PSoC Filter Block Tutorial

Eric Ponce
eaponce@mit.edu

May 5, 2017

1 Introduction

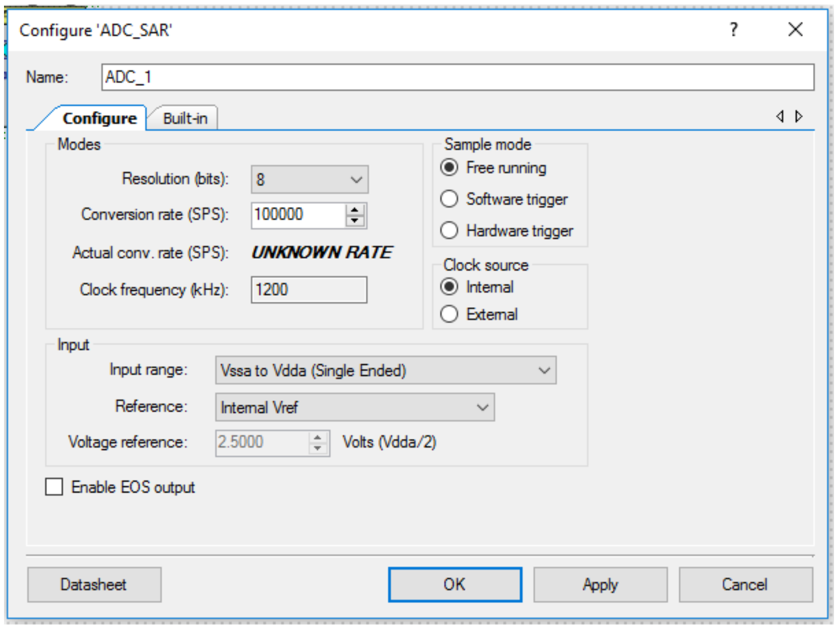
The goal of this tutorial is to take you through the steps to create a PSoC creator project that implements a digital low-pass filter almost entirely in hardware blocks. The PSoC 3 and 5 contain a DSP-esque hardware block that allows developers to implement high-speed processing of data without bogging down the core. Cypress created a drag-and-drop version of this block called a "Filter" that automatically generates the necessary DSP code to implement a few different types of filters.

2 Schematic Setup

In this tutorial, we will create a digital FIR (finite impulse response) low pass filter with a cutoff frequency of 3KHz. An ADC will sample the incoming signal at 100Ksps. A direct memory access (DMA) channel will be used to transfer that sample data to the filter block where it will undergo processing and when the filter block finishes its computation, another DMA channel will be used to move the filtered data over to an 8-bit DAC.

To start, create a new project with the proper device setting and an empty schematic

Next, open the .cysch file and drag in an ADC_SAR block and configure it as shown below. Firstly, because the DAC is only 8-bits, we will set the resolution of the ADC to the same. Then, we want to set the conversion rate to a value high enough to prevent aliasing of audio-frequency signals (less than 20KHz). Finally, we expand the input range to its maximum single ended value (Vssa to Vdda) and set the reference to internal.


Configure 'ADC_SAR'

Name: ADC_1

Modes

Resolution (bits): 8

Conversion rate (SPS): 100000

Actual conv. rate (SPS): UNKNOWN RATE

Clock frequency (kHz): 1200

Sample mode: [✓ Free running] Software trigger Hardware trigger

Clock source: [✓ Internal] External

Input

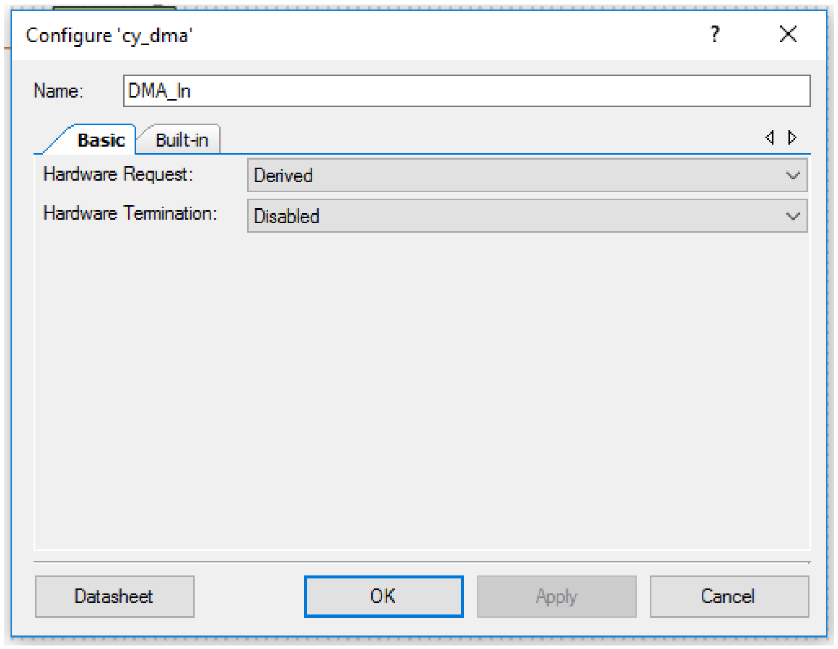
Input range: Vssa to Vdda (Single Ended)

Reference: Internal Vref

Voltage reference: 2.5000 Volts (Vdda/2)

☐ Enable EOS output

In order to get data from the ADC to the filter, we will use a DMA block. Add a DMA block to the schematic and configure it as shown. We want PSoC creator to automatically derive the correct functionality for the hardware request pin of the DMA block, which will be connected to the end-of-conversion signal of the ADC.

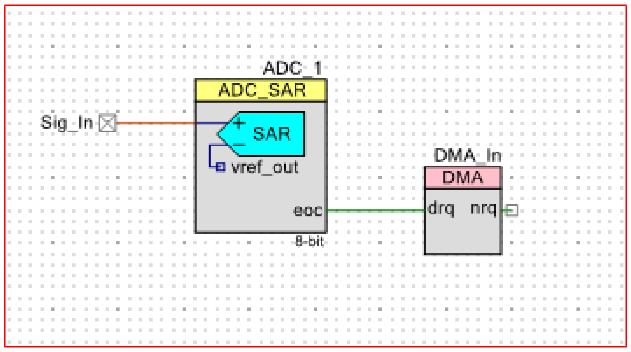

Configure 'cy_dma'

Name: DMA_In

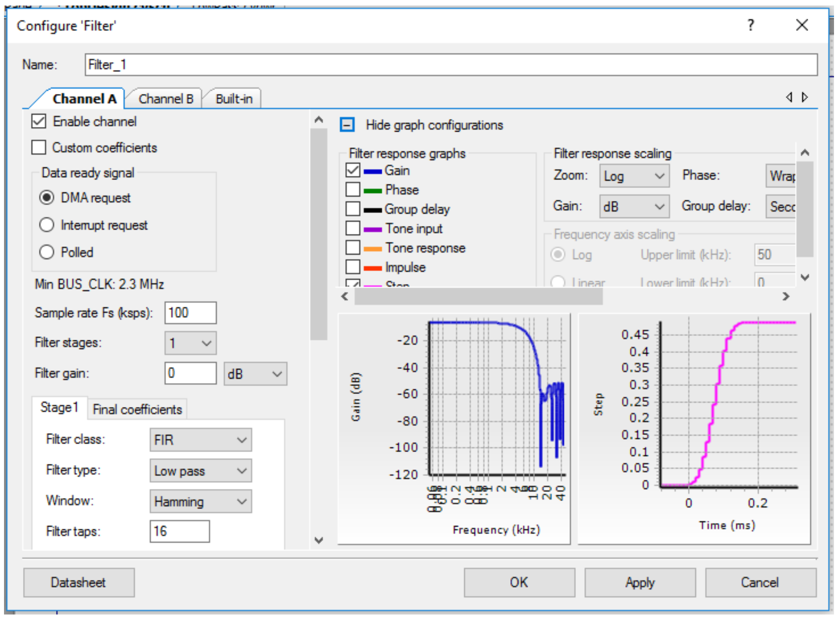
Hardware Request: Derived

Hardware Termination: Disabled

After setting up the ADC, add an analog pin to the schematic and wire the three blocks up as shown below.



Now that we have taken care of sampling the input signal and transferring the data to the filter, we must setup the filter block and the DMA block that will take data from it to the DAC. Drag in a filter block and double-click to edit its settings. First, enable channel A and set the "Data ready signal" to "DMA request". Next, ensure the sample rate matches that of your ADC.


Configure 'Filter'

Name: Filter_1

Channel A | Channel B | Built-in

[✓] Enable channel

[ ] Custom coefficients

Data ready signal

(●) DMA request

( ) Interrupt request

( ) Polled

Min BUS_CLK: 2.3 MHz

Sample rate Fs (ksps): 100

Filter stages: 1

Filter gain: 0 dB

Stage1 | Final coefficients

Filter class: FIR

Filter type: Low pass

Window: Hamming

Filter taps: 16

Hide graph configurations

Filter response graphs

[✓] Gain

[ ] Phase

[ ] Group delay

[ ] Tone input

[ ] Tone response

[ ] Impulse

[✓] Step

Filter response scaling

Zoom: Log

Phase: Wrap

Gain: dB

Group delay: Seco

Frequency axis scaling

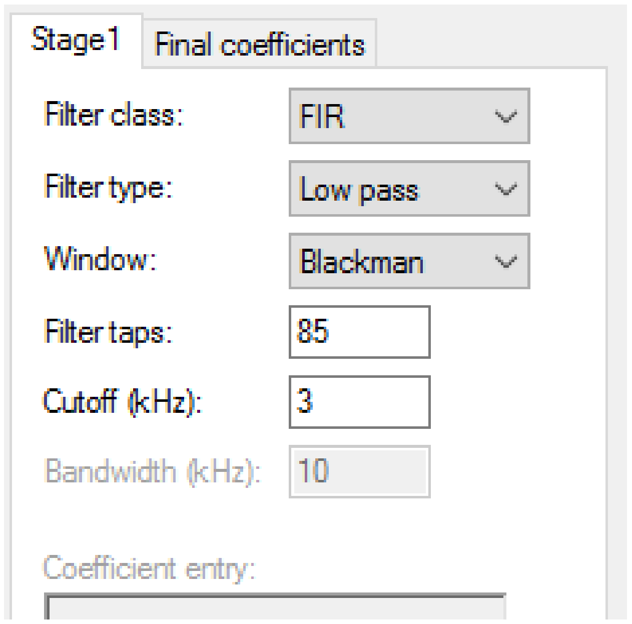
(●) Log

( ) Linear

Upper limit (kHz): 50

Lower limit (kHz): 0

After the baseline settings, scroll the left panel down to begin modifying the filter parameters. For the purposes of this tutorial, a FIR low-pass filter with a Blackman windowing and 85 taps was chosen to give a good balance of flat-response at low frequencies and sharp drop off after the cutoff. The settings are shown below.


Stage1 | Final coefficients

Filter class: FIR

Filter type: Low pass

Window: Blackman

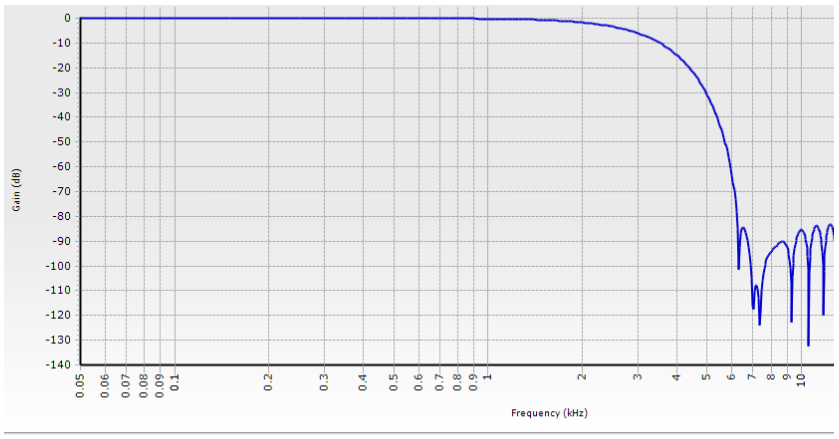
Filter taps: 85

Cutoff (kHz): 3

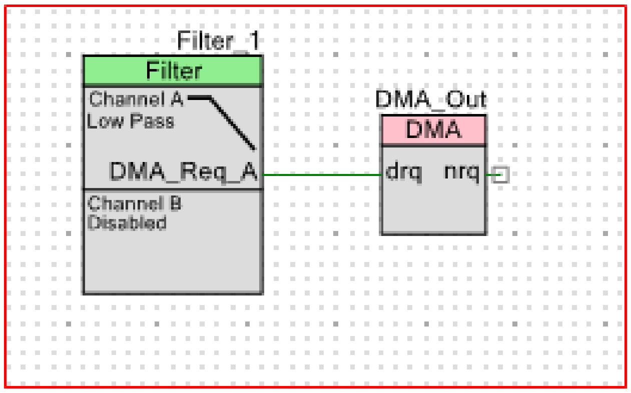
Bandwidth (kHz): 10

Coefficient entry:

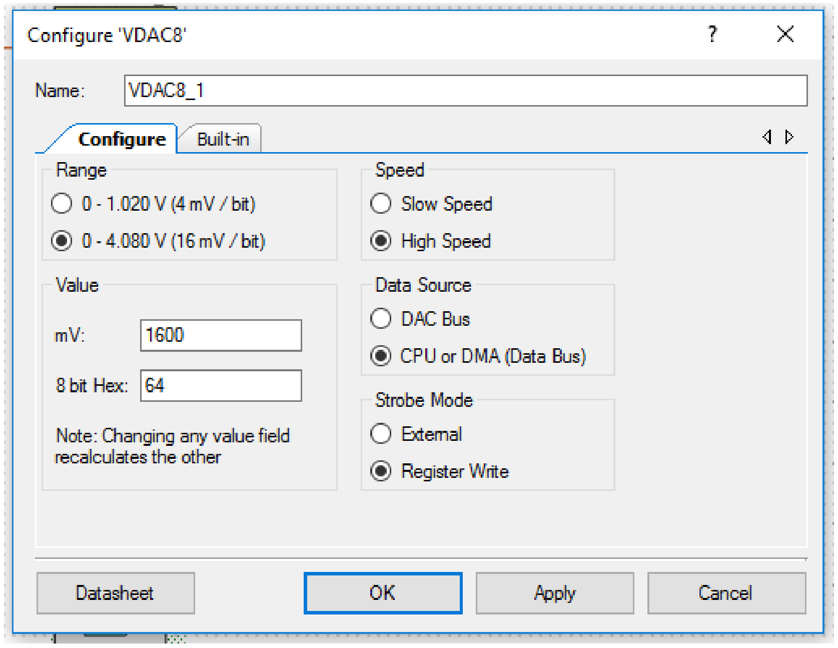
Now, you can look at the right panel and enable or disable different plots to show the characteristics of the filter that are important for your application. Below, the magnitude of the gain as a function of frequency is plotted.



After setting up the filter block, add another DMA block to the schematic and use the same settings as the first DMA block (Don't forget to use a different name though!). Connect the blocks as shown.



The last hardware block to add is the VDAC (or voltage digital-to-analog converter). This peripheral takes 8-bit data and outputs an analog representation. Add one to your schematic and change the settings to those shown below. We will want to put the VDAC into high-speed mode and expand the range.


Configure 'VDAC8'

Name: VDAC8_1

Configure | Built-in

Range
0 - 1.020 V (4 mV / bit) [✓ 0 - 4.080 V (16 mV / bit)]

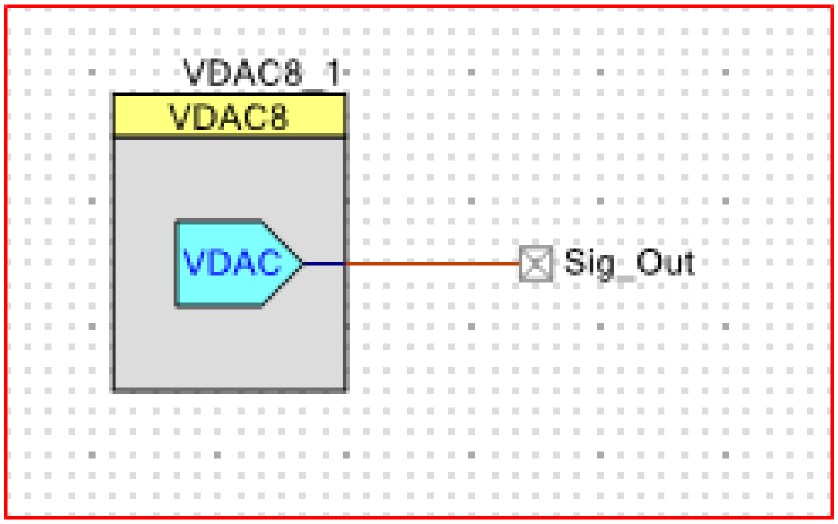
Value
mV: 1600
8 bit Hex: 64
Note: Changing any value field recalculates the other

Speed
Slow Speed [✓ High Speed]

Data Source
DAC Bus [✓ CPU or DMA (Data Bus)]

Strobe Mode
External [✓ Register Write]

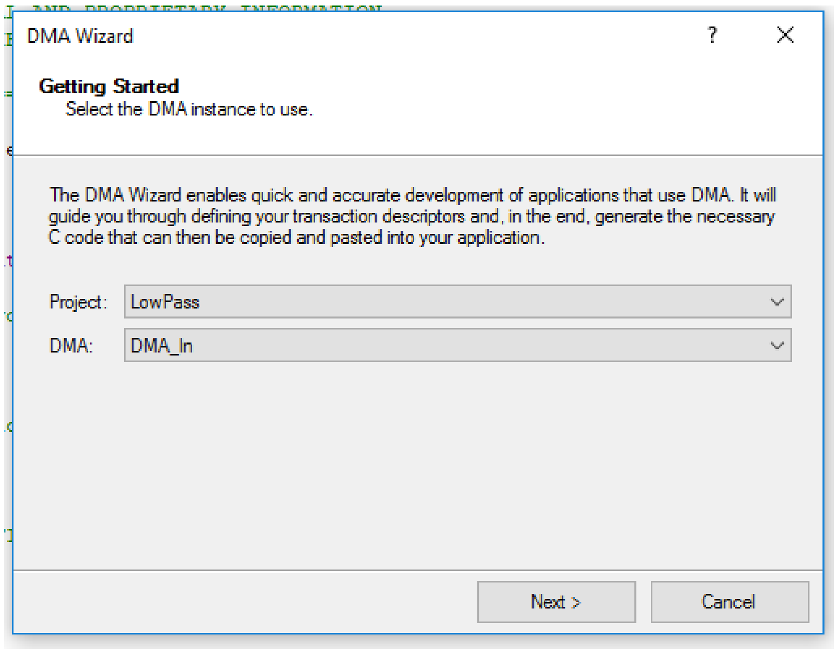
Now add the output analog pin and wire the block up!



3 Software

After setting up the hardware components. Generate the software APIs (Build→Generate Application) and navigate to your main.c file

The DMA hardware requires configuration to setup data transfer channels. This setup involves creating transaction descriptors (TD) that contain information about source/destination addresses and transfer size. TDs can be chained together to create transfers of large contiguous chunks of memory. To generate some of this code we will use PSoC Creator's DMA Wizard. It's a nifty little tool to do a lot of this setup work for you and can be found under the "Navigate" menu bar item. Mimic the first window as shown below to configure DMA_In.



DMA Wizard

Getting Started

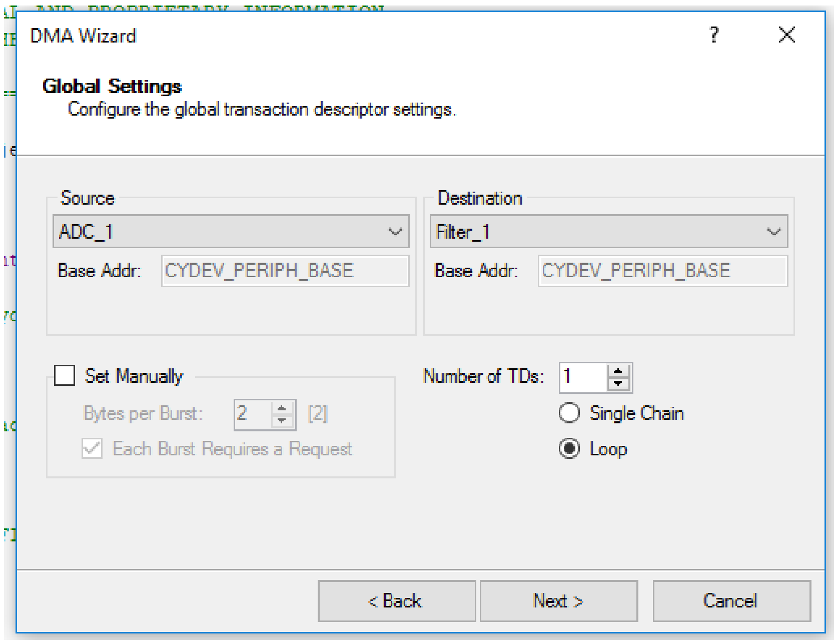
Select the DMA instance to use.

The DMA Wizard enables quick and accurate development of applications that use DMA. It will guide you through defining your transaction descriptors and, in the end, generate the necessary C code that can then be copied and pasted into your application.

Project: LowPass

DMA: DMA_In

The next window allows you to select the source and destination from a list of peripherals and memory. In this case our input is the ADC and our output is the filter. Because we will only be moving from and to one address, we want a singular TD. This TD should be a loop because we want the TD to go back to the same address after the DMA transfer is complete.



DMA Wizard

Global Settings

Configure the global transaction descriptor settings.

Source: ADC_1

Base Addr: CYDEV_PERIPH_BASE

Destination: Filter_1

Base Addr: CYDEV_PERIPH_BASE

☐ Set Manually

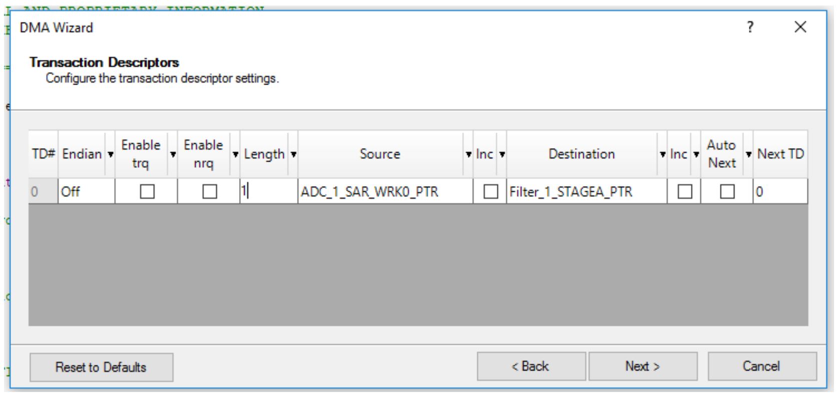
Bytes per Burst: 2 [2]

✓ Each Burst Requires a Request

Number of TDs: 1

Single Chain [✓ Loop]

The final configuration window in the the wizard allows you to further customize the TD. We will want to set the TD length to 1 (byte) and then press next. On the next screen you will be presented with auto generated code to copy into your main() function. Because of the way the DMA wizard works, it defaults to a burst width of 2 bytes (since the filter register width is 24-bits). Thus, in the copied code we need to change the "DMA_In_BYTES_PER_BURST" macro's value from 2 to 1.



DMA Wizard

Transaction Descriptors

Configure the transaction descriptor settings.

TD#	Endian	Enable trq	Enable nrq	Length	Source	Inc	Destination	Inc	Auto Next	Next TD
0	Off	☐	☐	1	ADC_1_SAR_WRK0_PTR	☐	Filter_1_STAGEA_PTR	☐	☐	0

For the output DMA TD, we need to follow a similar process. Open up a new wizard and select the output DMA peripheral. In the initial configuration screen, you will want to select the Filter as your source and the DAC as your destination. Similarly to the input DMA channel, we will want a singular TD configured as a loop. On the last configuration window, you will again want to set the length to 1. Paste the generated code under the code for the input DMA.

Aside from DMA initialization code, we need to initialize all the peripherals we are using (DAC, ADC, Filter). This can be accomplished with the following lines of code:

```
VDAC8_1_Start();
Filter_1_Start();
ADC_1_Start();
ADC_1_StartConvert();
```

Because the filter contains 24-bit input and output registers, we need to tell it to start the filtering process on a write to the least significant byte (LSB) in the input register and after a read from the LSB of the output register. This can be accomplished with a property called "coherency." Coherency allows you, the developer, to set byte alignment for different sized data types. In this case, we will want "Low" coherency so that the filter responds to reads/writes on the LSB of the register. This can be accomplished by adding the following line of code below the Filter start function:

```
Filter_1_SetCoherency(Filter_1_CHANNEL_A, Filter_1_KEY_LOW);
```

4 Hardware

The last step before you are ready to use your new digital filtering hardware, is to select your desired pins in the .cydwr file. If you are using the CY8CKIT-050 evaluation kit, it is recommended that you constrain analog pins to Port E and digital pins to Port D, as the board was designed with that use case in mind. Once you have selected your pins and built your code to verify that there is no typos or problems, you should be able to flash your device! Some notes to make sure you don't accidentally damage your evaluation kit:

- The ADC input range is Vssa to Vdda. Thus, ensure your signal does not exceed the power rails supplied to your PSoC! If necessary, you should be able to capacitively couple your input and output signals to move around your DC bias.
- Analog signals should be referenced to Vssa (Analog ground) to ensure the best immunity to noise caused by the PSoC's digital circuitry. Furthermore, keeping your wires short will greatly improve filter performance!
- Do NOT load your output DAC with non-capacitive loads. The VDAC is essentially a modified current DAC so it likely will not function correctly and may damage the PSoC. Use an op-amp unity-gain buffer if necessary to drive a resistive load (the PSoC contains internal op-amps if necessary).

5 Troubleshooting

- If you do not see output from your DAC, it may be useful to assign pins to the DMA request lines in the schematic. This will allow you to ensure that the DMA is working properly.
- If you find that the DMA is working improperly, please reference the available Creator project. It is likely an improperly set up DMA TD or the filter coherency was not set.

6 Extensions

- The DMA can easily be changed using the DMA wizard to take data from or to SRAM or other peripherals. This may allow you to process saved data or store processed data in a fast and efficient manner.
- The filter block has 2 channels that can be used to implement more than one filter. This can be used to process two separate streams of data or perform multi-stage filtering.
- Rather than using DMA to move data from the filter to the VDAC, an interrupt may be used which allows for more specific control of output

data. This comes at the cost of using up more core processing resources and can cause significant phase shifts between the input and output.

- If a digital filter is designed in MATLAB or some other computational software, the coefficients of the taps can be input manually rather than allowing PSoC creator to generate them for you