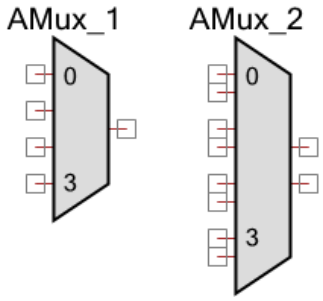
Analog Multiplexer (AMux)

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
- Multiple simultaneous connections
- Bidirectional (passive)

General Description

The analog multiplexer (AMux) component can be used to connect none, one, or more analog signals to a different common analog signal. The ability to connect more than one analog signal at a time provides cross-bar switch support, which is an extension beyond traditional mux functionality.

When to Use an AMux

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

Input/Output Connections

This section describes the various input and output connections for the AMux. 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 AMux is capable of having between 1 and 64 analog switchable connections.

0-32 (paired) – Analog *

The paired switchable connections are only used when the **MuxType** parameter is set to **Differential**.

common – Analog

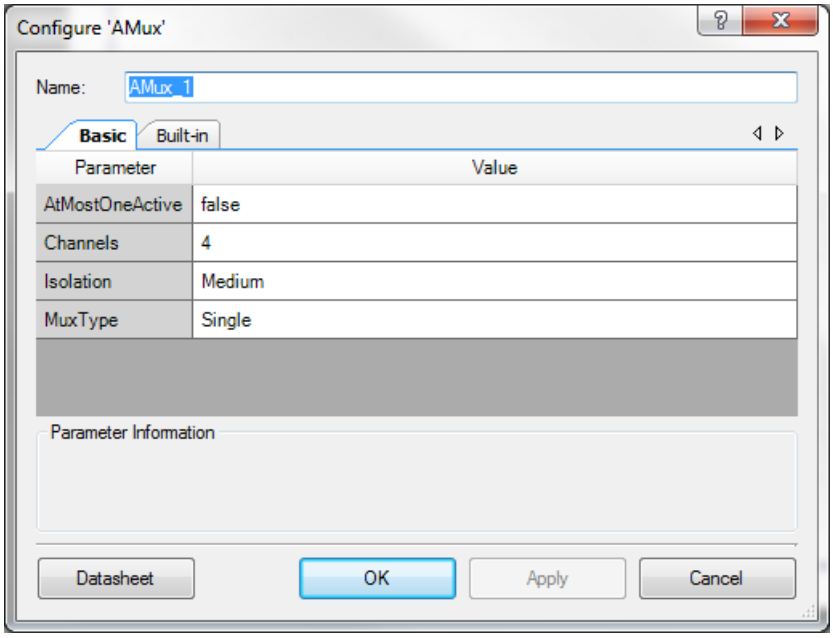
The “common” signal is the common connection; it is not labeled. The channel selected with the AMux_Select() function is connected to this terminal.

common (paired) – Analog *

The “common (paired)” signals are the common paired connections, when using a differential mux. The channels selected with the AMux_Select() function are connected to this terminal.

Component Parameters

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



The AMux provides the following parameters.

Channels

This parameter selects the number of switchable connections 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** switchable connection mux and a **Differential** switchable connections mux. **Single** is used when the connectable 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.

AtMostOneActive

When set to true, this parameter removes the cross-bar switch support from the AMux. This limits the AMux to at most one common connection. Setting this to true removes the “Connect” API from the generated code. It can optimize the performance of the AMux.

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

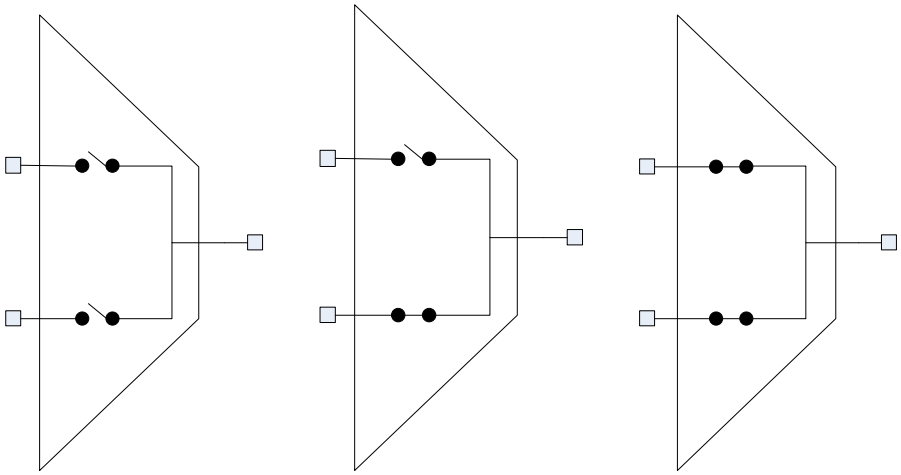


Figure 2. Double Switching, unique inner switches

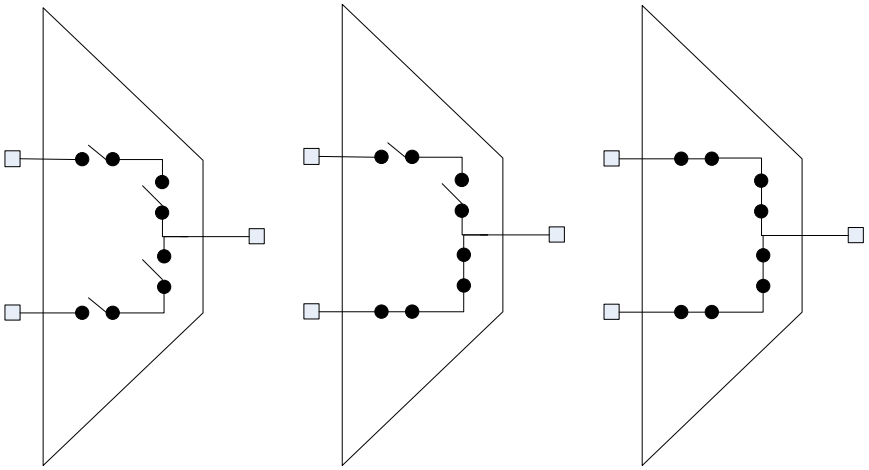
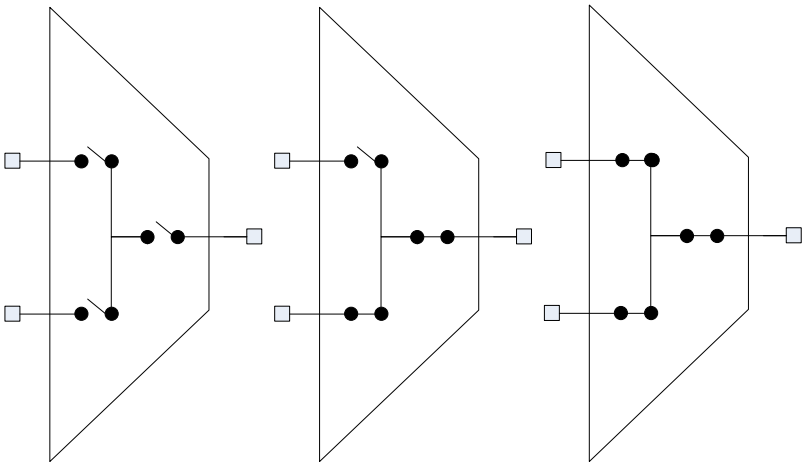


Figure 3. Dynamic Switching, shared inner switch



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 “AMux_1” to the first instance of a component in a given design. You can the 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 “AMux.”

Function	Description
AMux_Init()	Disconnects all channels
AMux_Start()	Disconnects all channels
AMux_Stop()	Disconnects all channels
AMux_Select()	Disconnects all channels, then connects “chan”. When **AtMostOneActive** is true, this is implemented as AMux_FastSelect().
AMux_Connect()	Connects “chan” signal, but does not disconnect other channels. When **AtMostOneActive** is true, this function is not available.
AMux_Disconnect()	Disconnects only “chan” signal
AMux_FastSelect()	Disconnects the last channel that was selected by the AMux_Select() or AMux_FastSelect() function, then connects the new signal “chan”
AMux_DisconnectAll()	Disconnects all channels

void AMux_Init(void)

Description: Disconnects all channels.

Parameters: None

Return Value: None

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

void AMux_Start(void)

Description: Disconnects all channels.

Parameters: None

Return Value: None

Side Effects: None

void AMux_Stop(void)

Description: Disconnects all channels.

Parameters: None

Return Value: None

Side Effects: None

void AMux_Select(uint8 chan)

Description: The AMux_Select() function first disconnects all other channels, then connects the given channel. When **AtMostOneActive** is true, this is implemented as AMux_FastSelect().

Parameters: chan: The channel to connect to the common terminal.

Return Value: None

Side Effects: Connections made either by AMux_Connect() or AMux_FastSelect() are disconnected when using AMux_Select().

void AMux_FastSelect(uint8 chan)

Description: This function first disconnects the last connection made with the AMux_FastSelect() or AMux_Select() functions, then connects the given channel. The AMux_FastSelect() function is similar to the AMux_Select() function, except that it is faster because it only disconnects the last channel selected rather than all possible channels.

Parameters: chan: The channel to connect to the common terminal

Return Value: None

Side Effects: If the AMux_Connect() function was used to select a channel prior to calling AMux_FastSelect(), the channel selected by AMux_Connect() is not disconnected. This is useful when parallel signals must be connected.

void AMux_Connect(uint8 chan)

Description: This function connects the given channel to the common signal without affecting other connections. When **AtMostOneActive** is true, this function is not available.

Parameters: chan: The channel to connect to the common terminal

Return Value: None

Side Effects: Calling the function AMux_Select() will disconnect any channel connected with the AMux_Connect() function before connecting the channel passed to the AMux_Select() command.

void AMux_Disconnect(uint8 chan)

Description: Disconnects only the specified channel from the common terminal.

Parameters: uint8 chan: The channel to disconnect from the common terminal

Return Value: None

Side Effects: None

void AMux_DisconnectAll(void)

Description: Disconnects all channels.

Parameters: None

Return Value: None

Side Effects: None

Sample Firmware Source Code

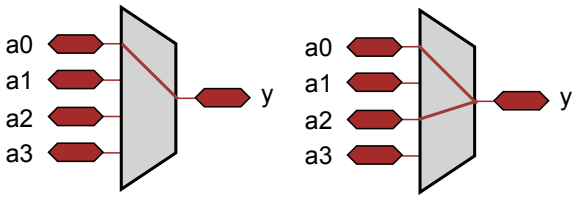
PSoC Creator provides many example projects that include schematics and example code in the Find Example Project dialog. For component-specific examples, open the dialog from the Component Catalog or an instance of the component in a schematic. For general examples, open the dialog from the Start Page 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 AMux is not like most hardware muxes. Two things make the AMux different from a standard fixed hardware mux. First, it is a collection of independent switches, and second, it is controlled by firmware not hardware.

Because of these two differences, the AMux is flexible and allows more than one signal at a time to be connected to the common signal. When the **AtMostOneActive** parameter is set to false, two or more signals can be connected to the common signal at any given time.



Performance

The 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. 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 at optimization level 5. For PSoC 4 and PSoC 5LP, the compiler setting is GNU optimized for Size.

Table 1. Performance

Function	Mux Single Inputs	PSoC 3 (µs)	PSoC 4 (µs)	PSoC 5LP (µs)
Connect*	2	4.1	1.3	1.9
	4	3.9	1.3	1.9
Disconnect	2	3.9	1.4	1.9
	4	3.8	1.4	1.8
Select*	2	14.6	3.5	5.5
	4	22.6	5.8	8.1
FastSelect	2	9.3	2.6	4.0
	4	9.3	2.6	4.0

* When “AtMostOneActive” is set to true, the Connect function is not available, and the Select function will has the same performance as FastSelect.

Resources

The AMux uses the 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	91	1	88	1	88	1
Differential	141	1	136	1	140	1

MISRA Compliance

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

DC and AC Electrical Characteristics

The AMux 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 requests from customers.
	Added MISRA Compliance section. This component was not verified for MISRA-C:2004 coding guidelines compliance.	
1.60	Changed data in performance table.	Previously published performance numbers were incorrect.
	Added AtMostOneActive and Isolation parameters. Updated the screen capture.	The AtMostOne parameter allows you to remove the cross-bar switch support from the AMux. The Isolation parameter 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 AMux_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	Symbol picture updated.	Updated to comply with corporate standard.

© 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.