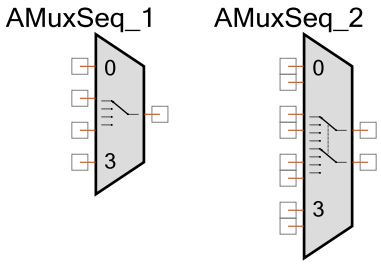
# Analog Multiplexer Sequencer (AMuxSeq)

1.80




## Features

- Single or differential connections
- Adjustable between 1 and 256 connections
- Software controlled
- Connections may be pins or internal sources
- No simultaneous connections
- Bidirectional (passive)

## General Description

The analog multiplexer sequencer (AMuxSeq) Component is used to connect one analog signal at a time to a different common analog signal, by breaking and making connections in hookup-order sequence. The AMuxSeq is primarily used for time division multiplexing.

### When to Use an AMuxSeq

Use the AMuxSeq Component any time you need to multiplex multiple analog signals into a single source or destination. Because the AMuxSeq Component is passive, it can be used to multiplex input or output signals.

The AMuxSeq has a simpler and faster API than the AMux. Use the AMuxSeq instead of the AMux when multiple simultaneous connections are not required and the signals will always be accessed in the same order.

## Input/Output Connections

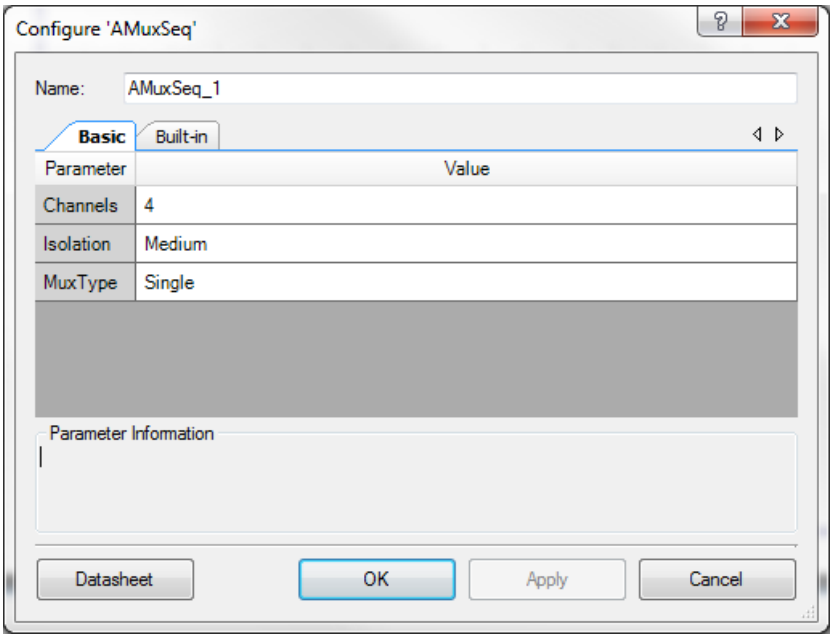
This section describes the various input and output connections for the AMuxSeq. An asterisk (*) in the list of I/Os indicates that the I/O may be hidden on the symbol under the conditions listed in the description of that I/O.

| I/O | Description |
|---|---|
| 0 - 255 – Analog | The AMuxSeq is capable of having between 1 and 256 analog switchable connections. The paired connections are present when the **MuxType** parameter is set to **Differential**. |

| I/O | Description |
|---|---|
| common – Analog | The “common” signal is the common connection; it is not labeled. The switchable connections signal selected with the AMuxSeq_Next() function is connected to this terminal. The paired signals are present when the **MuxType** parameter is set to **Differential**. |

# Component Parameters

Drag an AMuxSeq Component onto your design and double-click it to open the **Configure** dialog.




The AMuxSeq provides the following parameters.

## Channels

This parameter selects the number of inputs or paired inputs depending on the **MuxType**. Any value between 1 and 256 is valid.

## MuxType

This parameter selects between a single input per connection (**Single**) and a dual input **Differential** input mux. **Single** is used when the input signals are all referenced to the same signal, such as $V_{SSA}$. In cases where two or more signals may have a different signal reference, select the **Differential** option. The differential mode is most often used with an ADC that provides a differential input.

## Isolation

This parameter is used to select one of the following isolation modes:

- **Minimum –** Use single outer switching. This guarantees the fastest switching time.
- **Medium (default) –** Attempt to use double switching, with outer switch and unique inner switches only. If no unique inner switches are available, single outer switching will be used. Double switching will increase isolation but also increase switching time.
- **Maximum –** Use double switching, with outer switch and potentially shared inner switches. Inner switches do not have to be unique. A reference count allows sharing an inner switch. When non unique inner switches are used, switching time will be further impacted.

The following diagrams show the three possible switching implementations for an Amux with no arm connected, bottom arm connected, and both arms connected:

**Figure 1. Single Switching, no inner switches**

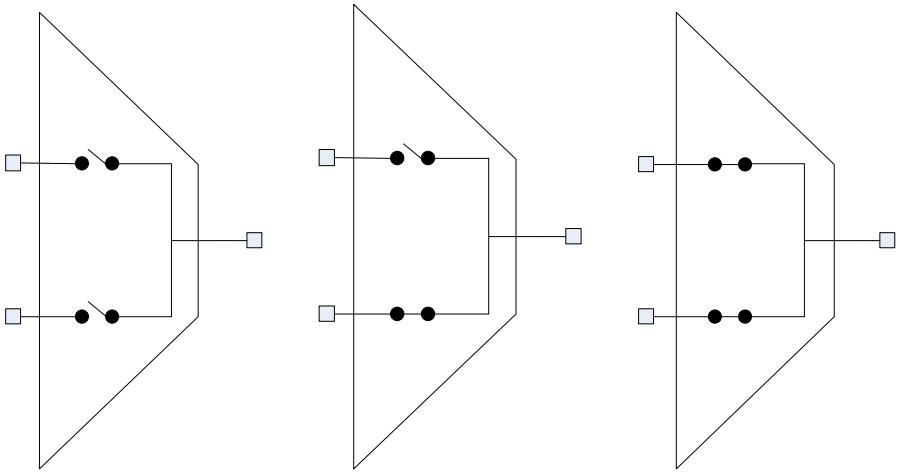


**Figure 2. Double Switching, unique inner switches**

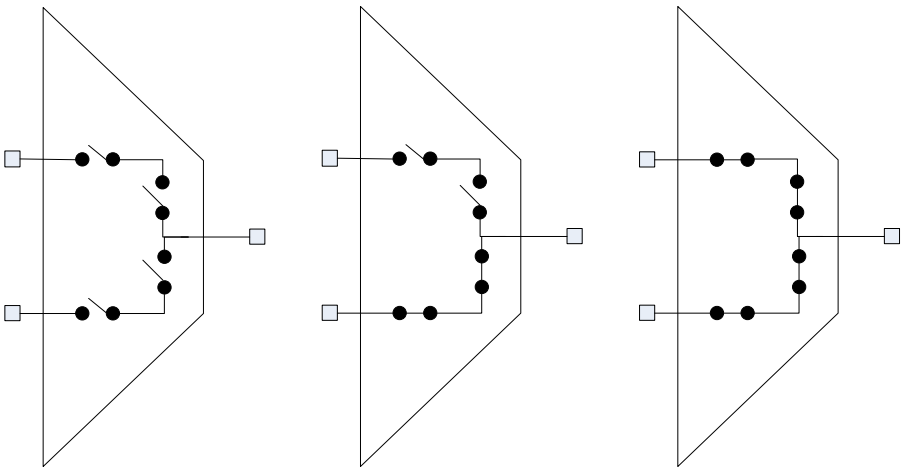
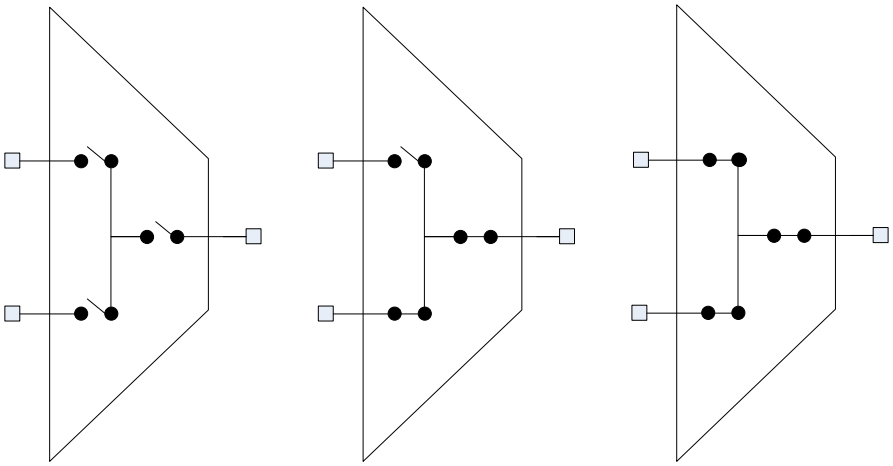


**Figure 3. Dynamic Switching, shared inner switch**



# Application Programming Interface

Application Programming Interface (API) routines allow you to configure the Component using software. The following table lists and describes the interface to each function. The subsequent sections cover each function in more detail.

By default, PSoC Creator assigns the instance name “AMuxSeq_1” to the first instance of a Component in a given design. You can rename the instance to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is “AMuxSeq.”

## Functions

| Function | Description |
|---|---|
| AMuxSeq_Init() | Disconnects all channels |
| AMuxSeq_Start() | Disconnects all channels |
| AMuxSeq_Stop() | Disconnects all channels |
| AMuxSeq_Next() | Disconnects the previous channel and connects the next one in the sequence. |
| AMuxSeq_DisconnectAll() | Disconnects all channels |
| AMuxSeq_GetChannel() | The currently connected channel is returned. If no channel is connected returns –1. |

### void AMuxSeq_Init(void)

**Description:** Disconnects all channels. The next time AMuxSeq_Next() is called, the first channel is selected.

**Side Effects:** All registers will be reset to their initial values.

### void AMuxSeq_Start(void)

**Description:** Disconnects all channels. The next time AMuxSeq_Next() is called, the first channel is selected.

### void AMuxSeq_Stop(void)

**Description:** Disconnects all channels. The next time AMuxSeq_Next() is called, the first channel is selected.

### void AMuxSeq_Next(void)

**Description:** Disconnects the previous channel and connects the next one in the sequence. When AMuxSeq_Next() is called for the first time or after AMuxSeq_Init(), AMuxSeq_Start(), AMuxSeq_Enable(), AMuxSeq_Stop(), or AMuxSeq_DisconnectAll(), it connects channel 0. The Component starts over from the first channel when the AMuxSeq_Next() API is called and the channel is already at the last channel (that is, it is cyclic).

### void AMuxSeq_DisconnectAll(void)

**Description:** This function disconnects all channels. The next time AMuxSeq_Next() is called, the first channel will be selected.

### int8 AMuxSeq_GetChannel(void)

**Description:** The currently connected channel is returned. If no channel is connected, returns –1.

**Return Value:** The current channel or –1.

## Sample Firmware Source Code

PSoC Creator provides many codes examples that include schematics and example code in the Find Code Example dialog. For Component-specific examples, open the dialog from the Component Catalog or an instance of the Component in a schematic. For general examples, open the dialog from the Start Page or **File** menu. As needed, use the **Filter Options** in the dialog to narrow the list of projects available to select.

Refer to the “Find Code Example” topic in the PSoC Creator Help for more information.

## API Memory Usage

The Component memory usage varies significantly, depending on the compiler, device, number of APIs used and Component configuration. The following table provides the memory usage for all APIs available in the given Component configuration.

The measurements have been done with the associated compiler configured in Release mode with optimization set for Size. For a specific design, the map file generated by the compiler can be analyzed to determine the memory usage.

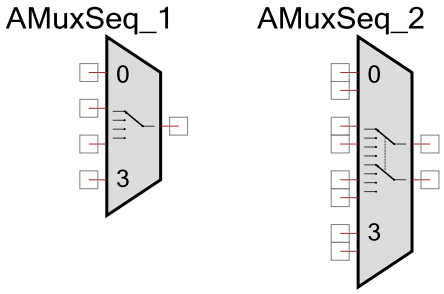
| Configuration | PSoC 3 (Keil_PK51) | | PSoC 4 (GCC) | | PSoC 5LP (GCC) | | PSoC 6 (GCC) | |
|---|---|---|---|---|---|---|---|---|
| | Flash Bytes | SRAM Bytes | Flash Bytes | SRAM Bytes | Flash Bytes | SRAM Bytes | Flash Bytes | SRAM Bytes |
| Single | 24 | 1 | 48 | 1 | 40 | 1 | 40 | 1 |
| Differential | 38 | 1 | 72 | 1 | 68 | 1 | 72 | 1 |

# Functional Description

The AMuxSeq is controlled by firmware, not by hardware. Only one signal at a time can be connected to the common signal.

The following shows the flow for an AMuxSeq configured as single and differential.

**Single** **Differential**




## Performance

The Sequential Analog Mux is controlled by software, so the switching performance depends on the execution time of the APIs provided. The performance varies depending on the exact configuration of the mux in the design, but is not sensitive to the number of inputs because a single input is disconnected and another connected with each call. Table 1 is intended to provide guidance on the switching performance.

All performance measurements were made with a CPU frequency of 48 MHz. The performance scales close to linearly with CPU frequency. The compiler optimization was configured for the highest optimization offered for the compilers bundled with PSoC Creator.

- For PSoC 3, the compiler setting is Keil optimized for Size or Speed at optimization level 5.
- For PSoC 4 and PSoC 5LP, the compiler setting is GNU optimized for Size or Speed.
- For PSoC 6, API performance is comparable to PSoC 4. Use those numbers as guidance for evaluating the switching performance.

**Table 1. Performance**

| Function | Optimization | PSoC 3 (µs) | PSoC 4 (µs) | PSoC 5LP (µs) |
|---|---|---|---|---|
| Next | Size | 8.8 | 1.8 | 1.4 |
| | Speed | 2.4 | 1.7 | 1.4 |

### Parasitics

The AmuxSeq Component uses many routing sources on the chip. Due to the parasitics of these routing resources, the AmuxSeq Component may appear to have current leak through an input to the output when the input is initially selected. Therefore, use strong voltage drive on AMuxSeq Component inputs to achieve more accurate voltage reading on the output. If this is not possible, then reduce the switching speed to account for the increased parasitic effects.

## Resources

The AMuxSeq uses individual switches that connect blocks and pins to analog buses.

## MISRA Compliance

This section describes the MISRA-C:2004 compliance and deviations for the Component. There are two types of deviations defined:

- project deviations – deviations that are applicable for all PSoC Creator Components
- specific deviations – deviations that are applicable only for this Component

This section provides information on Component-specific deviations. Non PSoC 6 project deviations are described in the MISRA Compliance section of the *System Reference Guide* along with information on the MISRA compliance verification environment. For PSoC 6, refer to PSoC Creator Help > Building a PSoC Creator Project > Generated Files (PSoC 6) for information on MISRA compliance and deviations for files generated by PSoC Creator.

The AMuxSeq Component does not have any specific deviations, and it does not have any embedded Components.

## DC and AC Electrical Characteristics

The AMuxSeq operates at all valid supply voltages.

## Component Changes

This section lists the major changes in the Component from the previous version.

| Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| 1.80.f | Added note on cyclic nature of AMuxSeq_Next(). | |
| | Added note regarding possible parasitic effects | |
| 1.80.e | Minor datasheet edit. | |
| 1.80.d | Updated MISRA section. | |

| Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| 1.80.c | Expanded the maximum number of channels to 256. | Enable valid designs in select devices |
| | Corrected API memory usage numbers for GCC | |
| 1.80.b | The Component was made visible for PSoC 6. | |
| 1.80.a | Minor datasheet edit. | |
| 1.80 | Updated API code for MISRA-C:2004 compliance. | |
| 1.70a | Added PSoC 4 resource usage and performance information. | |
| 1.70 | Relaxed AMux input connections' range. The input range is 1-64 for "Single" AMux, and 1-32 for "Differential" AMux. | To resolve demands from customers. |
| | Added MISRA Compliance section. This Component was not verified for MISRA-C:2004 coding guidelines compliance. | |
| 1.60 | Added Isolation parameter. Updated the screen capture. | Allows you to select an isolation mode to control switching time. |
| 1.50.c | Added Performance section to datasheet | |
| 1.50.b | Minor datasheet edits and updates | |
| 1.50.a | Minor datasheet edits and updates | |
| 1.50 | Added AMuxSeq_Init() function. | To comply with corporate standard and provide an API to initialize or restore the Component without starting it. |
| 1.20.a | Added information to the Component that advertizes its compatibility with silicon revisions. | The tool reports an error or warning if the Component is used on incompatible silicon. If this happens, update to a revision that supports your target device. |
| 1.20 | Updated the Symbol picture. | Updated to comply with corporate standard and indicate sequencing. |
| | Added the AMuxSeq_GetChanne()l API function. | To get currently connected channel. |
| | Added missing 'void' for functions with no arguments.<br>Changed type of AMux channel variable from unsigned to signed integer because -1 is used to indicate that no channel is selected. | These changes addressed warnings about deprecated declaration that appeared during compilation with MDK and RVDS compilers. |

© Cypress Semiconductor Corporation, 2013-2019. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC (“Cypress”). This document, including any software or firmware included or referenced in this document (“Software”), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress’s patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical Components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage (“Unintended Uses”). A critical Component is any Component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.

