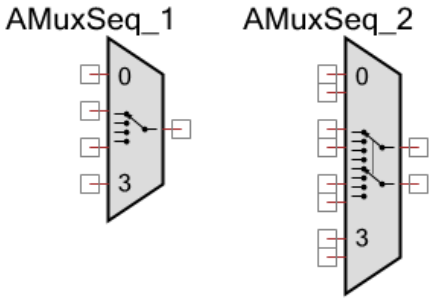
Analog Multiplexer Sequencer (AMuxSeq)

1.70

Features



- Single or differential connections
- Adjustable between 1 and 64 connections for single AMux, 1 and 32 connections for Differential AMux
- 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.

0-63 – Analog

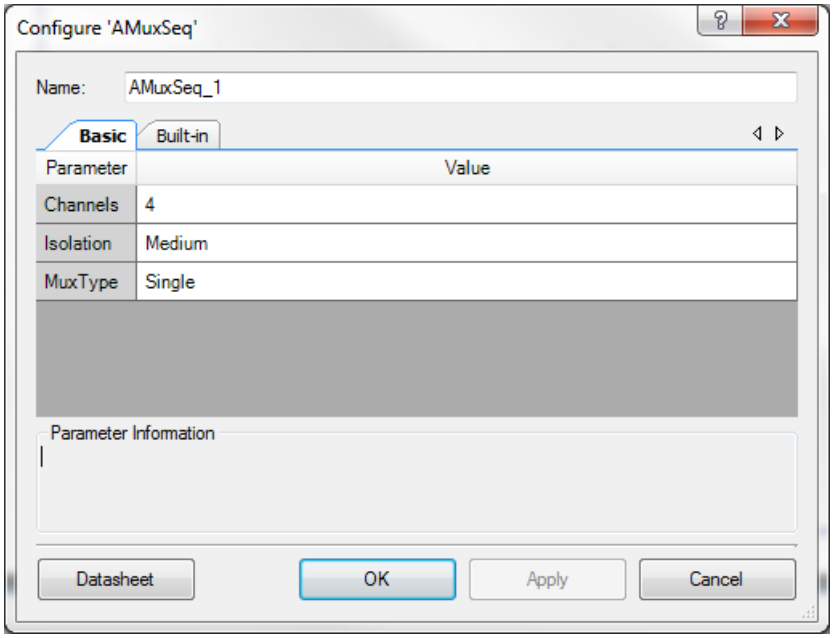
The AMuxSeq is capable of having between 1 and 64 analog switchable connections for Single AMux, between 1 and 32 analog switchable connections for Differential Amux. The paired connections are present when the **MuxType** parameter is set to **Differential**.

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 64 is valid for Single AMux, any value between 1 and 32 is valid for Differential AMux.

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

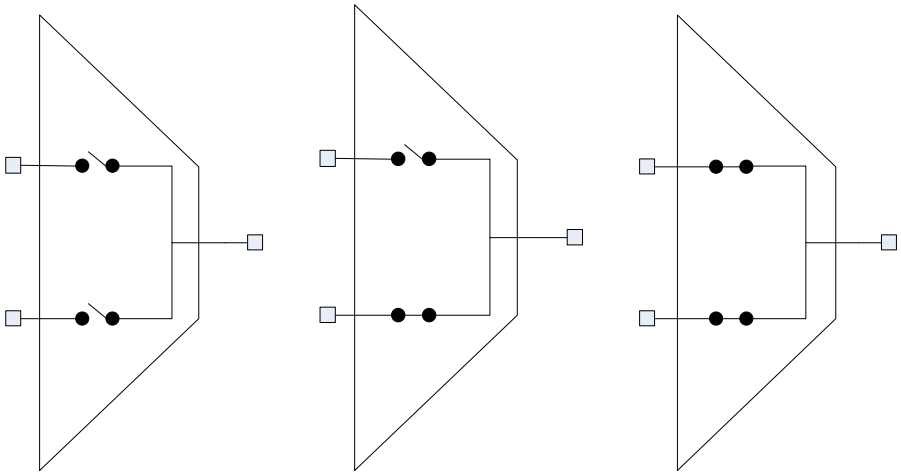


Figure 2. Double Switching, unique inner switches

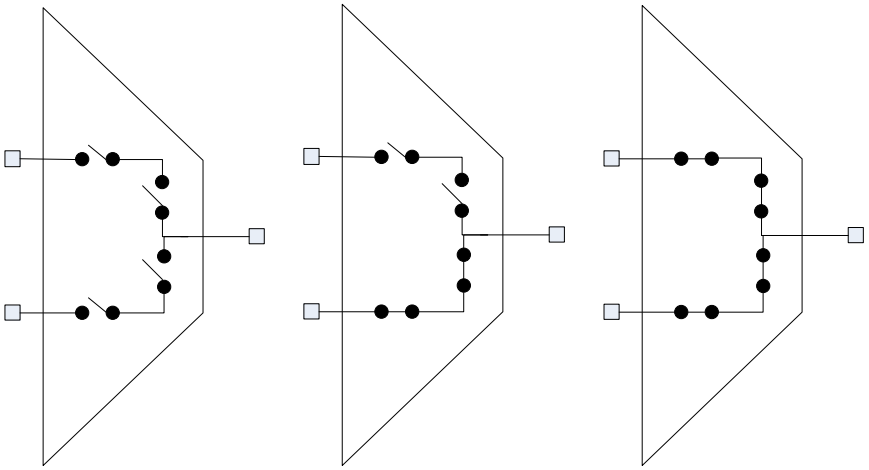
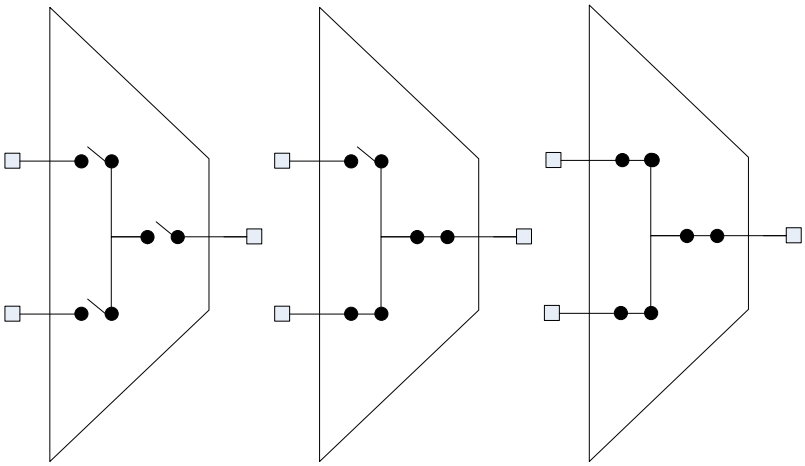


Figure 3. Dynamic Switching, shared inner switch



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.”

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.

Parameters: None

Return Value: None

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.

Parameters: None

Return Value: None

Side Effects: None

void AMuxSeq_Stop(void)

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

Parameters: None

Return Value: None

Side Effects: None

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.

Parameters: None

Return Value: None

Side Effects: None

void AMuxSeq_DisconnectAll(void)

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

Parameters: None

Return Value: None

Side Effects: None

int8 AMuxSeq_GetChannel(void)

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

Parameters: None

Return Value: The current channel or –1.

Side Effects: None

Sample Firmware Source Code

PSoC Creator provides many example projects that include schematics and example code in the Find Example Project 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.

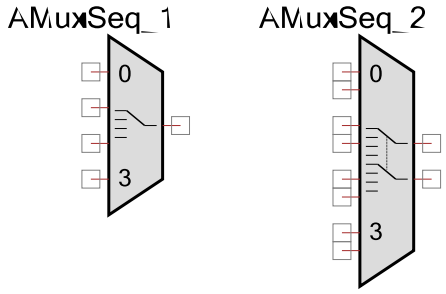
Refer to the “Find Example Project” topic in the PSoC Creator Help for more information.

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.

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

Resources

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

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)	
	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes
Single	24	1	48	1	40	1
Differential	38	1	72	1	68	1

MISRA Compliance

The component source code was not verified for MISRA-C:2004 coding guidelines compliance.

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
1.70a	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. 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. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC® is a registered trademark, and PSoC® Creator™ and Programmable System-on-Chip™ are trademarks of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.