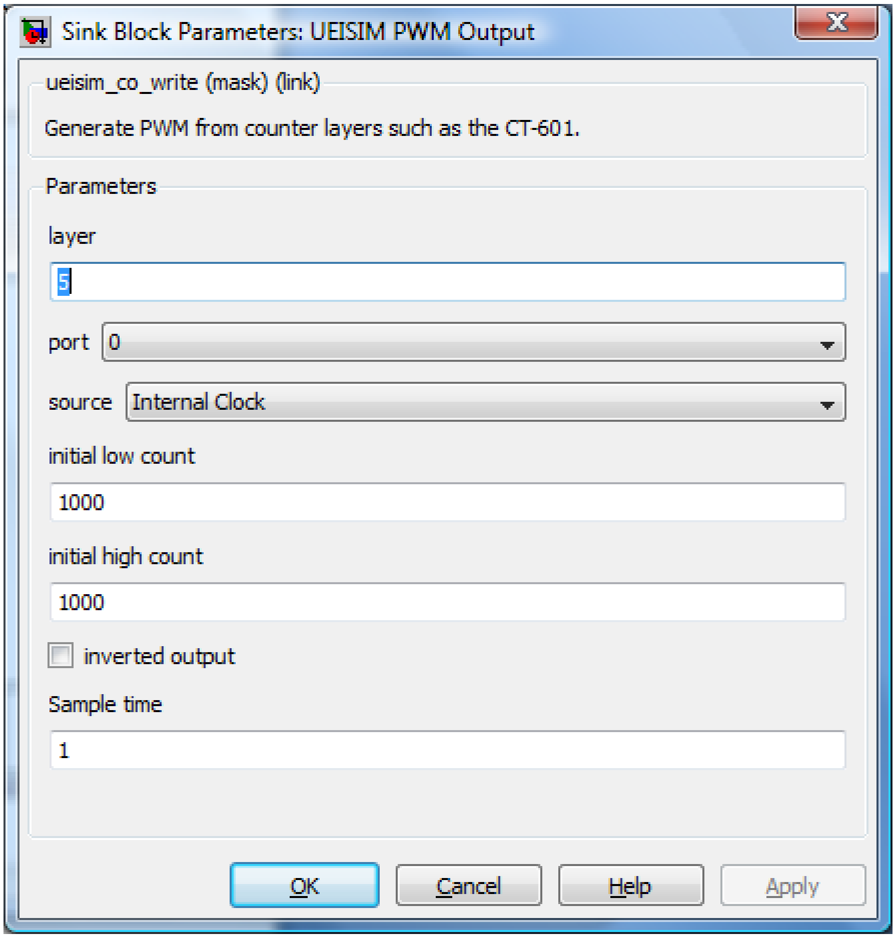
You can use Simulink’s “Data Type Conversion block” to convert those signals to a type that best fits your model.

5.18. PWM Output block

The PWM output block generates a continuous train of pulses out of the specified timer.
Use one instance of this block for each timer you wish to use as output.
The data type is uint32.

This block contains two inputs: The new low state width (in clock ticks) and the new high state width (in clock ticks) of each pulse.

The High-Performance Alternative



- **layer**: The Id of the counter output layer associated with this block (layer Ids start at 0 with the top layer)
- **port**: The port to read from.
- **source**: The source of the clock signal. Possible values are "Internal Clock" and "External Pin".
- **initial low count**: The initial width of each pulse low state in clock ticks.
- **initial high count**:The initial width of each pulse high state in clock ticks.
- **inverted output**: the output signal is inverted when this is checked.
- **Sample Time**: The rate at which the block executes during simulation (it also sets the hardware clock).

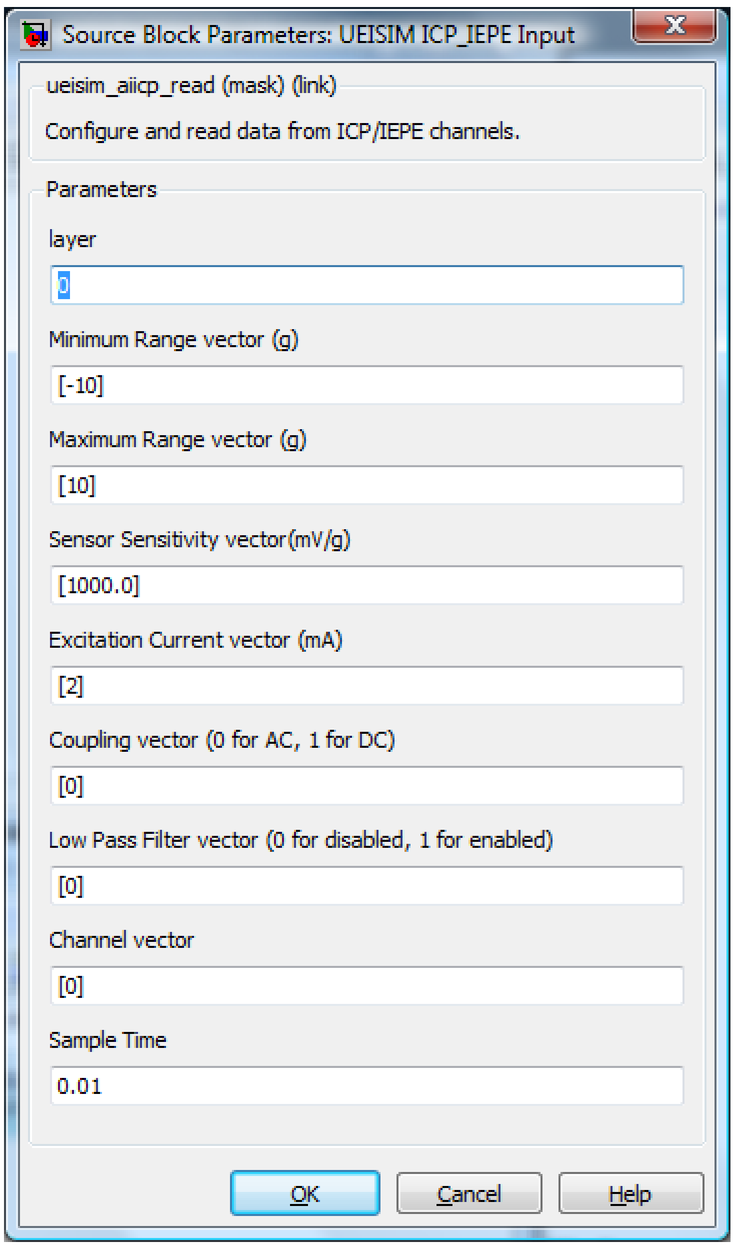
The type of the signals connected to the CO block must be "uint32". You can use Simulink's "Data Type Conversion block" to convert your signal

The High-Performance Alternative

5.19. ICP/IEPE block

Use the ICP/IEPE block to acquire data from ICP or IEPE sensors. Those sensors are only supported by analog input hardware that can provide excitation current to power the sensors (for example the AI-211).

The data type of the value returned for each configured channel is double.



United Electronic Industries
The High-Performance Alternative

- **layer**: The Id of the analog input layer associated with this block. (layer Ids start at 0 with the top/left layer)
- **Minimum Range vector**: The minimum value expected at the input of each channel
- **Maximum Range vector**: The maximum value expected at the input of each channel
- **Sensor Sensitivity vector**: The sensitivity of the sensor(s) connected to each channel
- **Excitation Current vector**: The excitation current used to power sensor(s) connected to each channel
- **Coupling vector**: The coupling (AC or DC) used on each channel
- **Low Pass Filter vector**: Turns on or off the anti-aliasing low pass filter on each channel
- **Channel vector**: Array of channels to acquire from
- **Sample Time**: The rate at which the block executes during simulation (it also sets the hardware ADC clock).

5.20. LVDT

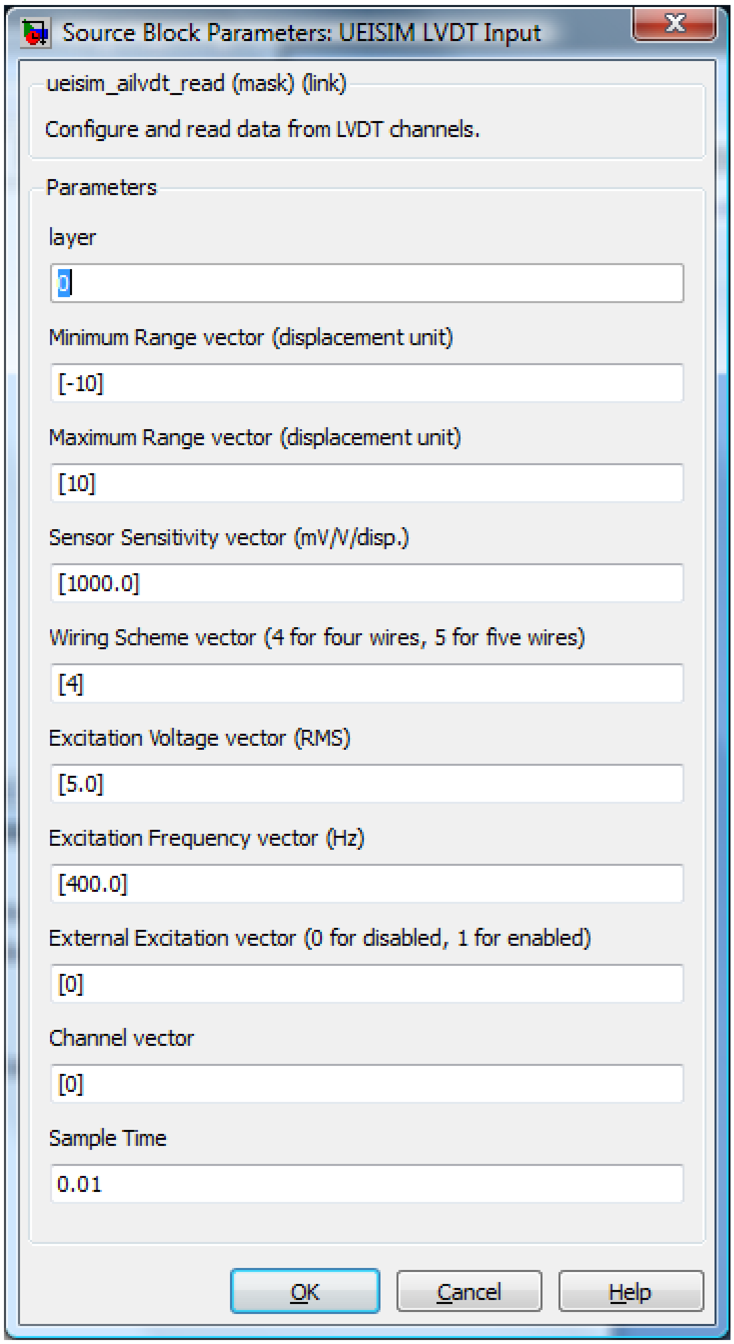
Use the LVDT blocks to acquire data from LVDT sensors and also simulate voltage emitted by real LVDT sensors.
Those sensors are only supported by analog input hardware that can provide excitation current to power the LVDTs (for example the AI-254).

5.20.1. LVDT Input block

The data type of the value returned for each configured channel is double.

The unit of the values read by this block is a displacement and depends on the sensor sensitivity unit.

For example, if you specify sensor sensitivity in mV/V/mm, the values read are millimeters.
With sensitivity set to 1000 mV/V/mm you will measure a displacement of -1mm to +1mm when moving the LVDT sensor across its full range.



- **layer**: The Id of the analog input layer associated with this block. (layer Ids start at 0 with the top/left layer)

The High-Performance Alternative

- **Minimum Range vector**: The minimum value expected at the input of each channel
- **Maximum Range vector**: The maximum value expected at the input of each channel
- **Sensor Sensitivity vector**: The sensitivity of the LVDT(s) connected to each channel
- **Wiring Scheme vector**: The wiring scheme (4 or 5 wires) used to connect LVDT(s) to each channel
- **Excitation Voltage vector**: The excitation voltage used to power LVDT(s) connected to each channel
- **Excitation Frequency vector**: The excitation frequency used to power LVDT(s) connected to each channel
- **External Excitation vector**: Specifies whether channel(s) provide excitation to LVDT(s) or whether excitation is supplied externally
- **Channel vector**: Array of channels to acquire from
- **Sample Time**: The rate at which the block executes during simulation (it also sets the hardware ADC clock).

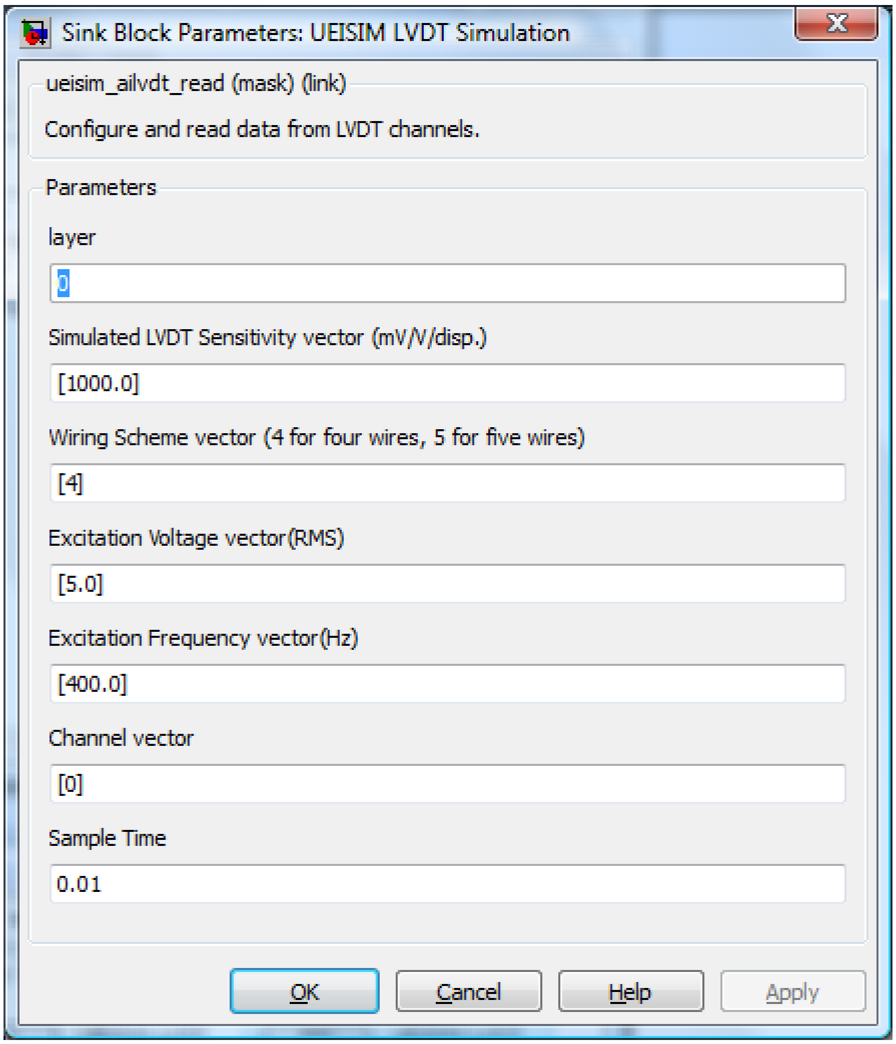
5.20.2. LVDT Simulation block

The data type of the value written to each configured channel is double

The unit of the value to simulate is a displacement and depends on the sensor sensitivity unit.

For example, if you set sensor sensitivity in mV/V/mm, the values written to the block must be specified in millimeters.

With sensitivity set to 1000 mV/V/mm, the values written to this block must be in the range [-1,+1] to simulate an LVDT sensor with a full range of -1mm to +1mm.



- **layer**: The Id of the analog input layer associated with this block. (layer Ids start at 0 with the top/left layer)
- **Simulated LVDT Sensitivity vector**: The sensitivity of the LVDT(s) simulated by each channel
- **Wiring Scheme vector**: The wiring scheme (4 or 5 wires) used to connect the LVDT(s) simulated by each channel
- **Excitation Voltage vector**: The excitation voltage used to power LVDT(s) simulated by each channel
- **Excitation Frequency vector**: The excitation frequency used to power LVDT(s) simulated by each channel
- **Channel vector**: Array of channels to simulate from

- **Sample Time**: The rate at which the block executes.

5.21. Synchro/Resolver

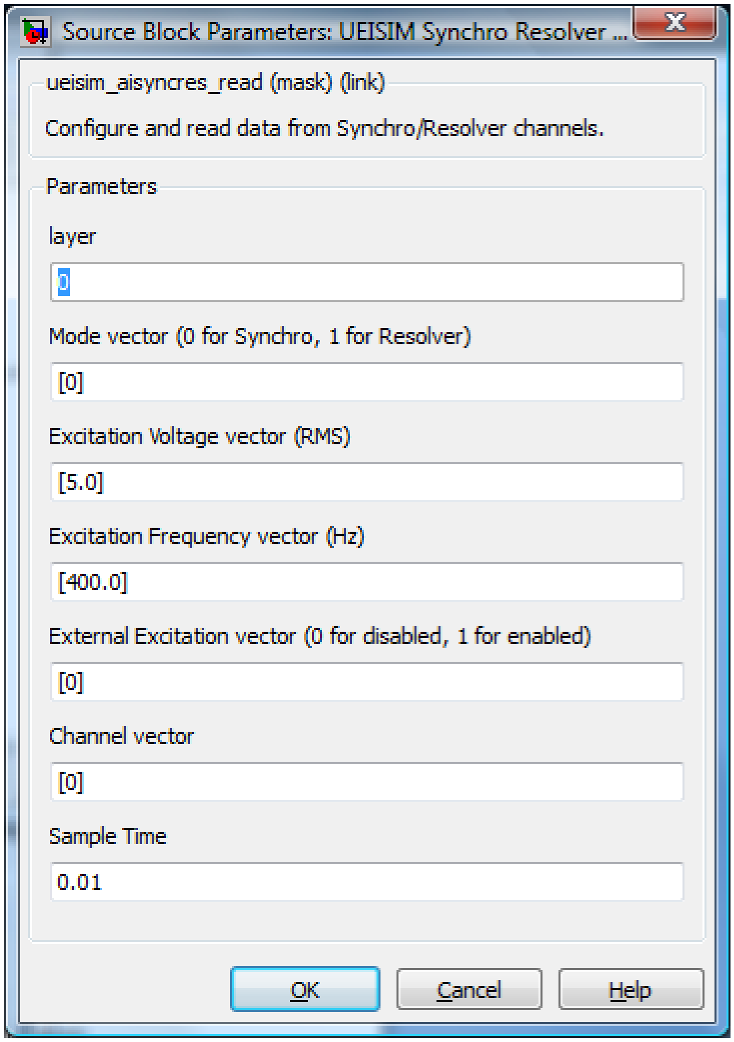
Use the Synchro/Resolver blocks to acquire data from Synchros or Resolvers and also simulate voltage emitted by real Synchros or Resolvers.
Those sensors are only supported by analog input hardware that can provide excitation current to power the Synchro/Resolvers (for example the AI-255 or AI-256).

5.21.1. Synchro/Resolver Input block

The data type of the value returned for each configured channel is double.

Measurements are returned as angles in radian.

The High-Performance Alternative



- **layer**: The Id of the analog input layer associated with this block. (layer Ids start at 0 with the top/left layer)
- **Mode vector**: Specifies whether a Synchro or a Resolver is connected to each channel
- **Excitation Voltage vector**: The excitation voltage used to power Synchro/Resolvers(s) connected to each channel
- **Excitation Frequency vector**: The excitation frequency used to power Synchro/Resolver(s) connected to each channel
- **External Excitation vector**: Specifies whether channel(s) provide excitation to Synchro/Resolver(s) or whether excitation is supplied externally
- **Channel vector**: Array of channels to acquire from

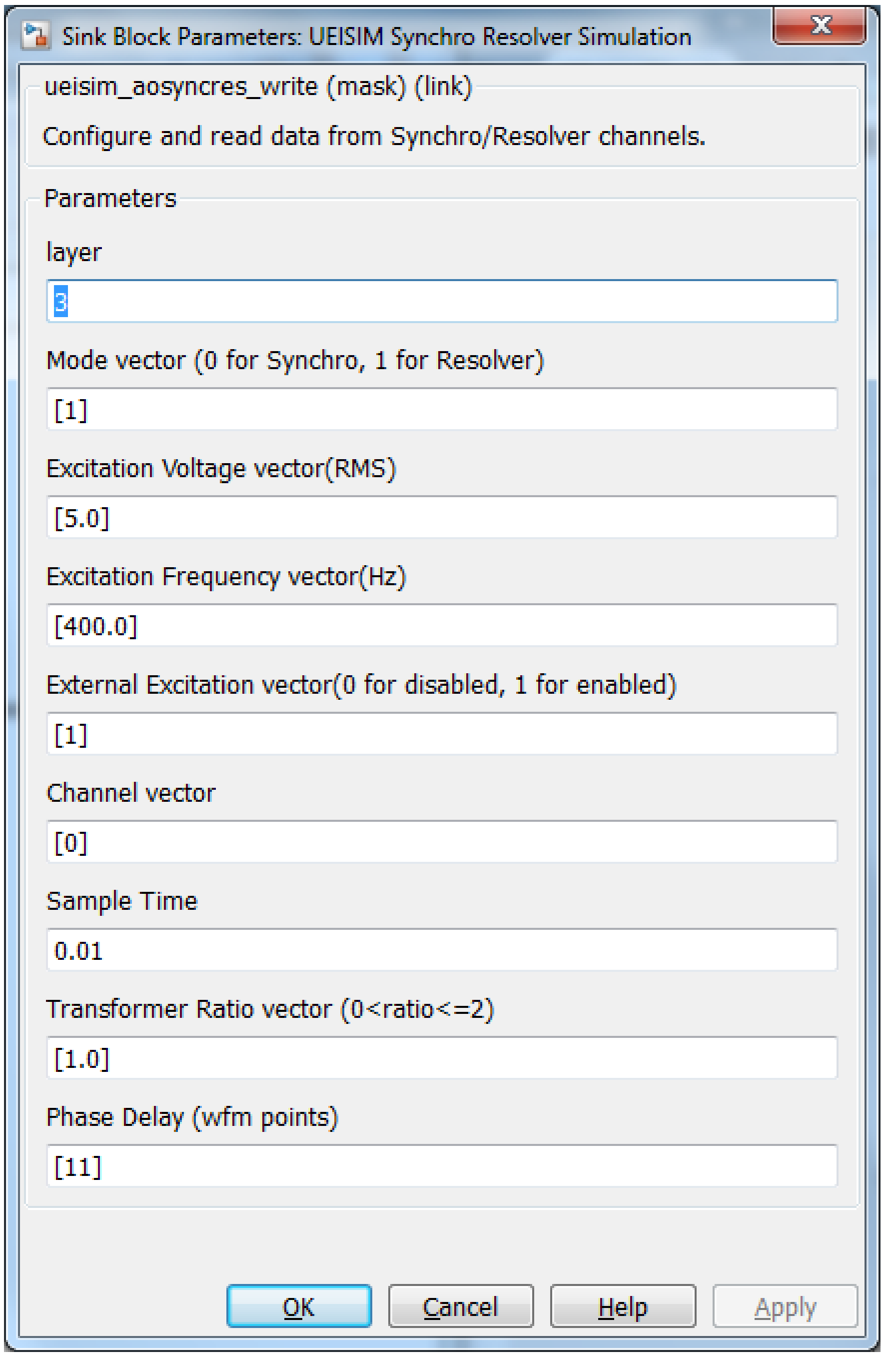
The High-Performance Alternative

- **Sample Time**: The rate at which the block executes during simulation (it also sets the hardware ADC clock).

5.21.2. Synchro/Resolver Simulation block

The data type of the value written to each configured channel is double

The value must be specified as an angle in radian.



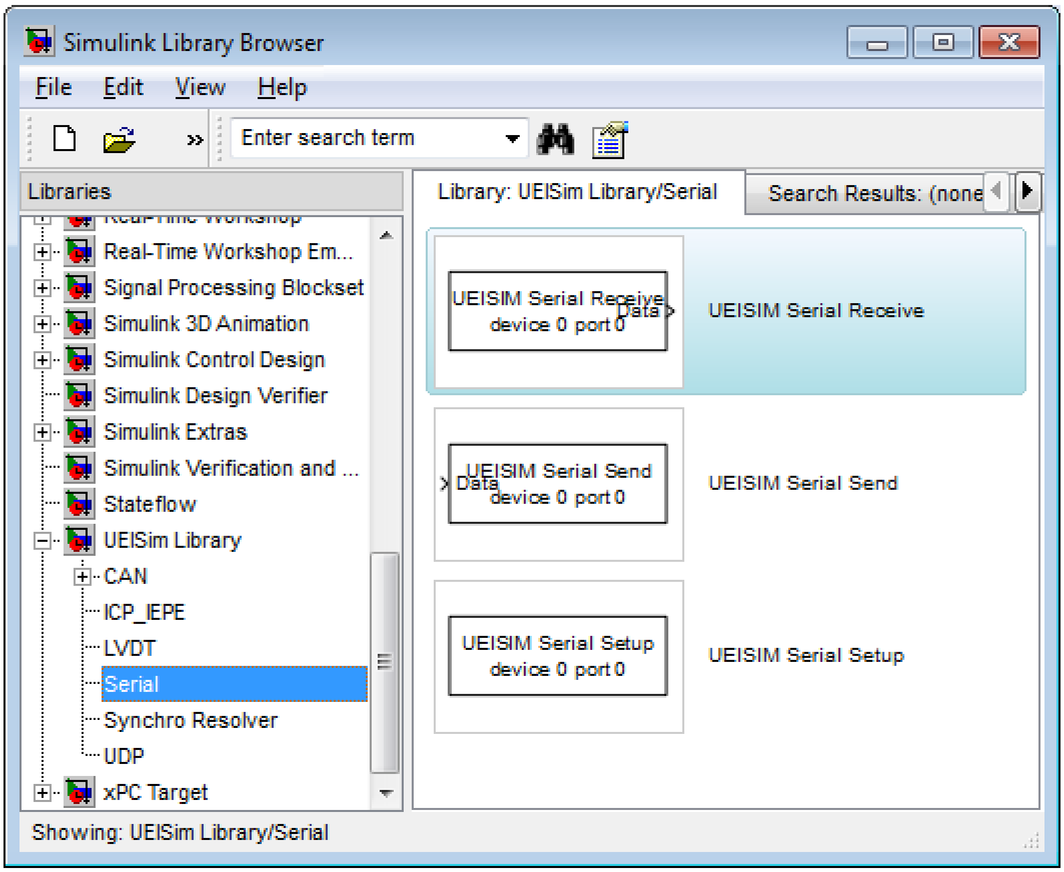
- **layer**: The Id of the analog input layer associated with this block. (layer Ids start at 0 with the top/left layer)

The High-Performance Alternative

- **Mode vector**: Specifies whether each channel is simulating a Synchro or a Resolver
- **Excitation Voltage vector**: The excitation voltage used to power Synchro/Resolver(s) simulated by each channel
- **Excitation Frequency vector**: The excitation frequency used to power Synchro/Resolver(s) simulated by each channel
- **External Excitation vector**: Specifies whether channel(s) provide excitation or whether excitation is supplied externally
- **Channel vector**: Array of channels to simulate from
- **Sample Time**: The rate at which the block executes
- **Transformer Ratio Vector**: Sets the ratio to apply to simulated waveforms amplitude. For example if excitation amplitude is 10vpp and ratio is 0.5. The simulated waveforms amplitude will be 5vpp
- **Phase Delay**: Sets the phase delay between the excitation and the simulated waveforms. Value is specified in number of samples of the simulated waveform. For example if the card is using 32 points to output one waveform cycle, a phase delay of 8 is equivalent to a 90 deg. phase shift.

5.22. Serial port communication

Serial communication blocks give access to the SL-501 and SL-508 serial ports. The configuration of each port is done using an independent setup block.
Sending and receiving bytes to/from a port is done using a send or receive block.



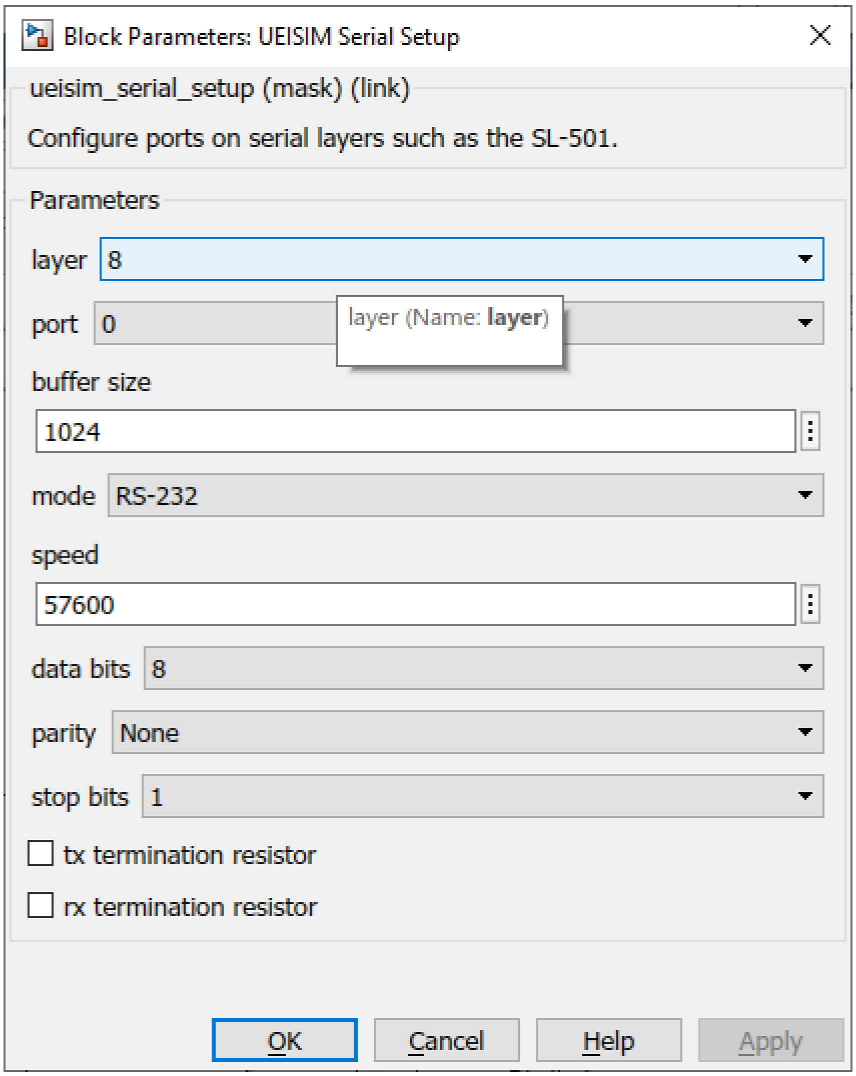
5.22.1. Serial Setup block

Configure communication settings on a given Serial port.

The setup block needs to run before the Send/Receive blocks are called (otherwise an error will be returned during model execution).
To view/change the execution context order: Select the menu option **Format > Block Displays > Sorted Order** and make sure that the setup block has a priority lower than the send and receive block for the same port.
To change a block priority: Right-click the block and select **Block Properties**. On the General tab, in the Priority field, enter the new priority.

There must be one setup block for each serial port used in the model.

The High-Performance Alternative

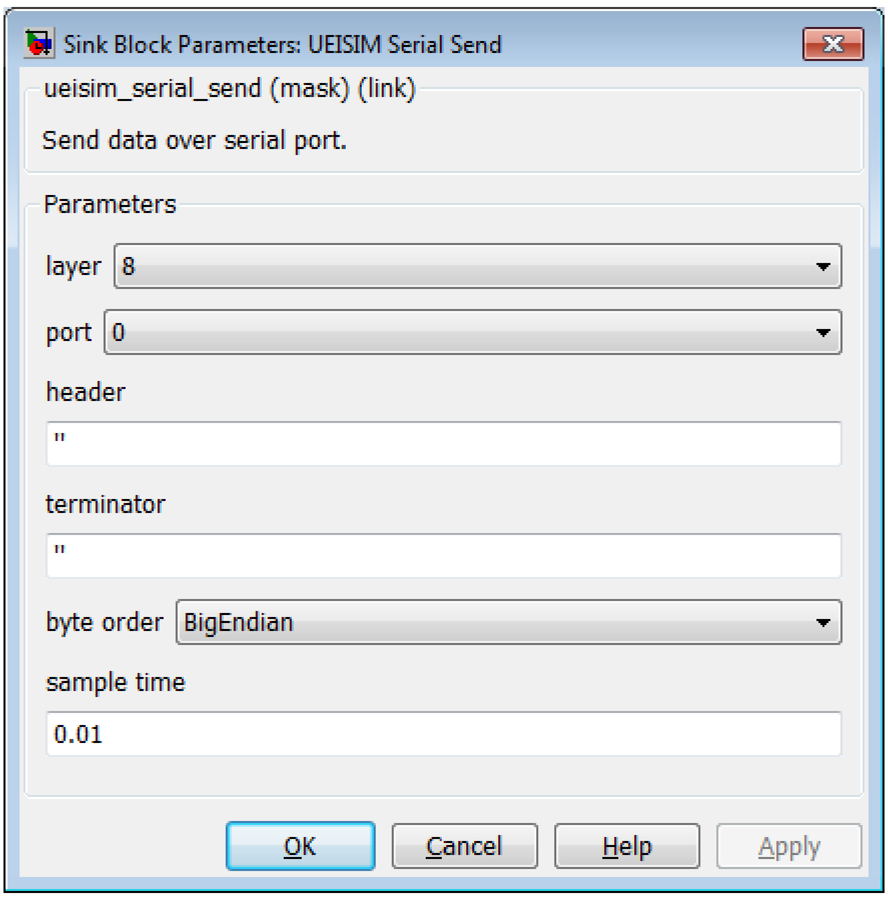


- **Layer**: The Id of the Serial layer associated with this block (layer Ids start at 0 with the top layer)
- **Port**: The Id of the port to configure (port Ids start at 0)
- **Buffer size**: Size in bytes of the send/receive buffers (determines the maximum number of bytes able to be received or sent)
- **Mode**: The serial link mode (RS-232/RS-485 HD/RS-485FD)
- **Speed**: The baud rate of the serial link
- **Data bits**: The number of data bits in each transmitted frame
- **Parity**: The method used to calculate the parity bit
- **Stop bits**: The number of stop bits in each transmitted frame

- **Tx termination resistor**: Enable/Disable termination resistor between Tx- and Tx+ (RS-485 mode only).
- **Rx termination resistor**: Enable/Disable termination resistor between Rx- and Rx+ (RS-485 mode only).

5.22.2. Serial Send block

Send a bytes to one Serial port. You can create multiple instance of this block to send data to the same port at different rate.



- **Layer**: The Id of the Serial layer associated with this block (layer Ids start at 0 with the top layer)
- **Port**: The Id of the port to send data through (port Ids start at 0)
- **Header**: String of bytes to be sent before the data
  Use the string notation (between single quotes) if the header uses printable characters. Otherwise for non-printable characters use a vector of chars.

The High-Performance Alternative

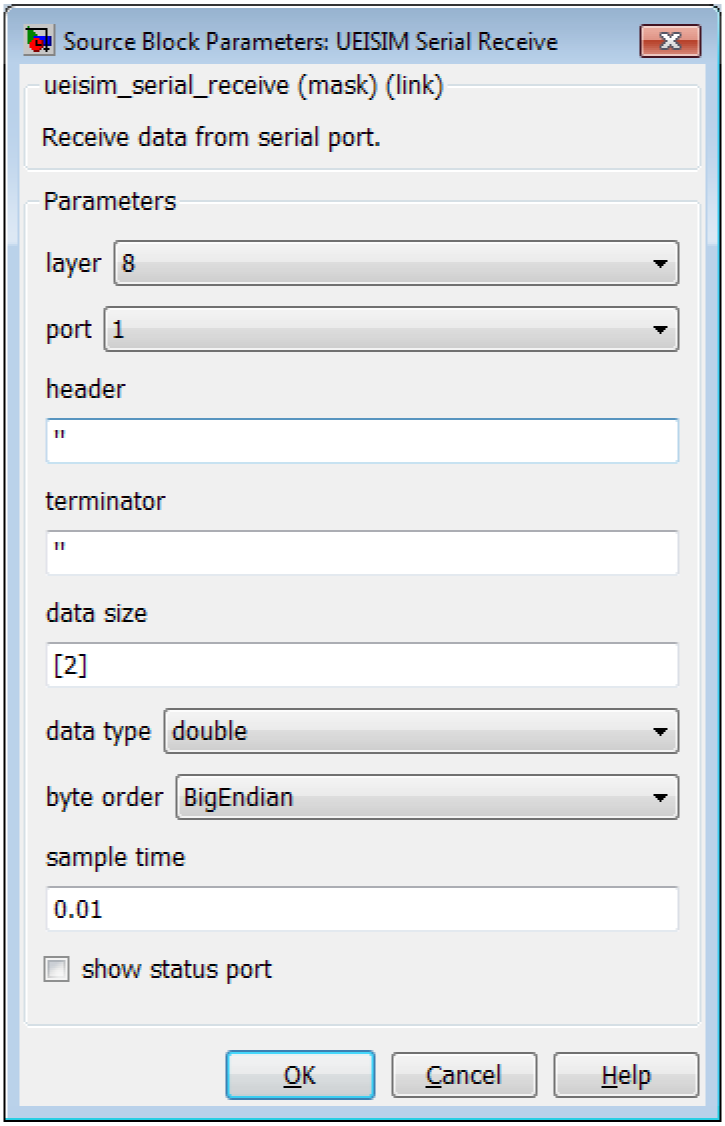
- **Terminator**: String of bytes to be sent after the data
  Use the string notation (between single quotes) if the terminator uses printable characters. Otherwise for non-printable characters use a vector of chars.
- **Byte Order**: The endianness used to convert signal(s) to bytes.
- **Sample Time**: The rate at which the block executes during simulation

The block displays an input port for connecting the value to send through the serial port, it automatically adapts to the data type and dimension of the signal connected.

Use the mux block to combine multiple signals that needs to be sent together.

5.22.3. Serial Receive block

Receives bytes from a serial port. You can create multiple instance of this block to receive data from different ports.



- **Layer**: The Id of the Serial layer associated with this block (layer Ids start at 0 with the top layer)
- **Port**: The Id of the port to send data through (port Ids start at 0)
- **Header**: String of bytes that signals the beginning of a data frame
  Use the string notation (between single quotes) if the header uses printable characters. Otherwise for non-printable characters use a vector of chars.
- **Terminator:** String of bytes that signals the end of a data frame
  Use the string notation (between single quotes) if the terminator uses printable characters. Otherwise for non-printable characters use a vector of chars.

The High-Performance Alternative

- **Data Size**: Dimension and size of the output signal (for ex [2 4] will output received data in a 2x4 matrix)
- **Data Type**: The data type used to decode received data
- **Byte Order**: The endianness used to convert received bytes to signal(s).
- **Sample Time**: The rate at which the block executes during simulation
- **Show Status Port**: Enable/disable status reporting

The block displays two output ports:
- Data: The signals extracted from the packet payload.
- Status: The status (see below).

The status output when enabled can take any of the following values:
- 0: No bytes were received
- N: Number of bytes received
- -1: A hardware error occurred
- -2: Buffer overrun, The receive block is not executed often enough to keep up with the pace of incoming bytes

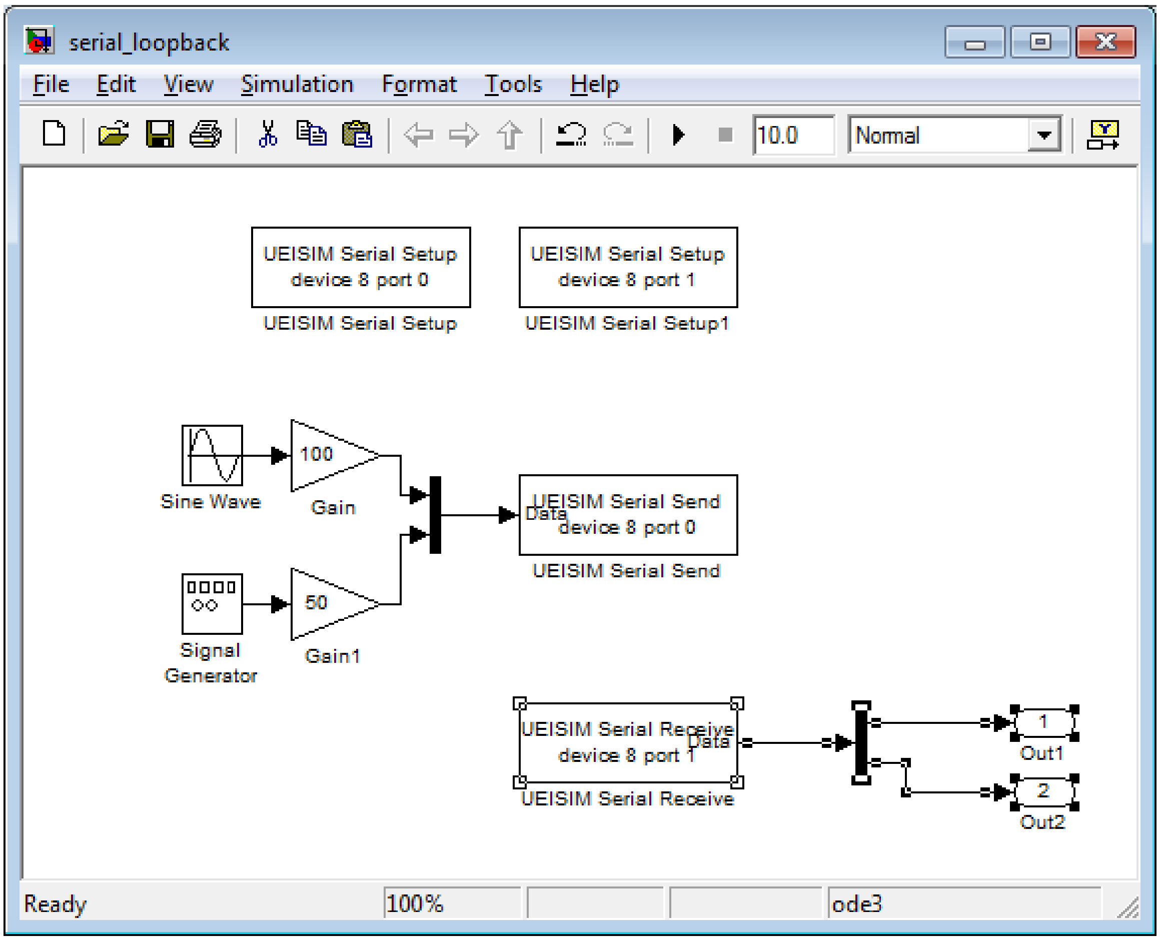
The data output port always returns a signal with the dimension specified by "Data size" parameter. However the number of values read might be less than the signal capacity. Use the status port to figure out how many values were actually read.

Use the demux block to separate received data into individual signals.

5.22.4. Serial example

The following example sends simulated data to one port receive data from another port. This example will read back the data sent if both ports are connected with a NULL modem cable.

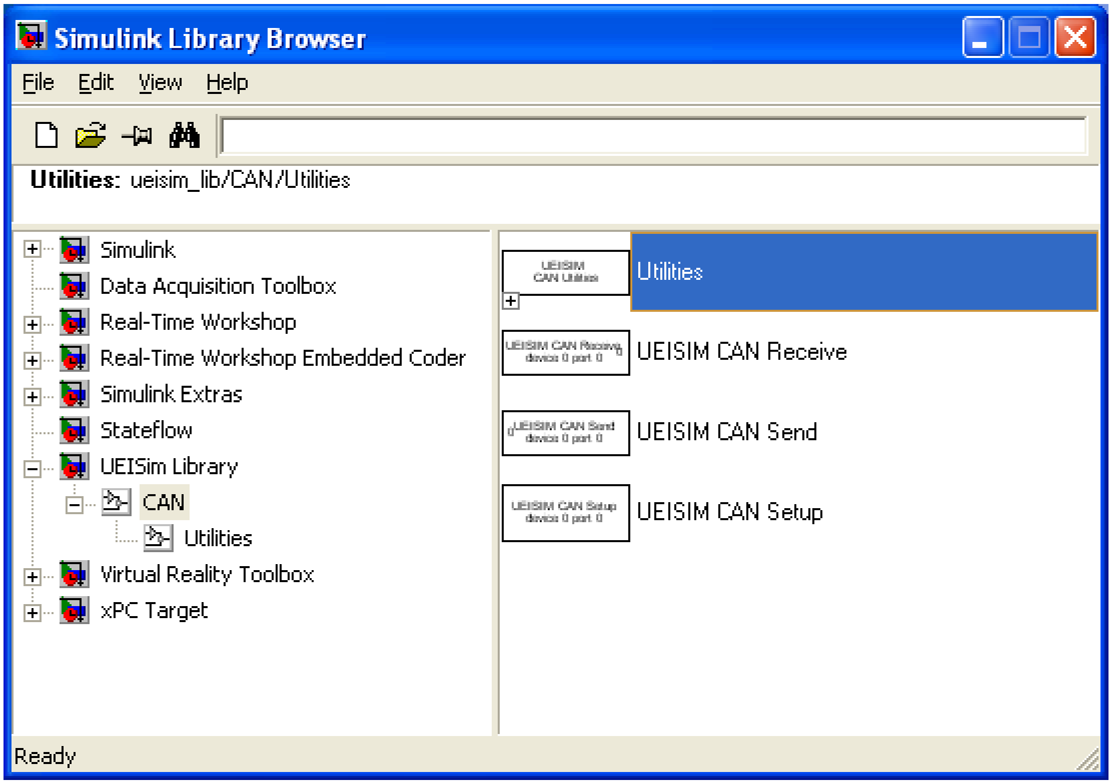
The High-Performance Alternative



5.23. CAN bus communication

CAN communication blocks give access to the CAN-503 CAN ports. The configuration of each port is done using an independent setup block.
Sending and receiving CAN frames to/from a port is done using a send or receive block.

The High-Performance Alternative

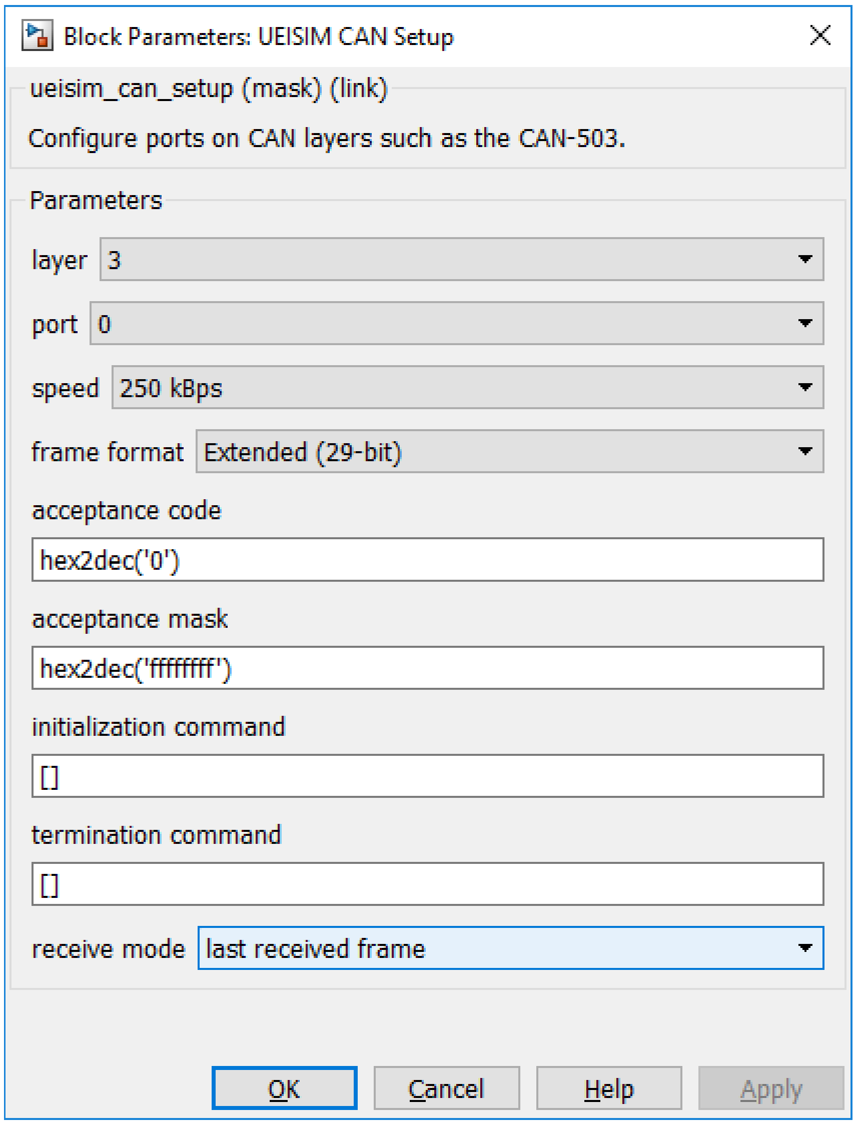


5.23.1. CAN Setup block

Configure communication settings on a given CAN port.

The setup block needs to run before the Send/Receive blocks are called (otherwise an error will be returned during model execution).
To view/change the execution context order: Select the menu option **Format > Block Displays > Sorted Order** and make sure that the setup block has a priority lower than the send and receive block for the same port.
To change a block priority: Right-click the block and select **Block Properties**. On the General tab, in the Priority field, enter the new priority.

There must be one setup block for each port used in the model.



- **layer**: The Id of the CAN layer associated with this block (layer Ids start at 0 with the top layer)
- **port**: The Id of the port to configure (port Ids start at 0)
- **speed**: The speed in bits/s used on the CAN bus connected to this port
- **frame format**: The type of frame sent or received (Standard or Extended)
- **acceptance code**: Acceptance filter code configuration
- **acceptance mask**: Acceptance filter mask configuration
- **initialization command**: A sequence of frames to send to the CAN bus right before the model start.

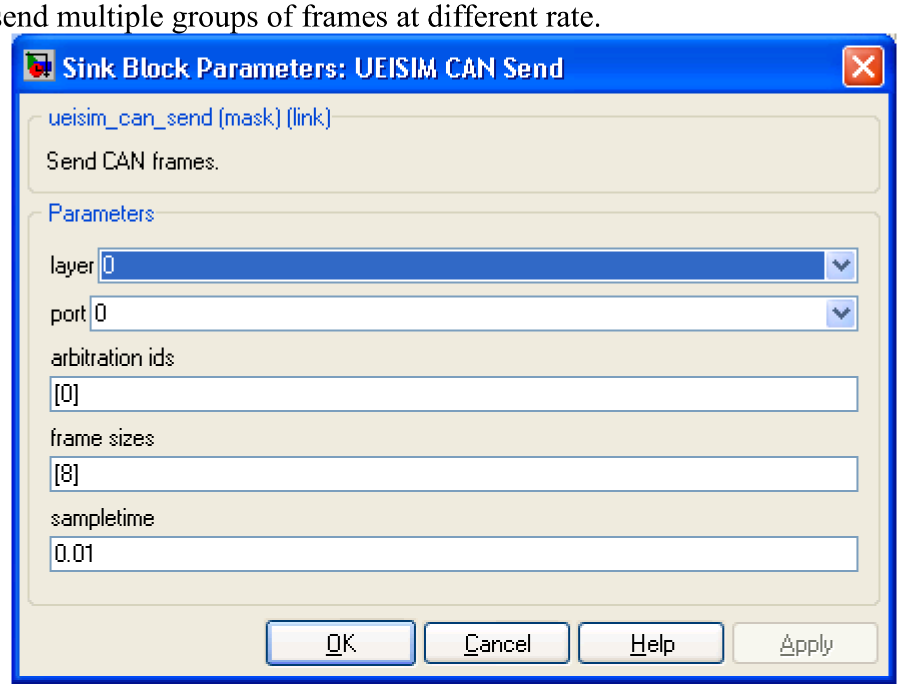
- **termination command**: A sequence of frames to send to the CAN bus right before the model terminates.
- **receive mode**: Selects the method used to process incoming frames.
  FIFO: Incoming frames are stored in a FIFO (one FIFO per arb. ID). The CAN Receive block dequeues received frames from the FIFOs.
  Last Received Frame: The CAN Receive block reads the latest received frame for each configured arb. ID.

The initialization and termination sequences use the following format [id1 len1 dataMSB1 dataLSB1 id2 len2 dataMSB2 dataLSB2 …]. For example to send a CAN frame with ID 0x12 and 5 bytes of data (0x01 0z02 0x03 0x04 0x05) use the following:

```
[ hex2dec('12') 5 hex2dec('05') hex2dec('04030201')]
```

5.23.2. CAN Send block

Send a group of CAN frames to one CAN port. You can create multiple instance of this block to send multiple groups of frames at different rate.



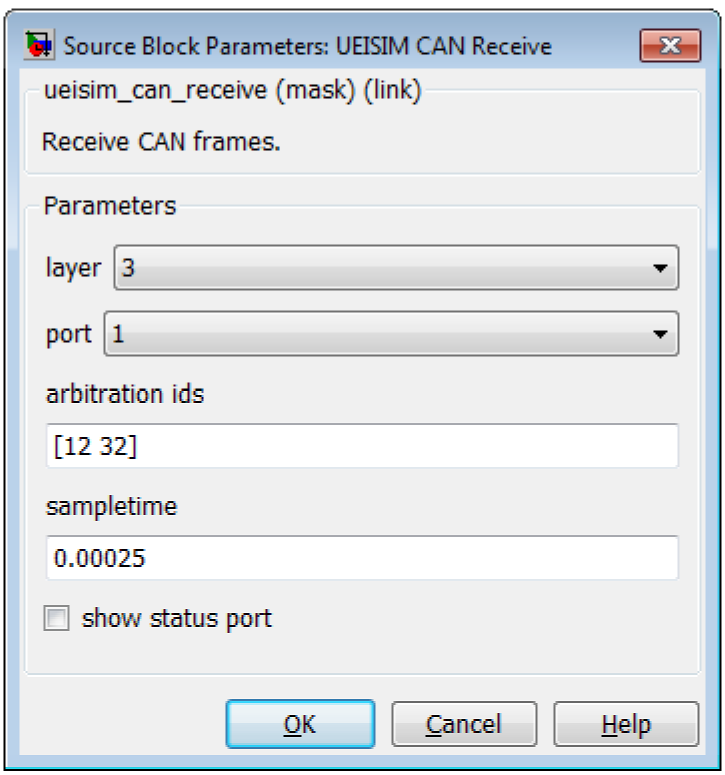
- **layer**: The Id of the CAN layer associated with this block (layer Ids start at 0 with the top layer)
- **port**: The Id of the port to send to (port Ids start at 0)
- **arbitration ids**: A list of arbitration IDs to send
- **frame sizes**: The size of the data payload for each frame
- **sample time**: The rate at which the block executes during simulation

The block displays an input port for connecting the value of the data payload for each frame.
The data payload is specified using the double data type, which is big enough to carry the 64 bits required for a full payload (8 bytes maximum).

Refer to section about packing/unpacking data into payload below.

5.23.3. CAN Receive block

Receive a group of CAN frames from one CAN port. You can create multiple instance of this block to receive multiple groups of frames at different rate.



- **layer**: The Id of the CAN layer associated with this block (layer Ids start at 0 with the top layer)
- **port**: The Id of the port to receive from (port Ids start at 0)
- **arbitration ids**: A list of arbitration IDs to receive
- **sample time**: The rate at which the block executes during simulation
- **Show Status Port**: Enable/disable status reporting

The block outputs the value of the data payload of each frame.
The data payload is specified using the double data type which is big enough to carry the 64 bits required for a full payload (8 bytes maximum).

Refer to section about packing/unpacking data into payload below.

The status output when enabled can take any of the following values:

- 0: No CAN frame was received, the signal output contains the data of the last received frame
- 1: A new CAN frame was received
- -1: A bus error occurred
- -2: Buffer overrun, The receive block is not executed often enough to keep up with the pace of incoming frames

5.23.4. Utility blocks

Utility blocks are used to pack and unpack data stored in the payload of CAN frames that are sent or received. You can specify the data types and position of multiple signals within a single CAN frame.

Each signal is specified using four parameters:

- data type: the type of the signal, possible values are boolean, int8, uint8, int16, uint16, int32, uint32, single or double.
- endianness: the endianness of the signal, possible values are:
  intel for little endian. Bits are counted to the left from the start bit.Bytes are also counted to the left.
  motorola for big endian, Bits are counted to the left from the start bit. Bytes are counted to the right.
  alorotom for backward Motorola format. Bits are counted to the left from the start bit. Bytes are counted to the right and the byte counting sequence is reversed.
- start bit: defines where the least significant bit of a signal's least significant byte is inserted into the message. It is always (even for big endian signals) counted from the start of the message (bit 0), and can be in the range (0..63).
- bit length: the number of bits used to represent the signal in the 8 bytes data payload.

5.23.4.1. Intel format

The least significant bit position, lsb, is specified as the start bit for signals in Intel format. The bits in an Intel CAN message are always counted as described in the layout below:

The High-Performance Alternative

Bit number within a byte									
7	**5**	**6**	**4**	**3**	**2**	**1**	**0**		Byte number within CAN message
7 **X**	6 **X**	5 **X**	4 **X**	3 **X**	2 **>lsb**	1	0	**0**	
15	14	13	12	11 **msb<**	10 **X**	9 **X**	8 **X**	**1**	
23	22	21	20	19	18	17	16	**2**	
31	30	29	28	27	26	25	24	**3**	
39	38	37	36	35	34	33	32	**4**	
47	46	45	44	43	42	41	40	**5**	
55	54	53	52	51	50	49	48	**6**	
63	62	61	60	59	58	57	56	**7**	

In the example above, a ten-bit long message begins at start bit 2 (the lsb of the LSB is at position 2), counting upward from the start of the message.

5.23.4.2. Motorola format

The start bit specifies the position of the least significant bit in Motorola format. The bits in a Motorola CAN message are always counted as described in the layout below:

The High-Performance Alternative

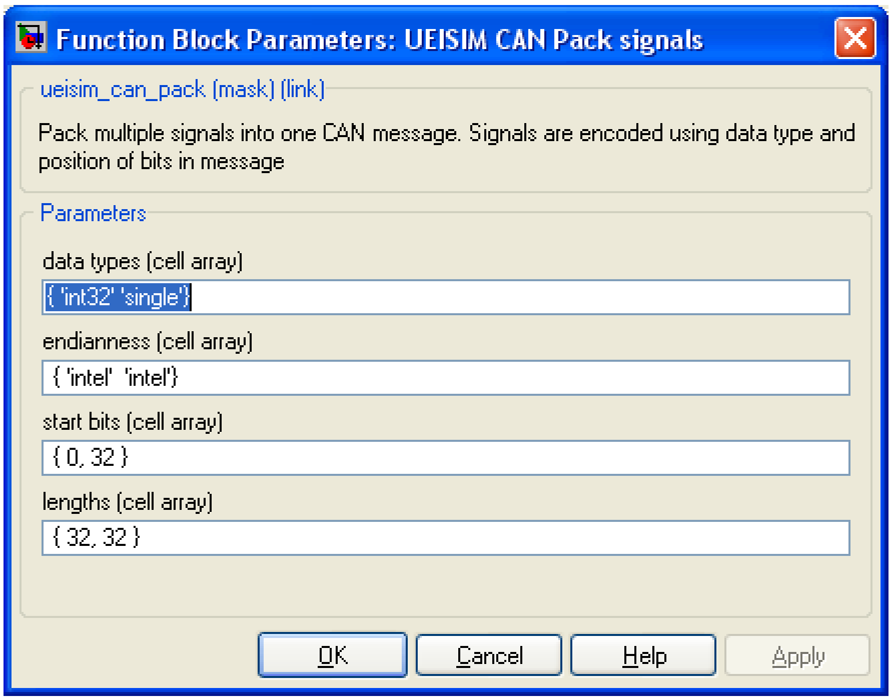
Bit number within a byte									
7	**5**	**6**	**4**	**3**	**2**	**1**	**0**		Byte number within CAN message
7	6	5	4	3	2	1	0	**0**	
15	14	13 **msb<**	12 **X**	11 **X**	10 **X**	9 **X**	8 **X**	**1**	
23 **X**	22 **X**	21 **X**	20 **X**	19 **X**	18 **>lsb**	17	16	**2**	
31	30	29	28	27	26	25	24	**3**	
39	38	37	36	35	34	33	32	**4**	
47	46	45	44	43	42	41	40	**5**	
55	54	53	52	51	50	49	48	**6**	
63	62	61	60	59	58	57	56	**7**	

In the example above, a twelve-bit long message begins at start bit 18 (the lsb of the LSB is at position 8), counting downward from the start of the message.

5.23.4.3. CAN pack block

Pack multiple signals into one CAN message. Signals are encoded using data type and position of bits in message.

United Electronic Industries
The High-Performance Alternative

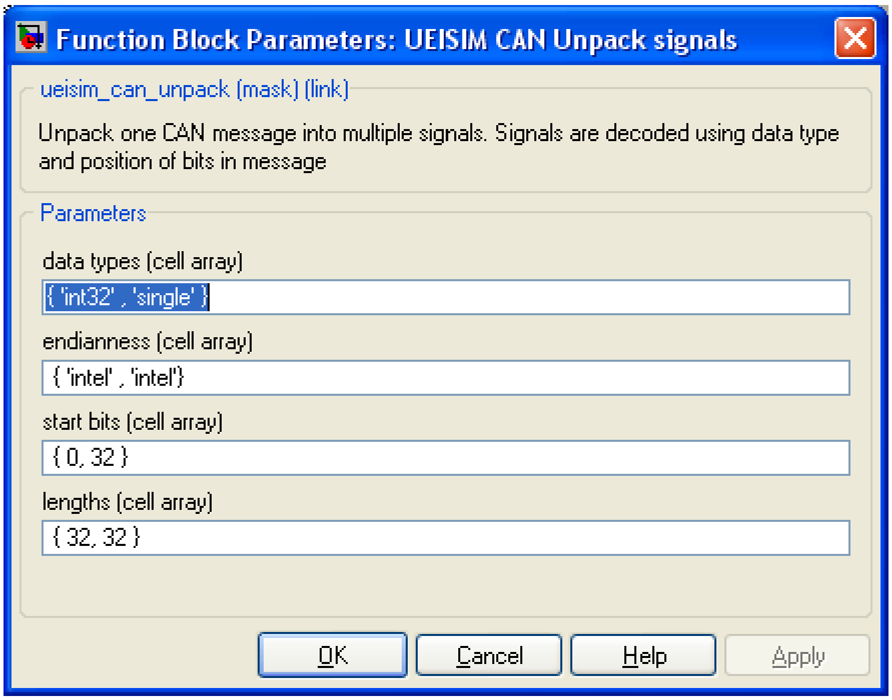


- **Data types**: A cell array containing the data types of the signals to pack in the message
- **Endianness**: A cell array containing the endianness of the signals to pack
- **Start bits**: A cell array containing the index of the first bit of the signals to pack
- **Bit length**: A cell array containing the number of bits of the signals to pack

The block automatically converts itself to one with the correct number of input ports. There is always one output port. The output value is ready to be connected to the CAN Send block.

5.23.4.4. CAN unpack block

Unpack one CAN message into multiple signals. Signals are decoded using data type and position of bits in message



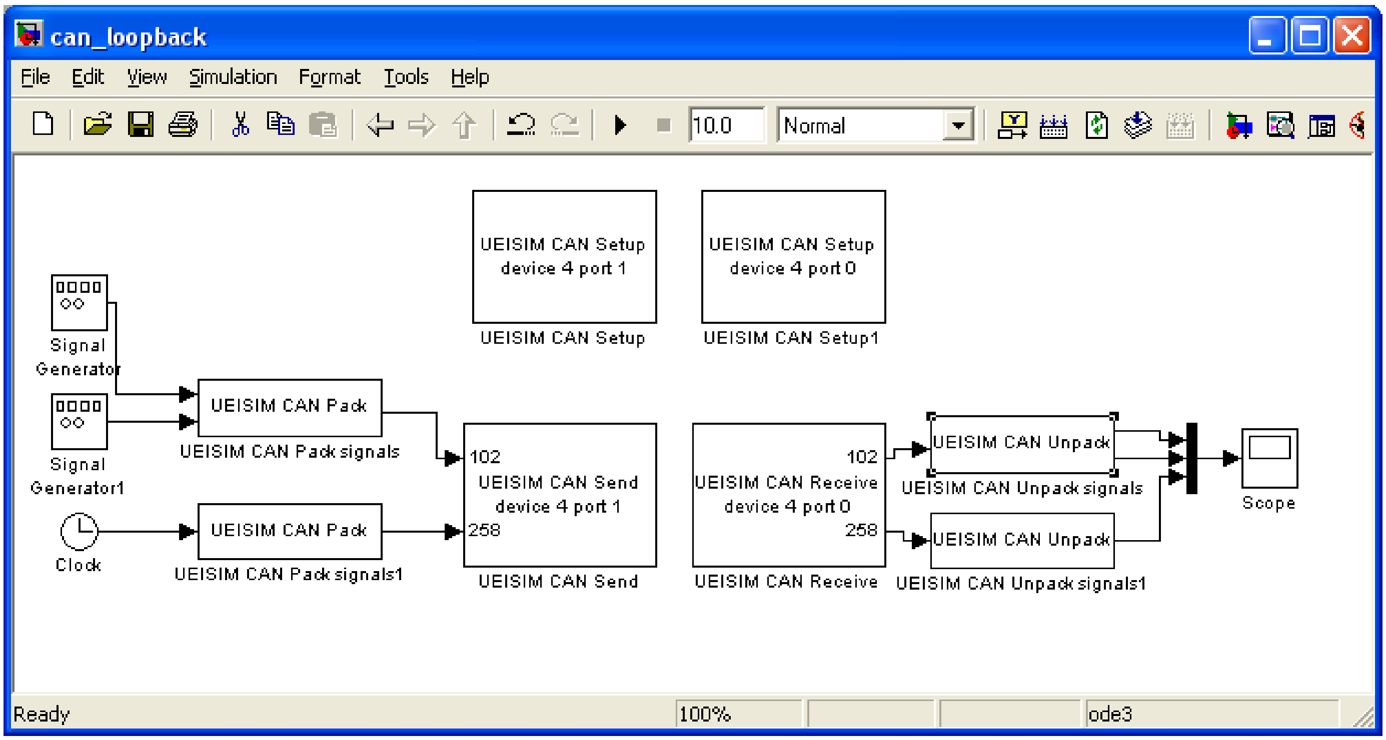
- **Data types**: A cell array containing the data types of the signals to unpack from the message
- **Endianness**: A cell array containing the endianness of the signals to unpack
- **Start bits**: A cell array containing the index of the first bit of the signals to unpack
- **Bit length**: A cell array containing the number of bits of the signals to unpack

The block displays one input port to connect a double value coming from the CAN Receive block. It also displays an output port for each signal to unpack from the CAN message.

5.23.5. CAN examples

The following example configures two ports on the same CAN-503, sends frames with Ids 102 and 258 out of port 0 and receives frames with Ids 102 and 258 from port 1. If port 0 and port1 are connected to the same CAN bus, you will receive what you send.

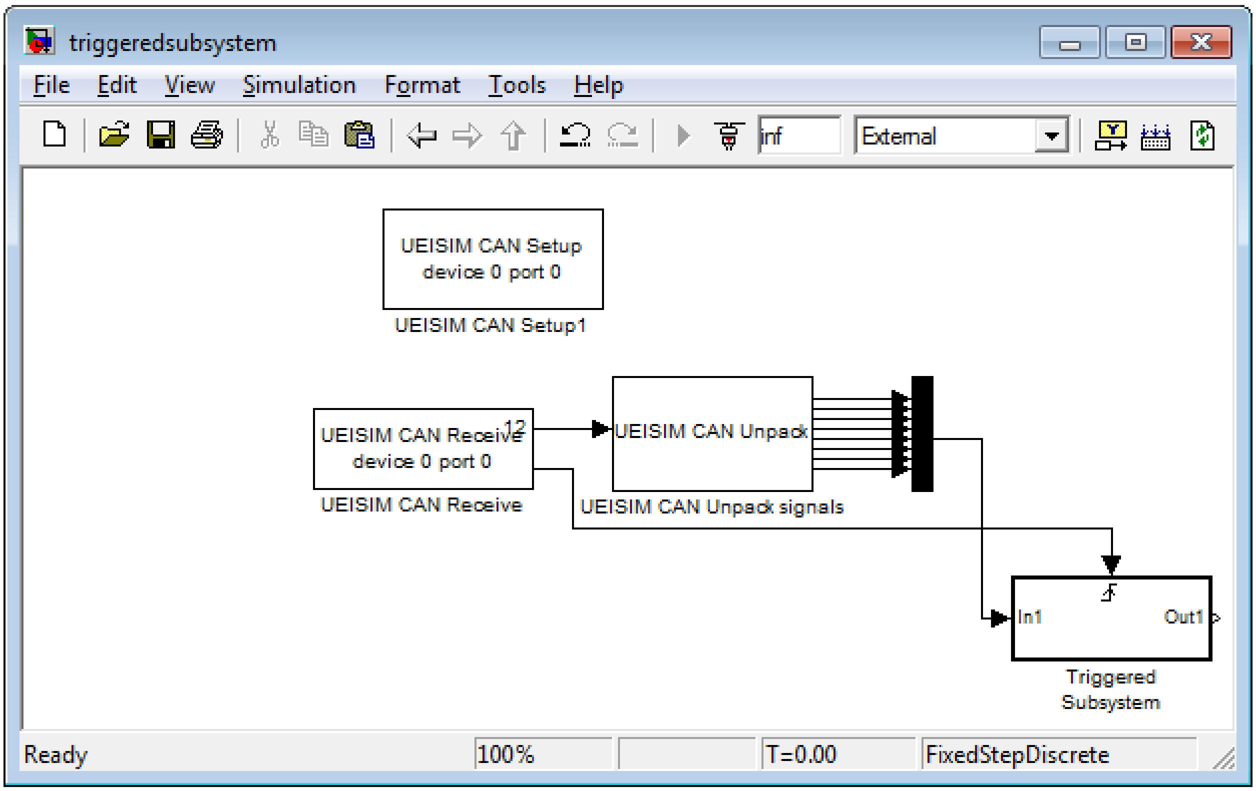
United Electronic Industries
The High-Performance Alternative



The example below shows how the status output can trigger a subsystem to only execute portion of your model when a fresh CAN frame has been received.

The triggered subsystem "Trigger Type" is configured to "Rising". It will execute when the CAN Receive status goes from 0 to 1 each time a new CAN frame is received.

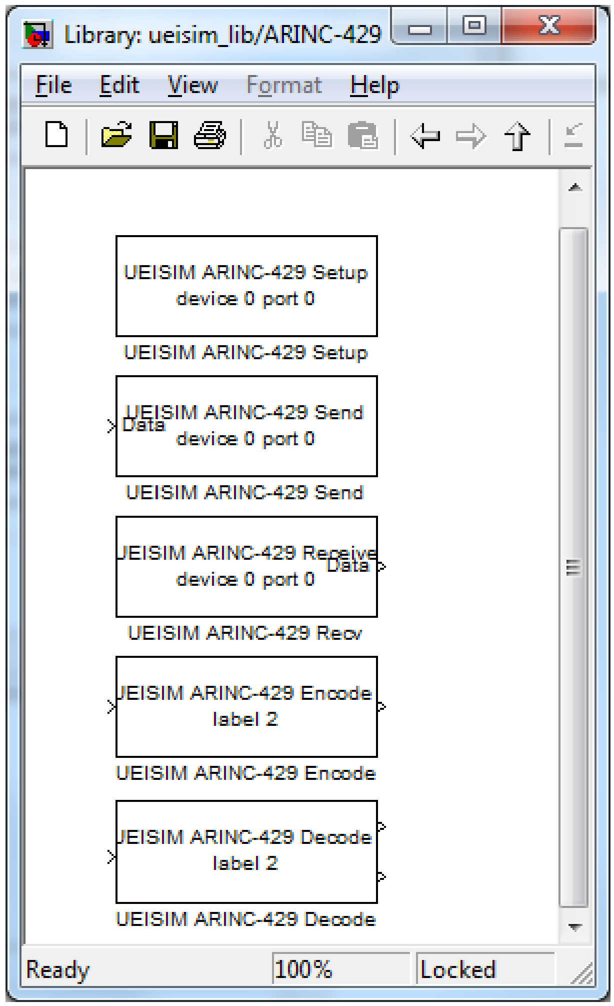
The High-Performance Alternative



5.24. ARINC-429 communication

ARINC-429 communication blocks give access to the 429-566 and 429-512 ARINC-429 ports.
The configuration of each port is done using an independent setup block.
Sending and receiving ARINC-429 words to/from a port is done using a send or receive block.

The High-Performance Alternative

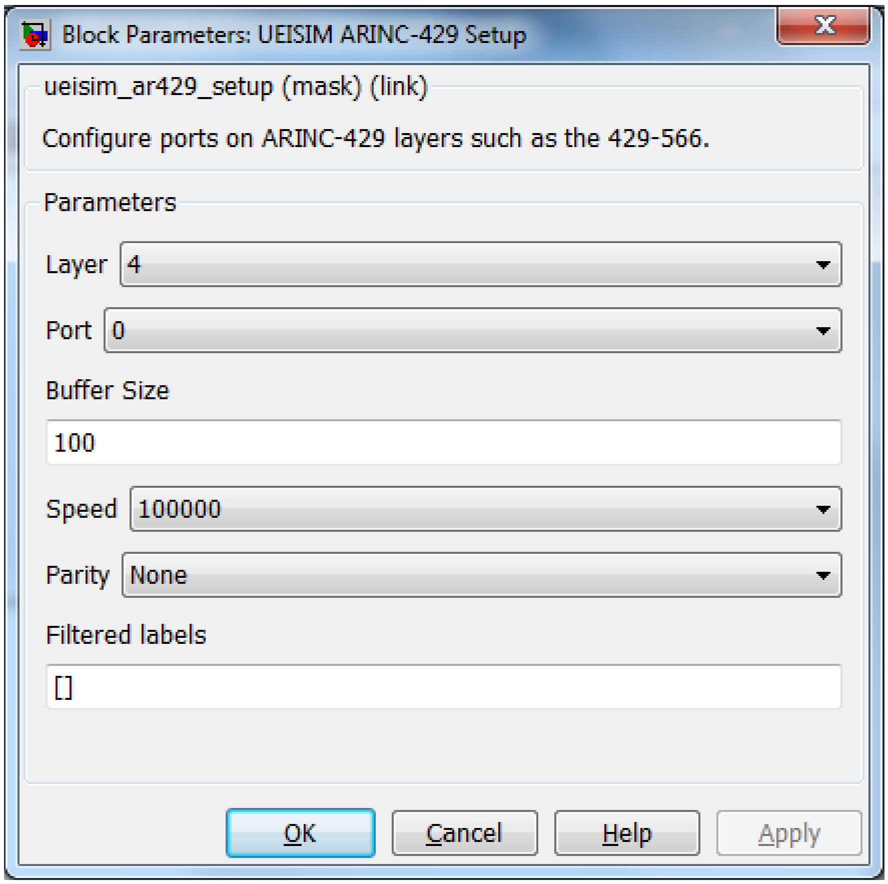


5.24.1. ARINC-429 Setup block

Configure communication settings on a given ARINC-429 port.

The setup block needs to run before the Send/Receive blocks are called (otherwise an error will be returned during model execution).
To view/change the execution context order: Select the menu option **Format > Block Displays > Sorted Order** and make sure that the setup block has a priority lower than the send and receive block for the same port.
To change a block priority: Right-click the block and select **Block Properties**. On the General tab, in the Priority field, enter the new priority.

There must be one setup block for each port used in the model.

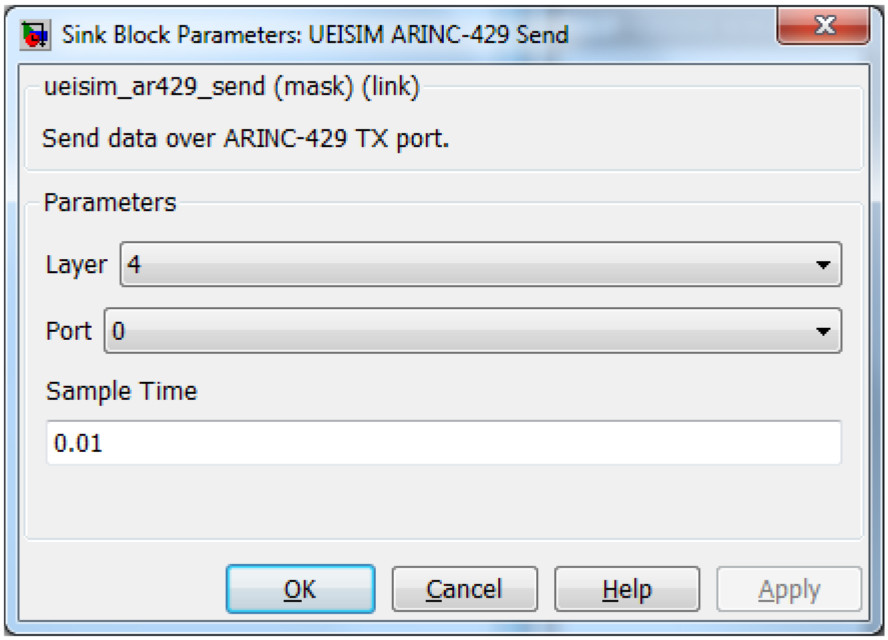


- **layer**: The Id of the ARINC-429 layer associated with this block (layer Ids start at 0 with the top layer)
- **port**: The Id of the port to configure (port Ids start at 0)
- **buffer size**: the size of the internal buffer allocated to store incoming words until they are actually received in the model.
- **speed**: The speed in bits/s used on the ARINC-429 bus connected to this port
- **parity**:The parity setting. Set it to None to have full control of the parity bit.
- **Filtered labels**: A sequence of labels to program the hardware filter. Matching words will be rejected by the ARINc-429 port.

5.24.2. ARINC-429 Send block

Send a group of words to one ARINC-429 TX port. You can create multiple instances of this block to send multiple groups of words at different rate.

The High-Performance Alternative



- **Layer**: The Id of the ARINC-429 layer associated with this block (layer Ids start at 0 with the top layer)
- **Port**: The Id of the port to send data through (port Ids start at 0)
- **Sample Time**: The rate at which the block executes during simulation

The block displays an input port for connecting an array of type UINT32 containing raw values for each word to transmit.

Raw word is a 32 bits value coded as follow:

32	31	30	29 ... 11	10	9	8 ... 1
P	SSM		Data	SDI		Label

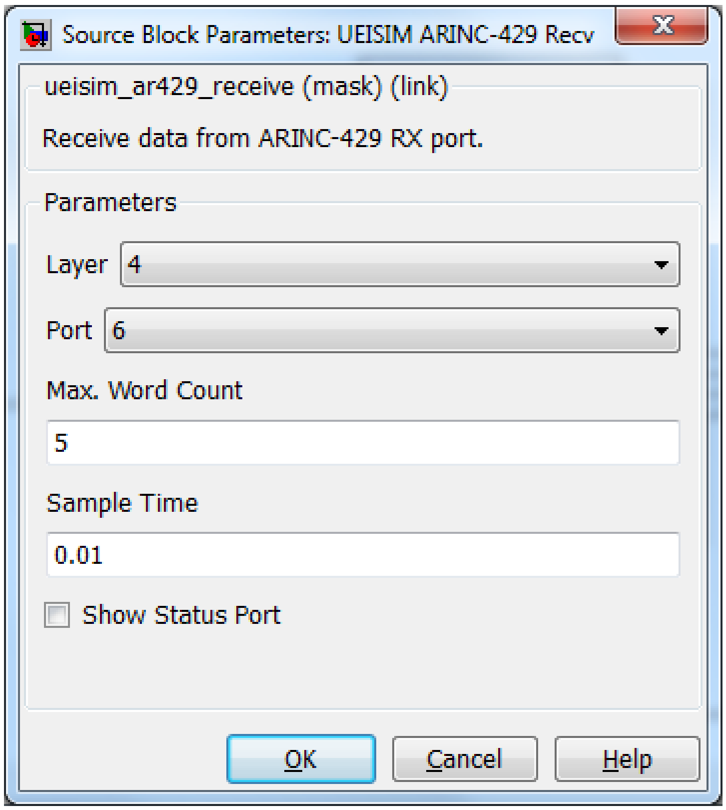
The parity bit in the raw word is ignored. It is automatically calculated at the time the word is transmitted.

Use the ARINC-429 Encode block to encode a value using BCD, BNR or Discrete data type in the data field.
Refer to section about encoding/decoding words below.

5.24.3. ARINC-429 Receive block

Receive a group of ARINC-429 words from one RX port. You can create multiple instances of this block to receive multiple groups of words at different rate.

The High-Performance Alternative



- **layer**: The Id of the ARINC-429 layer associated with this block (layer Ids start at 0 with the top layer)
- **port**: The Id of the port to receive from (port Ids start at 0)
- **max. word count**: The maximum number of word to read from the receive buffer
- **sample time**: The rate at which the block executes during simulation
- **Show Status Port**: Enable/disable status reporting

The block outputs a signal of type UINT32. The first value in the array contains the number of words actually retrieved followed by the raw values of each word.

Raw word is a 32 bits value coded as follow:

32	31	30	29 11	10	9	8 1
P	SSM		Data	SDI		Label

The parity bit in the received word is actually a parity status.

- It is set to 0, when parity is odd and the receiver counts an odd number of 1s (all Ok).
- It is set to 1, when parity is odd and the receiver counts an even number of 1s (parity error)

The High-Performance Alternative

- It is set to 1, when parity is even and the receiver counts an even number of 1s (all Ok).
- It is set to 0, when parity is even and the receiver counts an odd number of 1s (parity error)

Refer to section about encoding/decoding data field into word below.

The status output when enabled can take any of the following values:

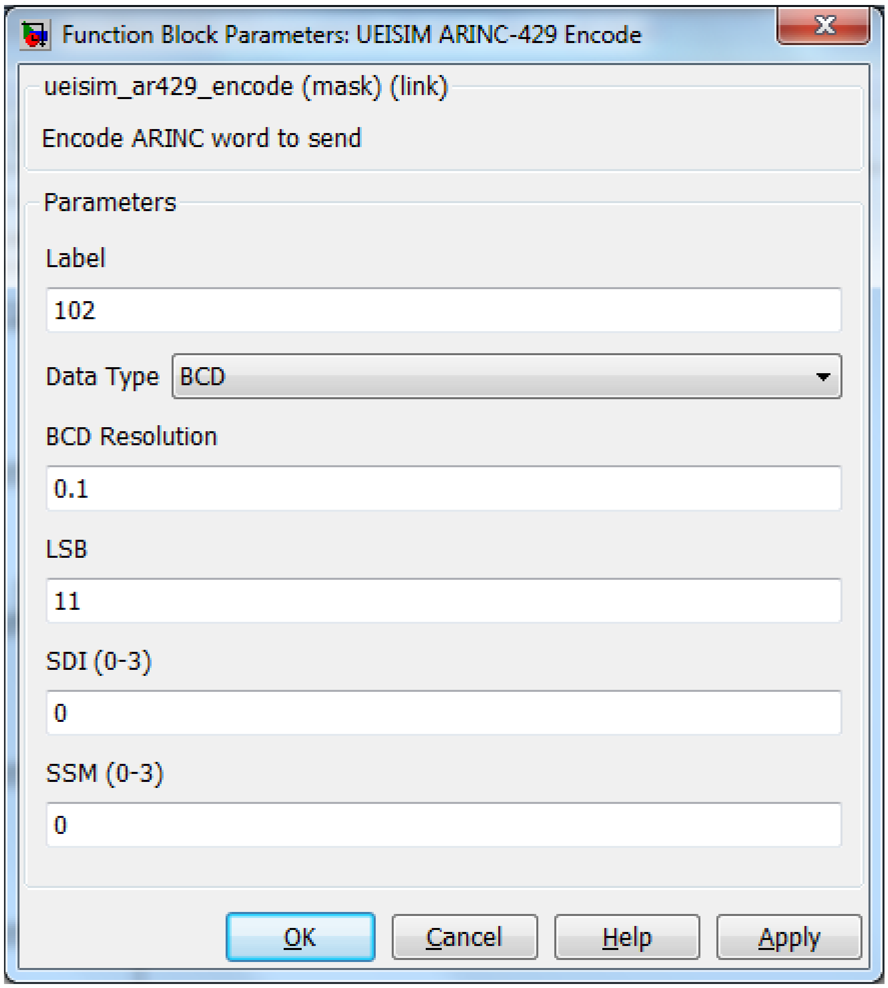
- N>=0: Number of words still available in the receive buffer
- -2: RX Buffer overrun, The receive block is not executed often enough to keep up with the pace of incoming words

5.24.4. ARINC-429 Encode block

Create ARINC-429 raw word and encode value using raw, discrete, BCD or BNR format.

5.24.4.1. BCD

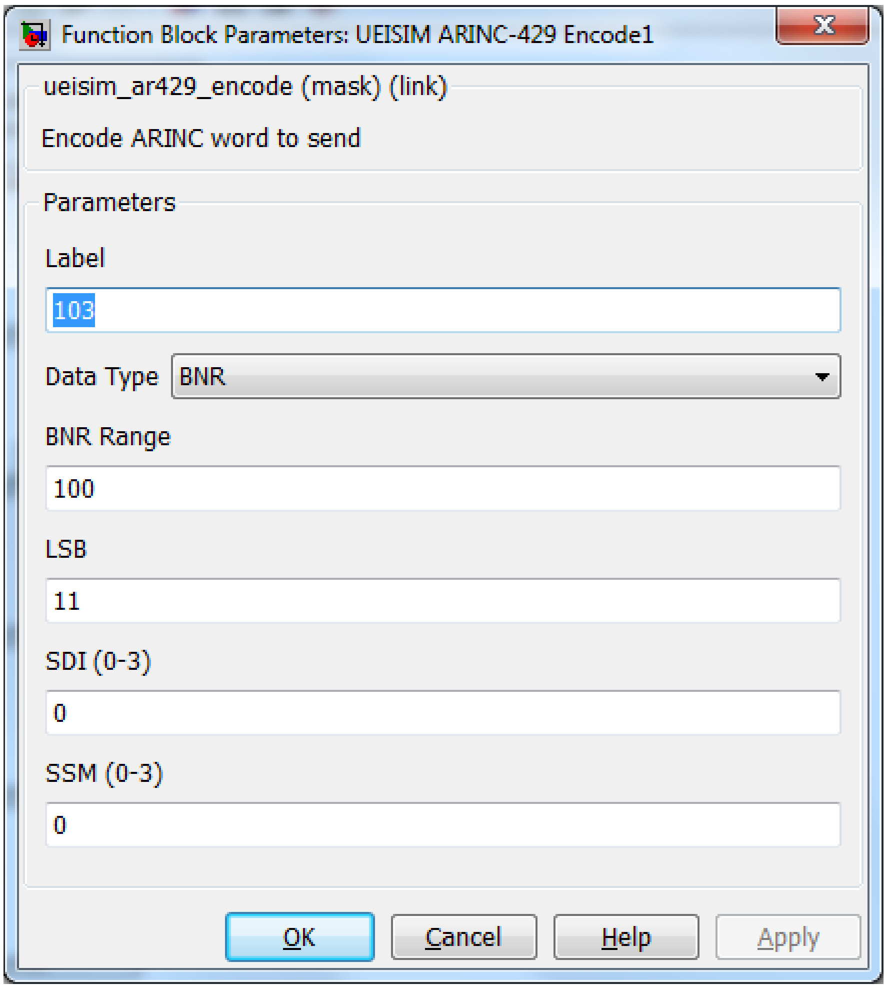
Scale and convert the input as a signed integer, limit it to the range representable by an ARINC five-character BCD value, and pack it into an ARINC word with the appropriate SSM, SDI, and Label parameter values.



- **label**: The 8-bit value inserted in the label field of the word sent over the output port
- **data type**: data type selector
- **BCD resolution**: the value of the least significant digit of the BCD data field to be encoded and sent. For example, if the associated resolution is .01 and the input signal contains the value 3.1415, the output ARINC word will contain the number 314 in its data field, encoded in BCD.
- **lsb**: defines where the encoded value is inserted in the ARINC word. Default is 11.
- **sdi**: if in the range 0 to 3, the block sets the SDI field of the word sent over the output port
- **ssm**: if in the range 0 to 3, the block sets the SSM field of the word sent over the output port

5.24.4.2. BNR

Scale the input and convert to two's complement binary notation, then pack it into an ARINC word with the appropriate SSM, SDI, and Label parameter values.



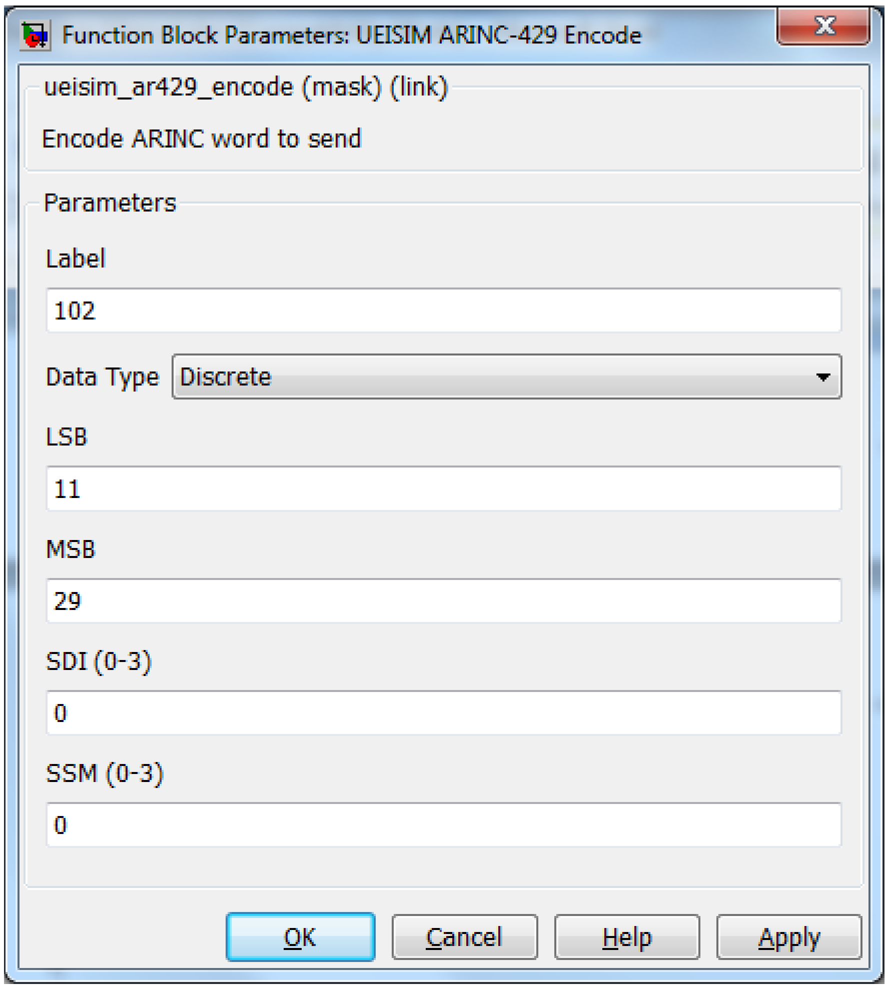
- **label**: The 8-bit value inserted in the label field of the word sent over the output port
- **data type**: data type selector
- **BNR range**: scale factor used to scale the input value which is then limited to [-range, range]. Input values outside that range will be limited to ±range.
- **lsb**: defines where the encoded value is inserted in the ARINC word. Default is 11.
- **sdi**: if in the range 0 to 3, the block sets the SDI field of the word sent over the output port

The High-Performance Alternative

- **ssm**: if in the range 0 to 3, the block sets the SSM field of the word sent over the output port

5.24.4.3. Discrete

Cast the input as an UINT32 and insert the low order 19 bits in the data field of the ARINC word along with the appropriate SSM, SDI, and Label parameter values



- **label**: The 8-bit value inserted in the label field of the word sent over the output port
- **data type**: data type selector
- **lsb**: defines where the encoded value is inserted in the ARINC word. Default is 11.
- **msb**: defines how much of the encoded value is truncated. Default value is 29.

The High-Performance Alternative

- **sdi**: if in the range 0 to 3, the block sets the SDI field of the word sent over the output port
- **ssm**: if in the range 0 to 3, the block sets the SSM field of the word sent over the output port

5.24.4.4. Raw

Cast the input to an unsigned 32-bit integer and output it as an ARINC word with no further processing.

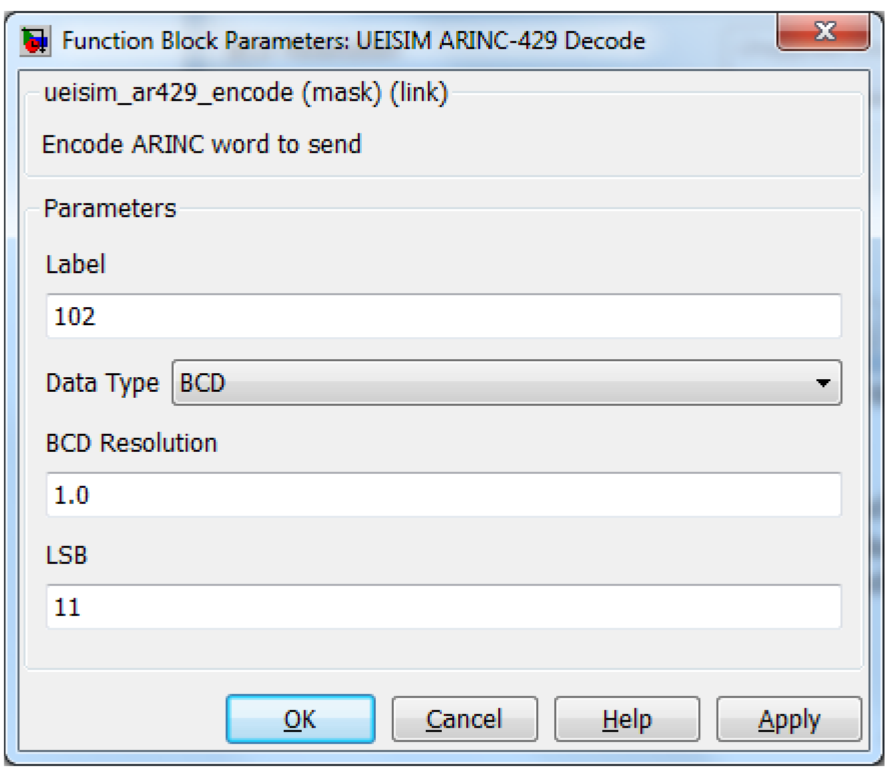
5.24.5. ARINC-429 Decode block

Compare label and decode raw word to scaled value.

The block displays one input port to connect a UINT32 coming from the ARINC-429 Receive block.
It also displays an output port for the decoded value and a status output port. Status is 0 if the input raw word's label field didn't match the label parameter and 1 otherwise.

5.24.5.1. BCD

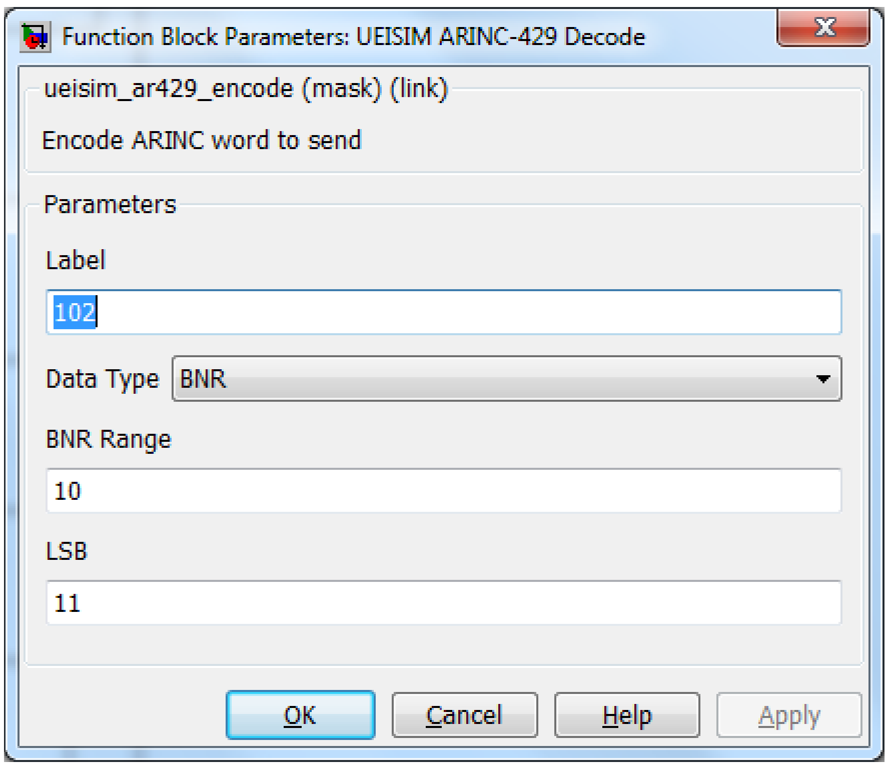
Decode the data field from 5 digit BCD value to double.



- **label**: The 8-bit value to compare with the label field of the word received on the input
- **data type**: data type selector
- **BCD resolution**: the value of the least significant digit of the BCD data field to be decoded.
- **lsb**: defines where the raw value is located in the input word. Default is 11.

5.24.5.2. BNR

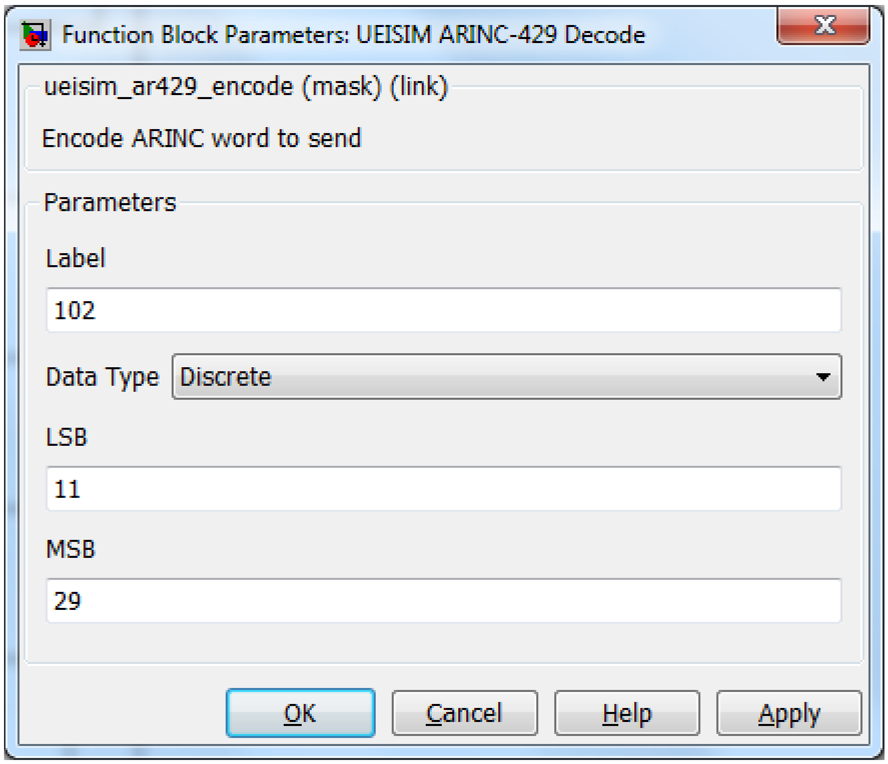
Decode the data field from two's complement binary notation and apply scaling factor.



- **label**: The 8-bit value to compare with the label field of the word received on the input
- **data type**: data type selector
- **BNR range**: scale factor used to scale the coded value back to its original value.
- **lsb**: defines where the coded value is located in the ARINC word. Default is 11.

5.24.5.3. Discrete

Extract the data field from the input word and cast it as a double.



- **label**: The 8-bit value inserted in the label field of the word sent over the output port
- **data type**: data type selector
- **lsb**: defines where the coded value is located in the ARINC word. Default is 11.
- **msb**: defines how much of the coded value to extract. Default value is 29.

5.24.5.4. Raw

Cast the input to a double with no further processing

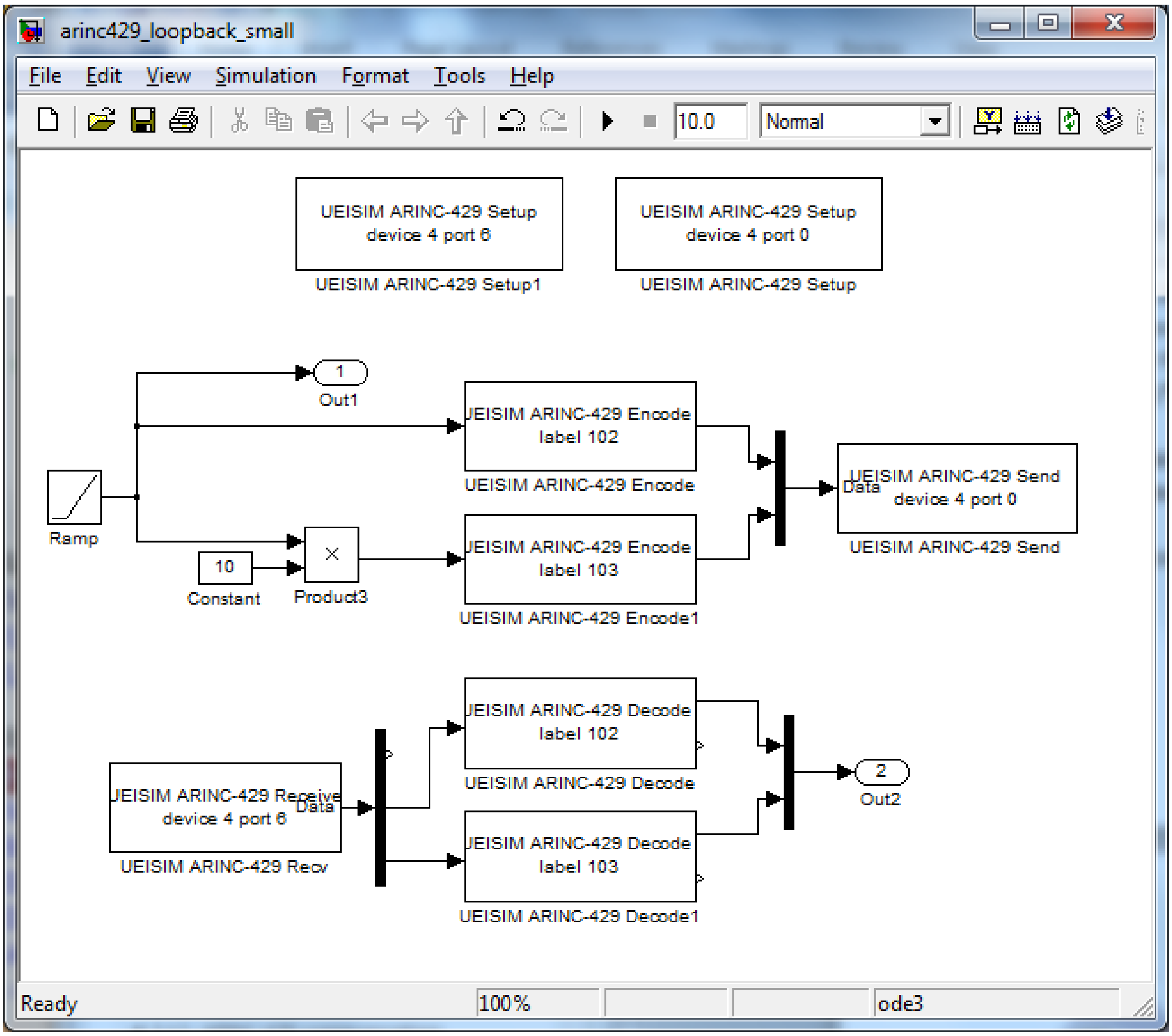
5.24.6. ARINC-429 examples

The following example configures two ports 0 and 6 to run at the same speed. (On 429-566, port 6 is internal loopback port; it automatically receives whatever is transmitted out of port 0).

Port 0 transmits two words where the value from a ramp function block is encoded using BCD format and labels 102 and 103.

Port 6 receives those words and decodes them back using the same parameters than the encode block.

The High-Performance Alternative



Note that the first output of the demux block connected to ARINC-429 receive is not connected. This output contains the number of words received and is ignored in this example.

The status output of the ARINC-429 decode blocks is also ignored. You could connect it to a triggered subsystem that would execute when the decoder status goes from 0 to 1 (each time a word that matches the label parameter is decoded).

The High-Performance Alternative

5.25. MIL-1553 communication

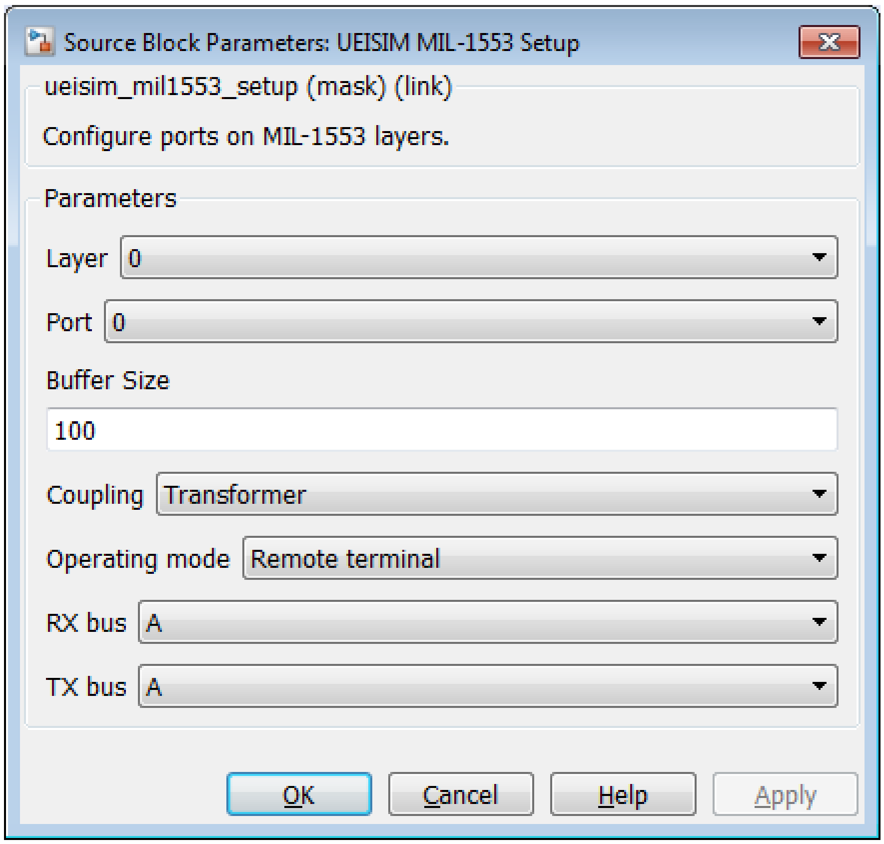
MIL-1553 communication blocks give access to the two MIL-1553 ports on the DNx-1553-553 device.
The configuration of each port is done using an independent setup block.
Each port can be configured as a bus monitor, a bus controller or a remote terminal.

5.25.1. MIL-1553 Setup block

Configure communication settings on a given MIL-1553 port.

The setup block needs to run before the Send/Receive blocks are called (otherwise an error will be returned during model execution).
To view/change the execution context order: Select the menu option **Format > Block Displays > Sorted Order** and make sure that the setup block has a priority lower than the send and receive block for the same port.
To change a block priority: Right-click the block and select **Block Properties**. On the General tab, in the Priority field, enter the new priority.

There must be one setup block for each port used in the model.



Block Parameters:

- **layer**: The Id of the MIL-1553 layer associated with this block (layer Ids start at 0 with the top layer)
- **port**: The Id of the channel to configure (channel Ids start at 0)
- **buffer size**: the size of the internal buffer allocated to store incoming messages until they are actually received in the model (only used in bus monitor mode).
- **coupling**: The coupling mode used to connect a terminal to the bus (transformer or direct)
- **operating mode**: The mode used to configure this channel: bus monitor, remote terminal or bus controller.
- **Rx Bus**: The bus to receive from A, B or both.
- **Tx Bus**: The bus to transmit to A, B or both.

5.25.2. Bus Monitor

MIL-1553 messages are monitored in two steps:

- **MIL-1553 BM Receive** reads messages and stores them in a custom data type.

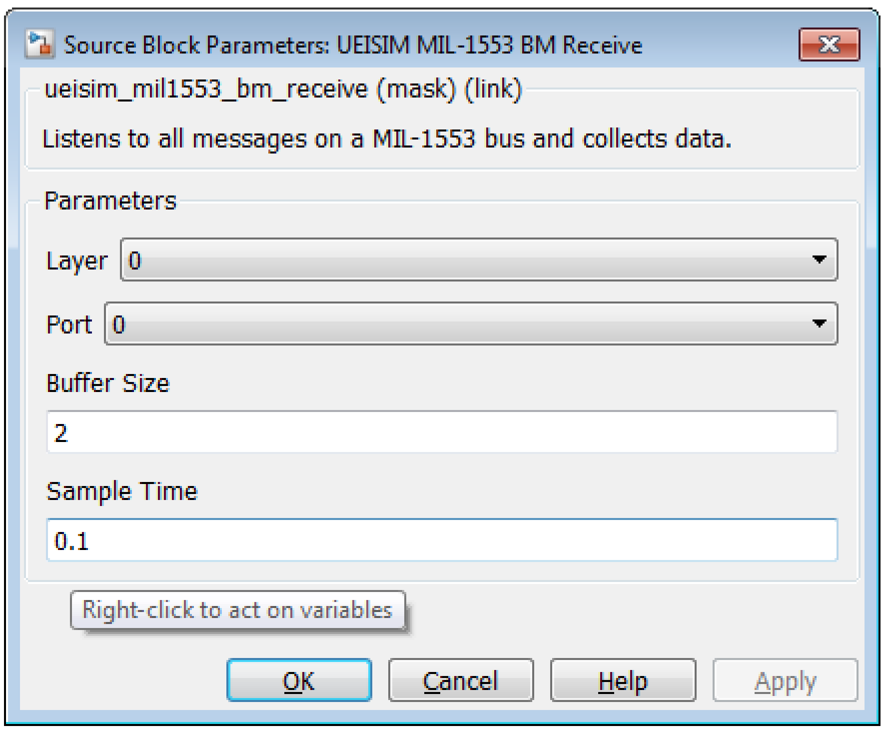
The High-Performance Alternative

- **MIL-1553 Decode BM Messages** extracts and decode messages from the list by index or by type/RT/SA.

5.25.2.1. MIL-1553 BM Receive

Listens to all messages on a MIL-1553 bus and collects data.

This block outputs the collected messages as a custom data type containing the number of messages and a pointer to the messages list.
Use the block "MIL-1553 Decode BM Messages" to decode received messages



Block Parameters:

- **layer**: The Id of the MIL-1553 layer associated with this block (layer Ids start at 0 with the top layer)
- **Port**: The channel to send message from
- **Buffer size:** Maximum number of 16-bit word to receive. The output data vector width will indicate how many messages were actually received.
- **Sample time:** The rate at which the block executes during simulation

The High-Performance Alternative

5.25.2.2. MIL-1553 Decode BM Messages

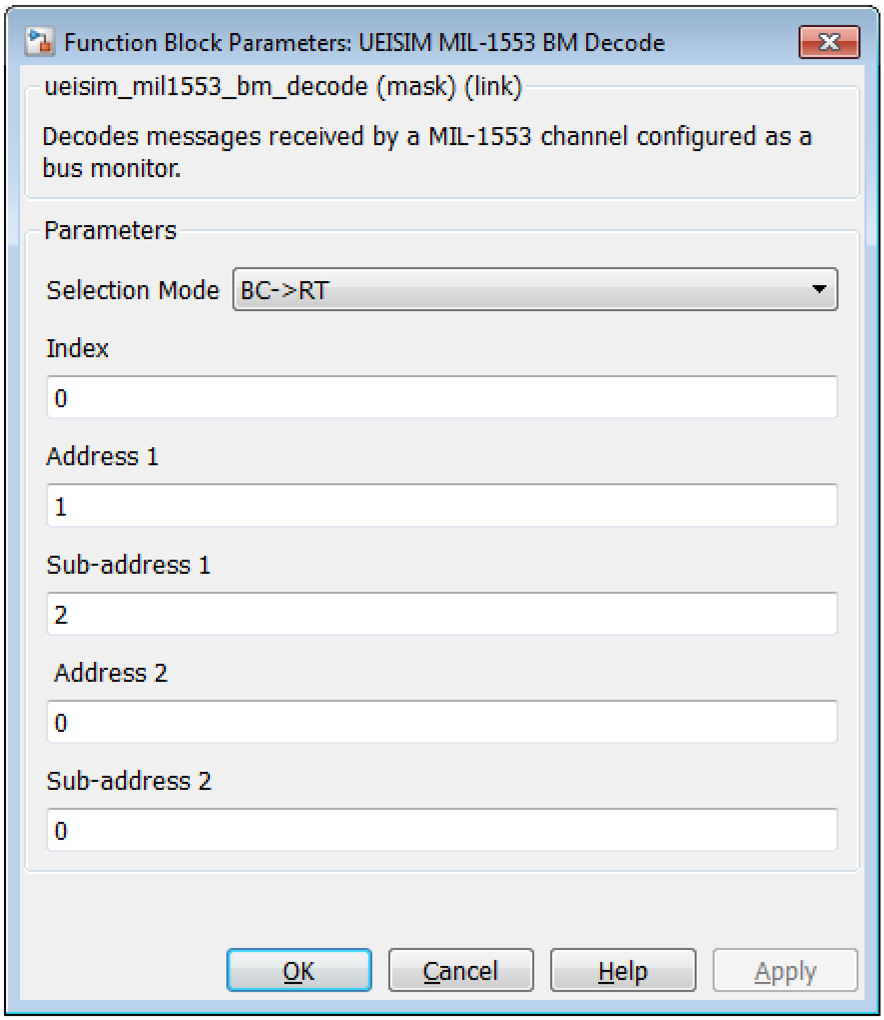
This block decodes messages received by a MIL-1553 channel configured as a bus monitor.
Connect the list of messages received by “MIL-1553 BM Receive” to this block’s input to decode messages.

You can decode messages sequentially by index or search the list for a given message type (BC->RT, RT->BC, RT->RT), RT address and sub-address.

The block outputs are:

- Message List (L): The input message list passed through
- Timestamp (T): The time this message was received
- Status (S): A 7 elements vector containing 16-bit commands and stauts values [index, cmd1, resp1, sts1, cmd2, resp2, sts2]
  Cmd* contains address information in bit fields:
  <RRRRR>T<SSSSS><CCCCC>
  RRRRR is the 5-bit field with the remote terminal address. T is 1 if a transmit message, 0 if receive. S is the subaddress. C is the count.
  Elements 5,6 and 7 are non-zero for RT->RT messages only.
- Data (D): A 32 elements vector containing 16-bit values. The count field of cmd1 contains the actual number of data words in that message.

The High-Performance Alternative



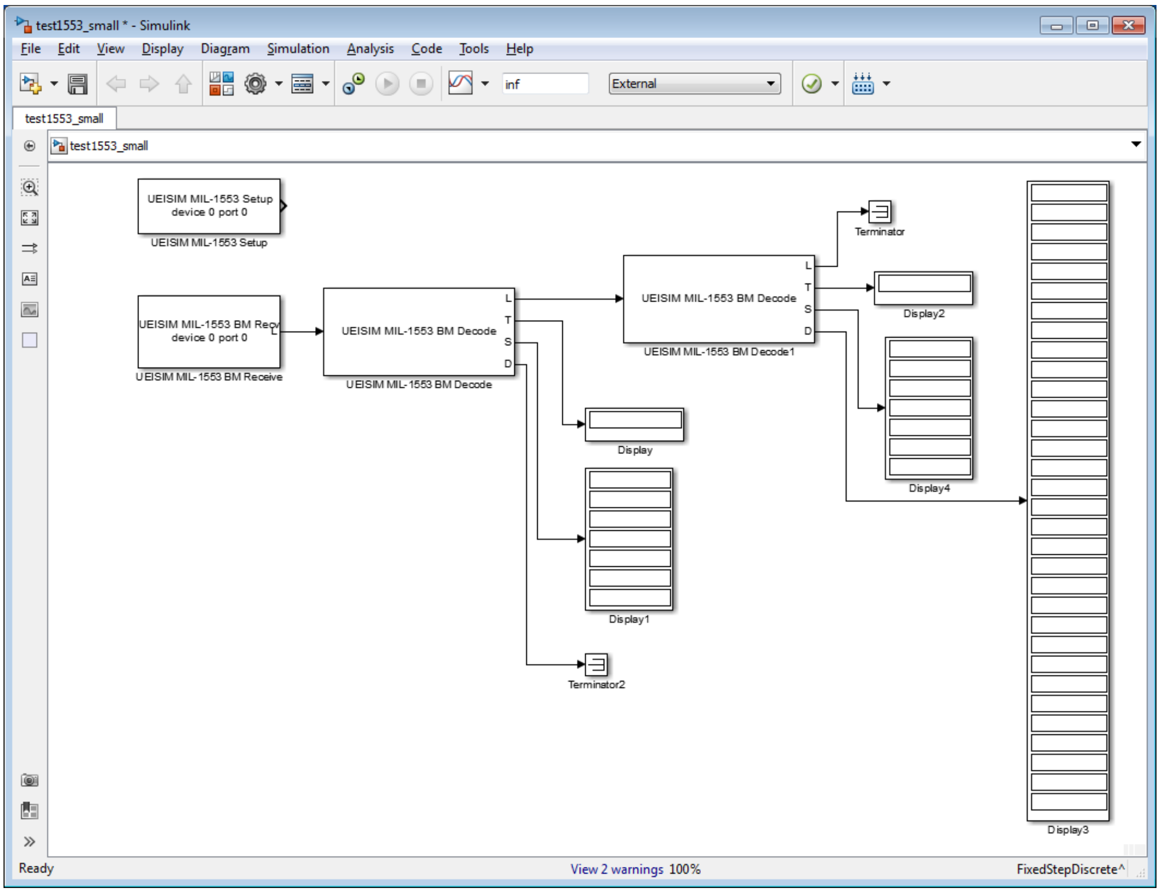
Block Parameters:

- **Selection Mode**: The mode used to select the message to decode in the messages list (message index, BC->RT, RT->BC or RT->RT)
- **Index**: The index of the message to decode (if mode is set to "message index")
- **Address1**: The address of the remote terminal from which to decode the message (if mode is set to BC->RT, RT->BC or RT->RT)
- **Sub-address1**: The sub-address from which to decode the message. (if mode is set to BC->RT, RT->BC or RT->RT)
- **Address2**: The destination address from which to decode the message. (if mode is set to RT->RT)
- **Sub-address2**: The sub-address from which to decode the message. (if mode is set to RT->RT)

United Electronic Industries
The High-Performance Alternative

5.25.2.3. Bus monitor example

The model below shows how BM decode blocks can be daisy chained to decode different messages received by the bus monitor port.

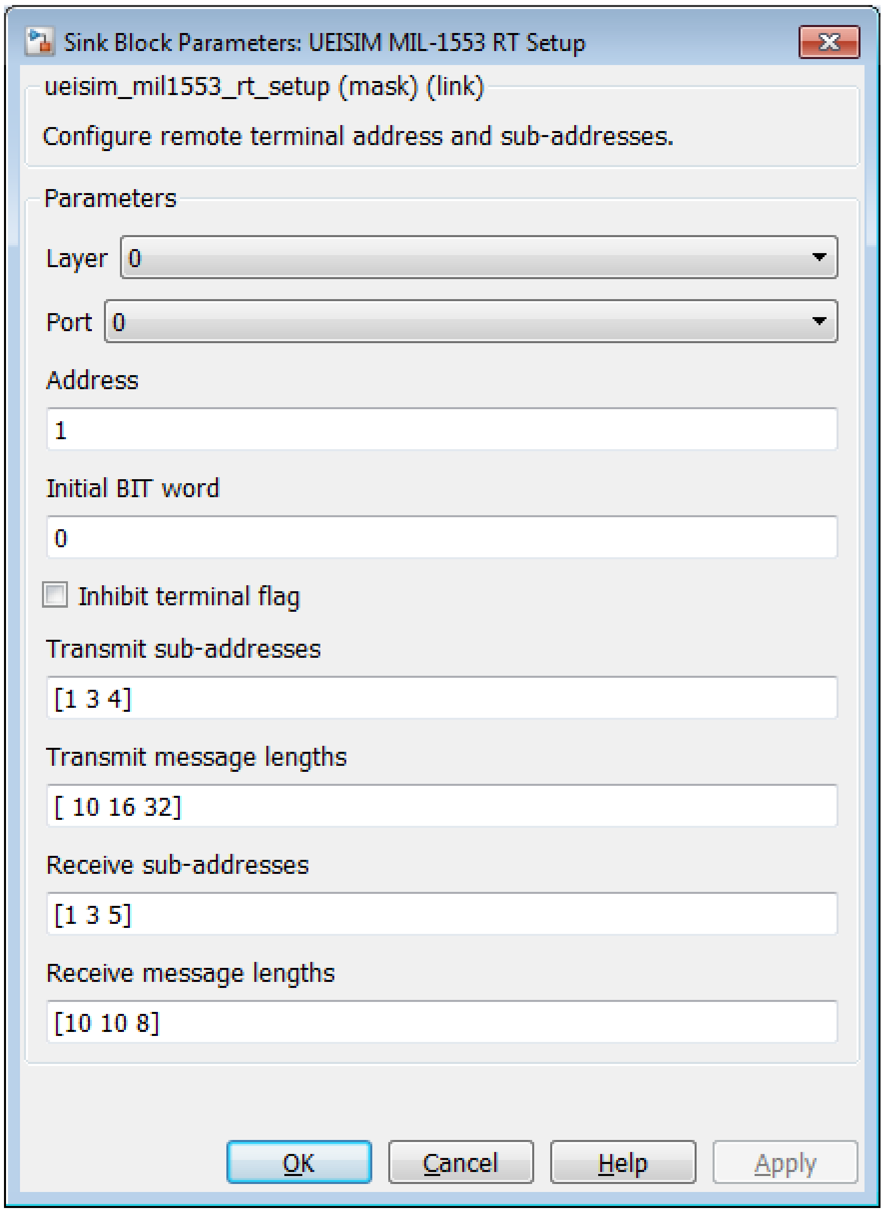


5.25.3. Remote terminal

5.25.3.1. MIL-1553 RT Setup

Configure remote terminal address and sub-addresses and associate RT with MIL-1553 channel. The MIL-1553 channel will only start accepting commands for a given RT/SA if it’s been setup with this block.

All instances of this block must execute after the global MIL-1553 setup. Connect the MIL-1553 setup block output signal to MIL-1553 RT Setup input to enforce correct execution order.



Block Parameters:

- **layer**: The Id of the MIL-1553 layer associated with this block (layer Ids start at 0 with the top layer)
- **port**: The Id of the channel that will emulate this RT
- **Address**: The address of this remote terminal (1 to 31)
- **Initial BIT word**: The initial value of the Buit-in test word sent in response to the BIT mode code.

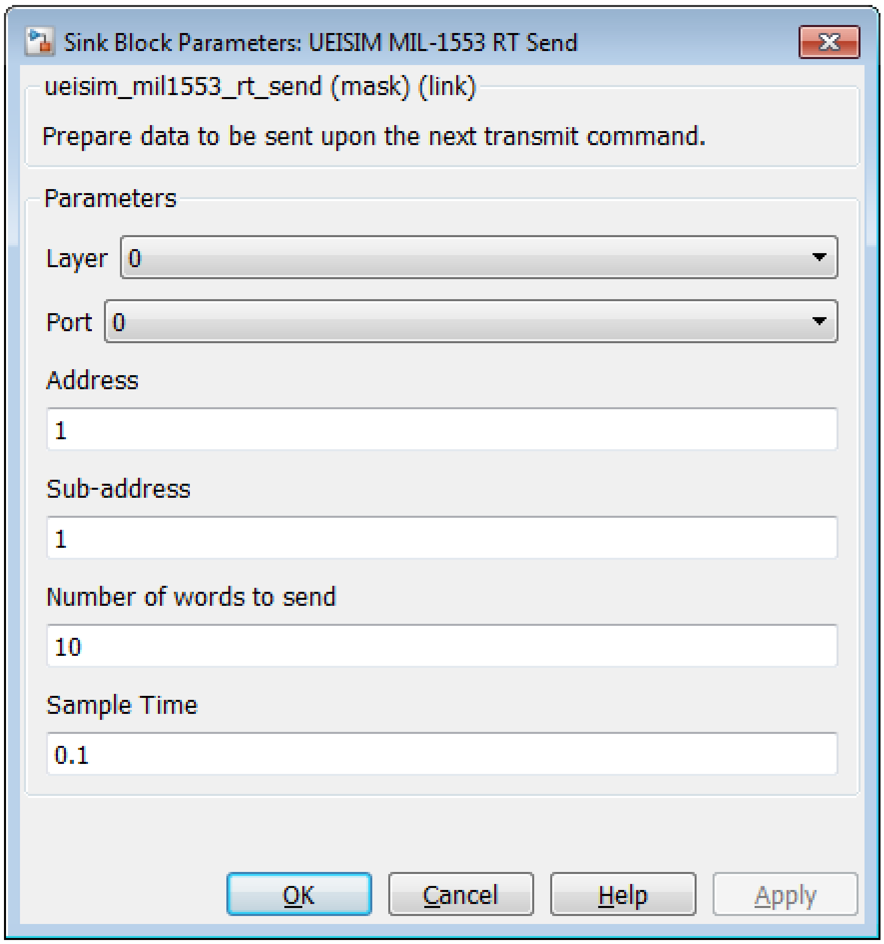
The High-Performance Alternative

- **Inhibit terminal flag**: Enable this check box to inhibit the RT flag.
- **Transmit sub-addresses**: A vector of sub-addresses to which this Remote Terminal will respond. These sub-addresses can transmit data when requested, the number of data words to transmit is specified in the **Transmit message lengths** vector.
- **Transmit message lengths**: A vector of transmit 'T'message lengths. There must be one message length for each sub-address. Message length must be between 1 and 32.
- **Receive sub-addresses**: A vector of sub-addresses that can accept a receive 'R' message. The number of data words to receive is specified in the **Receive message lengths** vector.
- **Receive message lengths**: A vector of receive 'R' message lengths. There must be one message length for each sub-address. Message length must be between 1 and 32.

5.25.3.2. MIL-1553 RT Send

A Remote Terminal sends data only if it receives a transmit command from the Bus Controller. This block prepares the data buffer for the next transmit command on this channel, Remote Terminal address, and sub-address.

This block takes a data vector on its input containing the 16-bit words to send during the next transmit command (RT->BC or RT->RT)



Block Parameters:

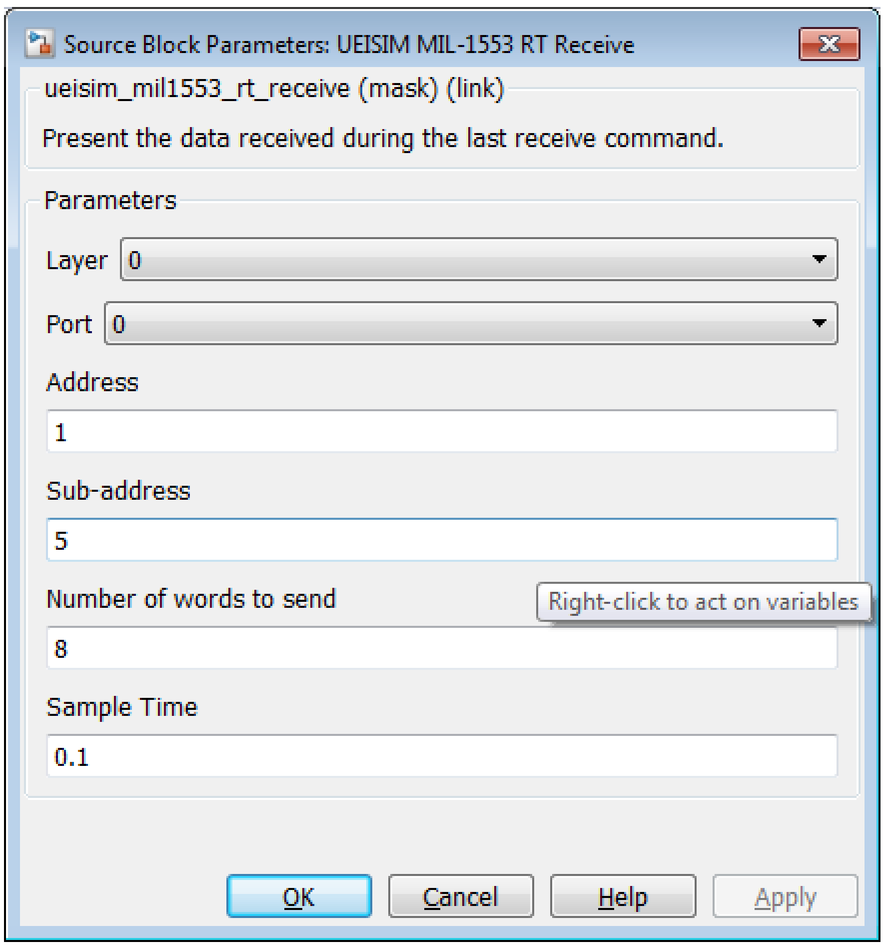
- **layer**: The Id of the MIL-1553 layer associated with this block (layer Ids start at 0 with the top layer)
- **port**: The channel to send message from
- **Address**: The RT address (1 to 31). The RT number must match one of the RTs configured in **MIL-1553 RT Setup**.
- **Sub-address** The RT sub-address.
- **Number of words to send:** Number of 16-bit word to send as part of this message, the input data vector must have the same width.
- **Sample time:** The rate at which the block executes during simulation

United Electronic Industries
The High-Performance Alternative

5.25.3.3. MIL-1553 RT Receive

A Remote Terminal receives data only if it receives a receive command from the Bus Controller. This block presents the data received during the last receive command on this channel, Remote Terminal address, and sub-address.

This block outputs a data vector containing the 16-bit words received during the last receive command (BC->RT or RT->RT)



Block Parameters:

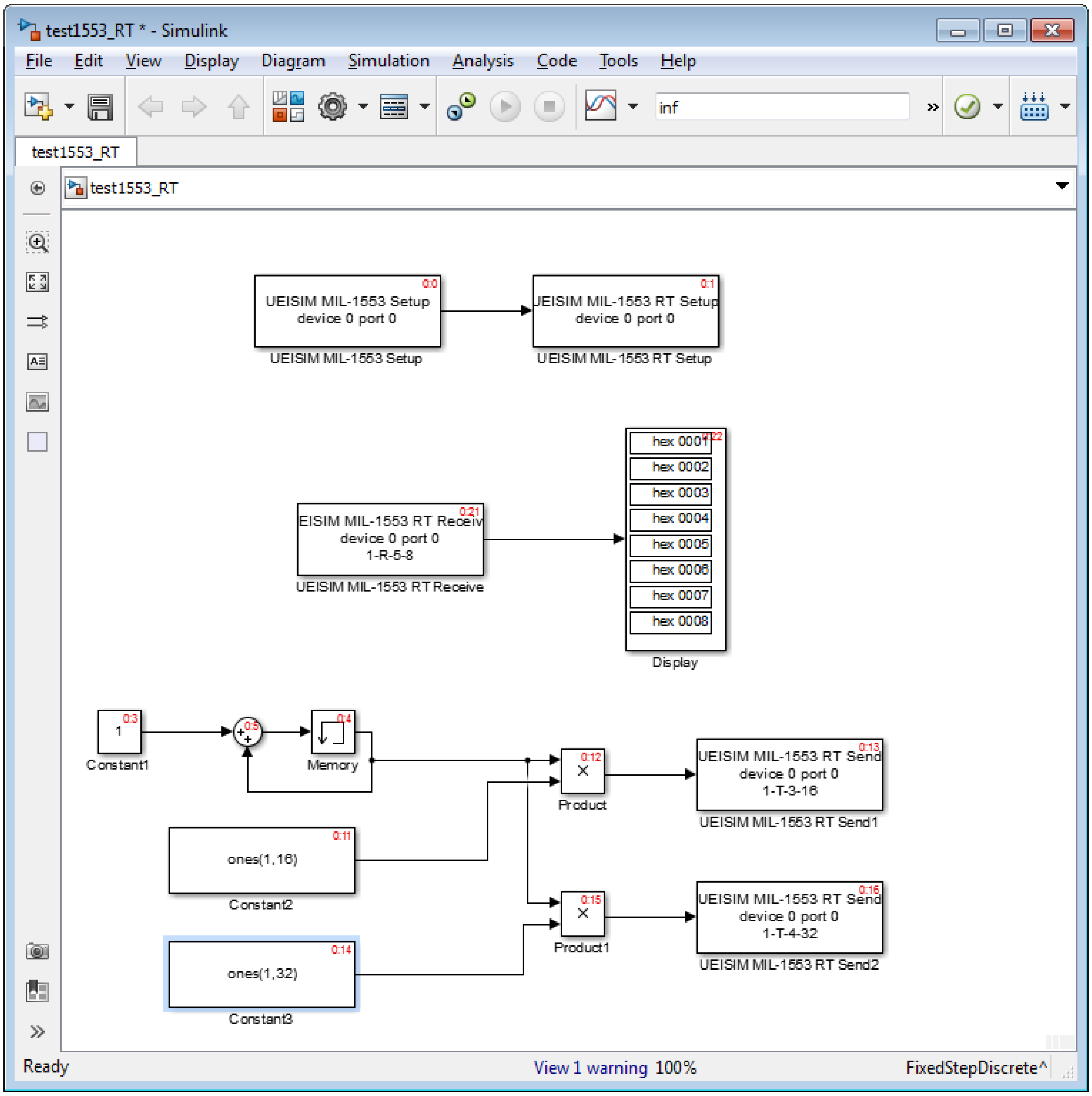
- **Layer**: The Id of the MIL-1553 layer associated with this block (layer Ids start at 0 with the top layer)
- **Port**: The channel to send message from
- **Address**: The RT address (1 to 31). The RT number must match one of the RTs configured in **MIL-1553 RT Setup**.

- **Sub-address** The RT sub-address.
- **Number of words to receive:** Maximum number of 16-bit word to receive. The output data vector will have this same width.
- **Sample time:** The rate at which the block executes during simulation

5.25.3.4. Remote terminal example

The model below reads 8 words on RT1/SA5, sends 16 words to RT1/SA3 and sends 32 words to RT1/SA4

The High-Performance Alternative

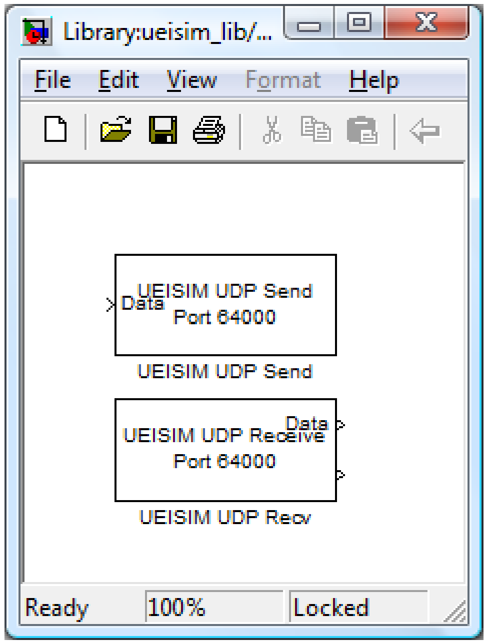


5.26. Network communication

5.26.1. UDP

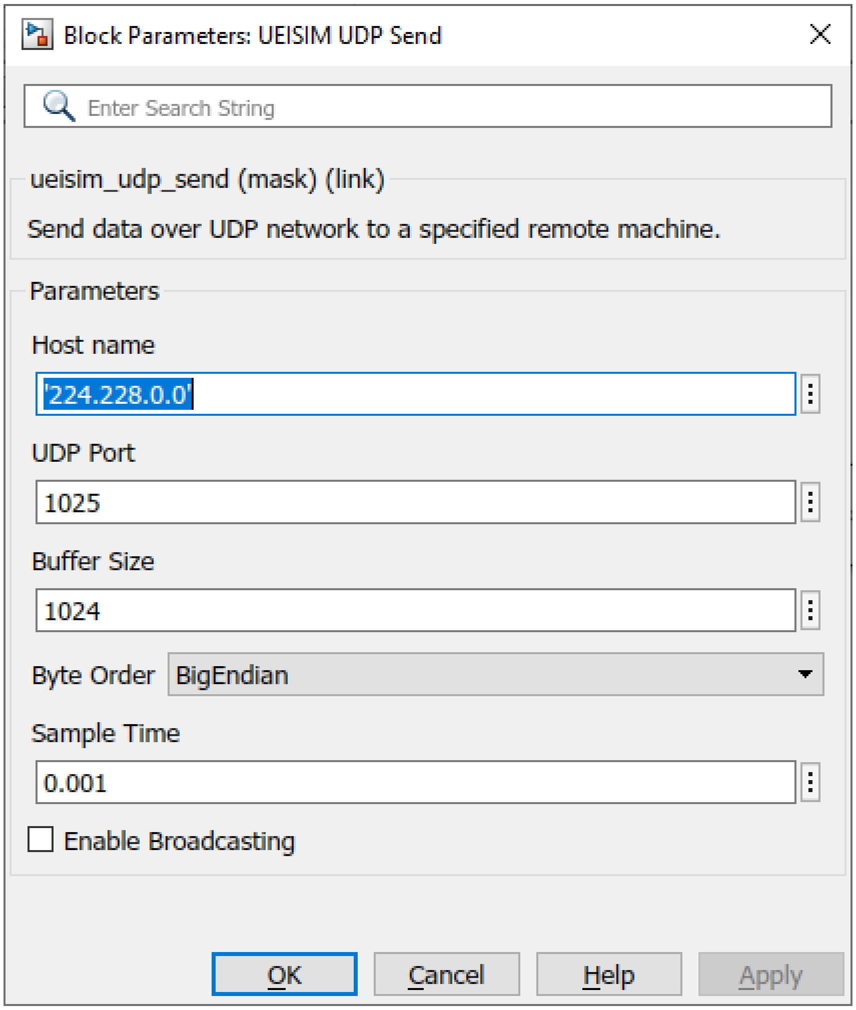
UDP communication blocks give access to the Ethernet port. Sending and receiving UDP packets to/from the Ethernet port is done using the UDP send or UDP receive block.

The High-Performance Alternative



5.26.1.1. UDP Send block

Send UDP packets to a network host. You can create multiple instance of this block to send packets to different ports at different rates.

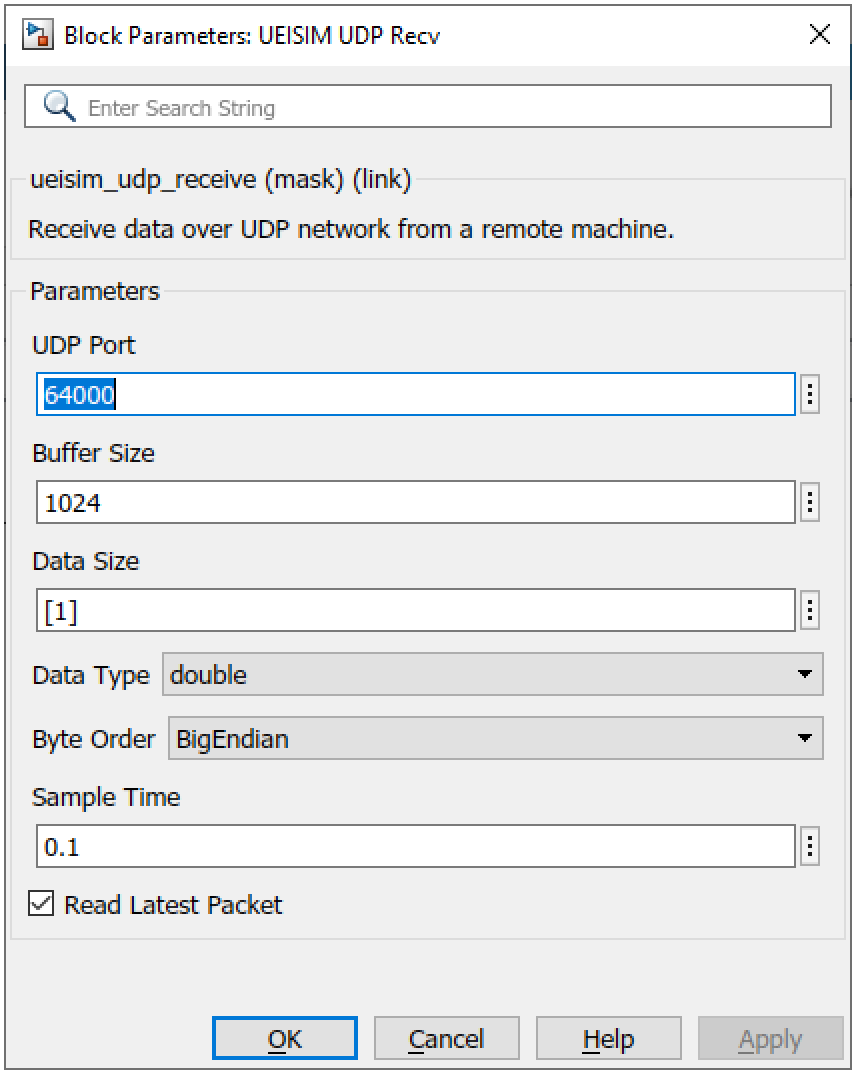


- **Host name**: The name or IP address of the destination host
- **UDP port**: The port to send to (must be > 1024 and < 65535)
- **Buffer size**: Size in bytes of the network buffer
- **Byte Order**: The endianness used to pack data in the UDP packet.
- **Sample Time**: The rate at which the block executes during simulation
- **Enable Broadcasting**: Enables broadcasting on the UDP socket

The block displays an input port for connecting the value of the packet payload, it automatically adapts to the data type and dimension of the signal connected.

5.26.1.2. UDP Receive block

Receive UDP packets from a network host. You can create multiple instance of this block to receive multiple packets from different ports.



- **UDP port**: The port to receive from (must be > 1024 and < 65535)
- **Buffer size**: Size in bytes of the network buffer
- **Data Size**: Dimension and size of the output signal (for ex [2 4] will output received data in a 2x4 matrix)
- **Data Type**: The data type used to decode received data
- **Byte Order**: The endianness used to unpack the UDP packet payload.
- **Sample Time**: The rate at which the block executes during simulation
- **Read Latest Packet**: Discard all pending packets and read the most recent one

The block displays two output ports:

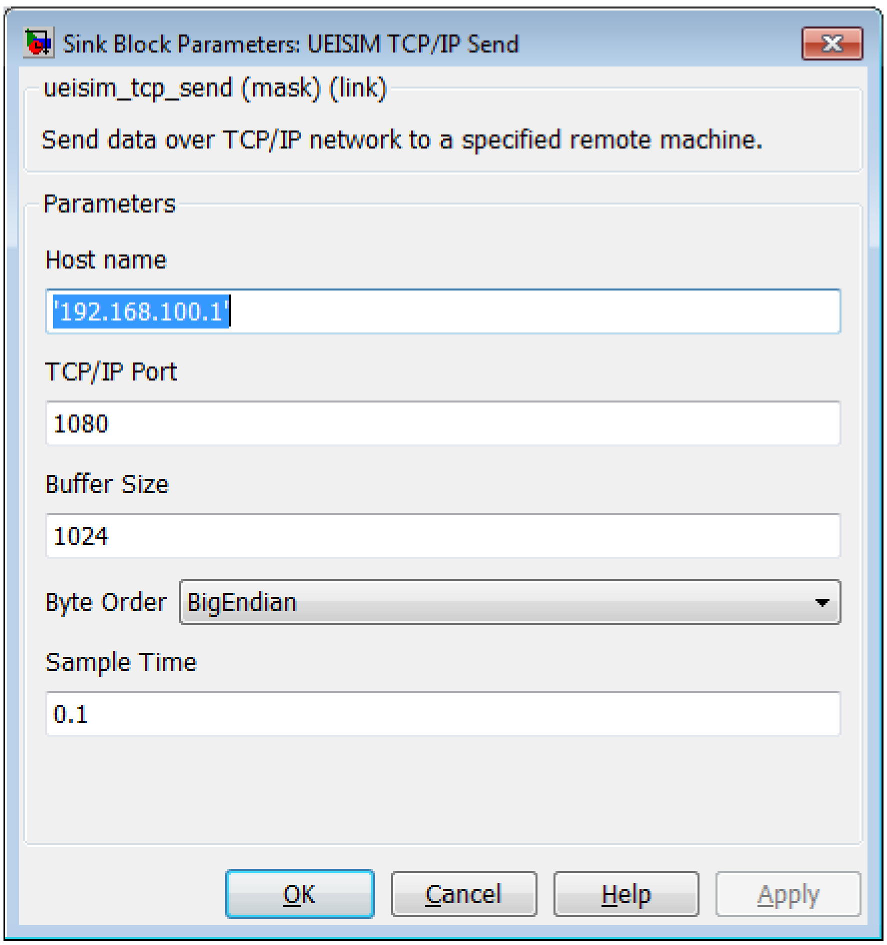
The High-Performance Alternative

- Data: The signal extracted from the packet payload.
- Status: The number of bytes in the payload (0 if no packet was received).

5.26.2. TCP/IP Client

5.26.2.1. TCP/IP Send block

Send TCP/IP packets to a TCP/IP server. You can create multiple instance of this block to send packets to different servers at different rates.



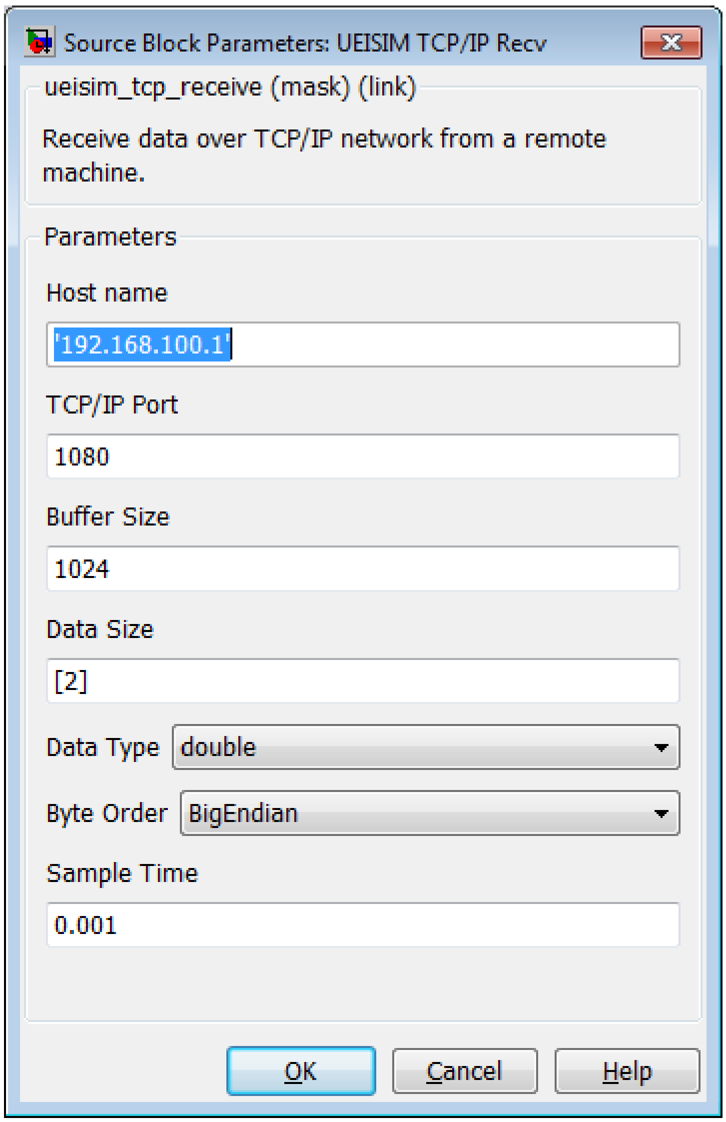
- **Host name**: The name or IP address of the server
- **TCP/Ip port**: The port to send to
- **Buffer size**: Size in bytes of the network buffer
- **Byte Order**: The endianness used to pack data in the TCP/IP packet.
- **Sample Time**: The rate at which the block executes during simulation

The High-Performance Alternative

The block displays an input port for connecting the value of the packet payload, it automatically adapts to the data type and dimension of the signal connected.

5.26.2.2. TCP/IP Receive block

Receive TCP/IP packets from a server. You can create multiple instance of this block to receive multiple packets from different servers.



- **Host name**: The name or IP address of the server
- **TCP/IP port**: The port to receive from
- **Buffer size**: Size in bytes of the network buffer

The High-Performance Alternative

- **Data Size**: Dimension and size of the output signal (for ex [2 4] will output received data in a 2x4 matrix)
- **Data Type**: The data type used to decode received data
- **Byte Order**: The endianness used to unpack the TCP/IP packet payload.
- **Sample Time**: The rate at which the block executes during simulation

The block displays two output ports:

- Data: The signal extracted from the packet payload.
- Status: The number of bytes in the payload (0 if no packet was received).

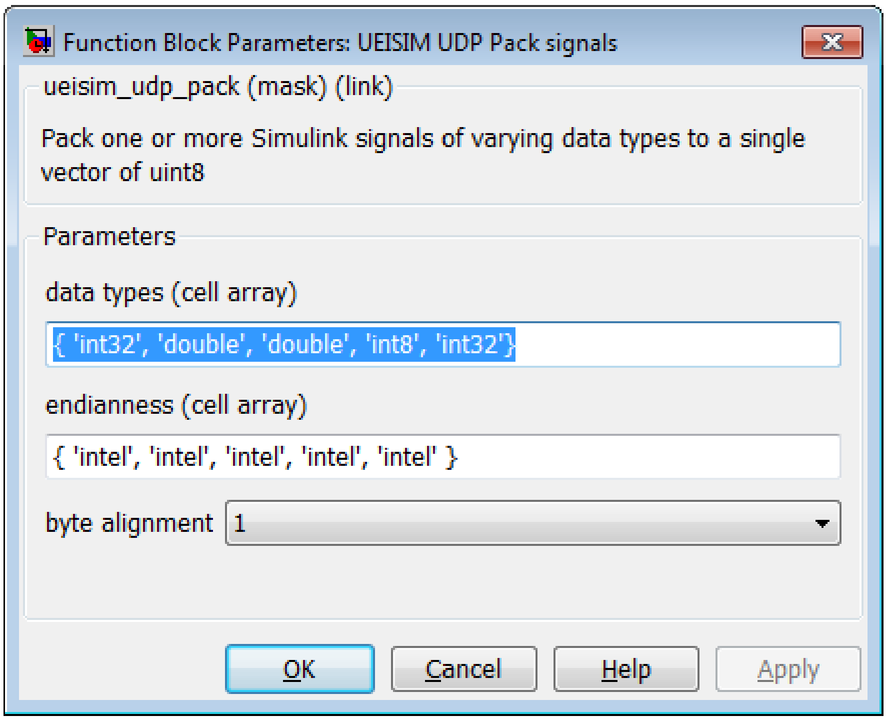
5.26.3. Utility blocks

Utility blocks are used to pack and unpack data structures stored in the TCP/IP or UDP packets that are sent or received. You can specify different data types for each member of the data structure.

Each member is specified using the following parameters:

- **data type**: the type of the member, possible values are boolean, int8, uint8, int16, uint16, int32, uint32, single or double.
- **endianness**: the endianness of the member, possible values are 'intel' for little endian, 'motorola' for big endian and 'alorotom' for backward Motorola format

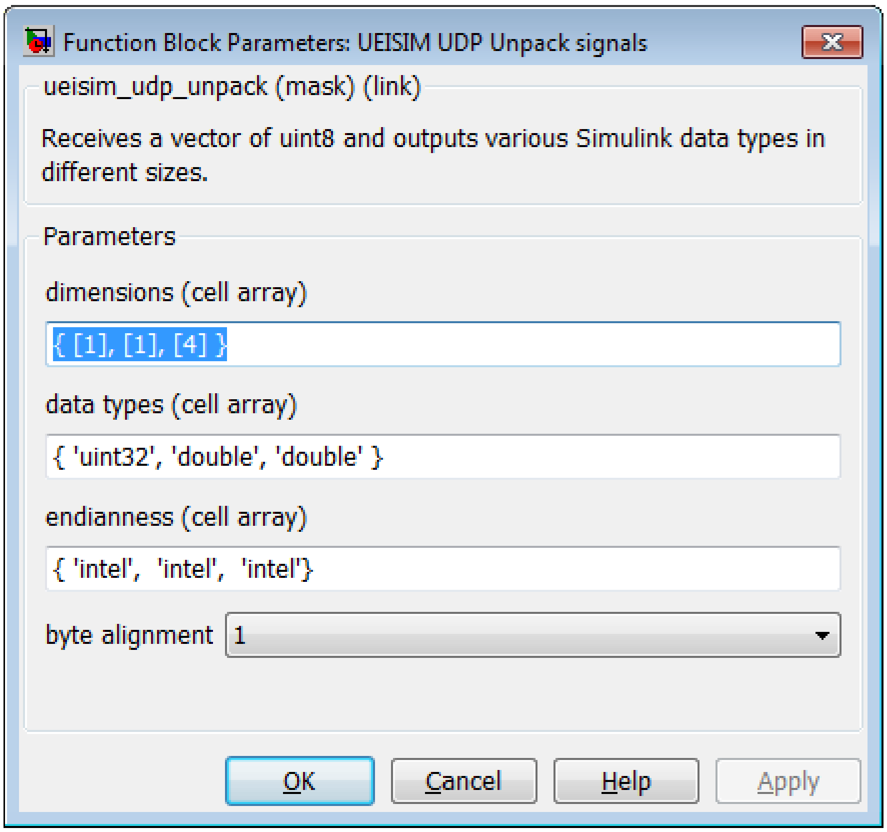
5.26.3.1. UEISIM Pack block



- **Data types**: A cell array containing the data types of the structure members to pack in the buffer
- **Endianness**: A cell array containing the endianness of the signals to pack
- **Byte alignment**: The minimum number of bytes occupied by each member. Possible values are 1,2,4 and 8. For example with align=4, int8 and uint8 members will occupy 4 bytes with 3 zero bytes for padding.

The block automatically converts itself to one with the correct number of input ports. There is always one output port of type uint8. The output value is ready to be connected to the UDP Send block.
The UEISIM UDP or TCP/IP Send block needs to be configured to send data of type **uint8**.

5.26.3.2. UEISIM Unpack block

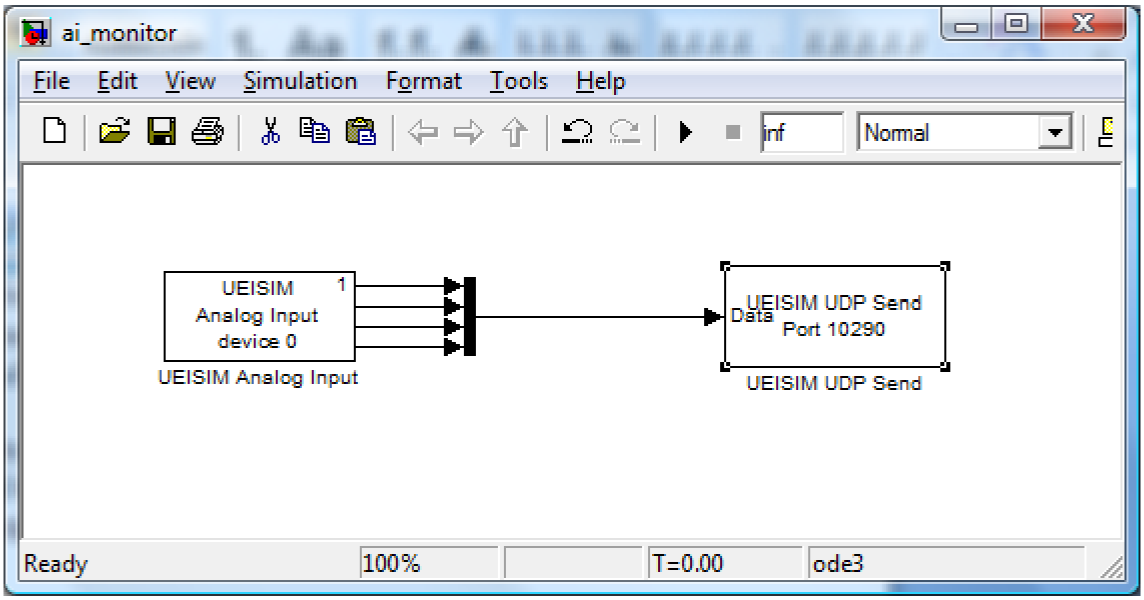


- **Dimensions**: A cell array containing the dimensions (as returned by MATLAB size() function) of the corresponding signal.
- **Data types**: A cell array containing the data types of the structure members to unpack from the buffer
- **Endianness**: A cell array containing the endianness of the signals to unpack
- **Byte alignment**: The minimum number of bytes occupied by each member. Possible values are 1,2,4 and 8. For example with align=4, **int8** and **uint8** members will occupy 4 bytes with 3 zero bytes for padding.

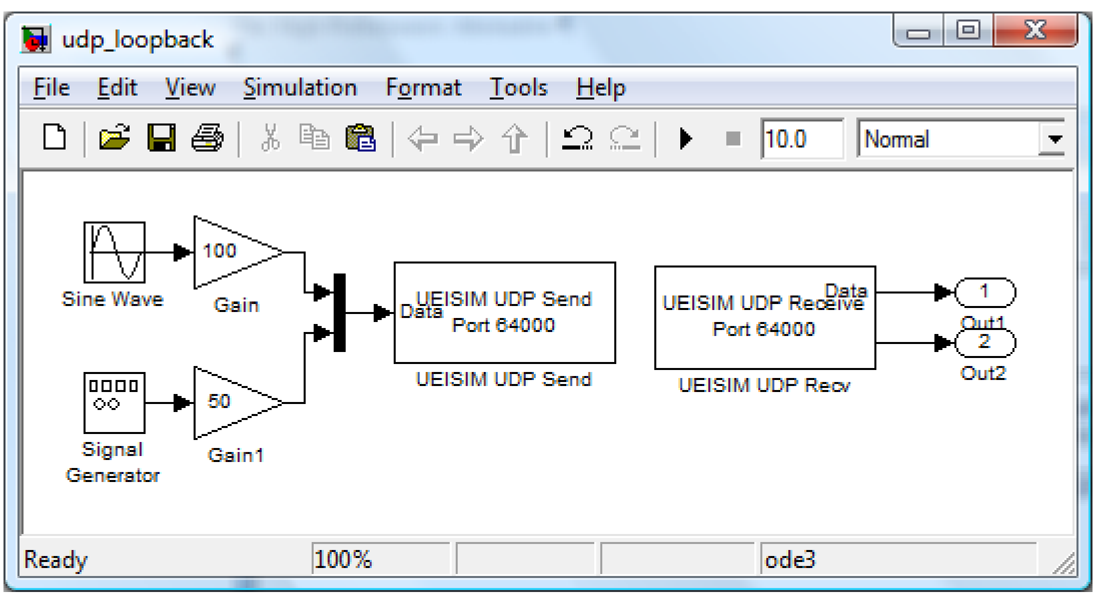
The block displays one input port to connect a **uint8** vector coming from the UDP Receive block. The block automatically converts itself to one with the correct number of output ports.
The UEISIM TCP/Ip or UDP Receive block needs to be configured to receive a vector of type **uint8** whose dimension is the size occupied by all members defined in the unpack block (in bytes).

5.26.4. UDP example

The following example acquires analog input channels and sends over the result to a network host.

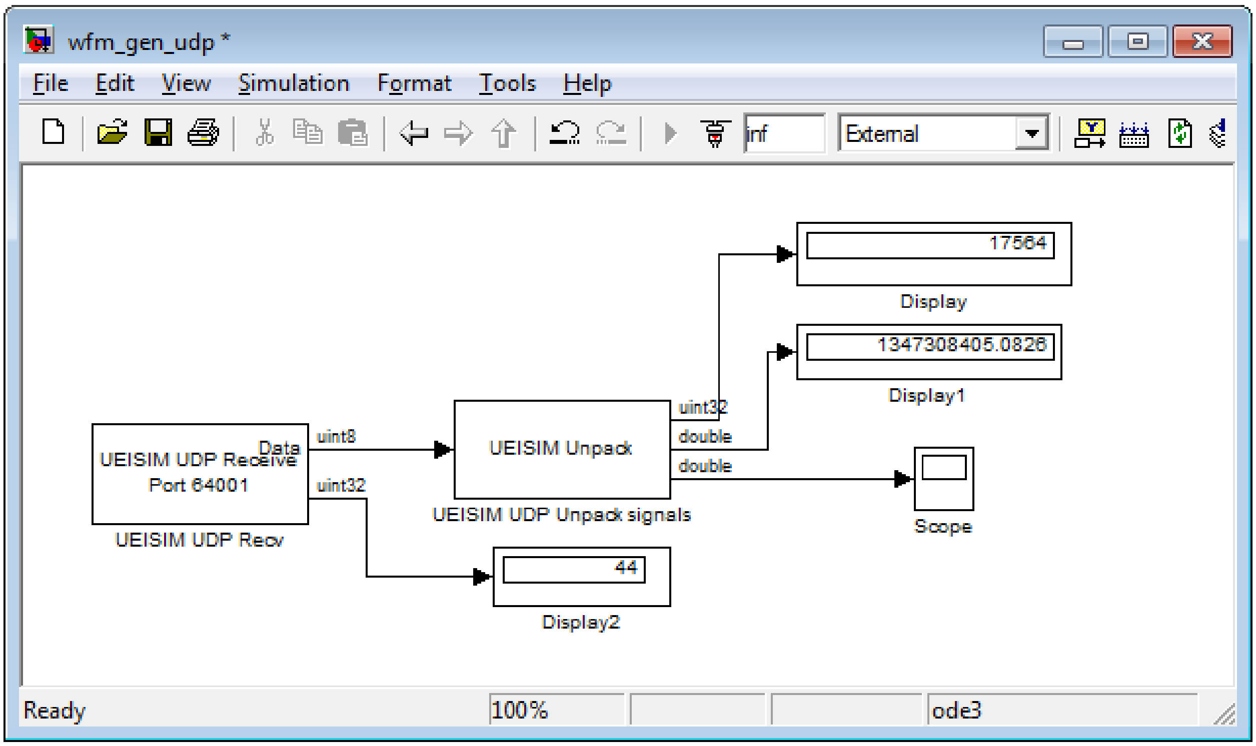


The following example sends simulated data and receives it too (IP address must be set to 127.0.0.1).



The High-Performance Alternative

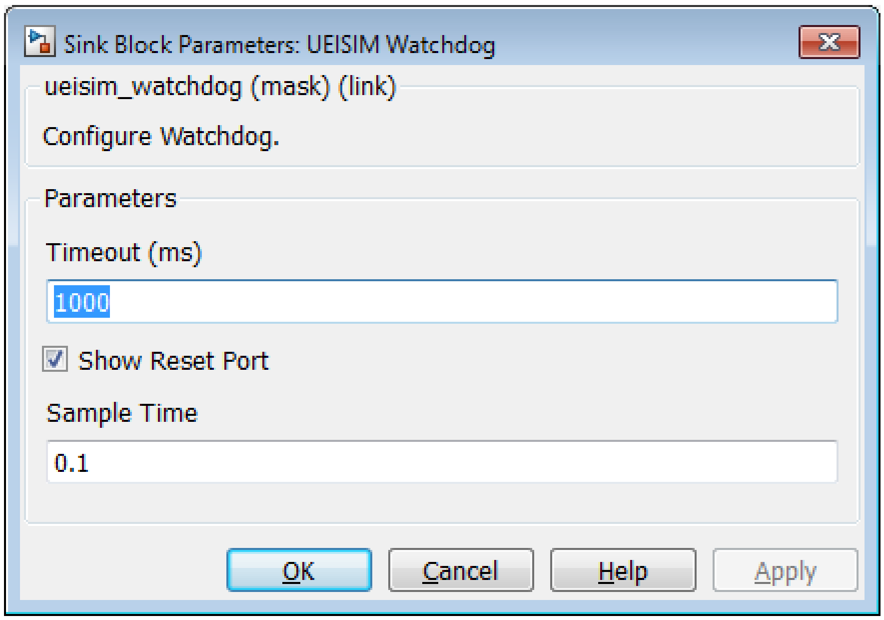
The following example receives 44 bytes from UDP port 64001 and decodes them as one uint32, one double and a vector of 4 doubles.



5.27. *Miscellaneous*

5.27.1. Watchdog block

A hardware watchdog can be configured to reboot the UEISIM if the model hangs or takes too long to complete a step.

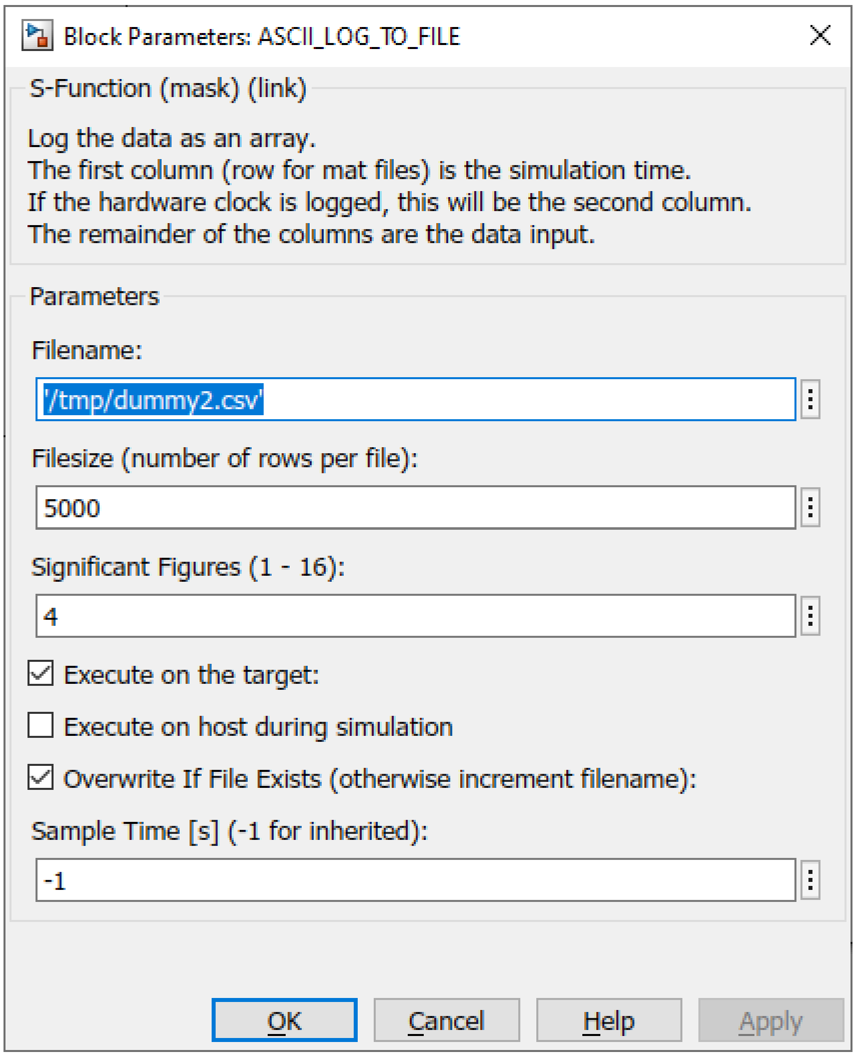


- **Timeout**: The watchdog timeout delay in milliseconds. UEISIM will reboot if watchdog isn't reset before timeout expires.
- **Show Reset port**: Allows to optionally connect a reset signal
- **Sample Time**: The rate at which the block executes during simulation

When **Show Reset Port** is checked, this block displays an input port for connecting a reset signal. The watchdog resets whenever the input signal value is greater or equal than 0.5.
Otherwise the watchdog is reset each time this block is executed.

5.27.2. Data logging to file

This blocks logs scalars, vectors and matrices to CSV files.
Data is logged across multiple. You can specify the number of bytes to log to a CSV file before creating a new file.
New files are named with the convention <filename>_####.csv, where #### starts at 0001 and increments with each new file.



- **Filename**: The name of the data log file. Note that the name will be modified to include a file index (<filename>_####.<ext>)
- **Filesize**: The maximum size of each data file
- **Significan figures**: Number of digits after decimal point in log file (useful to reduce file size)
- **Execute on target**: Data logging will execute when block is executed on UEISIM
- **Execute on host**: Data logging will execute when block is executed on host during Simulink execution.
- **Overwrite**: When checked, existing file with specified name will be deleted. Otherwise, file index is incremented until an available file name is found.

- **Sample Time**: The rate at which data logging is executed (typically -1 to inherit the sample time of logged signals)