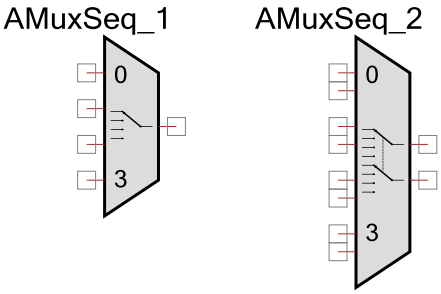
Analog Multiplexer Sequencer (AMuxSeq)

1.50

Features



- Single or differential connections
- Adjustable between 2 and 32 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.

0-31 – Analog

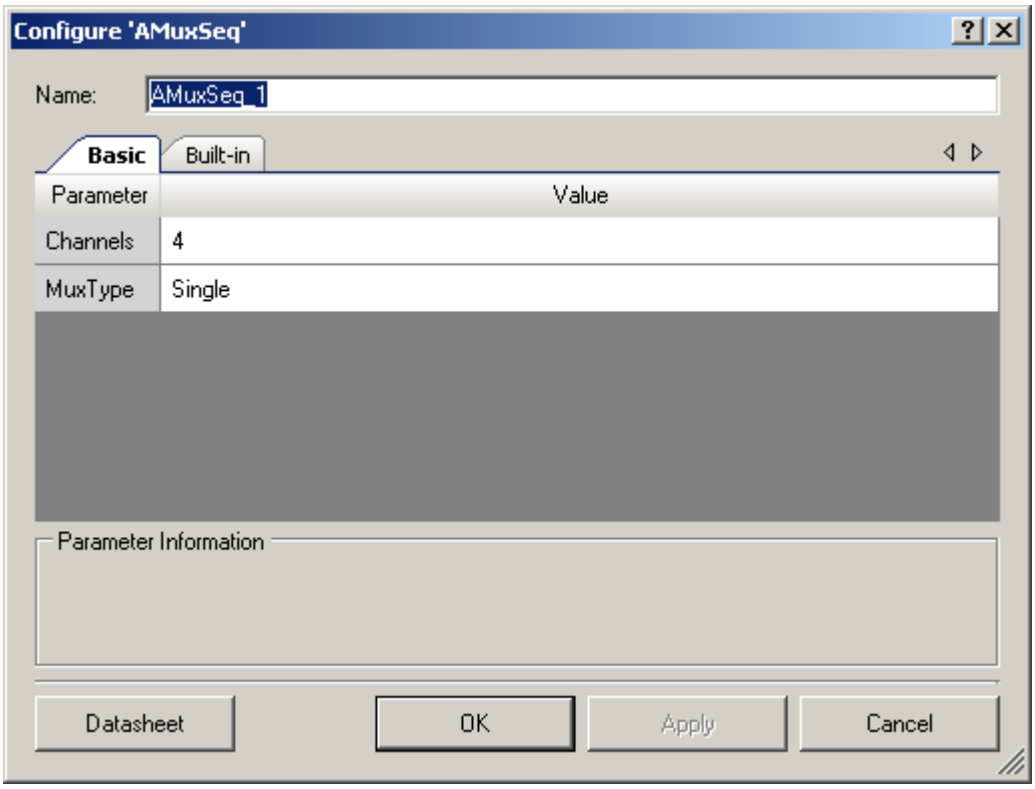
The AMuxSeq is capable of having between 2 and 32 analog switchable connections. The paired connections are present when the **MuxType** parameter is set to **Differential**.

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 2 and 32 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.

Resources

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

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.

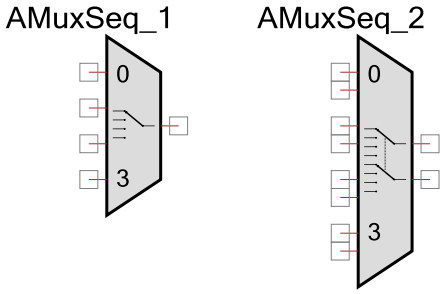
Refer to the “Find Example Project” topic in the PSoC Creator Help for more information.

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 5, the compiler setting is GNU optimized for Size or Speed.

Table 1. Performance

Function	Optimization	PSoC 3 (µs)	PSoC 5 (µs)
Next	Size	8.8	1.4
	Speed	2.4	1.4

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
1.50.e	Minor datasheet edit.	
1.50.d	Minor datasheet edit.	
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.	These changes addressed warnings about deprecated declaration that appeared during compilation with MDK and RVDS compilers.

Version	Description of Changes	Reason for Changes / Impact
	Changed type of AMux channel variable from unsigned to signed integer because -1 is used to indicate that no channel is selected.	

© Cypress Semiconductor Corporation, 2010-2016. 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, 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.