Analog Engineer's Circuit

High-Side Current Source LED Biasing Circuit Using Smart DACs

Data Converters

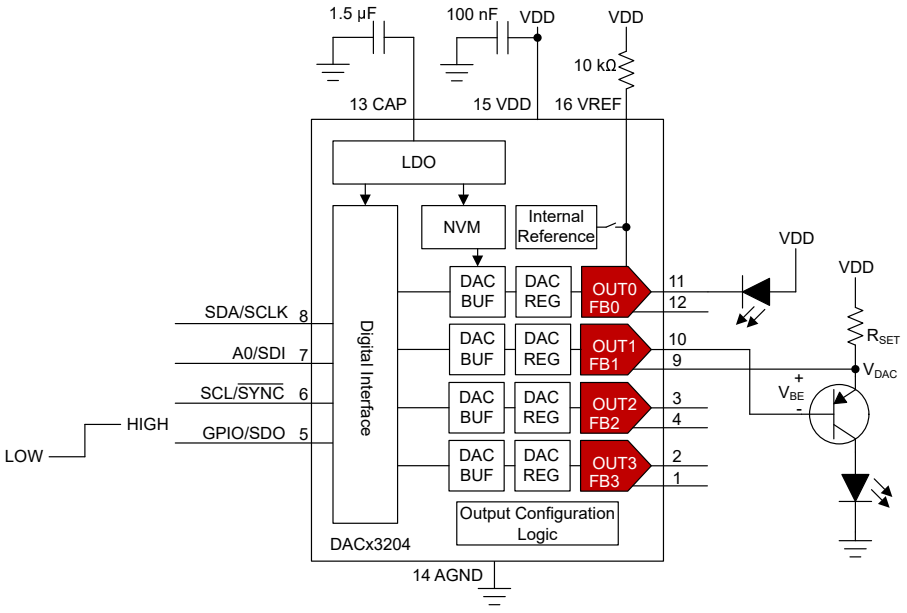
Design Objective

Key Input Parameter	Key Output Signal	Recommended Device
SPI or I^2C communication to program DAC codes 0x000 to 0xFFF, GPI trigger	0 A to 250 µA, and 0 mA to 20 mA LED current	DAC43204 (8 bit), DAC53204 (10 bit), DAC63204 (12 bit)

Objective: *Bias an LED with a high-side current source using a smart DAC.*

Design Description

This design uses a four-channel buffered voltage or current output smart DAC such as the DAC43204, DAC53204, or DAC63204 (DACx3204) to bias a light-emitting diode (LED). The smart DAC can configured in voltage output mode and be connected in a force-sense configuration with a bipolar junction transistor (BJT) in LED biasing applications needing a few milliamps of current as shown on channel 1 of the DACx3204 in the circuit schematic. The DAC sets the collector current of the PNP-type BJTx, such as the 2N2905, and controls the amount of current through the LED by varying the base voltage. The LED is connected between the collector of the BJT and ground. The DAC can be used in current output mode to drive the LED directly with up to 250 µA for low-current LED biasing applications as shown on channel 0 of the DACx3204 in the circuit schematic. The feedback pin (V_{FB}) of the DACx3204 compensates for the base-emitter voltage (V_{BE}) drop and the drift of the BJT when using the BJT configuration. The DACx3204 have a general-purpose input-output (GPIO) pin that can be used to switch the LEDs between two current values, or on and off. All register settings can be saved using the non-volatile memory (NVM) on the smart DAC, meaning that the device can be used without a processor, even after a power cycle. This circuit can be used in applications such as barcode scanners, barcode readers, currency counters, POS printers, optical modules, and appliance lighting.



Design Notes

1. The DACx3204 12-Bit, 10-Bit, and 8-Bit, Quad Voltage and Current Output Smart DACs With Auto-Detected I2C, PMBus™, or SPI data sheet recommends using a 100-nF decoupling capacitor for the VDD pin and a 1.5-µF or greater bypass capacitor for the CAP pin. The CAP pin is connected to the internal low-dropout regulator (LDO). Place these capacitors close to the device pins.
2. When the external reference is not used, the data sheet recommends the VREF pin be connected to VDD through a pullup resistor.
3. The example circuit shows two methods for controlling the LED current. Current can be set via a R_{SET} resistor and varying the base voltage of an external PNP-type BJT with the DACx3204 output, or the LED current can be set using the current output mode of the DACx3204.
   a. To adjust the current LED current with the external BJT, select an R_{SET} resistor and vary the base voltage with the DAC output. R_{SET} is calculated by:

      $$R_{SET} = \frac{V_{SET}}{I_{LED}}$$

      If the V_{SET} voltage range is chosen to be 0 V to 1 V, and the required LED current range is 0 mA to 20 mA, R_{SET} is calculated to be:

      $$R_{SET} = \frac{1\,V}{20\,mA} = 50\,\Omega$$

      The DAC codes for the 10-bit DAC53204 can be calculated by:

      $$Code = \frac{V_{DAC}}{V_{REF}} \times 1024$$

      V_{DAC} is calculated by:

      $$V_{DAC} = VDD - V_{SET}$$

      If a 5-V VDD is used as the reference, the high and low DAC codes become:

      $$Code = \frac{5\,V - 0}{5\,V} \times 1024 = 1024\,d$$

      $$Code = \frac{5\,V - 1V}{5\,V} \times 1024 = 819.2\,d$$

      This is rounded to 1023d and 819d to give a high value of 4.995 V and low value of 3.999 V. This configuration compensates the V_{BE} voltage drop caused by temperature, collector current, and aging of the BJT. A BJT provides a smaller V_{BE} drop as compared to a typical gate-source voltage (V_{GS}) drop of a MOSFET.
   b. The DAC can be used in current output mode to drive the LED directly with up to 250 µA. With the ±250 µA range selected, the DAC codes are calculated by:

      $$Code = \frac{(I_{DAC} - I_{MIN}) \times 256}{I_{MAX} - I_{MIN}}$$

      The high and low DAC53204 codes become:

      $$Code = \frac{(0\,\mu A + 250\,\mu A) \times 256}{250\,\mu A + 250\,\mu A} = 128\,d$$

      $$Code = \frac{(-250\,\mu A + 250\,\mu A) \times 256}{250\,\mu A + 250\,\mu A} = 0\,d$$

4. The slew rate between the high and low DAC codes can be programmed if these two values are stored in the MARGIN-HIGH and MARGIN-LOW DAC registers. The slew time is determined by the settings in the SLEW-RATE and CODE-STEP fields in the DAC-X-FUNC-CONFIG register. The slew time is given by:

$$Slew\ Time = \frac{(MARGIN\_HIGH\_CODE\ -\ MARGIN\_LOW\_CODE + 1)}{CODE\_STEP} \times SLEW\_RATE$$

If the CODE-STEP is set to 1 LSB, and the SLEW-RATE is set to 4 µs per step, the slew time for the voltage configuration becomes:

$$Slew\ Time = \frac{(1023 - 819 + 1)}{1} \times 4\ \mu s = 0.82\ ms$$

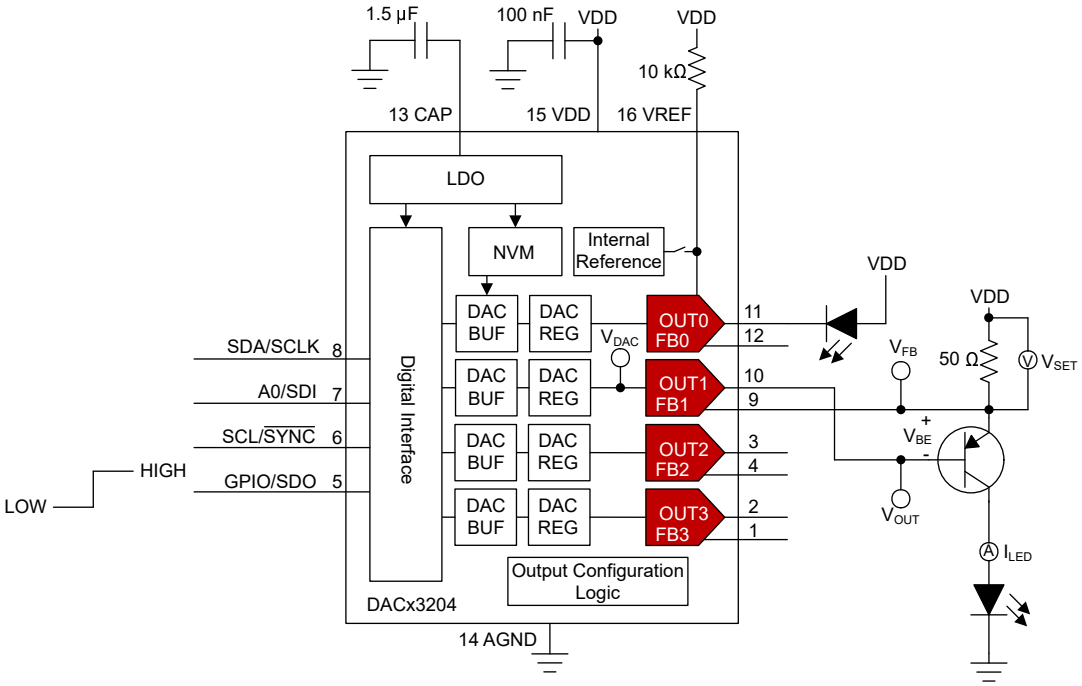
The slew time for the current output configuration becomes:

$$Slew\ Time = \frac{(128 - 0 + 1)}{1} \times 4\ \mu s = 0.516\ ms$$

5. The GPIO pin of the DACx3204 can be used to toggle between the margin high and margin low output value based on the slew time settings set in the DAC-X-FUNC-CONFIG register. A high on the GPI triggers the output to slew to the margin high value. A low on the GPI triggers the output to slew to the margin low value. The register settings to enable the GPIO for this function are described in the Register Settings section.
6. The DACx3204 can be programmed with the initial register settings described in the Register Settings section using I^2C or SPI. The initial register settings can be saved in the NVM by writing a 1 to the NVM-PROG field of the COMMON-TRIGGER register. After programming the NVM, the device loads all registers with the values stored in the NVM after a reset or a power cycle.

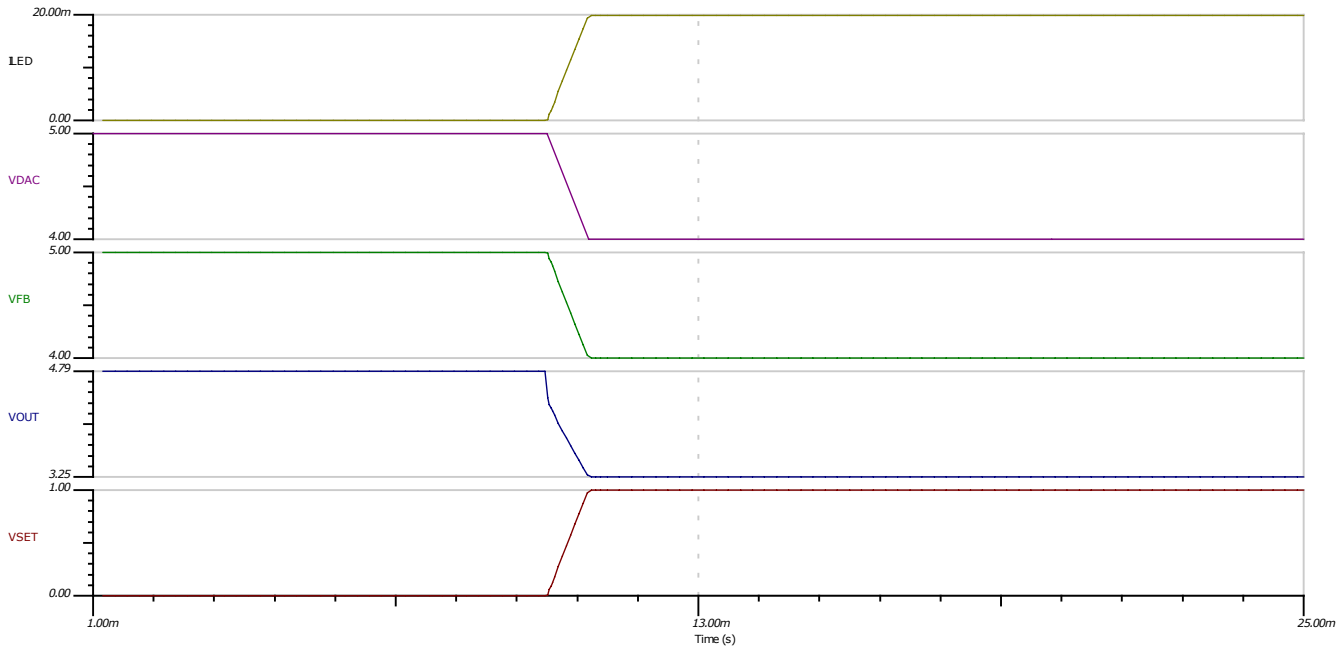
Design Simulations

This schematic is used for the following simulation of the DAC53204.



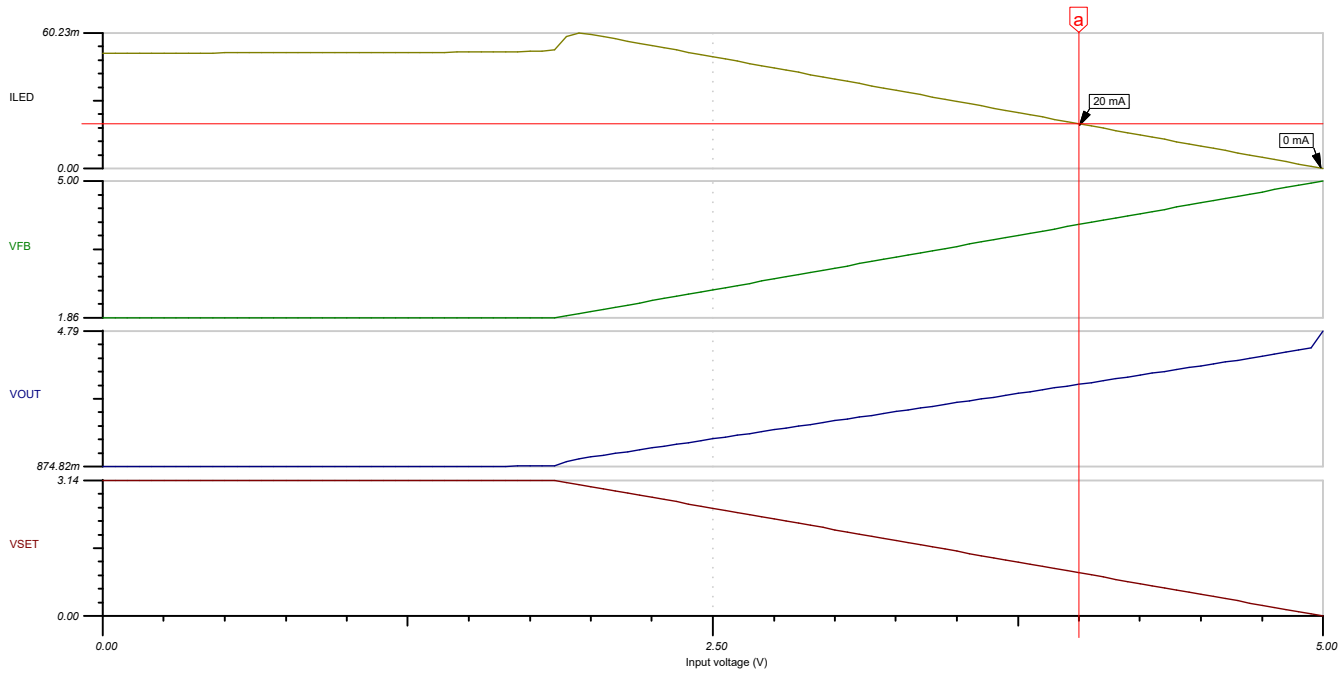
Transient Simulation Results

The simulation shows the LED current when the DAC slews from the margin high to margin low code.



DC Transfer Simulation Results

The simulation shows the LED current vs the DAC output voltage. The base voltage needs to be 0.7 V less than the emitter voltage for a typical PNP-type BJTs to conduct. In this circuit, the chosen 2N2905 transistor starts conducting when the base voltge exceeds 900 mV.



Register Settings

Register Settings for Voltage Output Configuration

Register Address	Register Name	Setting	Description
0x01	DAC-0-MARGIN-HIGH	0xFFC0	[15:4] 0xFFC: 10-bit data left adjusted updates the MARGIN-HIGH code
			[3:0] 0x0: Don't care
0x02	DAC-0-MARGIN-LOW	0xCCC0	[15:4] 0xCCC: 10-bit data left adjusted updates the MARGIN-LOW code
			[3:0] 0x0: Don't care
0x06	DAC-0-FUNC-CONFIG	0x0001	[15] 0b0: Write 0b1 to set DAC-0 clear setting to mid-scale
			[14] 0b0: Write 0b1 to update DAC-0 with LDAC trigger
			[13] 0b0: Write 0b1 to enable DAC-0 to be updated with broadcast command
			[12:11] 0b00: Selects phase for function generator
			[10:8] 0b000: Selects waveform generated by the function generator
			[7] 0b0: Write 0b1 to enable logarithmic slew
			[6:4] 0b000: Selects code-step of 1 LSB
			[3:0] 0b001: Selects slew-rate of 4 µs per step
0x1F	COMMON-CONFIG	0x0FF9	[15] 0b0: Write 0b1 to set window-comparator output to a latching output
			[14] 0b0: Write 0b1 to lock device. Unlock by writing 0b0101 to DEV-UNLOCK field in the COMMON-TRIGGER register
			[13] 0b0: Write 0b1 to set fault-dump read enable at address 0x01
			[12] 0b0: Write 0b1 to enables the internal reference
			[11:10] 0b11: Powers-down VOUT3
			[9] 0b1: Powers-down IOUT3
			[8:7] 0b11: Powers-down VOUT2
			[6] 0b1: Powers-down IOUT2
			[5:4] 0b11: Powers-down VOUT1
			[3] 0b1: Powers-down IOUT1
			[2:1] 0b00: Powers-up VOUT0
			[0] 0b1: Powers-down IOUT0
0x20	COMMON-TRIGGER	0x0002	[15:12] 0b0000: Write 0b0101 to unlock the device
			[11:8] 0b0000: Write 0b1010 to trigger a POR reset
			[7] 0b0: Write 0b1 to trigger LDAC operation if the respective SYNC-CONFIG-X bit in the DAC-X-FUNC-CONFIG register is 1
			[6] 0b0: Write 0b1 to set the DAC registers and outputs to zero-code or mid-code based on the respective CLR-SEL-X bit in the DAC-X-FUNC-CONFIG register
			[5] 0b0: Don't care
			[4] 0b0: Write 0b1 to trigger fault-dump sequence
			[3] 0b0: Write 0b1 to trigger PROTECT function
			[2] 0b0: Write 0b1 to read one row of NVM for fault-dump
			[1] 0b1: Write 0b1 to store applicable register settings to the NVM
			[0] 0b0: Write 0b1 to reload applicable registers with existing NVM settings

Register Settings for Voltage Output Configuration (continued)

Register Address	Register Name	Setting	Description
0x24	GPIO-CONFIG	0x0035	[15] 0b0: Write 0b1 to enable glitch filter on GPI
			[14] 0b0: Don't care
			[13] 0b0: Write 0b1 to enable output mode on GPIO pin
			[12:9] 0b0000: STATUS function setting mapped to GPIO as output
			[8:5] 0b0001: Determines channels affected by channel-specific GPI functions
			[4:1] 0b1010: Selects GPI to trigger margin high and low
			[0] 0b1: Enables input mode for GPIO pin

Register Settings for Current Output Configuration

Register Address	Register Name	Setting	Description
0x01	DAC-0-MARGIN-HIGH	0x8000	[15:4] 0x800: 8-bit data left adjusted updates the MARGIN-HIGH code
			[3:0] 0x0: Don't care
0x02	DAC-0-MARGIN-LOW	0x0000	[15:4] 0x000: 8-bit data left adjusted updates the MARGIN-LOW code
			[3:0] 0x0: Don't care
0x06	DAC-0-FUNC-CONFIG	0x0001	[15] 0b0: Write 0b1 to set DAC-0 clear setting to mid-scale
			[14] 0b0: Write 0b1 to update DAC-0 with LDAC trigger
			[13] 0b0: Write 0b1 to enable DAC-0 to be updated with broadcast command
			[12:11] 0b00: Selects phase for function generator
			[10:8] 0b000: Selects waveform generated by the function generator
			[7] 0b0: Write 0b1 to enable logarithmic slew
			[6:4] 0b000: Selects code-step of 1 LSB
			[3:0] 0b001: Selects slew-rate of 4 µs per step
0x1F	COMMON-CONFIG	0x0FFE	[15] 0b0: Write 0b1 to set window-comparator output to a latching output
			[14] 0b0: Write 0b1 to lock device. Unlock by writing 0b0101 to DEV-UNLOCK field in the COMMON-TRIGGER register
			[13] 0b0: Write 0b1 to set fault-dump read enable at address 0x01
			[12] 0b0: Write 0b1 to enables the internal reference
			[11:10] 0b11: Powers-down VOUT3
			[9] 0b1: Powers-down IOUT3
			[8:7] 0b11: Powers-down VOUT2
			[6] 0b1: Powers-down IOUT2
			[5:4] 0b11: Powers-down VOUT1
			[3] 0b1: Powers-down IOUT1
			[2:1] 0b11: Powers-down VOUT0
			[0] 0b0: Powers-up IOUT0

Register Settings for Current Output Configuration (continued)

Register Address	Register Name	Setting	Description
0x20	COMMON-TRIGGER	0x0002	[15:12] 0b0000: Write 0b0101 to unlock the device
			[11:8] 0b0000: Write 0b1010 to trigger a POR reset
			[7] 0b0: Write 0b1 to trigger LDAC operation if the respective SYNC-CONFIG-X bit in the DAC-X-FUNC-CONFIG register is 1
			[6] 0b0: Write 0b1 to set the DAC registers and outputs to zero-code or mid-code based on the respective CLR-SEL-X bit in the DAC-X-FUNC-CONFIG register
			[5] 0b0: Don't care
			[4] 0b0: Write 0b1 to trigger fault-dump sequence
			[3] 0b0: Write 0b1 to trigger PROTECT function
			[2] 0b0: Write 0b1 to read one row of NVM for fault-dump
			[1] 0b1: Write 0b1 to store applicable register settings to the NVM
			[0] 0b0: Write 0b1 to reload applicable registers with existing NVM settings
0x24	GPIO-CONFIG	0x0035	[15] 0b0: Write 0b1 to enable glitch filter on GPI
			[14] 0b0: Don't care
			[13] 0b0: Write 0b1 to enable output mode on GPIO pin
			[12:9] 0b0000: STATUS function setting mapped to GPIO as output
			[8:5] 0b0001: Determines channels affected by channel-specific GPI functions
			[4:1] 0b1010: Selects GPI to trigger margin high and low
			[0] 0b1: Enables input mode for GPIO pin

Pseudo Code Example

The following shows a pseudo code sequence to program the initial register values to the NVM of the DAC53204. The values given here are for the design choices made in the Design Notes.

Pseudo Code Example for Voltage Output Configuration

```
1:  //SYNTAX: WRITE <REGISTER NAME (Hex code)>, <MSB DATA>, <LSB DATA>
2:  //Configure GPI for margin high/low trigger
3:  WRITE GPIO-CONFIG(0x24), 0x00, 0x35
4:  //With 16-bit left alignment 0x3FF becomes 0xFFC0
5:  WRITE DAC-0-MARGIN-HIGH(0x01), 0xFF, 0xC0
6:  //With 16-bit left alignment 0x333 becomes 0xCCC0
7:  WRITE DAC-0-MARGIN-LOW(0x02), 0xCC, 0xC0
8:  //Set the CODE-SETP to 1 LSB and SLEW-RATE to 4 µs/step
9:  WRITE DAC-0-FUNC-CONFIG(0x06), 0x00, 0x01
10: //Power-up voltage output on channel 0, internal reference disabled
11: WRITE GENERAL_CONFIG(0x1F), 0x0F, 0xF9
12: //Save settings to NVM
13: WRITE COMMON-TRIGGER(0x20), 0x00, 0x02
```

Pseudo Code Example for Current Output Configuration

```
1:  //SYNTAX: WRITE <REGISTER NAME (Hex code)>, <MSB DATA>, <LSB DATA>
2:  //Configure GPI for deep-sleep trigger and enable deep-sleep function
3:  WRITE GPIO-CONFIG(0x24), 0x00, 0x35
4:  //With 16-bit left alignment 0x80 becomes 0x8000
5:  WRITE DAC-0-MARGIN-HIGH(0x01), 0x80, 0x00
6:  //Write DAC0 margin low code
7:  WRITE DAC-0-MARGIN-LOW(0x02), 0x00, 0x00
8:  //Set the CODE-SETP to 1 LSB and SLEW-RATE to 4 µs/step
9:  WRITE DAC-0-FUNC-CONFIG(0x06), 0x00, 0x01
10: //Power-up current output on channel 0, internal reference disabled
11: WRITE GENERAL_CONFIG(0x1F), 0x0F, 0xFE
12: //Save settings to NVM
13: WRITE COMMON-TRIGGER(0x20), 0x00, 0x02
```

Design Featured Devices

Device	Key Features	Link
DAC43204	4-channel, 8-bit, VOUT and IOUT smart DAC with I2C, SPI and PWM	DAC43204
DAC53204	4-channel, 10-bit, VOUT and IOUT smart DAC with I2C, SPI and PWM	DAC53204
DAC63204	4-channel, 12-bit, VOUT and IOUT smart DAC with I2C, SPI and PWM	DAC63204

Find other possible devices using the Parametric search tool.

Design References

See *Analog Engineer's Circuit Cookbooks* for TI's comprehensive circuit library.

Additional Resources

- Texas Instruments, DAC63204 Evaluation Module
- Texas Instruments, DAC63204 EVM User's Guide
- Texas Instruments, Precision Labs - DACs

For direct support from TI Engineers, use the E2E community:

e2e.ti.com

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265