1

Work with Arduino Hardware

Install Support for Arduino Hardware

This topic shows how to add support for Arduino® hardware to the Simulink® product. After you complete this process, you can run Simulink® models on your Arduino® hardware.

The installation process adds the following items to your host computer:

- Third-party software development tools, such as the Arduino® software
- A Simulink block library for configuring and accessing Arduino® sensors, actuators, and communication interfaces
- Demos for getting started and learning about specific features

To check for updates, repeat this process when a new version of MATLAB software is released. You can also check for updates between releases.

Note You can use this software on host computers running versions of 32-bit or 64-bit Windows® that Simulink® software supports.

To install support for Arduino® hardware:

1. Start Target Installer using one of the following methods:
   - In a MATLAB® Command Window, enter `targetinstaller`.
   - In a model, click the **Tools** menu, and select **Run on Target Hardware** > **Install/Update Support Package**.
2. Choose to get the support package from the Internet or from a folder.

Note The time required to downloading the support package and third-party software varies depending on the bandwidth and speed of your Internet connection.

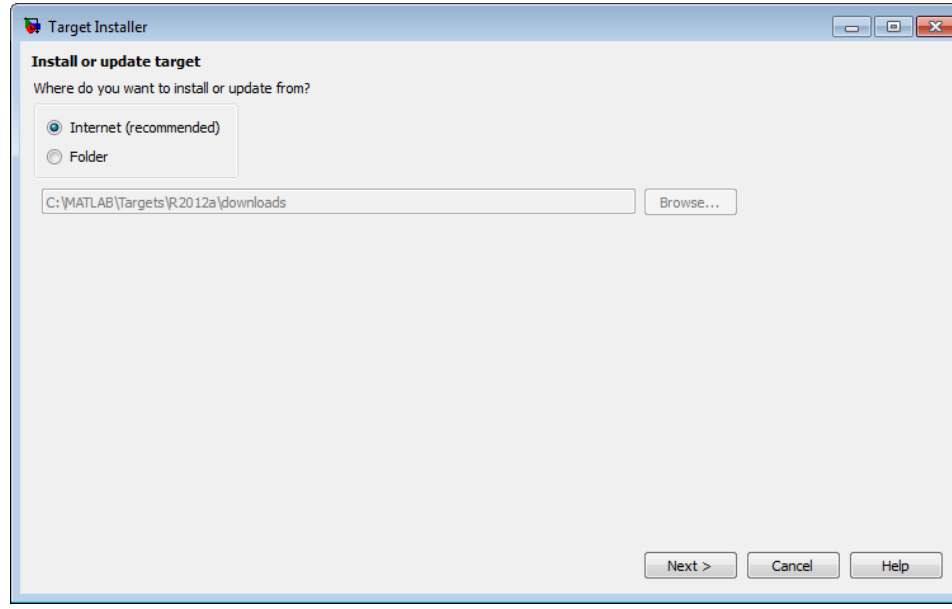
Internet: This option is selected by default. Target Installer downloads and installs the support package and third-party software from the Internet.

Folder: Target Installer gets the support package and third-party software installers from the specified folder. If the third-party software installers are not available, Target Installer downloads and installs those from the Internet.

You must have write privileges for this folder. Having write privileges for the default folder is typically not an issue. If you change to a new folder for which you do not have write permissions, such as a shared folder on a network, Target Installer generates an error message.

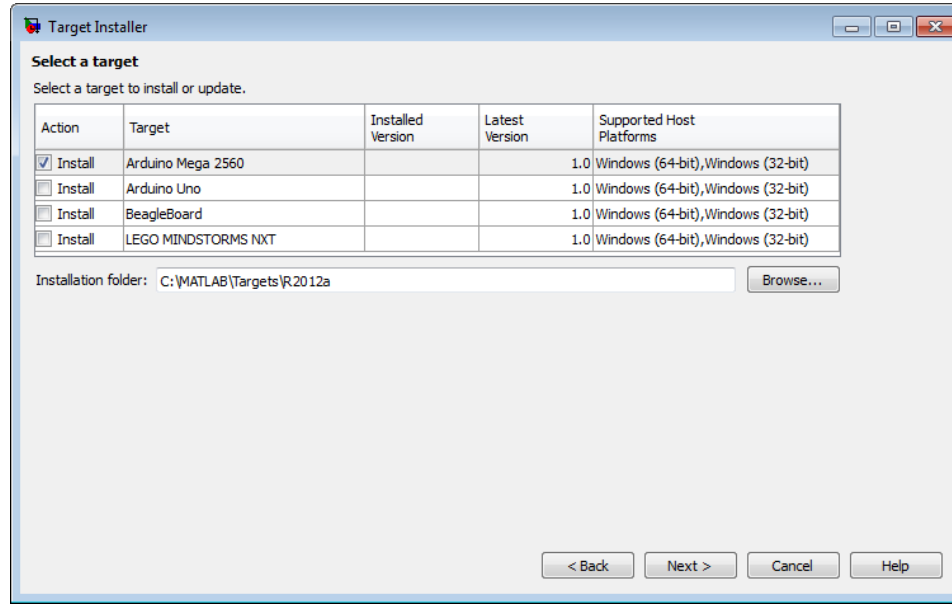
To solve this issue, copy the support package to a folder for which you have write privileges, and point Target Installer to that same folder. For example, copy the support package to `C:\MATLAB\Targets\`*`version`*`\downloads`. If they are available, also copy the third-party software installers associated with the support package.

To locate the support package in a folder, search for a filename that begins with `rtt_arduino` and ends with `.zip`. For example: `rtt_arduinouno_r2012a_v1_0.zip`



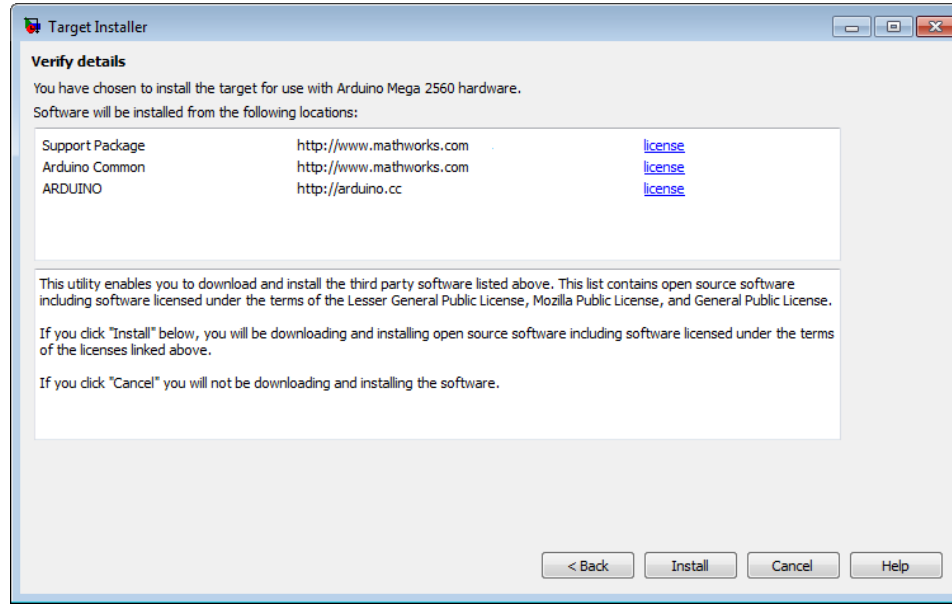
3 Select the **Install** check box for either Arduino® target, and click **Next >**. (To install another target, run Target Installer again later on.)

The **Installation folder** parameter tells Target Installer where to install the target and associated third-party software. You must have write privileges for this folder.

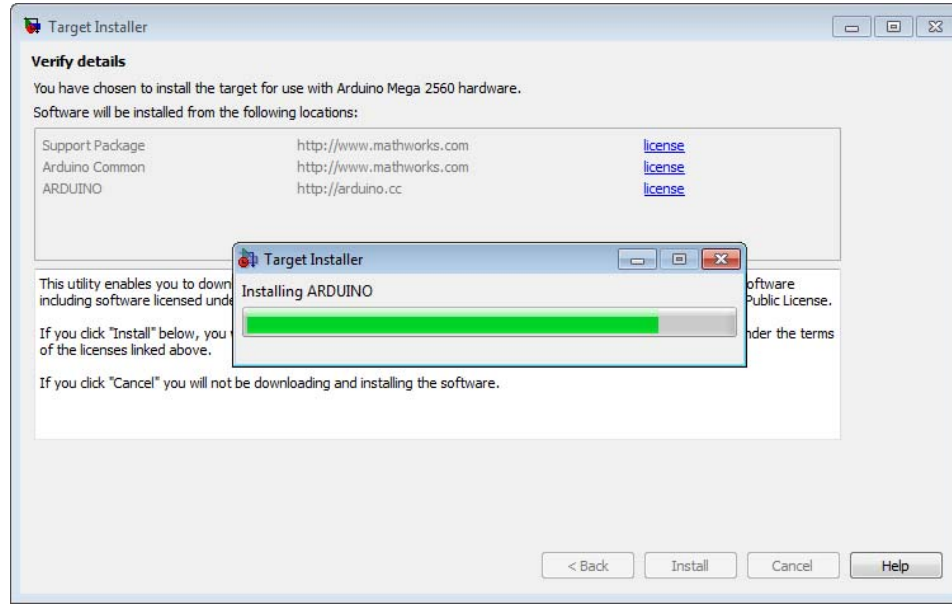


4 Target Installer confirms that you are installing the target, and lists third-party software it will install.

Review the information, including the license agreements, and click **Install**.

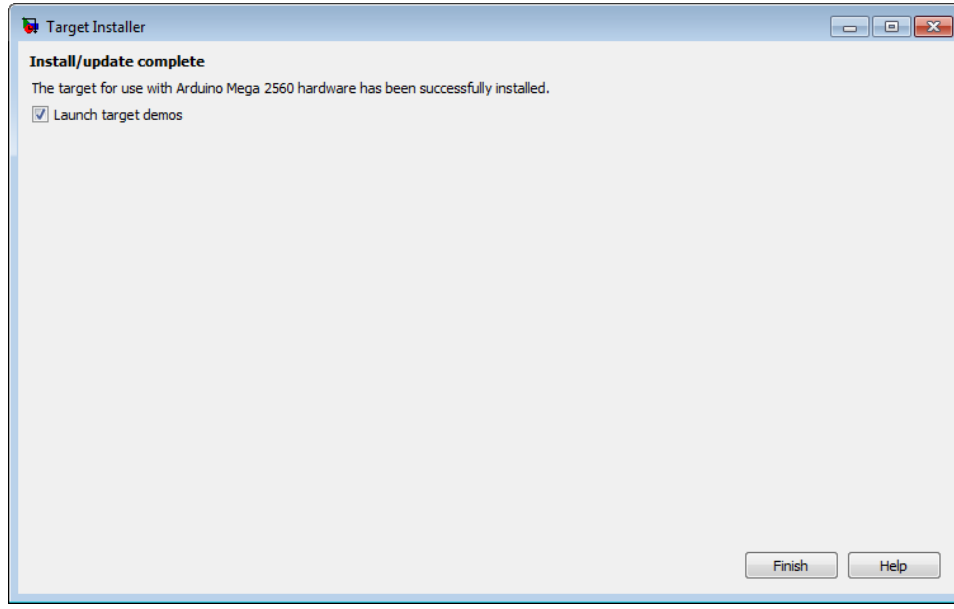


Target Installer displays a progress bar while it downloads and installs the third-party software.



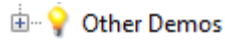
Note If you installed the target previously, Target Installer may remove files from that installation before installing the current target. If Target Installer cannot remove those files automatically, it instructs you to delete the files manually. Close the MATLAB® software before removing the files. Then restart MATLAB® software and run Target Installer again.

5 To view target demos, leave the **Launch target demos** checkbox selected and click **Finish**.



The Help that opens displays the appropriate demos for your hardware.

To find these demos again later, enter `doc` in the MATLAB Command Window. In the Help that opens, look for **Other Demos** at the bottom of the **Contents** list.



Open Block Libraries for Arduino Hardware

You can open the block libraries for your Arduino® hardware from the MATLAB Command Window or from the Simulink Library Browser.

The blocks in these block libraries provide support for various peripherals available on the Arduino® hardware.

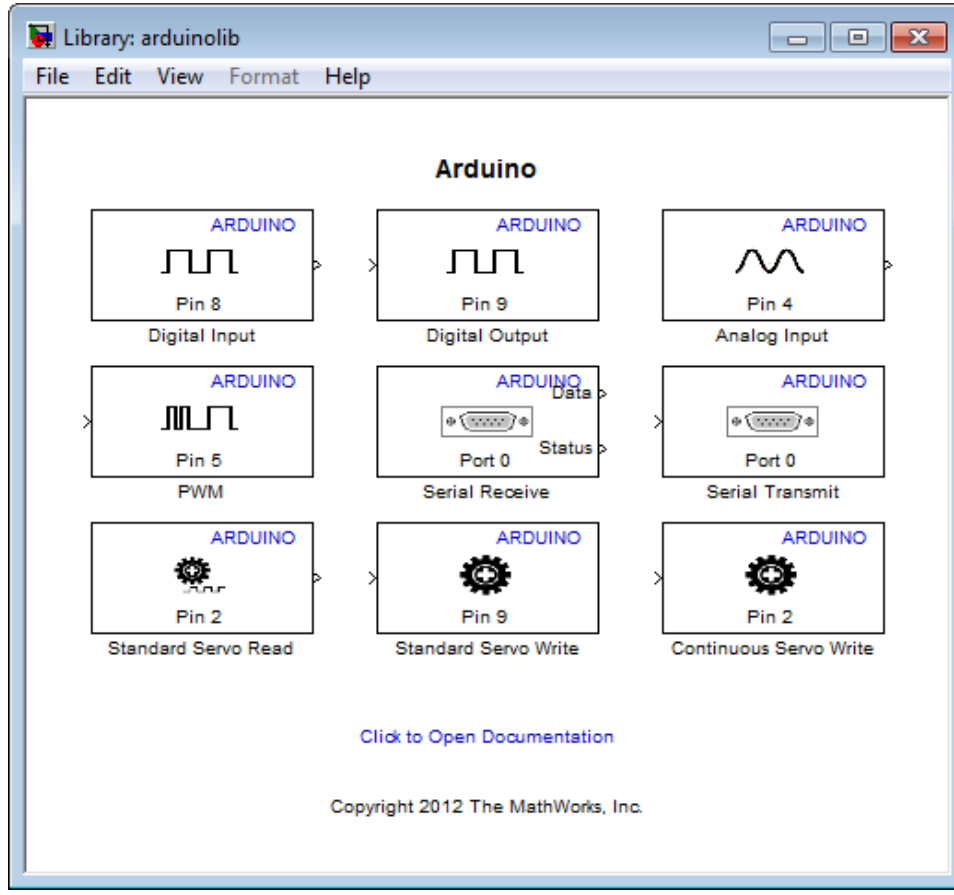
From the Command Line

After installing support for your Arduino® hardware, you can open its block library from the MATLAB® Command Window.

After installing support for Arduino® hardware, enter:

```
arduinolib
```

The software opens the corresponding block library.



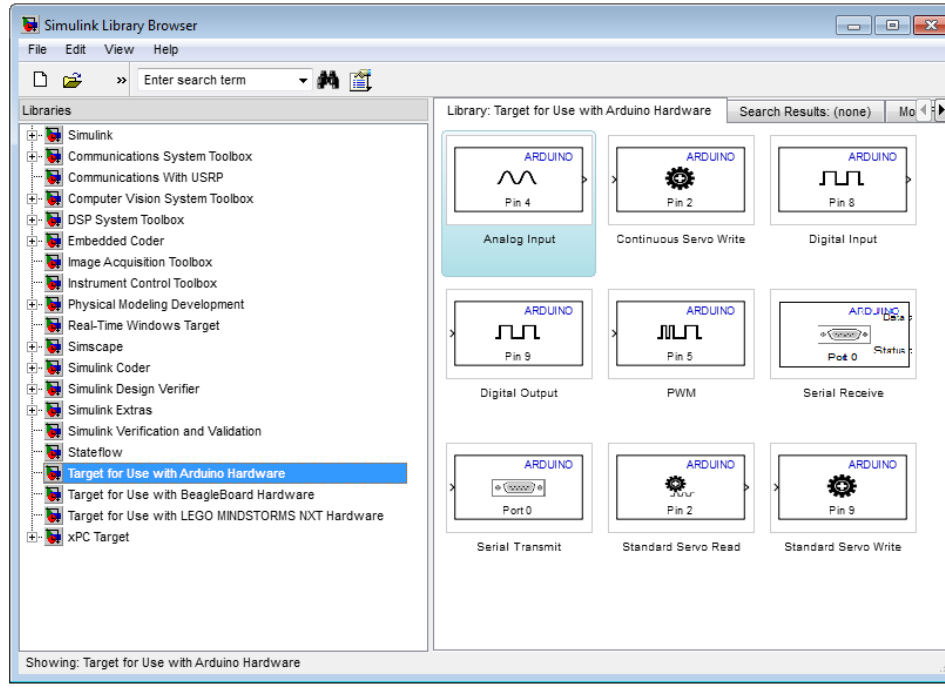
From the Simulink Library Browser

To open the block library from the Simulink® Library Browser:

Enter `simulink` in the MATLAB® Command Window, or click the following icon on the MATLAB® toolbar.

In the Simulink® Library Browser, click **Target for Use with Arduino Hardware**.

Simulink Library Browser displays the corresponding block library.



Run Model on Arduino Hardware

You can prepare, configure, and run a model on your Arduino® hardware.

Before starting:

- Connect your Arduino® hardware to the host computer using a USB cable.
- Create or open a Simulink® model located on a local drive or a mapped network drive that has a drive letter assigned to it.

  The software generates an error message if the location of the model contains a UNC path. For example, `\\server-00\user$\MATLAB\`

To prepare and run the model:

1. Use **File > Save As** to create a working copy of your model. Keep the original model as a backup copy.
2. Click the **Tools** menu in the model, and select **Run on Target Hardware > Prepare to Run**. This action changes the model Configuration Parameters.
3. In the Run on Target Hardware pane that opens, set the **Target hardware** parameter to `Arduino Mega 2560` or `Arduino Uno`.
4. Click the **Tools** menu, and select **Run on Target Hardware > Run**. This action automatically downloads and runs your model on the Arduino® hardware.

   The lower left corner of the model window displays status while Simulink® software prepares, downloads, and runs the model on the target hardware.

   To stop a model running on Arduino® hardware, you can:

   - Disconnect the power from the hardware. When you reconnect the power, the model will start running again.
   - Run a new or updated model on the hardware. This action automatically stops and erases the previous model running on the Arduino® hardware.

   To restart the model running on the Arduino® hardware, press the RESET button on the board.

Prepare Models That Use Model Reference

You can include one model in another by using Model blocks. Each instance of a Model block represents a reference to another model, called a *referenced model* or *submodel*. The model that contains a referenced model is its *parent model*. When you run the parent model on your target hardware, the submodel effectively replaces the Model block that references it. For more information, see "Overview of Model Referencing"

To run on target hardware, the parent model and the submodels must have the same Configuration Parameter settings.

For each Model block:

1. Open the model associated with the Model block.
2. Click the **Tools** menu, and select **Run on Target Hardware** > **Prepare to Run**.
3. Apply the same Configuration Parameters settings to the submodel as you applied to the parent model.

If the model and Model blocks have different settings, the software generates an error when you try to run the model on the target hardware.

See Also

- "Creating the Simple Model"
- "Tune and Monitor Models Running on Arduino Mega 2560 Hardware" on page 1-14
- "Overview of Model Referencing"

Tune and Monitor Models Running on Arduino Mega 2560 Hardware

In this section...

About External Mode

You can use External mode to tune parameters in, and monitor data from, your model while it is running on the Arduino® Mega 2560 hardware. This capability is not available with Arduino® Uno hardware.

External mode enables you to tune model parameters and evaluate the effects of different parameter values on the model results in real-time, in order to find the optimal values to achieve the desired performance. This process is called *parameter tuning*.

External mode accelerates parameter tuning because you do not have to re-run the model each time you change parameters. External mode also lets you develop and validate your model using the actual data and hardware for which it is designed. This software-hardware interaction is not available solely by simulating a model.

The following list provides an overview of the parameter tuning process with External mode:

- In the model on your host computer, you enable External mode in the Configuration Parameters.
- In the model on your host computer, you configure Simulink® software to run your model on the target hardware.
- You use the model on the host computer as a user interface for interacting with the model running on the target hardware:

  - When you open blocks and apply new parameter values on the host computer, External mode updates the corresponding values in the model running on the target hardware.
  - If your model contains blocks for viewing data, such as Scope or Display blocks, External mode sends the corresponding data from the target hardware to those blocks on the host computer.
- You determine the optimal parameter settings by adjusting parameter values on the host computer and observing data/outputs from the target hardware.

When you have finished tuning a model, you can disable External mode and run the tuned model on your hardware.

Some limitations apply while you are using External mode:

- Do not configure Serial Receive or Serial Transmit blocks in your model to use serial port 0. External mode uses serial port 0.
- Do not use the following Arduino® servo blocks: Standard Servo Read, Standard Servo Write, and Continuous Servo Write.

Run Your Model in External Mode

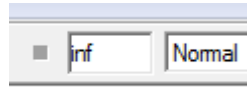
Before starting:

- Connect your Arduino® Mega 2560 hardware to the host computer using a USB cable.
- Create or open a Simulink® model located on a local drive or a mapped network drive that has a drive letter assigned to it.

  The software generates an error message if the location of the model contains a UNC path. For example, `\\server-00\user$\MATLAB\`

To prepare and run the model:

1. Open your Simulink® model.
2. In the model, set the **Simulation stop time** parameter, located on the model toolbar, as shown here.



- To run the model for an indefinite period, enter `inf`.
- To run the model for a finite period, enter a number of seconds. For example, entering `120` runs the model on the Arduino® hardware for 2 minutes.

3. Click the **Tools** menu, and select **Run on Target Hardware** > **Options**.

4. In the Run on Target Hardware pane that opens, select the **Enable External mode** checkbox.

5. Click **OK**, and then save the changes to your model.

6. Click the **Tools** menu, and select **Run on Target Hardware** > **Run**.

   The lower left corner of the model window displays status while Simulink® software prepares, downloads, and runs the model on the target hardware.

7. While the model is running in External mode, you can change tunable parameter values in the model on your host computer and observe the corresponding changes in the model running on the hardware.

   If your model contains blocks from the Simulink® Sinks block library, the sink blocks in the model on your host computer display the values generated by the model running on the hardware.

   If your model does not contain a sink block to which External mode can send data, the MATLAB® Command Window displays a "No data has been selected for uploading" warning. You can disregard this warning, or you can add a sink block to the model and rerun your model.

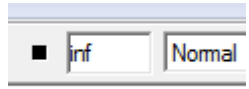
8. When you have finished tuning and monitoring your model, you can disable External mode.

To deploy model to your hardware without using External mode. See "Run Model on Arduino Hardware" on page 1-12:

Note External mode increases the processing burden of the model running on the hardware. If the software reports an overrun, you can apply the solutions described in "Detect and Fix Task Overruns on Arduino Hardware" on page 1-21.

Stop External Mode

To stop the model running in External mode, click the black square Stop button located on the model toolbar, as shown here.



This action stops the process for the model running on the Arduino® hardware, and stops the model simulation running on your host computer.

If it is set to a finite period, the **Simulation stop time** parameter stops External mode when the period elapses.

Use Serial Communications with Arduino Hardware

In this section...

Arduino® hardware has serial ports, also known as UARTs, that can communicate with other devices that have serial interfaces.

Hardware

The Arduino® Uno board has one serial port, serial port 0, connected to:

- The digital pins marked `TX 1` (transmit) and `RX 0` (receive).
- The USB port, through a serial-to-USB converter.

The Arduino® Mega 2560 board has four serial ports:

- Serial port 0 is connected to Communication pins marked `TX0 1` (transmit) and `RX0 0` (receive). Serial port 0 is also connected to the USB port through a converter.
- Serial port 1 is connected to Communication pins marked `TX1 18` (transmit) and `RX1 19` (receive).
- Serial port 2 is connected to Communication pins marked `TX2 16` (transmit) and `RX2 17` (receive).
- Serial port 3 is connected to Communication pins marked `TX3 14` (transmit) and `RX3 15` (receive).

You can use serial port 0 to communicate with other devices that have serial ports, or to communicate with a computer over the USB port.

Each serial port supports one Serial Transmit and one Serial Receive block, one block per pin.

If you intend to use External mode with Arduino® Mega 2560 hardware, use serial ports 1 through 3 for serial communications. Serial port 0 is not available for serial communications because it is connected to the USB port, which External mode uses to communicate with the host computer. This restriction does not apply to Arduino® Uno hardware, because External mode is not supported. For more information, see "Tune and Monitor Models Running on Arduino Mega 2560 Hardware" on page 1-14.

Serial communications are not supported in models that also use the Arduino® Standard Servo Read, Standard Servo Write, and Continuous Servo Write blocks.

Warning **Only connect serial port pins to devices that use 5 Volt TTL logic. Do not connect these pins to an RS-232 serial interface, such as the DE-9M connector on a computer, without limiting the voltage. The RS-232 standard allows higher voltages that can damage your hardware. For details, read the documentation for your Arduino® hardware.**

Transmit Serial Data

1. Add the Arduino® Serial Transmit block to your model.
2. Connect a data source to the block input on the Serial Transmit block.

   If the data type is not uint8, use a Data Type Conversion block to convert it to uint8.
3. In the Serial Transmit block, specify a **Port number**.
4. Click the **Tools** menu in the model, and select **Run on Target Hardware > Options**.

   In the Configuration Parameters dialog that opens, set the baud rate parameter of the serial port you specified in the Serial Transmit block.
5. Connect the appropriate digital transmit pin, or the USB port, to the hardware that receives the data.
6. Build and run the model.

Receive Serial Data

1. Add the Arduino® Serial Receive block to your model.
2. Connect the **Data** block output to a block that uses the data.
3. In the Serial Receive block, specify the **Port number**.
4. Click the **Tools** menu in the model, and select **Run on Target Hardware > Options**.

   In the Configuration Parameters dialog that opens, set the baud rate parameter of the serial port you specified in the Serial Receive block.
5. Connect the appropriate digital receive pin, or the USB port, to the hardware that transmits the data.
6. Build and run the model.

Detect and Fix Task Overruns on Arduino Hardware

You can configure a Simulink® model running on the target hardware to detect and notify you when a task overrun occurs. A task overrun occurs if the target hardware is still performing one instance of a task when the next instance of that task is scheduled to begin. You can fix overruns by decreasing the frequency with which tasks are scheduled to run, and/or by reducing the number of tasks defined by your model.

To enable overrun detection:

1. Click the **Tools** menu in the model, and select **Run on Target Hardware** and **Options**.
2. In the Run on Target Hardware pane that opens, select the **Enable overrun detection** check box.
3. Use the **Digital output to set on overrun** parameter to specify the pin number of a digital output.
4. Click **OK**.

To create a visual overrun indicator for your board, connect an appropriate resistor in series with an LED between the `GND` and the hardware pin specified by the **Digital output to set on overrun** parameter. Orient the LED so the longer leg (positive) is connected to the digital output pin.

When a task overrun occurs:

- The state of the digital output pin specified by the **Digital output to set on overrun** parameter changes from low (0 Volts) to high (5 Volts).
- The model continues running, but the effective sample time will be longer than specified.

To fix an overrun condition, reduce the processing burden of the model by applying one or more of the following solutions:

- Increase the sample times for the model. For example, increase the values of the **Sample time** parameters in all of your data source blocks.

- Simplify the model.

If you are using External mode, and the preceding solutions do not fix the task overrun condition, consider clearing the **Enable External mode** checkbox in the Run on Target Hardware pane. External mode adds a lightweight server to the model running on the target hardware. This server increases the processing burden upon the target hardware, which can contribute to a task overrun condition.

Troubleshoot Running Models on Arduino Hardware

In this section...

Block Produces Zeros in Simulation

If you simulate a model on your host computer without running it on your target hardware, input blocks produce zeros, and output blocks do nothing. This is the expected behavior.

For example, if you select **Simulation > Run** in a model that contains a Digital Input block and a Digital Output block:

- The block output on the Digital Input block produces zeros.
- The Digital Output block does nothing.

To solve this issue, run your model on the target hardware as described in:

- "Run Model on Arduino Hardware" on page 1-12
- "Run Your Model in External Mode" on page 1-15

"Could not automatically set host COM port"

If you try to run a model on your Arduino® hardware and Simulink generates an error message similar to this one: "The call to realtime_make_rtw_hook, during the entry hook generated the following error: Could not automatically set host COM port for your Arduino® hardware. This may be due to a disconnected or unrecognized board. If the board is not connected to your host computer, connect it and let the operating system install the board driver."

First, resolve any connection issues:

1. Verify that your Arduino® hardware is powered on and connected to your host computer.
2. Try running the model again on your Arduino® hardware.

If you get the error message while your board is powered on and connected to your host computer, resolve any issues with Arduino® drivers in Windows:

1. In Windows, click the **Start** menu and select **Devices and Printers**.
2. If you find an **Unknown Device** under **Other Devices** or **COM Ports**, double click the **Unknown Device**.
3. In the **Unknown Device Properties** dialog box that opens, click the **Hardware** tab, and click **Properties**.
4. In the **Unknown device Properties** dialog box that opens, click **Update Driver**.
5. In the **Update Driver Software - Unknown Device** dialog box that opens, click **Browse my computer for driver software**.
6. Select the **Include subfolders** checkbox and click **Browse**.
7. Navigate to the **Installation folder** that Target Installer used when you installed support for your Arduino® hardware, and then click **Next**. By default, this folder location is `C:\MATLAB\Targets\`*`releasenumber`*`\arduino-`*`version`*. For example: `C:\MATLAB\Targets\R2012a\arduino-1.0`.
8. If prompted by Windows Security, choose **Install this driver software anyway**, and let Windows complete the process of installing the driver.
9. Try running the model again on your Arduino® hardware.

If you get the error message after resolving issues with Arduino® drivers, resolve any issues with the COM port settings. The drivers for some Arduino® board revisions do not identify the board as an Arduino® device in Windows®. In that case, set the COM port number manually:

1. Click the **Tools** menu in the model, and select **Run on Target Hardware > Options**.
2. In the Run on Target Hardware pane, change the **Set host COM port** parameter to `Manually` and leave the Configuration Parameters dialog open.

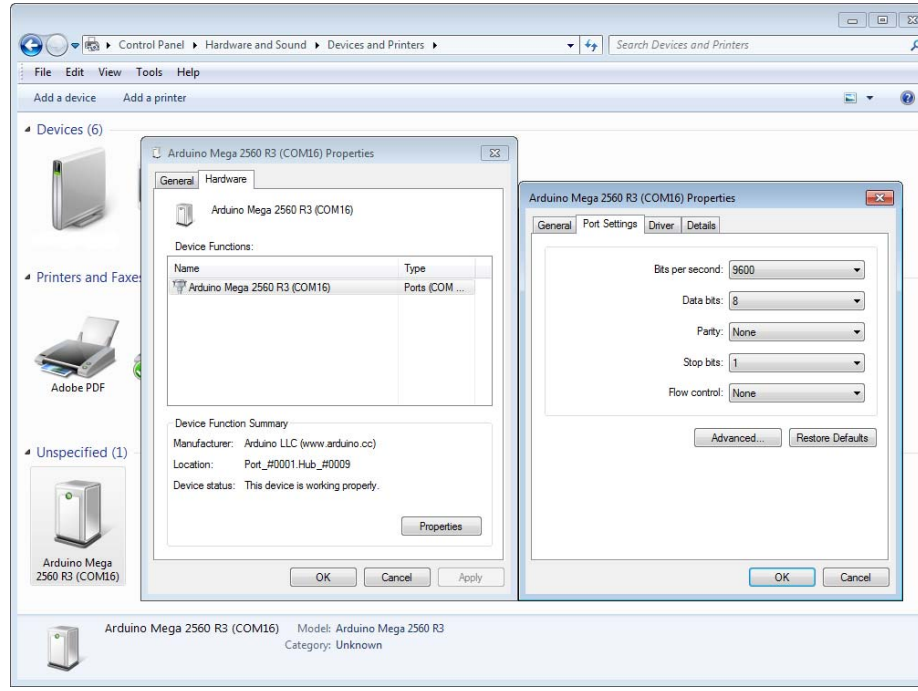
3 Open **Devices and Printers** in Windows.

4 Double-click **Arduino Uno** or **Arduino Mega 2560** device.

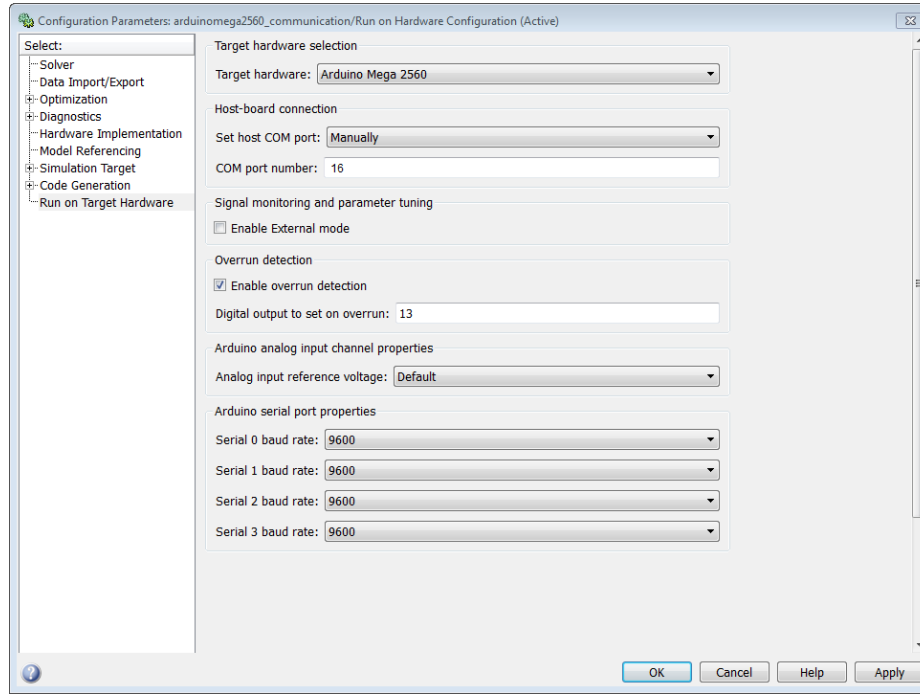
5 In the device properties dialog, click the **Hardware** tab, and then click the **Properties** button.

6 Click the **Port Settings** tab.

For example, the following image shows an Arduino® device in Devices and Printers, the Hardware tab, and the Port Settings tab.



7 In Configuration Parameters, update the **COM port number** and **Serial 0 baud rate** parameters to match those of the Arduino® device in Windows.



8 Apply the new Configuration Parameter values, and try running the model on your Arduino® hardware again.

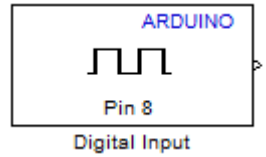
Arduino Digital Input

Purpose

Get logical value of digital input pin

Library

Target for Use with Arduino® Hardware



Description

Get the logical value of a digital pin on the Arduino® hardware:

- If the logical value of the digital pin is LOW, the block outputs 0.
- If the logical value of the digital pin is HIGH, the block outputs 1.

The data type of the block output is uint8.

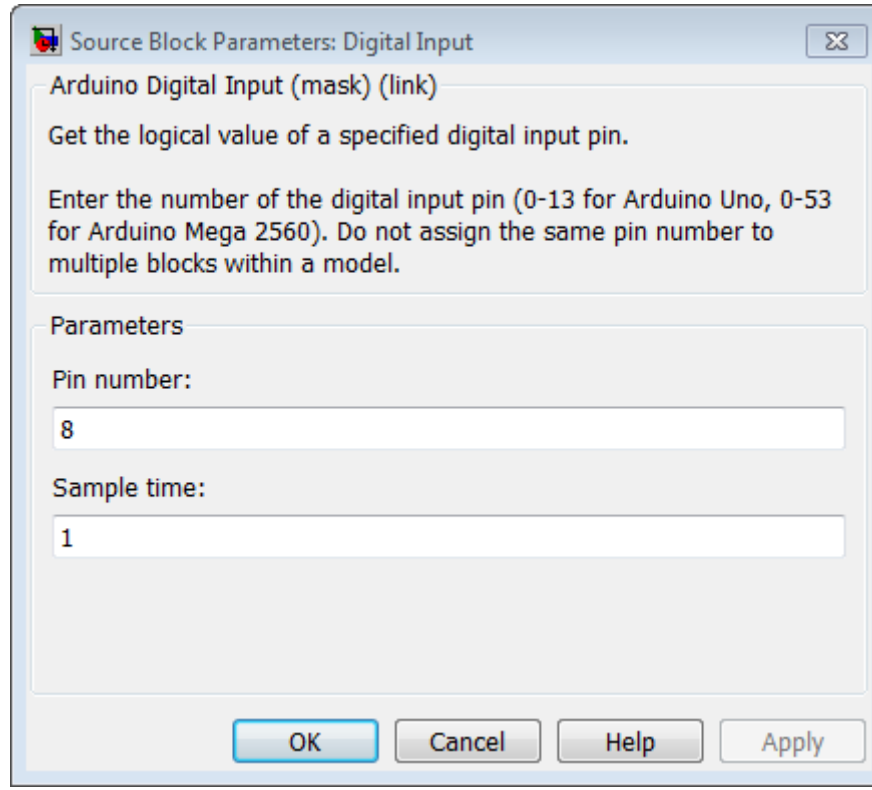
If you simulate your model without running it on the target hardware, this block outputs zeroes. See “Block Produces Zeros in Simulation”

Warning

Only connect digital pins to devices that use 5 Volt TTL logic. For details, read the documentation for your Arduino® hardware.

Arduino Digital Input

Dialog



Pin number

Enter the number of the digital pin.

Do not assign the same pin number to multiple blocks within the model.

For Arduino® Mega 2560, enter a pin number from `0` to `53`.

For Arduino® Uno, enter a pin number from `0` to `13`.

Sample time

Specify how often this block reads the pin value, in seconds. Enter a value greater than zero. This value defaults to a sample time of `1` second. The minimum value is `0.000001` second.

Smaller values require the processor to complete the same number of instructions in less time, which can cause task overruns.

See Also

Arduino Digital Output | "Install Support for Arduino® Hardware" | "Detect and Fix Task Overruns on Arduino® Hardware" |

External Links

- http://arduino.cc/en/Reference/DigitalRead

Arduino Digital Output

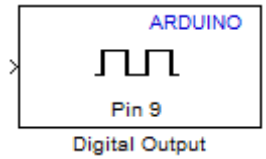
Purpose

Set logical value of digital output pin

Library

Target for Use with Arduino® Hardware

Description



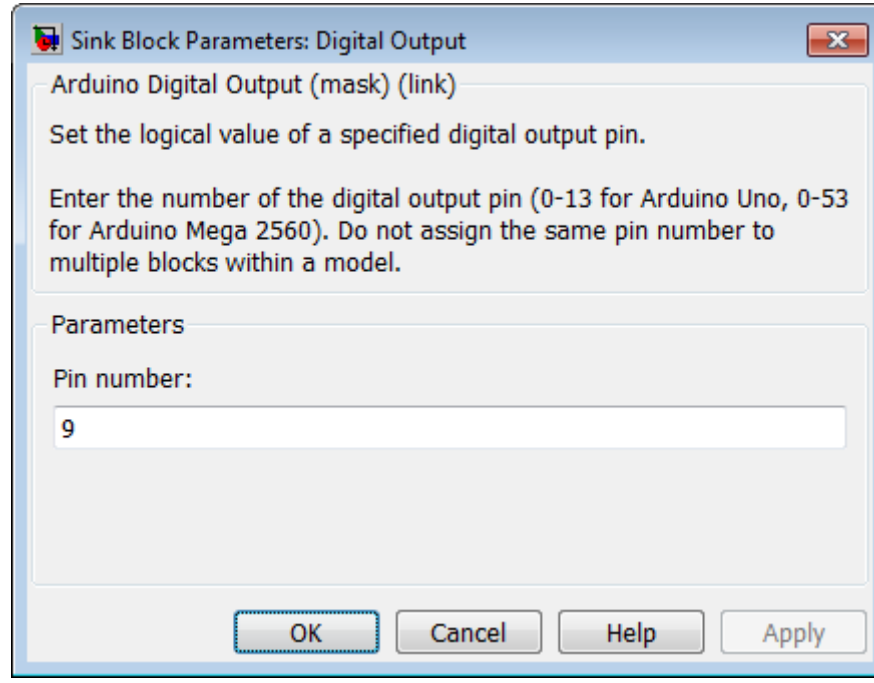
Set the logical value of a digital pin on the Arduino® hardware:

- Sending `1` to the block input sets the logical value of the digital pin HIGH to 5 V or 3.3 V, depending on the board voltage.
- Sending `0` to the block input sets the logical value of the digital pin LOW to 0 V.

The block input inherits the data type of the upstream block, and internally converts it to boolean.

If you simulate your model without running it on the target hardware, this block does nothing. See “Block Produces Zeros in Simulation”.

Dialog



Pin number

Enter the number of the digital output pin.

Do not assign the same pin number to multiple blocks within the model.

For Arduino® Mega 2560, enter a pin number from 0 to 53.

For Arduino® Uno, enter a pin number from 0 to 13.

See Also

Arduino Digital Input | “Install Support for Arduino® Hardware” |

External Links

- http://arduino.cc/en/Reference/DigitalWrite

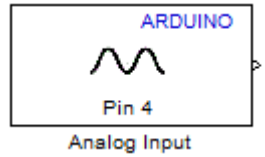
Arduino Analog Input

Purpose

Measure voltage of analog input pin

Library

Target for Use with Arduino® Hardware



Description

Measure the voltage of an analog pin relative to the analog input reference voltage on the Arduino® hardware. Output the measurement as a 10-bit value that ranges from 0 to 1023.

- If the measured voltage equals the ground voltage, the block outputs `0`.
- If the measured voltage equals the analog reference voltage, the block outputs `1023`.

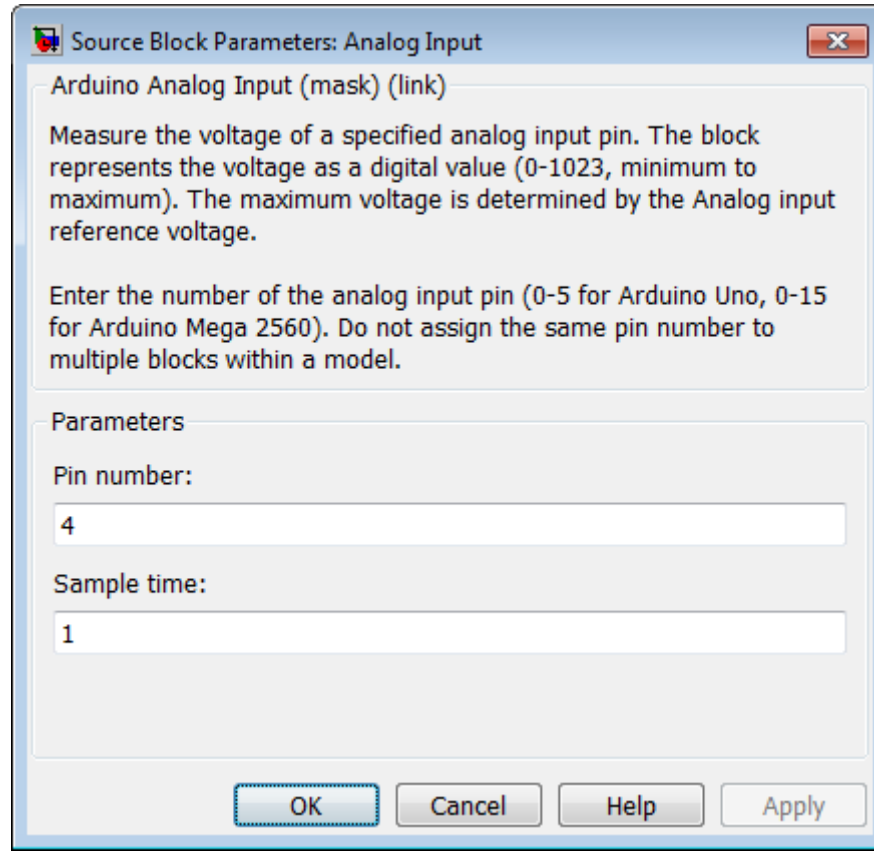
The default value of the analog input reference voltage is 0 to 5 V. To change the **Analog input reference voltage** parameter in your model Configuration Parameters, select **Tools > Run on Target Hardware > Options...**.

If you simulate your model without running it on the target hardware, this block outputs zeroes. See “Block Produces Zeros in Simulation”

Warning

The range of the voltage that can be applied to the analog input pin depends on the analog input reference voltage. For details, read the documentation for your Arduino® hardware.

Dialog



Pin number

Enter the number of the analog input pin.

Do not assign the same pin number to multiple blocks within the model.

For Arduino® Mega 2560, enter a pin number from `0` to `15`.

For Arduino® Uno, enter a pin number from `0` to `5`.

Sample time

Specify how often this block reads the pin value. Enter a value greater than zero. This value defaults to a sample time of `1` second. The minimum value is `0.000001` second.

Smaller values require the processor to complete the same number of instructions in less time, which can cause task overruns.

See Also

Arduino PWM | "Analog input reference voltage" | "Install Support for Arduino® Hardware" | "Detect and Fix Task Overruns on Arduino® Hardware" |

External Links

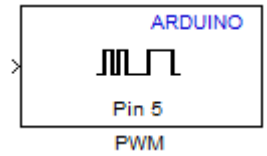
- http://arduino.cc/en/Reference/AnalogRead
- http://arduino.cc/en/Reference/AnalogReference

Purpose

Generate PWM waveform on analog output pin

Library

Target for Use with Arduino® Hardware



Description

Use pulse-width modulation (PWM) to change the duty-cycle of square-wave pulses output by a PWM pin on the Arduino® hardware. PWM enables a digital output to provide a range of different power levels, similar to that of an analog output.

The value sent to the block input determines the width of the square wave, called *duty-cycle*, that the target hardware outputs on the specified PWM pin. The range of valid outputs is `0` to `255`.

For example:

- Sending the maximum value, `255`, to the block input produces 100% duty-cycle, which results in full power on a PWM pin.
- Sending the minimum value, `0`, to the block input produces 0% duty-cycle, which results in no power on a PWM pin.
- Sending an intermediate value to the block input produces a proportional duty-cycle and power output on a PWM pin. For example, sending 192 to the block input produces 75% duty cycle and power (192/256 = 0.75).
- Sending out-of-range values, such as `500` or `-500`, to the block input has the same effect as sending the maximum or minimum input values.

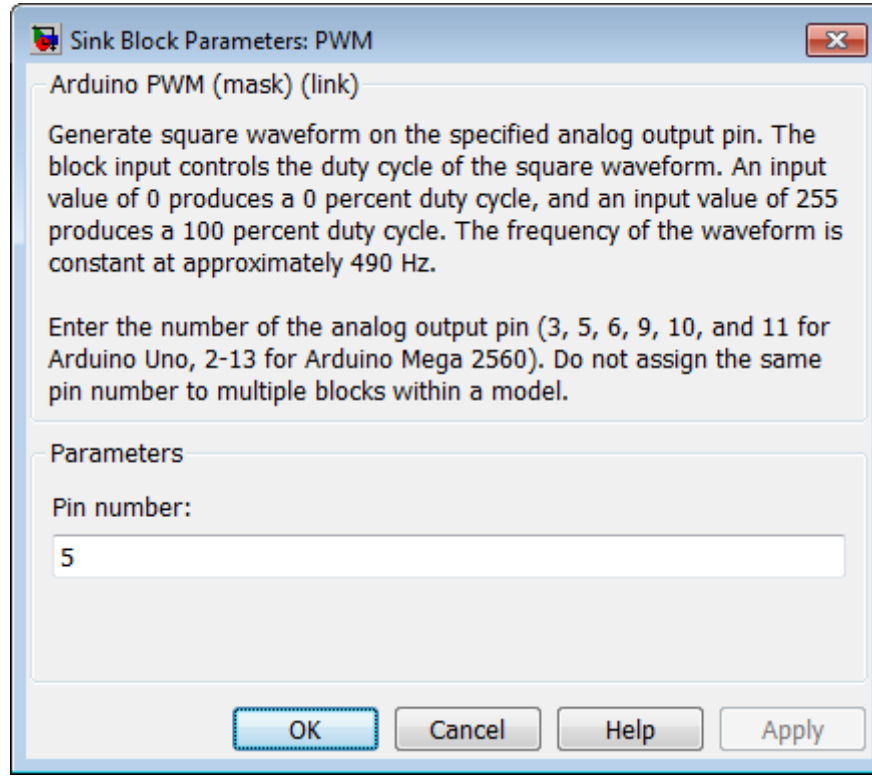
The frequency of the square wave is ~490 Hz.

The block input inherits the data type of the upstream block, and internally converts it to uint8.

Some limitations:

- With Arduino® Uno hardware, the Arduino® PWM block cannot use digital pins 9 or 10 when the model contains Servo blocks.
- With Arduino® Mega 2560 hardware, the Arduino® PWM block cannot use digital pins 11 or 12 when the model contains more than 12 Servo blocks.

Dialog



Pin number

Enter the number of the PWM pin.

Do not assign the same pin number to multiple blocks within the model.

For Arduino® Mega 2560, enter a pin number from 2 to 13.

For Arduino® Uno, enter one of the following pin numbers, 3, 5, 6, 9, 10, and 11, which are marked with a ~ symbol.

See Also

"Install Support for Arduino® Hardware" |

External Links

- http://arduino.cc/en/Reference/AnalogWrite

Arduino Serial Receive

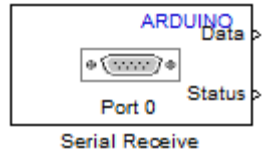
Purpose

Get one byte of data from serial port

Library

Target for Use with Arduino® Hardware

Description



Get one byte of data per sample period from the receive buffer of the specified serial port. For more information, see "Use Serial Communications with Arduino® Hardware".

The Serial Receive block has two block outputs, **Data** and **Status**.

When data is available:

- The **Data** block output emits data from the serial receive buffer.
- The **Status** block output emits `1`.

When data is not available:

- The **Data** block output emits `255`.
- The **Status** block output emits `0`.

The datatype of the **Data** block output is uint8.

The datatype of the **Status** block output is int. You can use the Status block output to determine whether a value of `255` emitted by the **Data** block output is data, or an indication that no data was received.

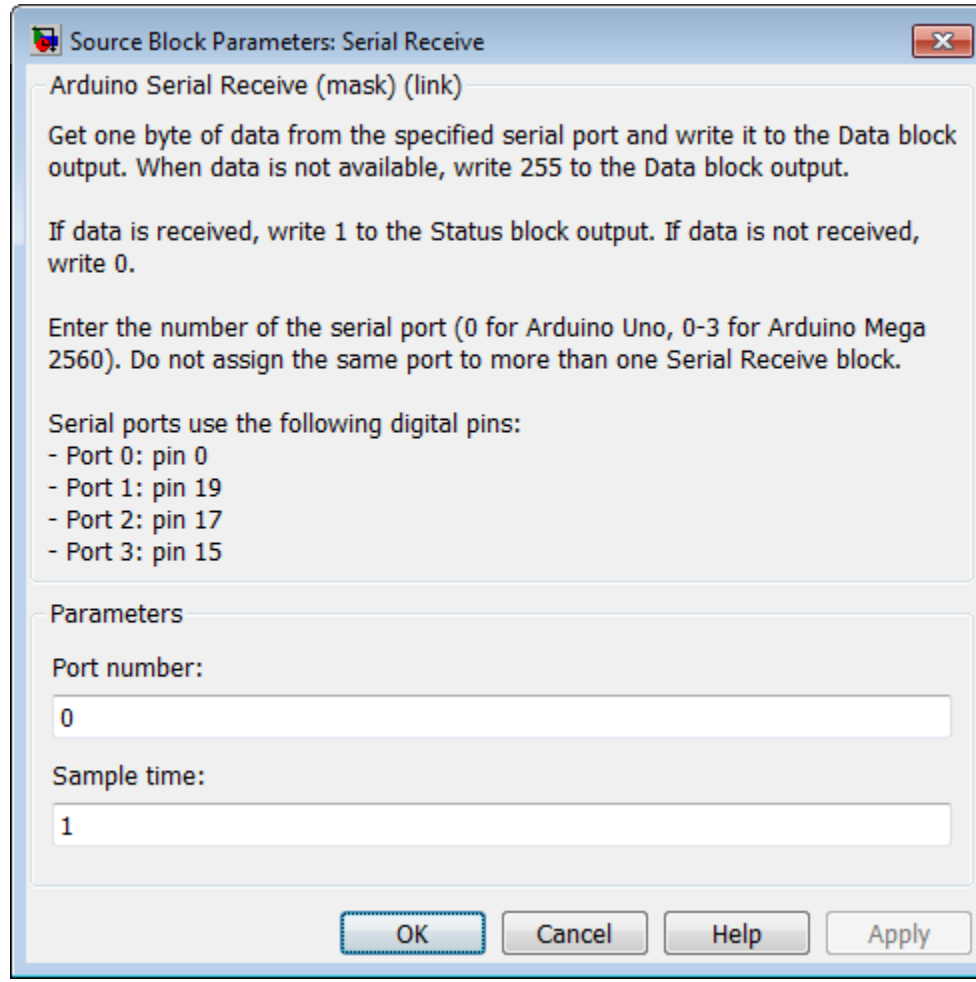
If you simulate your model without running it on the target hardware, this block outputs zeroes. See "Block Produces Zeros in Simulation"

Do not use this block in models with the Standard Servo Read, Standard Servo Write, and Continuous Servo Write blocks.

Warning

Only connect serial port pins to devices that use 5 Volt TTL logic. Do not connect these pins to an RS-232 serial interface, such as the DE-9M connector on a computer, without limiting the voltage. The RS-232 standard allows higher voltages that can damage your hardware. For details, read the documentation for your Arduino® hardware.

Dialog



Port Number

Enter the number of the serial port. For Arduino® Mega 2560, enter 0 - 3. For Arduino® Uno, enter 0.

You can assign a Serial Transmit block and a Serial Receive block to the same serial port.

Do not assign more than one Serial Receive block to the same serial port.

Do not assign the pin numbers used by the serial port to other blocks within the model.

Serial port 0 is connected to the USB port through a converter. Do not use both serial port 0 and the USB port at the same time. For example, do not use serial port 0 if you intend to use External mode, because External mode requires the USB port.

Sample time

Specify how often this block reads the serial port buffer. Enter a value greater than zero. This value defaults to a sample time of `1` second. The minimum value is `0.000001` second.

Smaller values require the processor to complete the same number of instructions in less time, which can cause task overruns.

See Also

`Arduino Serial Transmit` | "Install Support for Arduino® Hardware" | "Use Serial Communications with Arduino® Hardware" | "Tune and Monitor Models Running on Arduino® Mega 2560 Hardware" | "Detect and Fix Task Overruns on Arduino® Hardware" |

External Links

- http://arduino.cc/en/Serial/Read

Arduino Serial Transmit

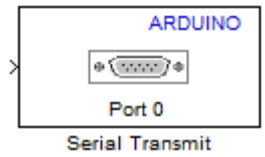
Purpose

Send buffered data to serial port

Library

Target for Use with Arduino® Hardware

Description



Send buffered data to the specified serial port. For more information, see "Use Serial Communications with Arduino® Hardware".

The Arduino® Uno hardware has one serial port device, serial port 0, connected to the digital pins marked `TX 1` and `RX 0`. If you set the **Port number** parameter to `0`, this block transmits over the digital pin marked `TX 1`.

The block input accepts vector or scalar uint8 data. To convert a data source to uint8, use a Data Type Conversion block.

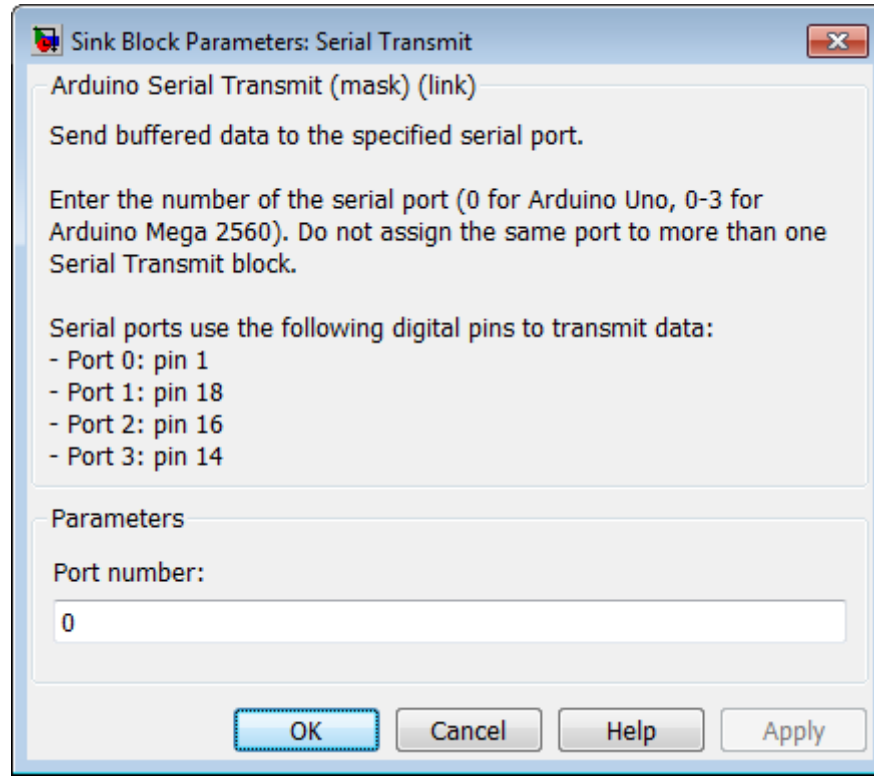
If you simulate your model without running it on the target hardware, this block does nothing. See "Block Produces Zeros in Simulation".

Do not use this block in models with the Standard Servo Read, Standard Servo Write, and Continuous Servo Write blocks.

Warning

Only connect serial port pins to devices that use 5 Volt TTL logic. Do not connect these pins to an RS-232 serial interface, such as the DE-9M connector on a computer, without limiting the voltage. The RS-232 standard allows higher voltages that can damage your hardware. For details, read the documentation for your Arduino® hardware.

Dialog



Port Number

Enter the number of the serial port. For Arduino® Mega 2560, enter 0 - 3. For Arduino® Uno, enter 0.

You can assign a Serial Transmit block and a Serial Receive block to the same serial port.

Do not assign multiple Serial Transmit blocks to the same serial port.

Do not assign the pin numbers used by the serial port to other blocks within the model.

Serial port 0 is connected to the USB port through a converter. Do not use both serial port 0 and the USB port at the same time. For example, do not use serial port 0 if you intend to use External mode, because External mode requires the USB port.

See Also

`Arduino Serial Receive` | "Install Support for Arduino® Hardware" | "Use Serial Communications with Arduino® Hardware" | "Tune and Monitor Models Running on Arduino® Mega 2560 Hardware" |

External Links

- http://arduino.cc/en/Serial/Write

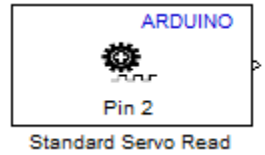
Purpose

Get position of standard servo motor shaft in degrees

Library

Target for Use with Arduino® Hardware

Description



Measure the angle of a standard servo motor shaft in degrees, from 0 to 180.

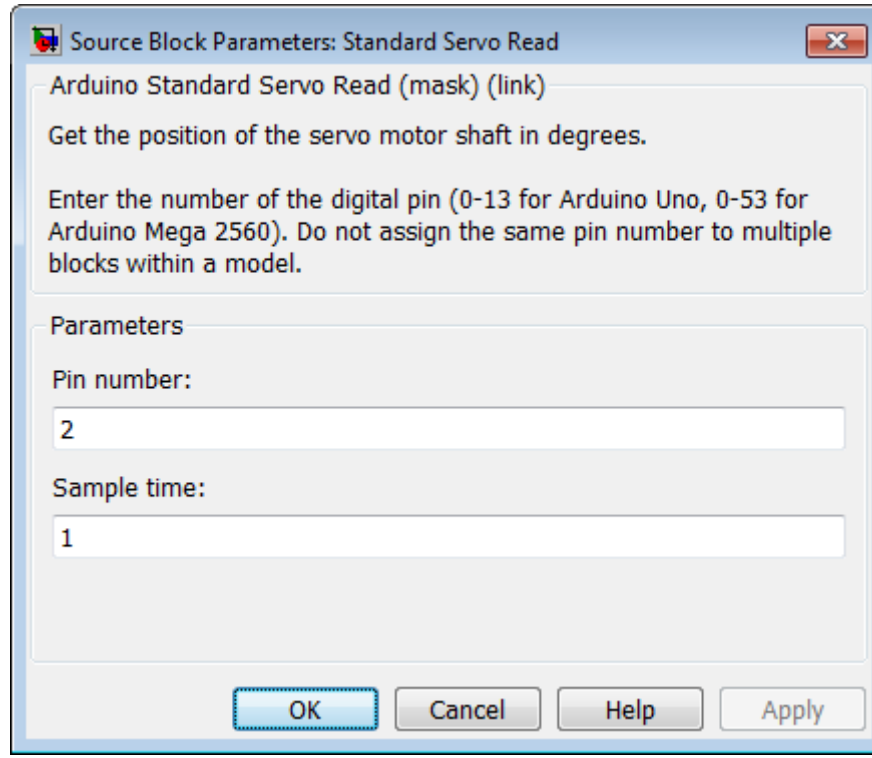
The data type of the block output is uint8.

If you simulate your model without running it on the target hardware, this block outputs zeroes. See “Block Produces Zeros in Simulation”.

Some limitations:

- Do not use Servo blocks with External mode or with models that contain Serial Transmit or Serial Receive blocks.
- The maximum number of Servo blocks per model is 12 for Arduino® Uno hardware, and 48 for Arduino® Mega 2560 hardware.
- With Arduino® Uno hardware, the Arduino® PWM block cannot use digital pins 9 or 10 when the model contains Servo blocks.
- With Arduino® Mega 2560 hardware, the Arduino® PWM block cannot use digital pins 11 or 12 when the model contains more than 12 Servo blocks.

Dialog



Pin number

Enter the number of the digital pin.

Do not assign the same pin number to multiple blocks within the model.

For Arduino® Mega 2560, enter a pin number from `0` to `53`.

For Arduino® Uno, enter a pin number from `0` to `13`.

Sample time

Specify how often this block reads the pin value. Enter a value greater than zero. This value defaults to a sample time of `1` seconds. The minimum value is `0.000001` second.

Smaller values require the processor to complete the same number of instructions in less time, which can cause task overruns.

See Also

`Arduino Standard Servo Write` | `Arduino Continuous Servo Write` | "Install Support for Arduino® Hardware" | "Detect and Fix Task Overruns on Arduino® Hardware" |

External Links

- http://arduino.cc/en/Reference/ServoRead

Arduino Standard Servo Write

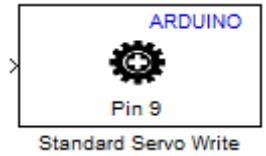
Purpose

Set shaft position of standard servo motor

Library

Target for Use with Arduino® Hardware

Description



Set the shaft position of a standard servo motor, from 0 to 180 degrees.

To rotate the servo shaft, send values from 0 to 180 to the block input.

Sending out-of-range values, such as `-5` or `200`, to the block input has the same effect as sending the maximum or minimum input values.

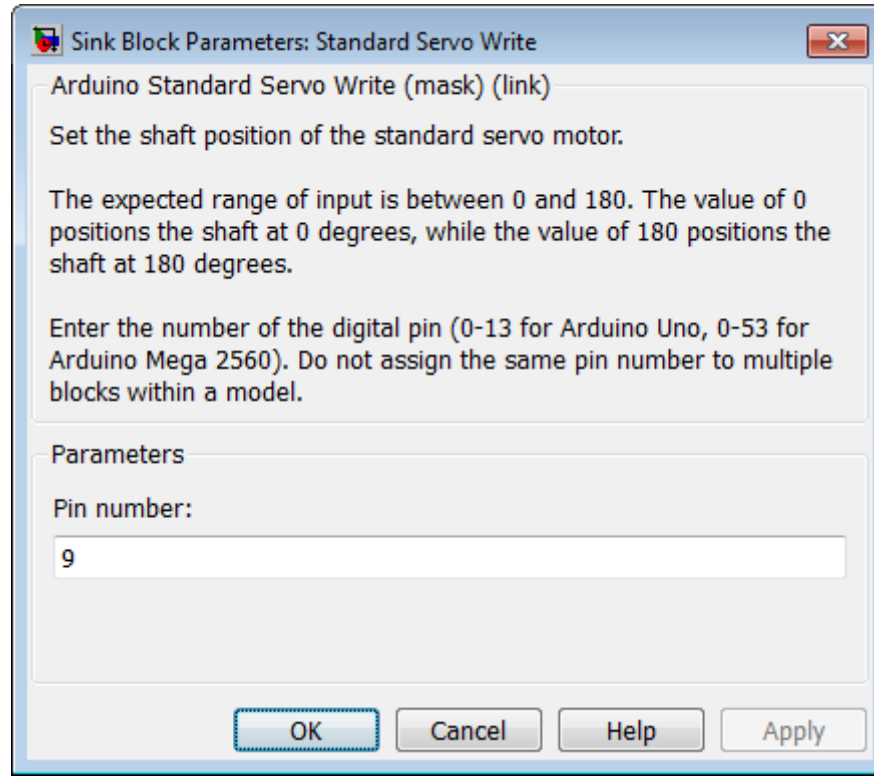
The block input inherits the data type of the upstream block, and internally converts it to uint8.

If you simulate your model without running it on the target hardware, this block does nothing. See “Block Produces Zeros in Simulation”.

Some limitations:

- Do not use Servo blocks with External mode or with models that contain Serial Transmit or Serial Receive blocks.
- The maximum number of Servo blocks per model is 12 for Arduino® Uno hardware, and 48 for Arduino® Mega 2560 hardware.
- With Arduino® Uno hardware, the Arduino® PWM block cannot use digital pins 9 or 10 when the model contains Servo blocks.
- With Arduino® Mega 2560 hardware, the Arduino® PWM block cannot use digital pins 11 or 12 when the model contains more than 12 Servo blocks.

Dialog



Pin number

Enter the number of the digital output pin.

Do not assign the same pin number to multiple blocks within the model.

For Arduino® Mega 2560, enter a pin number from 0 to 53.

For Arduino® Uno, enter a pin number from 0 to 13.

Arduino Standard Servo Write

See Also

`Arduino Standard Servo Read` | `Arduino Continuous Servo Write` | "Install Support for Arduino® Hardware" |

External Links

- http://arduino.cc/en/Reference/ServoWrite

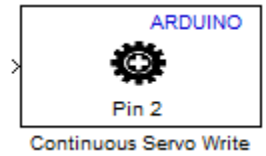
Purpose

Set shaft speed of continuous rotation servo motor

Library

Target for Use with Arduino® Hardware

Description



Set the direction and speed of a continuous rotation servo motor:

- Sending -90 to the block input produces the maximum rate of rotation in one direction.
- Sending `90` to the block input produces the maximum rate of rotation in the opposite direction.
- Sending 0 to the block input stops the servo motor.
- Sending out-of-range values, such as `-95` or `200`, to the block input has the same effect as sending the maximum or minimum input values.

The characteristics of some motors cause them to continue rotating when the block input value is 0. In that case, you can experiment to find an offset value that stops the motor.

With Arduino® Mega 2560 hardware, you can use External mode to determine the offset while your model is running on the hardware. See "Tune and Monitor Models Running on Arduino® Mega 2560 Hardware".

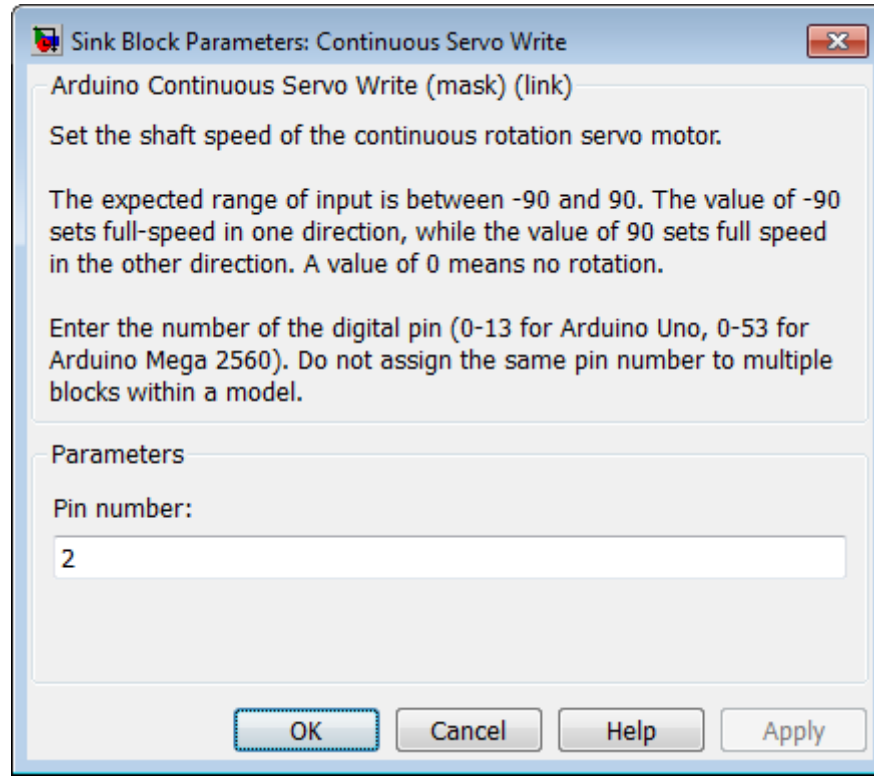
The block input inherits the data type of the upstream block, and internally converts it to uint8.

If you simulate your model without running it on the target hardware, this block does nothing. See "Block Produces Zeros in Simulation".

Some limitations:

- Do not use Servo blocks with External mode or with models that contain Serial Transmit or Serial Receive blocks.
- The maximum number of Servo blocks per model is 12 for Arduino® Uno hardware, and 48 for Arduino® Mega 2560 hardware.
- With Arduino® Uno hardware, the Arduino® PWM block cannot use digital pins 9 or 10 when the model contains Servo blocks.
- With Arduino® Mega 2560 hardware, the Arduino® PWM block cannot use digital pins 11 or 12 when the model contains more than 12 Servo blocks.

Dialog



Pin number

Enter the number of the digital output pin.

Do not assign the same pin number to multiple blocks within the model.

For Arduino® Mega 2560, enter a pin number from 0 to 53.

For Arduino® Uno, enter a pin number from 0 to 13.

Arduino Continuous Servo Write

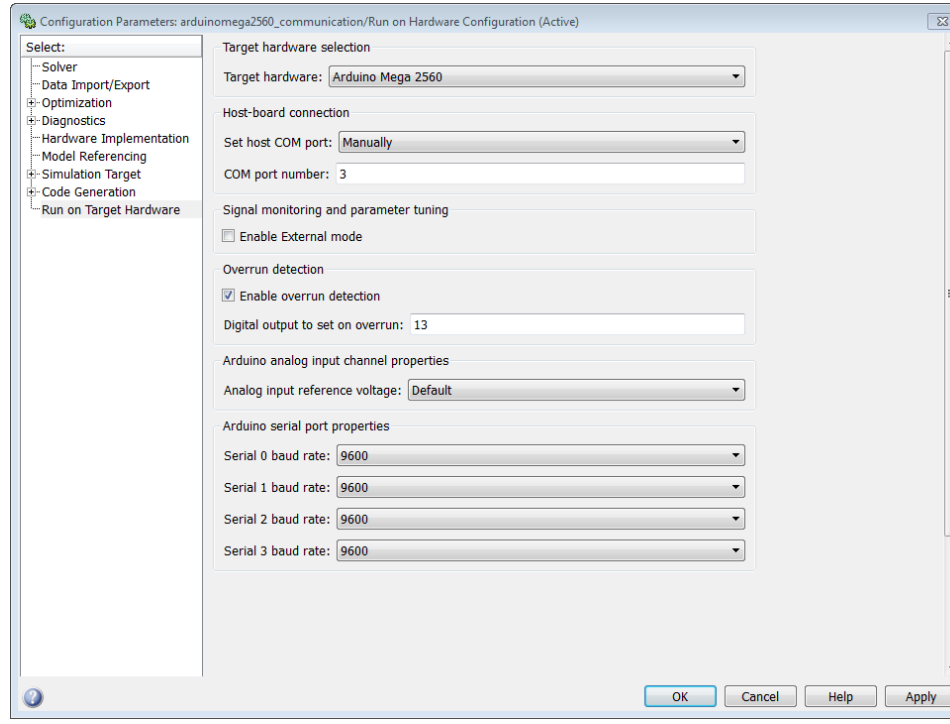
See Also

`Arduino Standard Servo Read` | `Arduino Standard Servo Write` | “Install Support for Arduino® Hardware” |

External Links

- http://arduino.cc/en/Reference/ServoWrite

Run on Target Hardware Pane



In this section...

In this section...

Run on Target Hardware Pane Overview

Specify the options for creating and running applications on target hardware.

Configuration

1. Choose the **Target hardware** in the Run on Target Hardware pane.
2. Set the parameters displayed for the selected device type.
3. Apply the changes.

To get help on an option

1. Right-click the option's text label.
2. Select **What's This** from the popup menu.

Target hardware

Select the type of hardware upon which to run your model.

Changing this parameter updates the Configuration Parameters dialog so it only displays parameters that are relevant to your target hardware.

If your target hardware is supported, but not available in the **Target hardware** parameter options, use Target Installer to install support for your target hardware. To use Target Installer, close the Configuration Parameters dialog and enter `targetinstaller` in the MATLAB® Command Window. After installing support for your target hardware, reopen the Configuration Parameters dialog and select your target hardware.

Settings

Default: `None`

`None`
: This setting means your model has not been configured to run on target hardware. Choose your target hardware from the list of options.

`Arduino Mega 2560`
: This setting displays the following configuration parameters for Arduino® Mega 2560 hardware:

  - "Enable External mode" on page 1-7
  - "Enable overrun detection" on page 1-8
  - "Digital output to set on overrun" on page 1-9
  - "Set host COM port" on page 1-10
  - "COM port number" on page 1-10
  - "Analog input reference voltage" on page 1-11
  - "Serial 0 baud rate, Serial 1 baud rate, Serial 2 baud rate, Serial 3 baud rate" on page 1-12

`Arduino Uno`
: This setting displays the following configuration parameters for Arduino Uno hardware:

  - "Enable overrun detection" on page 1-8

- "Digital output to set on overrun" on page 1-9
- "Set host COM port" on page 1-10
- "COM port number" on page 1-10
- "Analog input reference voltage" on page 1-11

See Also

- "Install Support for Arduino Hardware"

Enable External mode

Enable External mode to tune and monitor a model while it runs on your target hardware.

With External mode, changing a parameter value in the model on the host changes the corresponding value in the model running on the target hardware. Similarly, scopes in the model display data from the model running on target hardware.

Enabling External mode adds a lightweight server to the model running on the target hardware. This server increases the processing burden upon the target hardware, which can result in an overrun condition. If you enable the **Enable overrun detection** check box, and the software reports an overrun, consider disabling External mode.

Settings

Default: Disabled

Disabled
: The model application does not support External mode.

Enabled
: The model application supports External mode.

See Also

- “Enable overrun detection” on page 1-8
- “Set host COM port” on page 1-10
- “Tune and Monitor Models Running on Arduino Mega 2560 Hardware”
- “Use Serial Communications with Arduino Hardware”

Enable overrun detection

Detect when a task overruns occurs in a Simulink® model running on the target hardware. Indicate when an overrun has occurred.

A task overrun occurs if the target hardware is still performing one instance of a task when the next instance of that task is scheduled to begin.

The "Detect and Fix Task Overruns" topics listed in the following "See Also" subtopic describe how your target hardware indicates that an overrun has occurred.

You can fix overruns by decreasing the frequency with which tasks are scheduled to run, and by reducing the number or complexity of the tasks defined by your model.

If those solutions do not fix the task overrun condition, and you are using External mode, consider disabling External mode by clearing the **Enable External mode** checkbox.

Settings

Default: Disabled

Disabled
: Do not detect overruns.

Enabled
: Detect overruns and generate an error message when an overrun occurs.

See Also

- "Digital output to set on overrun" on page 1-9
- "Enable External mode" on page 1-7
- "Detect and Fix Task Overruns on Arduino Hardware"

Digital output to set on overrun

This parameter appears when the **Enable overrun detection** check box is selected.

Select the digital output pin the Arduino hardware uses to signal a task overrun.

Do not use a pin that is assigned to another block within the model.

Settings

Default: 13

See Also

- “Enable overrun detection” on page 1-8
- “Detect and Fix Task Overruns on Arduino Hardware”

Set host COM port

This parameter only appears when the **Target hardware** parameter is set to `Arduino Mega 2560` or `Arduino Uno`.

Automatically detect or manually set the COM port your host computer uses to communicate with the target hardware.

Warning **Do not connect Arduino Uno and Arduino Mega 2560 to a RS-232 serial interface, commonly found on computers and equipment. RS-232 interfaces can use voltages greater than 5 Volts, which can damage your Arduino hardware.**

Settings

Default: `Automatically`

`Automatically`
: Let the software determine which COM Port your host computer uses.

`Manually`
: Select this option to display the **COM port number** parameter.

See Also

- "COM port number" on page 1-10

COM port number

This parameter only appears when the **Set host COM port** parameter is set to `Manually`.

Manually set the number of the COM Port the host computer uses to communicate with the target hardware, and then enter it here.

Warning **Do not connect Arduino Uno and Arduino Mega 2560 to a RS-232 serial interface, commonly found on computers and equipment. RS-232 interfaces can use voltages greater than 5 Volts, which can damage your Arduino hardware.**

Settings

Default: `0`

See Also

- "Set host COM port" on page 1-10

Analog input reference voltage

This parameter only appears when the **Target hardware** parameter is set to `Arduino Mega 2560` or `Arduino Uno`.

Set the reference voltage used to measure inputs to the ANALOG IN pins.

Warning **Only connect an external power source to AREF while this parameter is set to `External`. Connecting an external power source to AREF while this parameter is set to any other option exposes the internal voltage references to the external voltage. This voltage difference can damage your hardware.**

Do not connect Arduino Uno and Arduino Mega 2560 to voltages greater than 5 Volts. These greater voltages can damage your Arduino hardware.

Settings

Default: `Default`

`Default`
: Use the default operating voltage of the board. For Arduino Uno and Arduino Mega 2560 the operating voltage is 5 Volts.

`Internal (1.1 V)`
: Valid for Arduino Mega 2560 only: Use the internal 1.1 Volt reference.

`Internal (2.56 V)`
: Valid for Arduino Mega 2560 only: Use the internal 2.56 Volt reference.

`External`
: On the Arduino Uno and Arduino Mega 2560, use an external 0-5 volt power supply connected to the AREF pin. This voltage should match

the voltage of the power supply connected to the Arduino hardware. If your application requires low-noise measurements, use this option with a filtered power supply.

See Also

- Arduino Analog Input

Serial 0 baud rate, Serial 1 baud rate, Serial 2 baud rate, Serial 3 baud rate

Arduino Uno hardware has one serial port, Serial 0. Arduino Mega 2560 hardware has four serial ports, Serial 0 through Serial 3.

Set the baud rate of the serial port on the Arduino hardware.

Settings

Default: 9600

300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 76800, 115200, 128000, 500000, 1000000

See Also

- Arduino Serial Receive
- Arduino Serial Transmit