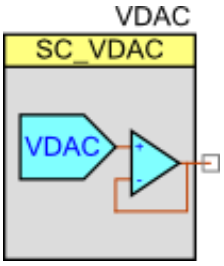
Switched-Cap VDAC

1.0



Features

- 13-bit signed input data
- Monotonic to 11 bits
- Up to two reference voltage terminals for flexible transfer functions
- Can be used as an MDAC
- Buffered output can drive loads with up to 10 mA

General Description

The Switched-Cap VDAC is a general-purpose digital-to-analog converter which is monotonic to 11 bits. It has up to two user-selectable reference voltage input terminals (one of which is referred to as “Analog Ground”, a potential somewhere between the rails used to set system operating point). It can also accept a signal from the analog routing of the PSoC device, thus forming an MDAC.

When to Use a Switched-Cap VDAC

This component can be used wherever a digital code needs to be converted into a buffered analog signal.

Input/Output Connections

This section describes the various input and output connections for the Switched-Cap VDAC component. An asterisk (*) near the terminal name in the following list indicates that the terminal may not be shown on the component symbol for the conditions listed in the description of that I/O.

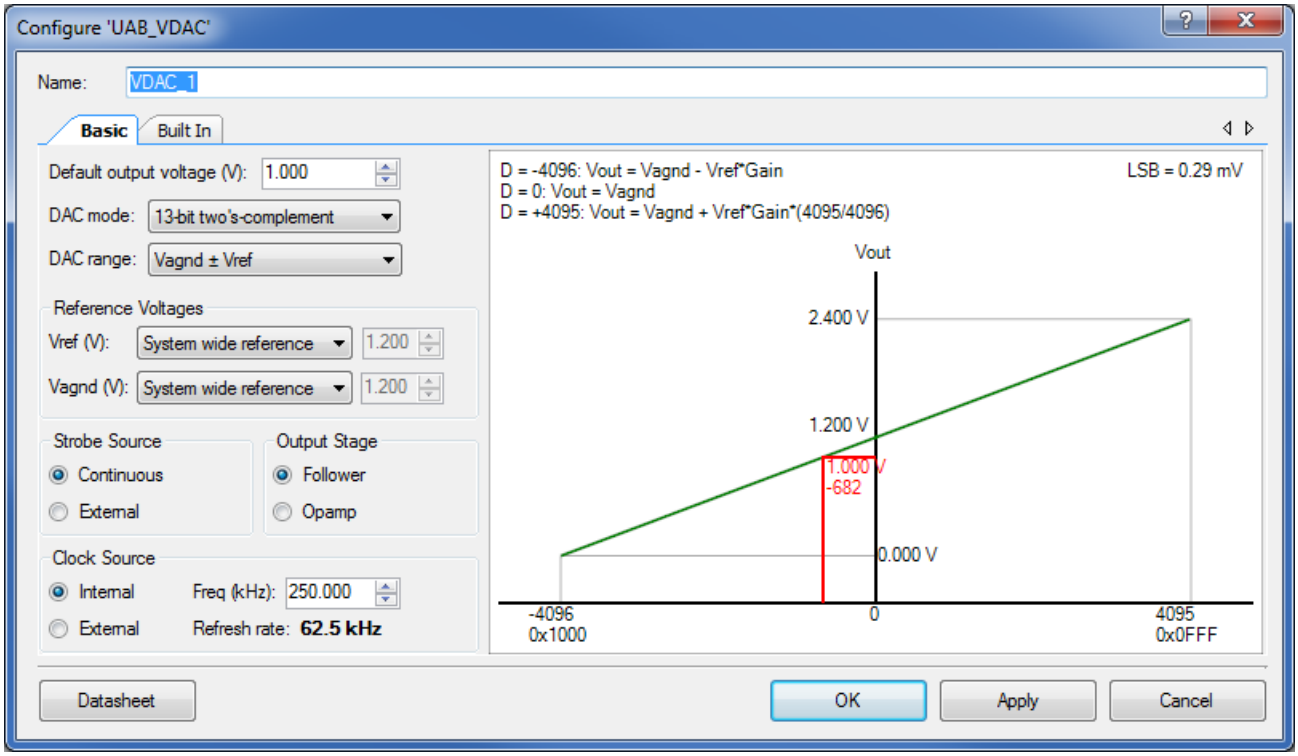
Terminal Name	I/O Type	Description
fb* (feedback)	Analog Input	This input routes to the inverting terminal of the output stage. By default, the output stage is simply a follower, and this terminal is hidden.

Terminal Name	I/O Type	Description
vref*	Analog Input	This input is for a signal that is the maximum deflection from vagnd. The voltage must be 0.6 V or greater. By default, the system-wide reference voltage, 1.2 V by default, is used, so this terminal is hidden.
vagnd*	Analog Input	This input is for a signal that represents the zero code. It must be 0.6 V or greater. By default, the system-wide reference voltage is used, so this terminal is hidden.
strobe*	Digital Input	This input is for a digital signal which can be used to control when the VDAC output changes. By default, the VDAC output changes as soon as it is passed a value, so this terminal is hidden.
clk*	Clock Input	This terminal is for a clock which controls the timing of the switching capacitors. By default, the component configures its own clock, rather than using one from the schematic, so this terminal is hidden.
vout (unlabeled)	Analog Output	This terminal is for the converted voltage output.

Component Parameters

Drag a Switched-Cap VDAC component onto your design and double-click it to open the Configure dialog. This dialog has the following tab with different options.

Basic Tab

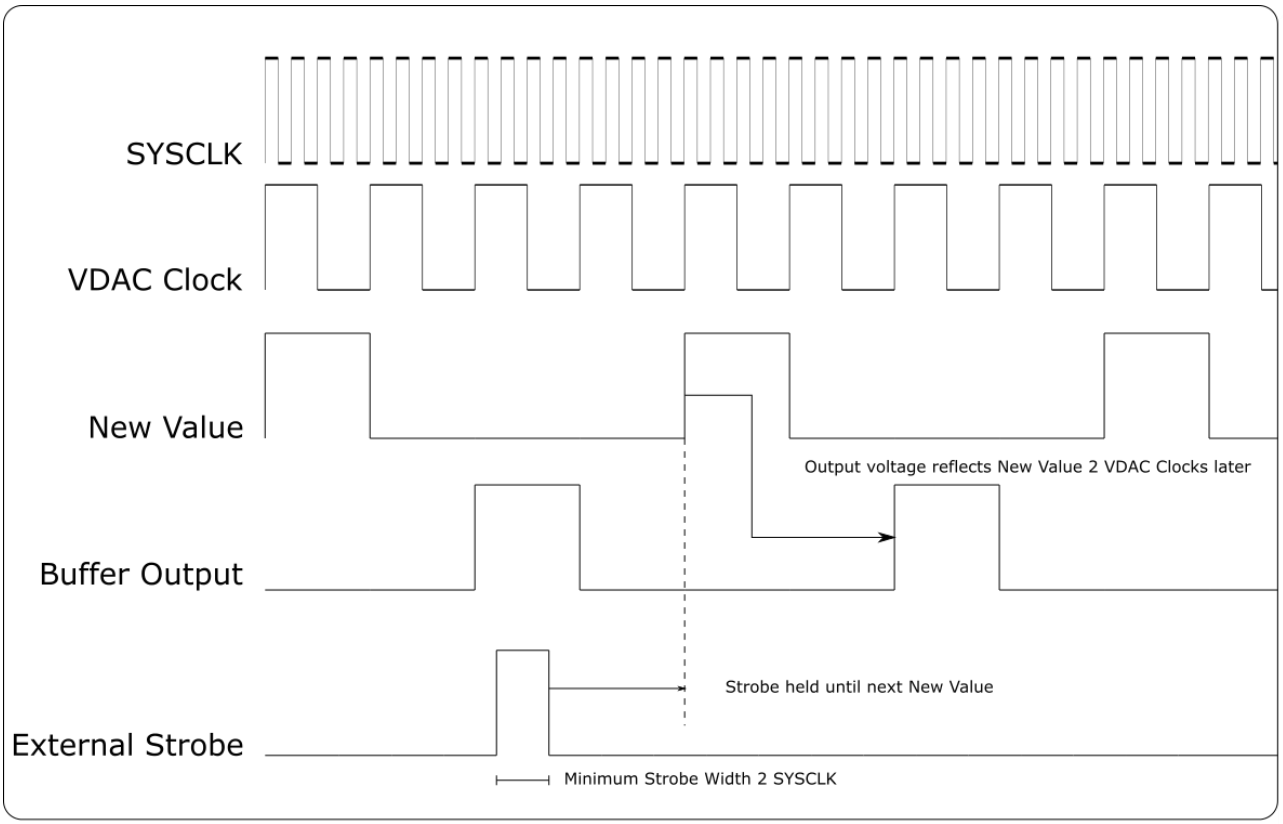


Parameter Name	Description
Default output voltage (V)	The entry box sets the default output voltage for the VDAC. The corresponding point on the transfer graph is updated when these change. **Default 1 V.**
DAC mode	This dropdown box selects between supported numeric formats for the VDAC data. Choices are: "**13-bit two's complement**" and "13-bit sign-and-magnitude". The transfer graph is updated when this changes.
DAC range	This dropdown box selects the range of the VDAC. A code of '0' always corresponds to Vagnd. Using gain, the output can have a linear range to within about 60 mV of either rail. Choices are: **Vagnd ± Vref**, Vagnd ± 2Vref, or Vagnd ± 4Vref. The transfer graph is updated when this changes.
Reference Voltage: Vref (V)	This dropdown box selects the source of the Vref signal (magnitude that the output can deflect above or below Vagnd). Choices are: "**System wide reference**" or "External". When "System wide reference" is selected, the numeric entry box is disabled, and the system wide vref is loaded for calculations. When "External" is selected, the vref terminal is exposed and the numeric entry box is enabled, allowing the user is able to specify what the expected voltage will be. Vref must be at least 0.6 V.

Parameter Name	Description
Reference Voltage: Vagnd (V)	This control is in most ways similar to Vref (V). In particular, Vagnd must be at least 0.6 V.
Strobe Source	These radio buttons configure how the DAC value is updated. "**Continuous**" means the whenever a value is passed to the DAC, it updates on the next refresh cycle. "External" means a digital strobe terminal (strobe) is exposed, described earlier in Input/Output Connections. .
Output Stage	These radio buttons configure the output stage into one of two modes. "**Follower**" configures the output stage into a unity-gain buffer. "Feedback" exposes a terminal (fb), described earlier in Input/Output Connections which allows external components to be connected within the feedback path.
Clock Source	These radio buttons choose whether the clock is internal or routed from a terminal. The "Freq (kHz)" entry box shows the rate of the clock used by the analog block. The "Refresh rate" label shows the resulting update rate, which is one fourth of the clock rate. The clock frequency must be between 100 kHz and 2 MHz. When "**Internal**" is selected, the entry box is enabled, and the clock can be set by the user. When "External" is selected, the entry box is disabled and the customizer attempts to discover clock component connected to its clk terminal, described earlier in Input/Output Connections. .

Block Timing

The Switched-Cap VDAC is built on the hardware of the Universal Analog Block (UAB). It uses four clock phases to create the desired output voltage. To use the strobe terminal, the signal must be asserted for at least two SYSCLK and will cause an update on the next New Value positive edge. This loads the value stored in VDAC_VALUE_REG. (VDAC_VALUE_REG is set using VDAC_SetValue() or by using DMA).



If an external strobe is not used, the new value is loaded every time New Value asserts, and it is output when Buffer Output asserts.

DMA

DMA is available in PSoC Analog Coprocessor devices.

The VDAC API defines an alias to the update register, which can be used for DMA. For example, to update the VDAC via DMA, with a DMA Channel component instance named ‘DMA’ and a Switched-Cap. VDAC component instance named ‘VDAC’:

```
DMA_SetDstAddress(VDAC_VALUE_REG);
```

Application Programming Interface

Application Programming Interface (API) routines allow you to configure the component using software. The following sections list and describe each function and dependencies.

Functions

API functions allow configuration of the component using the CPU.

By default, PSoC Creator assigns the instance name “VDAC_1" to the first instance of a Switch Cap. VDAC component in a given design. You can rename it 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 "VDAC”.

Function	Description
VDAC_Start()	Initializes and enables the hardware underlying the component.
VDAC_Stop()	Disables the hardware underlying the component.
VDAC_SetValue()	Sets the value to be converted.
VDAC_SaturateTwosComp()	Takes a 32-bit two’s-complement number and returns a 13-bit two’s-complement number within the VDAC’s valid range.
VDAC_SaturateSignMagnitude()	Takes a sign and a magnitude and returns a 13-bit sign-and-magnitude number within the VDAC’s valid range.
VDAC_SetFormat()	Changes the numeric format of the VDAC.
VDAC_SetStrobeMode()	Enables or disables the external strobe.
VDAC_SetHiZ()	Enables or disables the output buffer.
VDAC_SetGain()	Adjusts the gain.
VDAC_Init()	Called by VDAC_Start(). Configures the VDAC into its initial condition.
VDAC_Enable()	Called by VDAC_Start(). Enables the hardware underlying the VDAC.
VDAC_Sleep()	Saves the buffer state and disables the VDAC.
VDAC_Wakeup()	Enables the VDAC and restores buffer state.

void VDAC_Start(void)

Description: Initializes the component and enables hardware blocks. Calls both VDAC_Init() and VDAC_Enable(). Always call this before trying to use the features of the VDAC.

Parameters: None

Return Value: None

Side Effects: None

void VDAC_Stop(void)

Description: Disables the underlying hardware. Configuration registers clear due to analog reset.

Parameters: None

Return Value: None

Side Effects:

void VDAC_SetValue(int32 value)

Description: Sends value to the VDAC. If using a strobe signal, the value will update on next strobe, otherwise the value will update on the next clock cycle.

Parameters: value

This is a signed integer assumed to be formatted correctly according to the VDAC's mode. It is passed, unaltered, to the VDAC.

Return Value: None

Side Effects: None

int32 VDAC_SaturateTwosComp(int32 value)

Description: Saturates a 32-bit value to be within the VDAC's valid range. Output is valid for VDAC in two's-complement mode.

Parameters: value

This is a signed integer of any value. If it is outside the VDAC's valid range, it is saturated before being returned.

Return Value: The value returned is correctly formatted, and ready to use with VDAC_SetValue.

Side Effects: None

void VDAC_SaturateSignMagnitude(VDAC_sign_enum sign, uint32 magnitude)

Description: Saturates a 32-bit magnitude to be within the VDAC's valid range and applies the provided sign. Output is valid for VDAC in sign-and-magnitude mode.

Parameters: magnitude

This is an unsigned integer of any value. If it is outside the VDAC's valid range, it is saturated before sign is applied.

sign

This is the intended sign for the number.

Constant	Description
VDAC_SIGN_POSITIVE	Apply a positive sign.
VDAC_SIGN_13TWOSCOMP	Inputs to SetValue() assumed to be 13-bit two's-complement format.

Return Value: Returns an integer properly formatted for use with SetValue(), with VDAC in sign-and-magnitude mode.

Side Effects: None

Int32 VDAC_ConvertUnsigned2TwosComp(uint32 value)

Description: Saturates input to VDAC's valid range, and converts to a number valid for VDAC's two's-complement mode.

Parameters: value

The value to saturate and convert.

Return Value: Returns an integer properly formatted for use with SetValue(), with the VDAC in two's-complement mode.

Side Effects: None

Void VDAC_SetFormat(VDAC_format_enum dacFormat)

Description: Changes the mode of the VDAC.

Parameters: dacFormat

The new mode to use

Constant	Description
VDAC_FORMAT_13SIGNMAG	Inputs to SetValue() assumed to be 13-bit sign-and-magnitude format.
VDAC_FORMAT_13TWOSCOMP	Inputs to SetValue() assumed to be 13-bit two's-complement format.

Return Value: None

Side Effects: None

Void VDAC_SetStrobeMode(VDAC_strobe_mode_enum isHWStrobed)

Description: Changes the VDAC from requiring a strobe signal to continuously updating, and vice versa.

Parameters: isHWStrobed

Constant	Description
VDAC_STROBE_CONTINUOUSLY	VDAC output will update as soon as it receives a new value.
VDAC_STROBE_FROM_TERMINAL	VDAC output will update only after the signal connected to the strobe terminal is asserted.

Return Value: None

Side Effects: None

Void VDAC_SetHiZ(VDAC_output_state_enum isHiZ)

Description: Put the VDAC output into a high-impedance state, or restore it to a buffering state.

Parameters: isHiZ

The desired output state of the VDAC.

Constant	Description
VDAC_OUTSTATE_HIZ	Put the VDAC into a high-impedance output state.
VDAC_OUTSTATE_DRIVEN	Restore the VDAC to a buffered output state.

Return Value: None

Side Effects: None

Void VDAC_SetRange (VDAC_range_enum rangeFactor)

Description: Set the range of the VDAC. The range adjusts the voltage per least significant bit. An input of zero always corresponds to a voltage of analog ground.

Parameters: gainFactor

Constant	Description
VDAC_RANGE_VREF	Default range; Vref reflects around analog ground.
VDAC_ RANGE_2VREF	Double range; 2Vref reflects around analog ground.
VDAC_ RANGE_4VREF	Quadruple range; 4Vref reflects around analog ground.

Return Value: None

Side Effects: None

Void VDAC_Init(void)

Description: Configures hardware, after reset, according to parameters defined in the customizer.

It is not necessary to call Init() because the Start() API calls this function and is the preferred method to begin the component operation. Init will not necessarily restore a default state if the block is not in a reset state. VDAC_Init only writes registers when the resulting register will be nonzero, and it only writes to registers that the VDAC utilizes.

Parameters: None

Return Value: None.

Side Effects: Component output will be in a high impedance state until VDAC_Enable() is called.

Void VDAC_Enable(void)

Description: Enables the component. It is not necessary to call Enable() because the Start() API calls this function and is the preferred method to begin the component operation. Activates the UAB and CTB. Requires that the blocks have been configured, such as by calling VDAC_Init().

Parameters: None.

Return Value: None.

Side Effects: Causes output state to be driven until set to analog high impedance by, for example, VDAC_Stop().

Void VDAC_Sleep(void)

Description: Disables block's operation and saves its configuration. Should be called just prior to entering sleep.

Parameters: None.

Return Value: None.

Side Effects: None.

Void VDAC_Wakeup(void)

Description: Enables block's operation and restores its configuration. Should be called just after awaking from sleep.

Parameters: None.

Return Value: None.

Side Effects: None.

Global Variables

Variable	Description
VDAC_ initVar (static)	The initVar variable is used to indicate initial configuration of this component. This variable is prepended with the component name. The variable is initialized to zero and set to 1 the first time VDAC_Start() is called. This allows for component initialization without reinitialization in all subsequent calls to the VDAC_Start() routine. It is necessary to reinitialize the component when the device is going through sleep cycles. Therefore, the variable is set to zero when going into sleep VDAC_Sleep() and set during the reinitialization done in VDAC_Wakeup().
VDAC_backup (static)	Contains configuration of VDAC before sleep. Enable and HiZ states are the only data required to be preserved.

Interrupt Service Routine

The UAB can generate an interrupt when the pending value becomes the current value. For example, a project could set the pending value in the interrupt and wait for an external strobe to cause the pending value to load and trigger the interrupt again.

All UABs on one chip share one entry in the interrupt table, so interrupts are not handled on a per-component basis.

To use an interrupt, add a Global Signal Reference (GSR) component to the schematic, and attach an Interrupt Service Routine component to it. Configure the GSR to use the “Combined UAB interrupt (UABInt)” signal source.

Code Examples and Application Notes

This section lists the projects that demonstrate the use of the component.

Code Examples

PSoC Creator provides access to code examples in the 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 "Code Example" topic in the PSoC Creator Help for more information.

There are also numerous code examples that include schematics and example code available online at the Cypress Code Examples web page.

API Memory Usage

Shows the Flash, SRAM, and stack usage of the component.

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 an 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.

PSoC Analog Coprocessor (GCC)

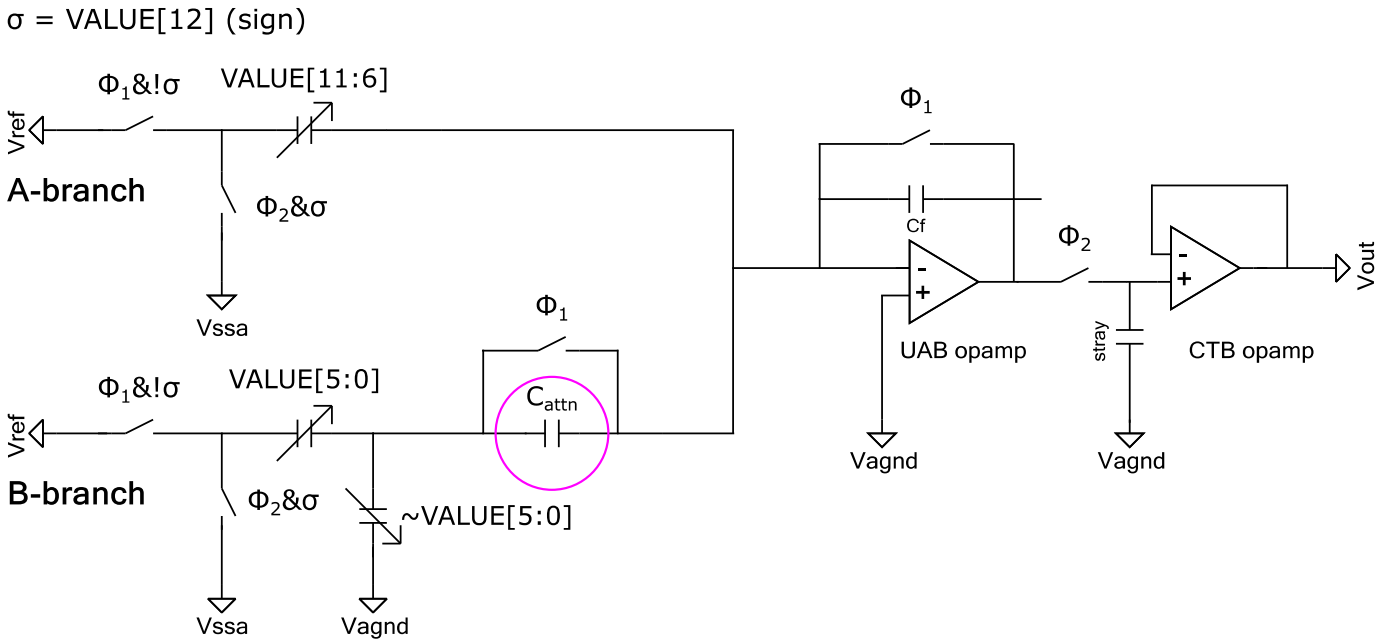
Configuration	Flash Bytes	SRAM Bytes
All functions	1536	312
No Saturation/Conversion	1408	312

Functional Description

PSoC devices that contain a Universal Analog Block (UAB) can configure it as a VDAC. Due to the discrete-time nature this design, the desired output is only driven intermittently. To create a continuously driven output, the UAB output is sampled and buffered.

The following diagram shows the switched-capacitor topology of the UAB configured as a sign-and-magnitude VDAC, connected to an output buffer. Two of the input capacitor arrays are used, and the feedback capacitor array is used. The clocks' waveforms are defined in configuration registers and assigned on a per-switch basis.

When the VDAC is configured as two's-complement, hardware logic decodes the input value to its corresponding sign-and-magnitude value, so no application-level decoding is required. Two's-complement is the default mode of the Switch Cap. VDAC, which means it works out-of-the-box with the signed format used by the PSoC Analog Coprocessor's Cortex M0+.



Definitions

- Clk_hf – High frequency clock. The vast majority of logic operates from this or a divided version of it.
- CTB – Continuous Time Block. Cypress's highly configurable opamp with resistor network.
- GSR – Global Signal Reference. When many components must share an interrupt, they can't each have a schematically-routable ISR (Interrupt Service Routine) component. Instead a single component, GSR, provides a place for interrupts to be handled.
- MDAC – Multiplying digital-analog converter. The digital value as well as reference voltage are inputs, and together they correspond to the voltage output. The reference voltage is multiplied by the digital value.
- PRB – Programmable Reference Block. A voltage source with many taps, which serve as references.

- SYSCLK - The main system clock driving buses, registers and processor, this is a pre-scaled version of clk_hf. This clock must be the higher than all other clocks in the system that are divided off clk_hf.
- UAB – Universal Analog Block. Cypress's highly configurable switched-capacitor network.
- VDAC – Voltage digital-analog converter. The digital value is considered the only input, and it corresponds to a voltage output.
- VDDA – Drain voltage in the analog domain. Also the supply voltage. Commonly 1.8, 3.3, or 5 V.
- VSSA – Source voltage in the analog domain. How we refer to 0 V.

Clock Selection

This component allows the user to use:

1. an internal clock or
2. route a clock from their schematic

In either case, the VDAC output refreshes at a rate that is one-fourth that of the used clock.

Industry Standards

This section lists the industry standard compliance of the Switched-Cap VDAC component.

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. Project deviations are described in the MISRA Compliance section of the *System Reference Guide* along with information on the MISRA compliance verification environment.

The Switched-Cap VDAC component has the following specific deviations:

Rule	Rule Class	Rule Description	Description of Deviation(s)
3.1	R	All usage of implementation-defined behaviour shall be documented.	Embedded component, UABPRIM, source code has comments containing one of the characters '$', '@' or '".

Rule	Rule Class	Rule Description	Description of Deviation(s)
19.7	A	A function should be used in preference to a function-like macro.	UABPRIM source code uses function-like macros to take generic functions and rename them for specific use cases and use predefined parameters making the API easier to use. Also, there are read-modify-write macros that are in most UABPRIM API functions and are used to improve readability of the code so that the intent is clearly understood. The macros have proper parenthesis shielding of the parameters as well as the whole macro.

This component has the following embedded components: OpAmp_P4_v1_20 (Opamp [v1.20]) and UABPRIM_v1_0 (UABPRIM [v1.0]). Refer to the corresponding component datasheets for information on their MISRA compliance and specific deviations.

Resources

The switched-capacitor VDAC component uses the following device resources:

- Analog fixed function:
  - Half of a UAB
  - Half of a CTB
- Analog routing, two of:
  - Design-wide voltage reference
  - Programmable reference bus
  - General purpose analog

Registers

Refer to the chip Technical Reference Manual (TRM) for more information about the UAB and CTB registers.

DC and AC Electrical Characteristics

Note Final characterization data for PSoC 4000S, PSoC 4100S and PSoC Analog Coprocessor devices is not available at this time. Once the data is available, the component datasheet will be updated on the Cypress web site.

Component Errata

Cypress ID	Component Version	Problem	Workaround
238079	1.0	Due to unexpected interactions in the first silicon of the Analog Coprocessor, this VDAC component does not meet our targets for linearity. DNL 'spikes' of around 8 LSBs may be experienced. This component, running on first silicon, should be considered a prototype for project development, not a production component.	None.

Component Changes

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

Version	Description of Changes	Reason for Changes / Impact
1.0.a	Edited datasheet.	Final characterization data for PSoC 4000S, PSoC 4100S and PSoC Analog Coprocessor devices is not available at this time. Once the data is available, the component datasheet will be updated on the Cypress web site.
1.0	Initial version	New component

© Cypress Semiconductor Corporation, 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.