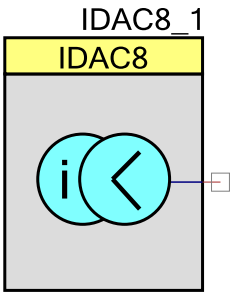
8-Bit Current Digital to Analog Converter (IDAC8)

1.50

Features



- Three ranges 2040 uA, 255 uA, and 32.875 uA
- Software or clock driven output strobe
- Data source may be CPU, DMA, or UDB
- Current sink or source selectable

General Description

The IDAC8 component is an 8-bit current output DAC (Digital to Analog Converter). You may configure the IDAC8 component a number of ways depending on your needs. The IDAC8 may be controlled by hardware, software, or with a combination of both hardware and software. The IDAC8 is also easily configured as a current sink or source.

When to use a IDAC8

Use the IDAC8 when a fixed or programmable current source is required in an application.

Input/Output Connections

This section describes the various input and output connections for the IDAC8. 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.

output – Analog

The output terminal is the connection to the current source/sink. It may be routed to any analog compatible pin on the device. When the highest current range is selected (2048 uA), the output should only be routed to a specific set of pins that provide a direct low resistive path. These port pins are P0[6], P0[7], P3[0], or P3[1].

PRELIMINARY

data[7:0] – Input *

This 8-bit wide data signal connects the IDAC8 directly to the DAC bus. The DAC bus may be driven by UDB based components or control registers, or routed directly from GPIO pins. This input is enabled by setting the **Data_Source** parameter to "DAC Bus". If the "DMA or CPU" option is selected instead, the bus connection will disappear from the component symbol.

Use the data[7:0] input when hardware is capable of setting the proper value without CPU intervention. When using this option, **Strobe_Mode** should be enabled as well.

For many applications this input is not required, but instead the CPU or DMA will write a value directly to the data register. In firmware, you may use the SetRange() function or directly write a value to the IDAC8_1_Data register (assuming an instance name of "IDAC8_1").

strobe – Input *

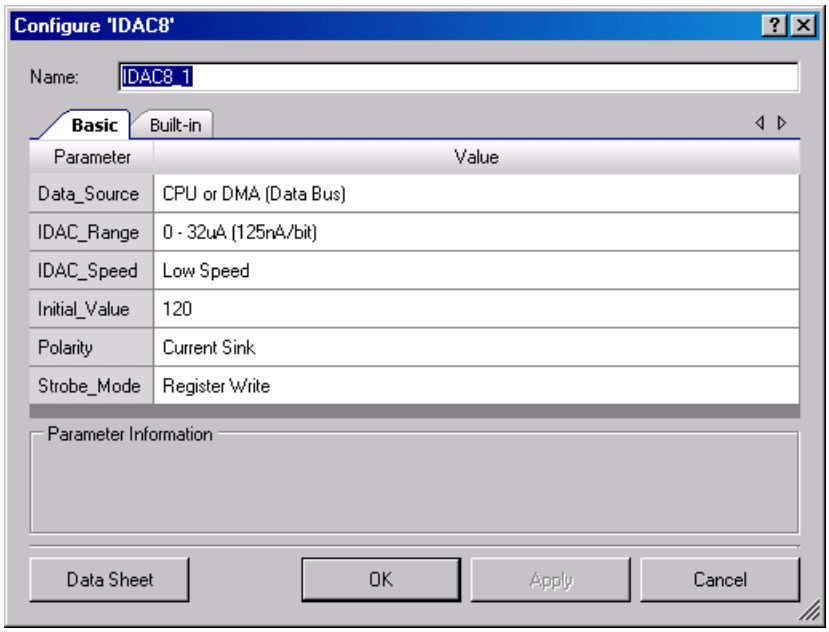
The strobe input is an optional signal input and is selected with the **Strobe_Mode** parameter. If **Strobe_Mode** is set to "External", the pin will be visible and must be connected to a valid digital source. In this mode the data is transferred from the IDAC8 register to the DAC on the next positive edge of the strobe signal.

If this parameter is set to "Register Write" the pin will disappear from the symbol and any write to the data registers will be immediately transferred to the DAC.

For audio or periodic sampling applications, the same clock used to clock the data into the DAC could also be used to generate an interrupt. Each rising edge of the clock would transfer data to the DAC and cause an interrupt to get the next value loaded into the DAC register.

Parameters and Setup

Drag an IDAC8 component onto your design and double-click it to open the Configure dialog.



The IDAC8 component provides the following parameters.

Data_Source

This parameter selects the source of the data to be written into the DAC register. If the CPU (firmware) or the DMA will write data to the IDAC8, then select "DMA or CPU". If data is written directly from the UDBs or UDB-based component, then the "DAC bus" should be selected. Note that there is only one DAC bus that is shared by all of the viDAC8 analog blocks.

IDAC_Range

This parameter enables the designer to set one of three current ranges as the default value. The range may be changed at anytime during runtime with the SetRange() function. If the highest current range, "0 - 2040uA" is selected, the output should be routed to one of the special pins that provide a low resistive path. These pins are P0[6], P0[7], P3[0], and P3[1].

Range	Lowest Value	Highest Value	Step Size
0 – 32 uA	0.0 uA	31.875 uA	0.125 uA
0 – 255 uA	0.0 uA	255 uA	1 uA
0 – 2040 uA	0.0 uA	2040 uA	8 uA

IDAC_Speed

This parameter provides two settings for the designer, Low Speed and High Speed. In the Low Speed mode, the settling time is slower but it consumes less operating current. In the High Speed mode, the current settle much faster, but at a cost of more operating current.

Initial_Value

This is the initial value the IDAC8 will present after the Start() command is executed. The SetValue() function or a direct write to the DAC register will override the default value at anytime. Legal values are between 0 and 255 inclusive.

Polarity

The Polarity parameter allows the designer to select whether the IDAC8 sinks or sources current to its load. When the "Current Source" option is selected, the output of the DAC will source current to a load that is connected to Vss. In the "Current Sink" mode, it will supply current to a load that is connected to Vdd.

Strobe_Mode

This parameter selects whether the data is immediately written to the DAC as soon as the data is written into the IDAC8 data register. This mode is selected when the "Register Write" option is selected. When the "External" option is selected, a clock or signal from UDBs controls when the data is written from the DAC register to the actual DAC.

Resources

The IDAC8 uses one vIDAC8 analog block.

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 "IDAC8_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 "IDAC8."

Function	Description
void IDAC8_Init(void)	Initializes or restores default IDAC8 configuration

Function	Description
void IDAC8_Enable(void)	Enables the IDAC8.
void IDAC8_Start(void)	Initialize the IDAC8 with default customizer values. Enable and power up the IDAC8.
void IDAC8_Stop(void)	Disables the IDAC8 and sets it to the lowest power state.
Void IDAC8_SetSpeed(uint8 speed)	Set DAC speed.
void IDAC8_SetPolarity(uint8 polarity)	Sets the output mode to current sink or source.
void IDAC8_SetRange(uint8 range)	Sets full scale range for IDAC8.
void IDAC8_SetValue(uint8 value)	Sets value between 0 and 255 with the given range.
void IDAC8_SaveConfig(void)	Empty function. Provided for future usage.
void IDAC8_RestoreConfig(void)	Empty function. Provided for future usage.
void IDAC8_Sleep(void)	Stops and saves the user configuration.
void IDAC8_WakeUp(void)	Restores and enables the user configuration.

Global Variables

Variable	Description
IDAC8_initVar	Indicates whether the IDAC8 has been initialized. The variable is initialized to 0 and set to 1 the first time IDAC8_Start() is called. This allows the component to restart without reinitialization after the first call to the IDAC8_Start() routine. If reinitialization of the component is required, then the IDAC8_Init() function can be called before the IDAC8_Start() or IDAC8_Enable() function.

void IDAC8_Init(void)

Description: Initializes/Restores default IDAC8 configuration.

Parameters: None

Return Value: None

Side Effects: All registers will be set to their initial values. This will re-initialize the component. Calling the Init() function requires a call to SetValue() if you intend to set a new value other than what is currently in the register.

void IDAC8_Enable(void)

Description: Enables the IDAC8.

Parameters: None

Return Value: None

Side Effects: None

void IDAC8_Start(void)

Description: Initialize the IDAC8 with default customizer values. Enable and power up the IDAC8 to the given power level. A power level of 0 is the same as executing the stop function.

Parameters: None

Return Value: None

Side Effects: None

void IDAC8_Stop(void)

Description: Powers down IDAC8 to lowest power state and disables output.

Note This API is not recommended for use on PSoC 3 ES2 and PSoC 5 ES1 silicon. These devices have a defect that causes connections to several analog resources to be unreliable when not powered. The unreliability manifests itself in silent failures (e.g. unpredictably bad results from analog components) when the component utilizing that resource is stopped. It is recommended that all analog components in a design should be powered up (by calling the <INSTANCE_NAME>_Start() APIs) at all times. Do not call the <INSTANCE_NAME>_Stop() APIs.

Parameters: None

Return Value: None

Side Effects: None

void IDAC8_SetSpeed(uint8 speed)

Description: Set DAC speed.

Parameters: (uint8) speed: Sets DAC speed, see table below for valid parameters.

Option	Description
IDAC8_LOWSPEED	Low speed (low power)
IDAC8_HIGHSPEED	High speed (high power)

Return Value: None

Side Effects: None

void IDAC8_SetPolarity(uint8 polarity)

Description: Sets output polarity to sink or source.

Parameters: (uint8) polarity: Sets current sink or source functionality, see table below.

Option	Description
IDAC8_SOURCE	Set mode as current source.
IDAC8_SINK	Set mode to current sink.

Return Value: None

Side Effects: None

void IDAC8_SetRange(uint8 range)

Description: Sets full scale range for IDAC8

Parameters: (uint8) range: Sets full scale range for IDAC8. See table below for ranges.

Option	Description
IDAC8_RANGE_32uA	Set full scale range to 31.875 uA
IDAC8_RANGE_255uA	Set full scale range to 255 uA
IDAC8_RANGE_2mA	Set full scale range to 2.040 mA

Return Value: None

Side Effects: None

void IDAC8_SetValue(uint8 value)

Description: Sets value to output on IDAC8. Valid values are between 0 and 255.

Parameters: (uint8) value: Value between 0 and 255. A value of 0 is the lowest (zero) and a value of 255 is the full scale value. The full scale value is dependent on the range which is selectable with the SetRange API.

Return Value: None

Side Effects: On PSoC 3 ES2 and PSoC 5 ES1, the SetValue() function should be called after enabling the power to the IDAC.

void IDAC8_Sleep(void)

Description: Stops the operation. Saves the user configuration and the component enable state. Should be called just prior to entering sleep.

Parameters: None

Return Value: None

Side Effects: None

void IDAC8_Wakeup(void)

Description: Restores the configuration registers and component enable state. Should be called just after awaking from sleep.

Parameters: None

Return Value: None

Side Effects: Calling this function without previously calling IDAC8_Sleep() may lead to unpredictable behavior.

void IDAC8_SaveConfig(void)

Description: Saves the user configuration.

Parameters: None

Return Value: None

Side Effects: Empty function. Implemented for future usage.

void IDAC8_RestoreConfig(void)

Description: Restores the user configuration.

Parameters: None

Return Value: None

Side Effects: Empty function. Implemented for future usage.

DMA

IDAC8 components do not require implementation of a DMA Request signal. The typical usage is signal generation and the data rate to IDAC8 components should be controlled externally. The DMA Wizard can be used to configure DMA operation as follows:

Name of DMA source / destination in DMA Wizard	Direction	DMA Req Signal	DMA Req Type	Description
IDAC8_Data_PTR	Destination	N/A	N/A	Stores the DAC value between 0 to 255

Sample Firmware Source Code

The following is a C language example demonstrating the basic functionality of the IDAC8 component. This example assumes the component has been placed in a design with the default name "IDAC8_1."

Note If you rename your component you must also edit the example code as appropriate to match the component name you specify.

```
#include <device.h>

void main()
{
    IDAC8_1_Start();                          // Enable IDAC8
    IDAC8_1_SetRange(IDAC8_1_RANGE_255uA);  // Set full scale range to 255uA
    IDAC8_1_SetValue(100);                    // Set value to 100uA
}
```

Functional Description

The DAC generates either a voltage or a current output. It is built using current mirror architecture; current is mirrored from a reference source to a mirror DAC. Calibration and value current mirrors are responsible for the 8-bit calibration [DACx_TR] and the 8-bit DAC value. The current is then diverted into the scaler to generate the current corresponding to the DAC value. The DAC value can either be given from the register DACx_D or from 8 lines from the UDB. This selection is made using the DACx_CR1[5] bit.

The DAC is strobed to get its output to change for the input code. The strobe control is enabled by the DACx_STROBE[3] bit. The strobe sources for the DAC can be selected from the bus write strobe, analog clock strobe to any UDB signal strobe. This selection is done on the basis of setting in DACx_STROBE[2:0].

Current (IDAC) Mode

The two mirrors for the current source and sink provide output as a current source or current sink, respectively. These mirrors also provide range options in the current mode.

When used as an IDAC, the output is an 8-bit digital-to-analog conversion current. This is done by setting the DACx_CR0 [4] register. The reference source is a current reference from the analog reference called IREF(DAC). In this mode, there are three output ranges selected by register DACx_CR0 [3:2]:

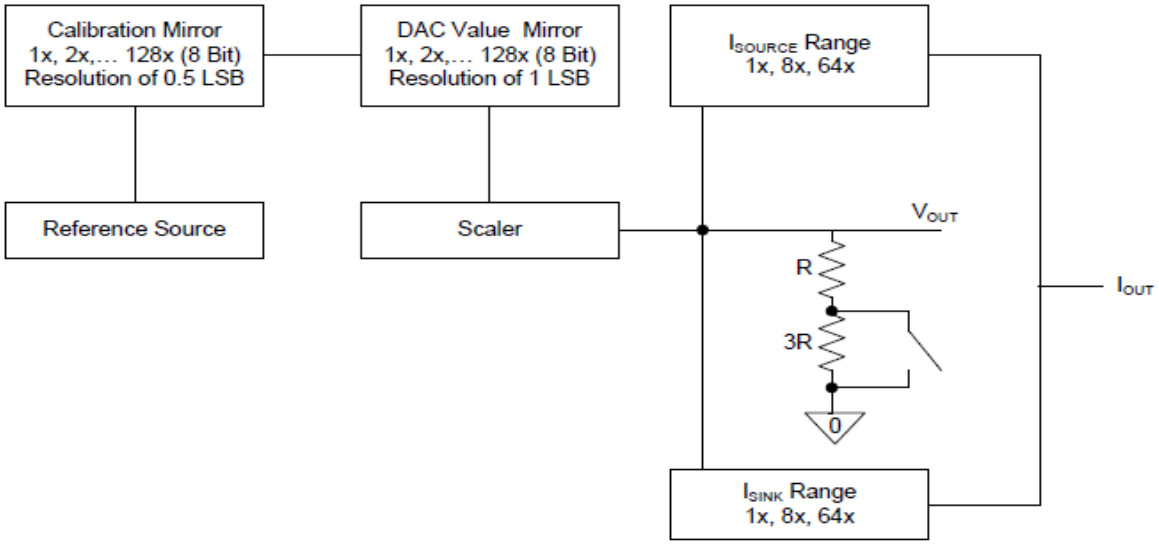
- 0 to 2.048 mA, 8 µA/bit
- 0 to 256 µA, 1 µA/bit
- 0 to 32 µA, 0.125 µA/bit

For each level, there are 255 equal steps of M/256 where M = 2.048 mA, 256 µA, or 32 µA. In the 2.048 mA configuration, the block is intended to output a current into an external 600Ω load.

The IDAC is capable of converting up to 8 Msps. You also have the option of selecting whether the output is a current source or a sink. This is done by the DACx_CR1[2] register. The selection between source and sink for the IDAC can also be done using a UDB input. UDB control for the source sink selection is enabled using the DACx_CR1[3] bit.

Block Diagram and Configuration

The following shows the block diagram for the IDAC8 component.



Registers

The functions provided support most of the common runtime functions that are required for most applications. The register reference below provides a brief description for the advanced user. The IDAC8_Data register may be used to write data directly to the DAC without using the API. This may be useful for either the CPU or DMA.

Table 1 IDAC8_CR0

Bits	7	6	5	4	3	2	1	0
Value	reserved			mode	Range[1:0]		hs	reserved

- mode: Sets DAC to either voltage or current mode.
- range[1:0]: DAC range settings.
- hs: Use to set data speed.

Table 2 IDAC8_CR1

Bits	7	6	5	4	3	2	1	0
Value	reserved		mx_data	reset_udb_en	mx_idir	idirbit	Mx_ioff	ioffbit

- mx_data: Select data source.
- reset_udb_en: DAC reset enable.
- mx_idir: Mux selection for DAC current direction control.
- idirbit: Register source for DAC current direction.
- mx_off: Mux selection for DAC current off control.
- ioffbit: Register source for DAC current off

Table 3 IDAC8_DATA

Bits	7	6	5	4	3	2	1	0
Value	Data[7:0]							

- Data[7:0]: DAC data register.

DC and AC Electrical Characteristics

DC Characteristics

Parameter	Description	Conditions	Min	Typ	Max	Units
Resolution						
	Output current					
Iout	High	Code = 255, Vdda ≤ 2.7 V, RL 300 Ω	-	-	2.048	mA
	Medium	Code = 255, RL 600 Ω	-	-	256	µA
	Low	Code = 255, RL 600 Ω	-	-	32	µA
INL	Integral non linearity	RL 600 Ω, CL=15 pF	-	-	±1	LSB
DNL	Differential non linearity	RL 600 Ω, CL=15 pF	-	-	±0.5	LSB
Ezs	Zero scale error		-	0	±1	LSB
Eg	Gain error	Uncompensated	-	-	2.5	%
		Temperature compensated	-	-	TBD	%
IDAC_ICC	DAC current low speed mode	Code = 0	-	-	100	µA
IDAC_ICC	DAC current high speed mode	Code = 0	-	-	500	µA

AC Characteristics

Parameter	Description	Conditions	Min	Typ	Max	Units
Fdac	Update rate		-	-	8	Msps
Tsettle	Settling time to 0.5LSB	Full scale transition, 600 Ω load, CL = 15 pF				
	Fast mode	Independent of IDAC range setting (Iout)	-	-	100	ns
	Slow mode	Independent of IDAC range setting (Iout)	-	-	1000	ns

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	Added Sleep/Wakeup and Init/Enable APIs.	To support low power modes, as well as to provide common interfaces to separate control of initialization and enabling of most components.
	Added DMA capabilities file to the component.	This file allows the IDAC8 to be supported by the DMA Wizard tool in PSoC Creator.

© Cypress Semiconductor Corporation, 2009-2010. 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.