

SC/CT block superprimitive!

This is the way primitive components should be made. This documents and distributes the SC/CT block super primitive. This component is designed to be imported into your workspace and provide you with a template component that you can start to use and modify immediately. The component brings out all connections for the hardware and makes them available for connections on the top level schematic. The component includes pre-made header and C files that include all definitions for all registers needed to configure the SC/CT block in any way possible. The header file also includes bit masks, modes, and shifts for all associated registers, making writing code for the super primitive incredibly easy. The C file comes with pre made stub init, enable and start APIs with critical clock and pump configuration already taken care of. The C file also includes a commented out example init function for enabling the DeltaSigma modulator mode of the SC/CT block as reference for creating your own init function.

Sounds good you say? I'm not done! The component also comes packaged with a complete component debug file, allowing you to see, and reconfigure your super primitive component in the debugger with insane ease. This is by far one of the most useful features of the super primitive.

But wait! There's more! The super primitive also comes pre-packed with a DMA capability file, giving you easy access to all registers associated with the SC/CT block in the DMA wizard tool. Why would you want that? I DON'T KNOW, BUT I GAVE IT TO YOU ANYWAY!

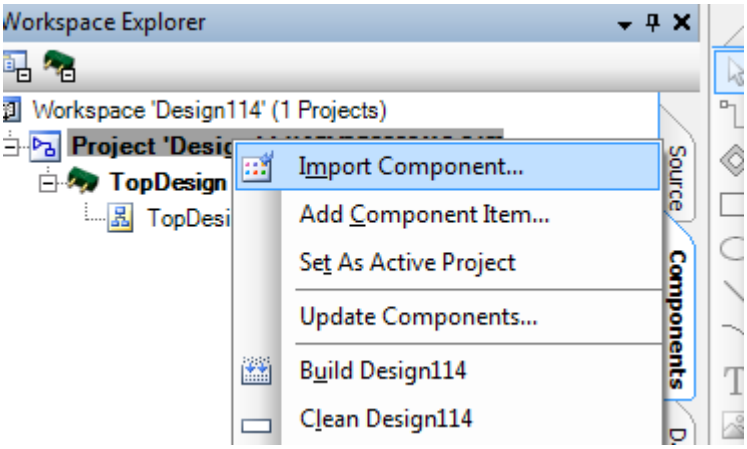
The super primitive eliminates 99% of the mundane work when creating a component, leaving you to design, create, and explore the limitless capabilities of PSoC. Feel free to use the super primitive as the base for any new component you wish to build on the SC/CT blocks. If you find any mistakes, please let me know so I can update the super primitive.

Attached File Summary:

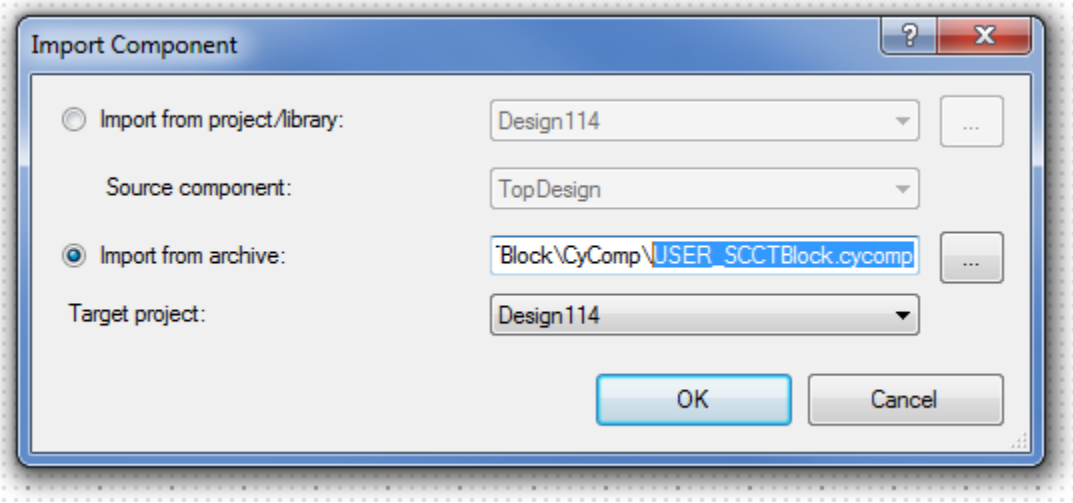
File # 2 is a ZIP file of the super primitive component. To use the file, download it and change the extension from .pdf to .zip. The cycomp file is contained in this zip file.

Details:

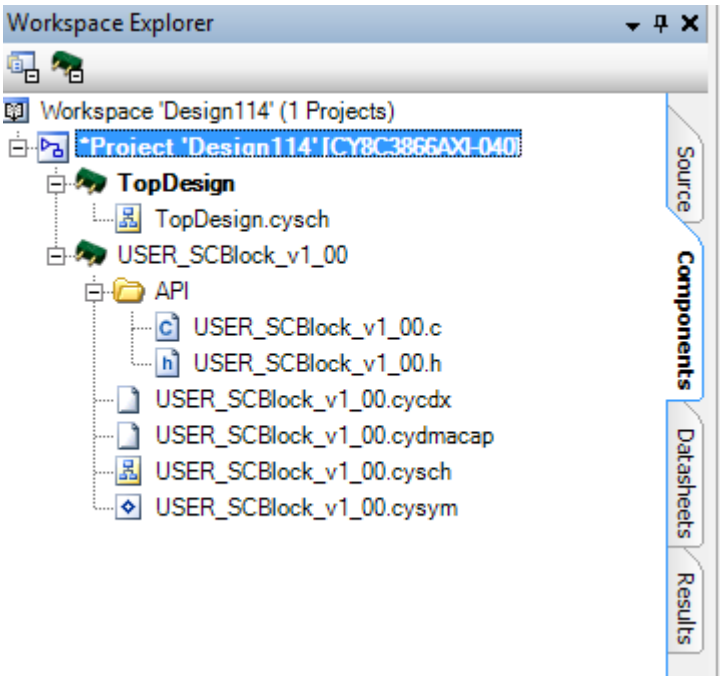
To get started with the super primitive, open a project, navigate to the component tab in the workspace explorer, right click on you project and select “Import Component”



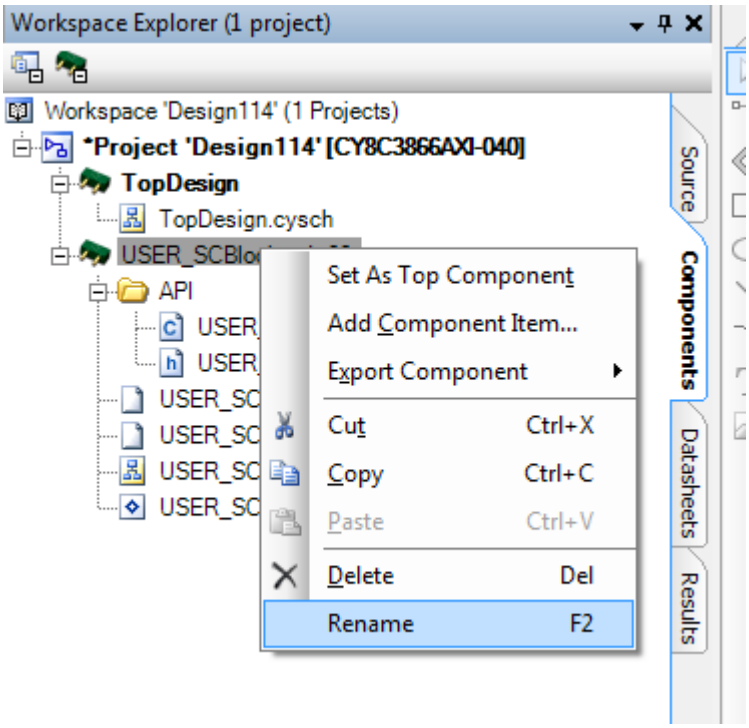
When the dialog box pops up, select “Import from archive” and navigate to the locations of the “USER_SCCTBlock.cycomp” file. Click OK



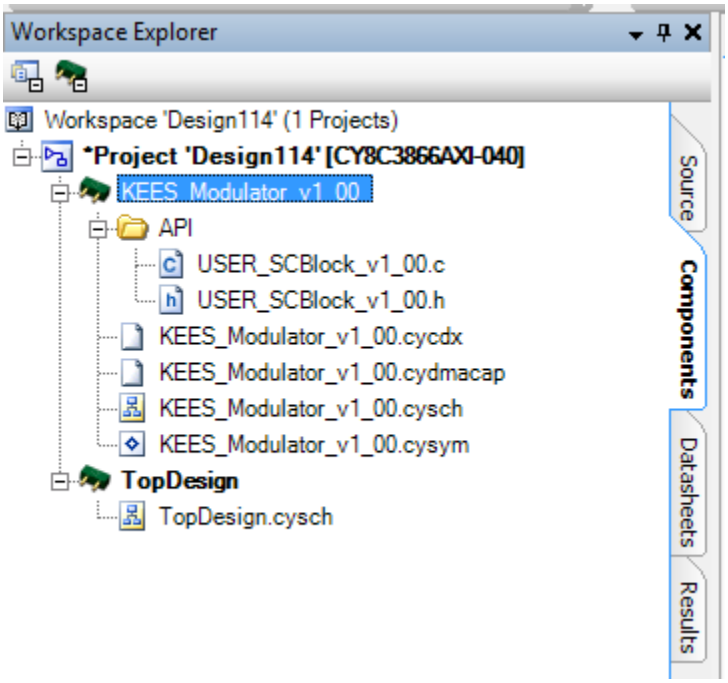
Your workspace explorer will now contain all the files for your component



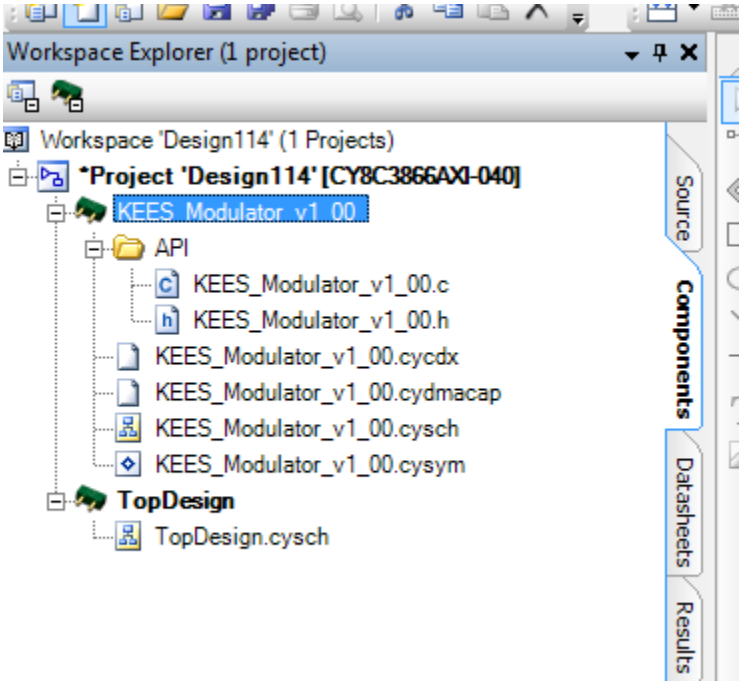
First thing you should do is rename the component by right clicking on the component and selecting “Rename.”



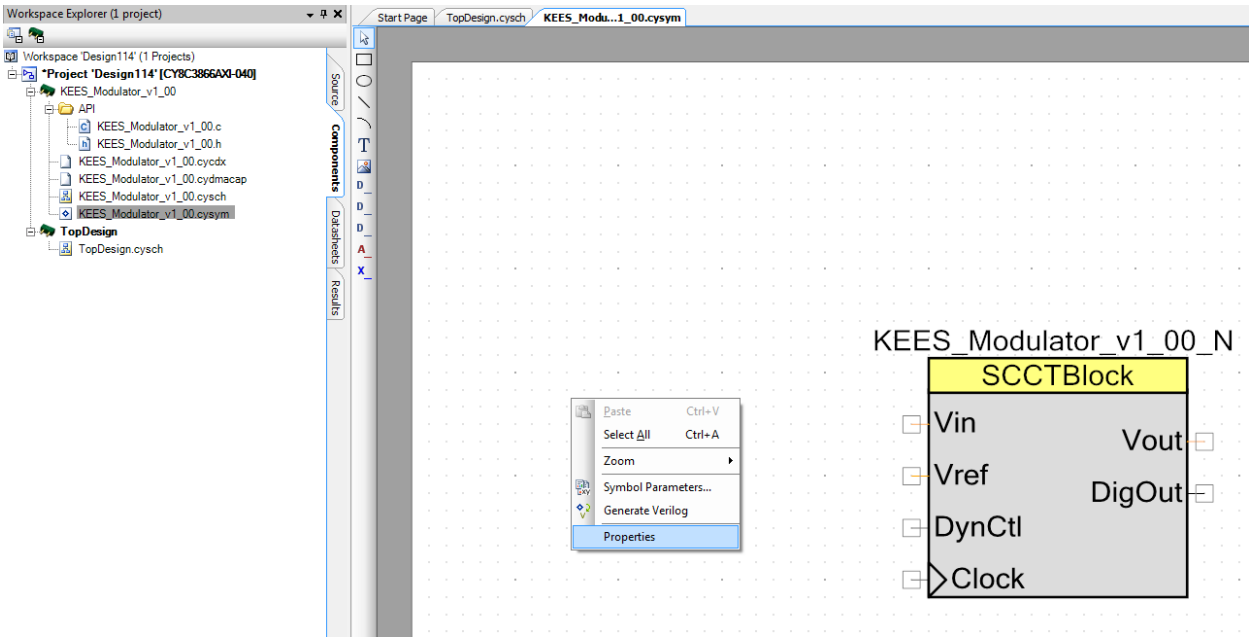
Rename it to something with your initials, or your name instead of USER and either leave the rest alone, or give a name that reflects your final purpose. Make sure to leave the _v1_00 appended to the end so that you can make new versions later if you choose.

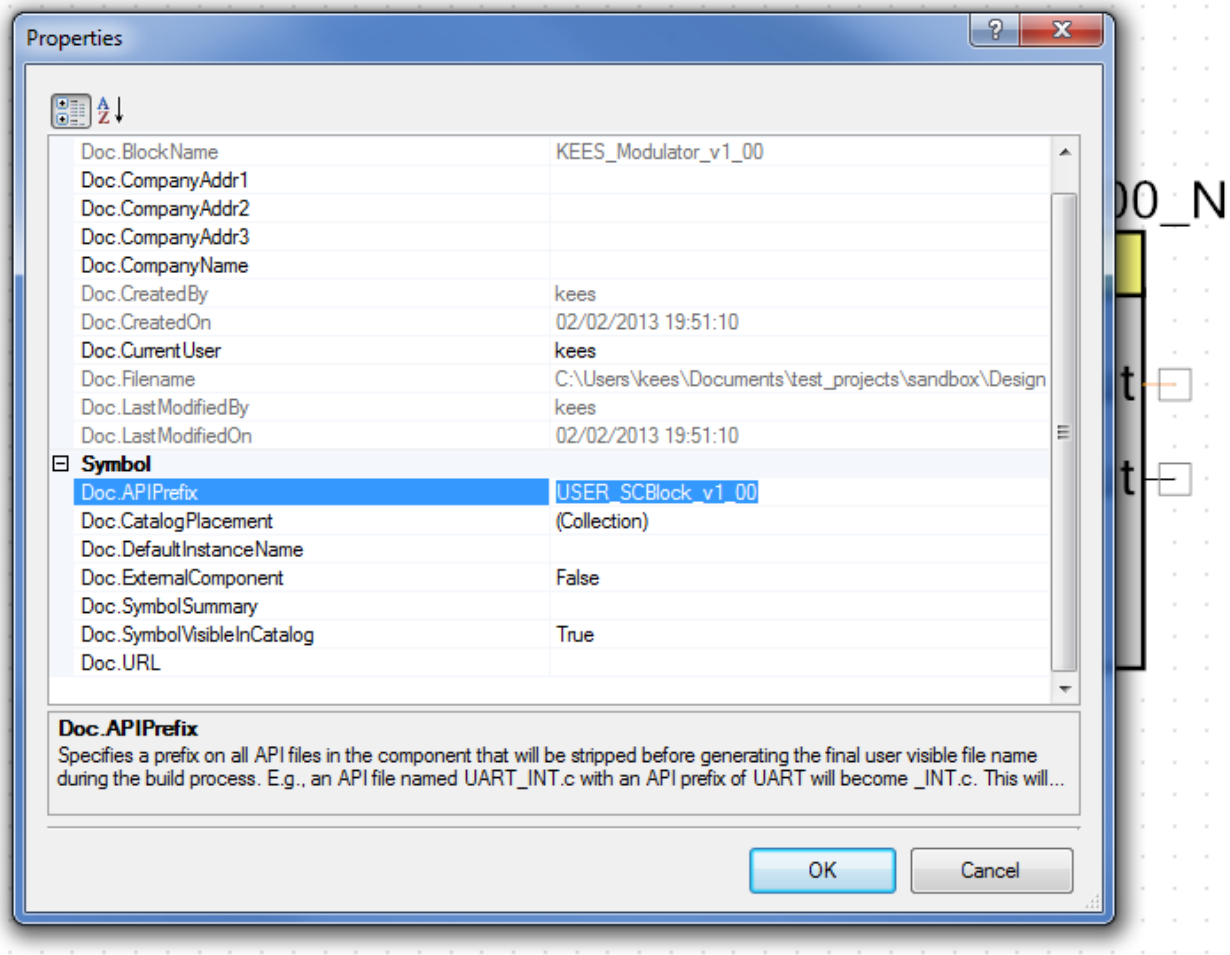


Notice how the API files did not change. You need to manually change them to match the new component name by right clicking on them and selecting "Rename"

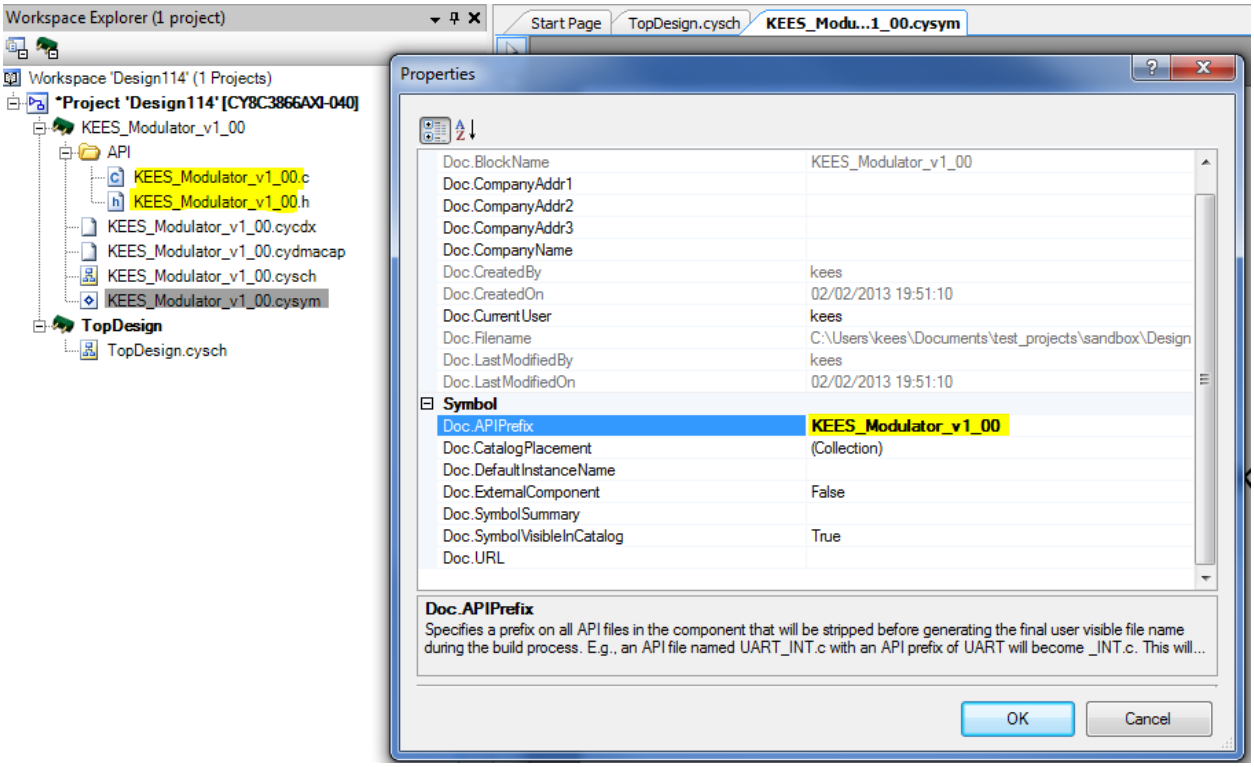


Now this step is important. Open the **.cysym** file, right click in the symbol area (not on the symbol itself) and select "Properties"

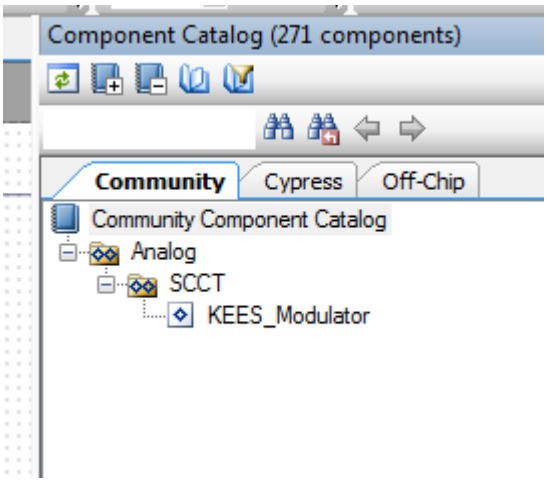

Workspace Explorer (1 project)
Workspace 'Design114' (1 Projects)
*Project 'Design114' [CY8C3866AXI-040]
KEES_Modulator_v1_00
API
KEES_Modulator_v1_00.c
KEES_Modulator_v1_00.h
KEES_Modulator_v1_00.cycdx
KEES_Modulator_v1_00.cydmacap
KEES_Modulator_v1_00.cysch
KEES_Modulator_v1_00.cysym
TopDesign
TopDesign.cysch
Start Page
TopDesign.cysch
KEES_Modu...1_00.cysym
Source
Components
Datasheets
Results
Paste Ctrl+V
Select All Ctrl+A
Zoom
Symbol Parameters...
Generate Verilog
Properties
KEES_Modulator_v1_00_N
SCCTBlock
Vin
Vref
DynCtl
Clock
Vout
DigOut


Properties
Doc.BlockName KEES_Modulator_v1_00
Doc.CompanyAddr1
Doc.CompanyAddr2
Doc.CompanyAddr3
Doc.CompanyName
Doc.CreatedBy kees
Doc.CreatedOn 02/02/2013 19:51:10
Doc.CurrentUser kees
Doc.Filename C:\Users\kees\Documents\test_projects\sandbox\Design
Doc.LastModifiedBy kees
Doc.LastModifiedOn 02/02/2013 19:51:10
Symbol
Doc.APIPrefix USER_SCBlock_v1_00
Doc.CatalogPlacement (Collection)
Doc.DefaultInstanceName
Doc.ExternalComponent False
Doc.SymbolSummary
Doc.SymbolVisibleInCatalog True
Doc.URL
Doc.APIPrefix
Specifies a prefix on all API files in the component that will be stripped before generating the final user visible file name during the build process. E.g., an API file named UART_INT.c with an API prefix of UART will become _INT.c. This will...
OK
Cancel

Change the Doc.APIPrefix to match the renamed API files with.

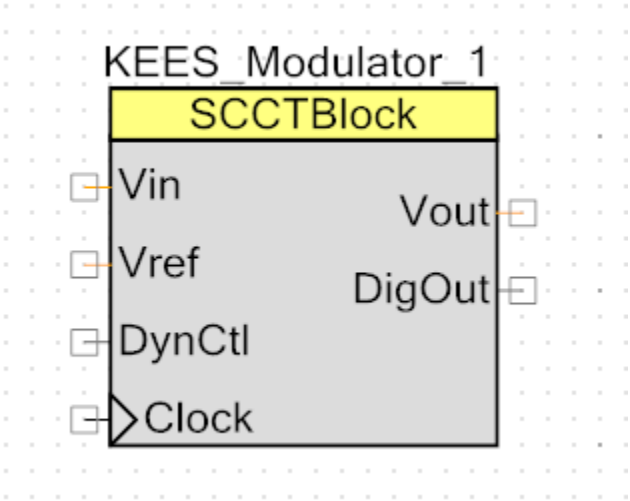


And that concludes all the component customization that you MUST do. Any further customization is optional. At this point, your component will be located in the “Community/Analog/SCCT/…” library tab.

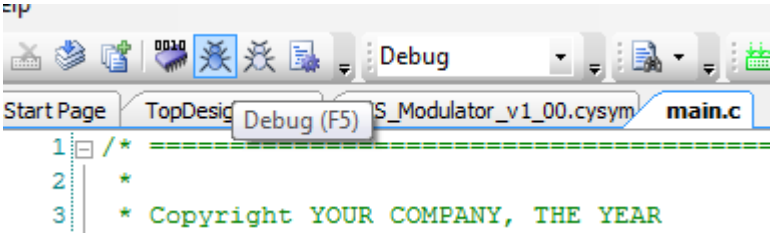


Getting started with you super primitve

Drag one onto your schematic and call the Start() API in main. The Start() API is just an incomplete stub, configuring the 3 configuration registers with zero so it will compile. You are not required to connect any pin. ***Hit debug to see one of the greatest features of the super primitive.***

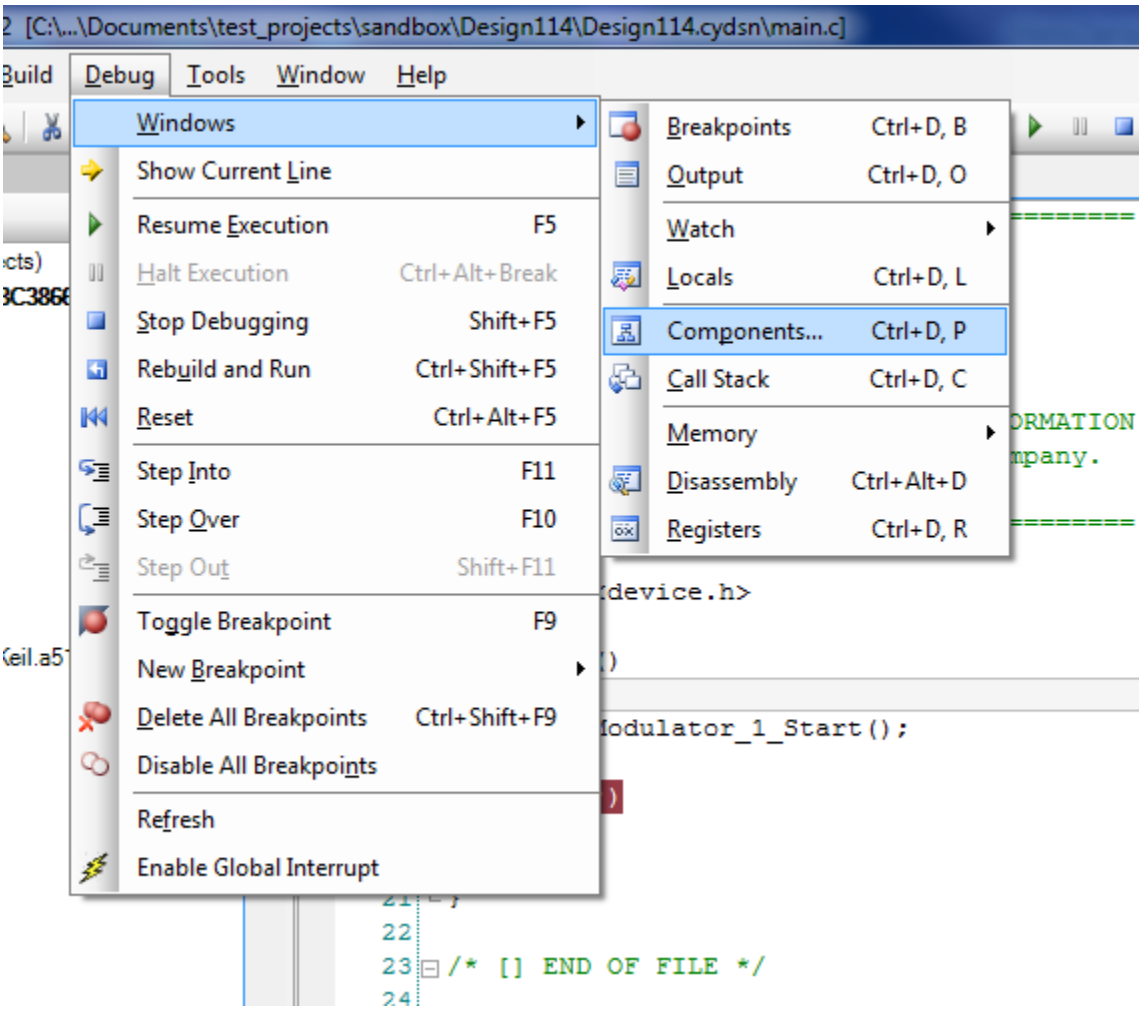



```
#include <device.h>

void main()
{
    KEES_Modulator_Start();

    for(;;)
    {
    }
}
```

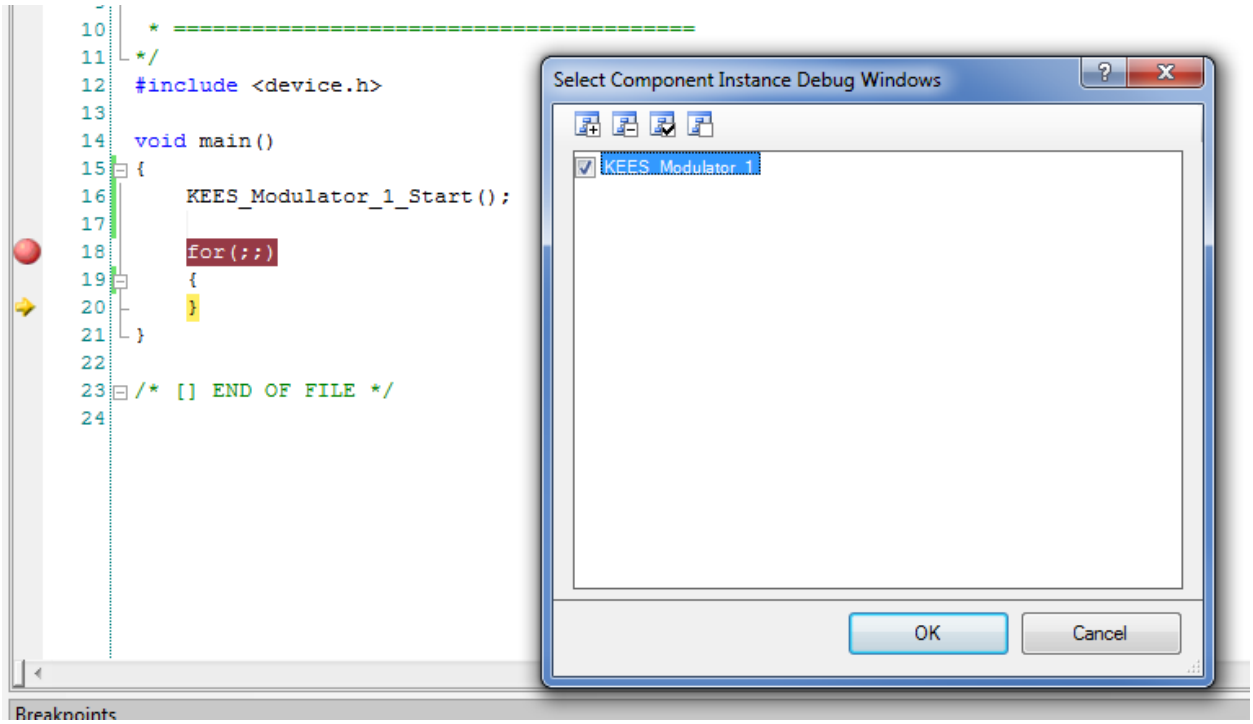


Component Debug features

When the debugger starts, go to Debug->Windows->Components



This will bring up the component debug selection window. Check the box next to your component and click OK.



A new window in the debugger will appear filled with all of the registers associated with your SC/CT block



Note! There a few bugs in Creator 2.2 related to the component debugger.

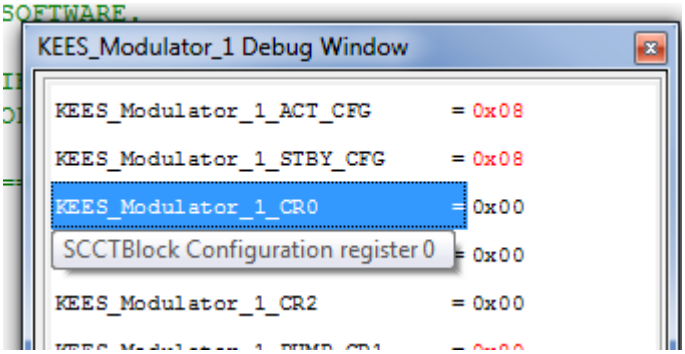
1. ***When the window is displayed, all the values will be zero. You have to refresh the debugger to see the current contents of the memory. Select Debug->Refresh to update the window. Sadly, it seems that you need to refresh the window any time you wish to see a change. Hitting a breakpoint, stepping code or anything else that should cause it to update automatically does not seem to work.***
2. ***The debugger will forget that you have a component debug window open if you stop and restart the debugger. For now, you have to re-enable the window every time you start a new debugging session.***
3. ***A radix change will not show up (left click on register, then right click and select "Radix") unless you refresh the debugger. You***

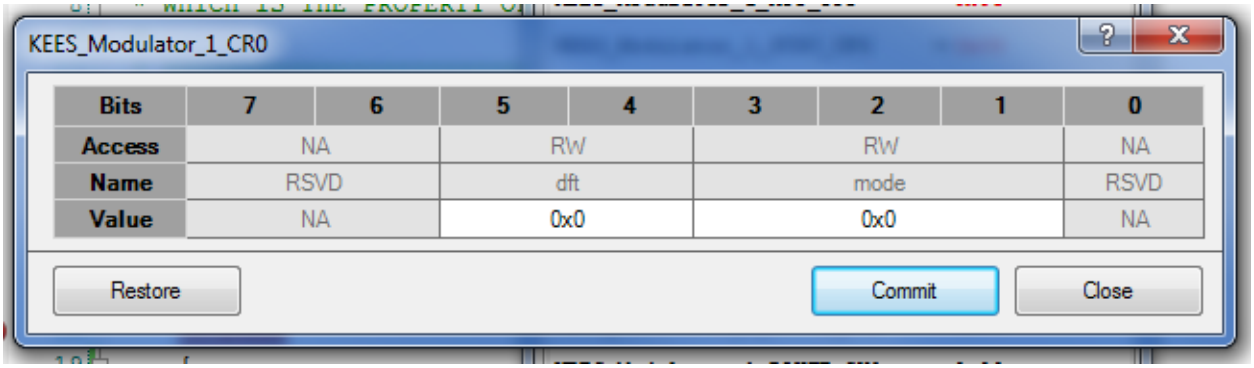
must also have a register selected when you right click, or you will get an unhandled exception. If this happens, click "continue" as it does not seem to cause the debugger to crash.

4.) ***When changing the radix for one register, all registers will change to the new radix.***

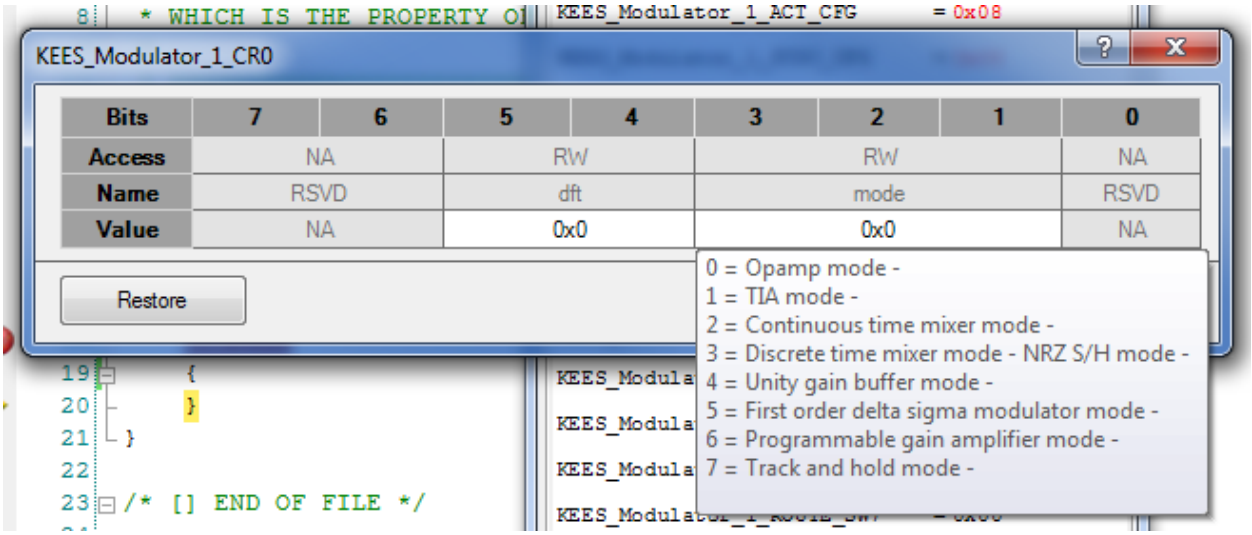
5.) ***CDT 144250 was filed on all of these problems.***

Double clicking on any register will bring up detailed information on each bit field in that register, as well as the current configuration information.

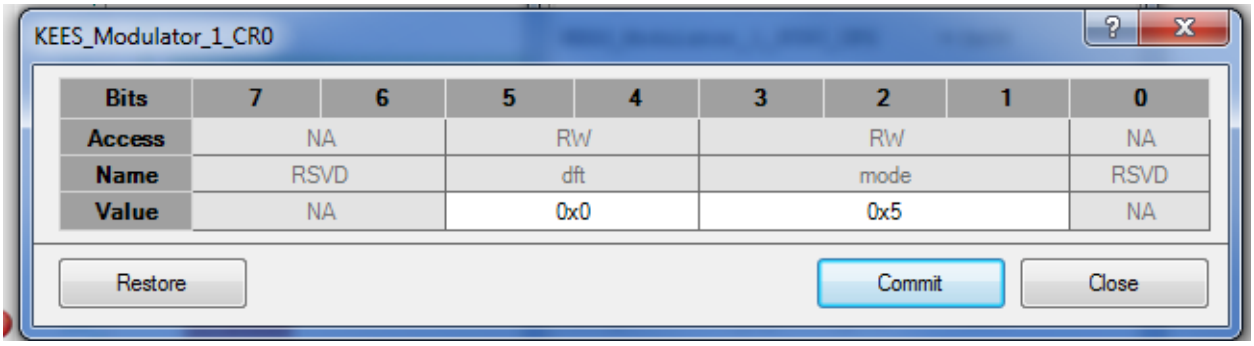


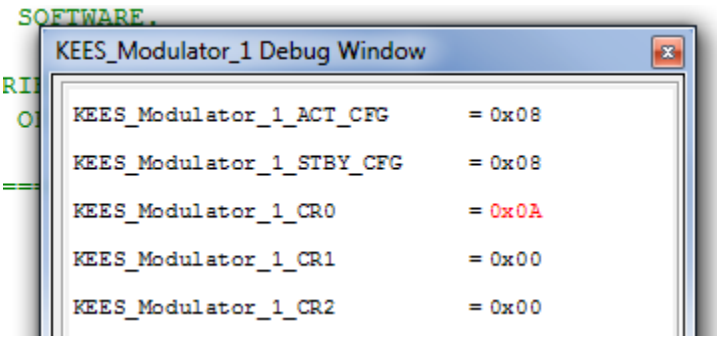


Hovering your cursor over a value will show all the options (unless the field is self explanatory like Clock_Enable)



You can modify the registers directly within these detailed views. Simply enter your new desired value and click Commit.





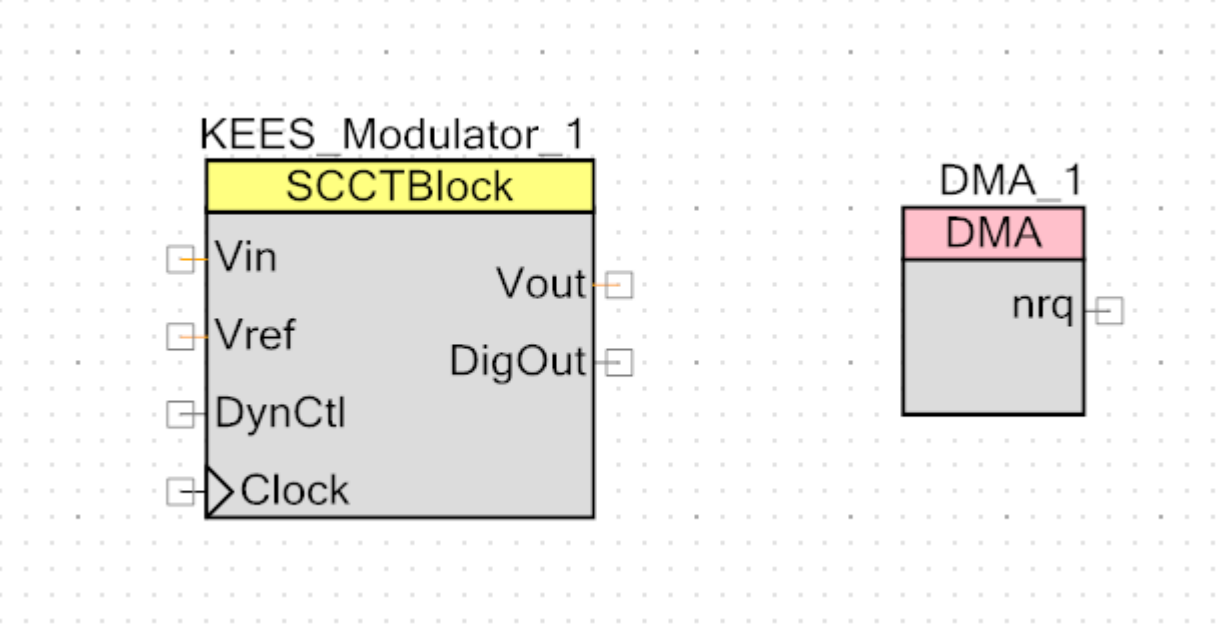
Explore the registers! Try out the different modes! Debug your new component with incredible ease! These register details are pulled directly from the RDL. If the RDL lists the register and the mode, it is included in the component debugger.

The component debug window has all registers associated with the SC/CT blocks, including power and routing registers. Please note, that some routing options will not be possible given the placement of the SC/CT block, but the

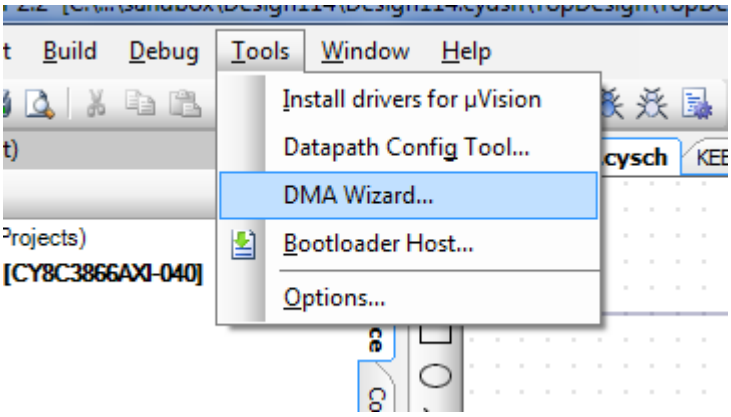
component debug tool will show you all possible options, regardless of placement of the SC/CT block within PSoC. There may be a way to update the debug file to reflect the options available, but I am not aware of how to do that at this time. Hover your cursor over a routing register in the component debug window and look for a comment that states if the contents of the register depend on block placement to find out which ones have placement dependent routing. If you are curious about the component debug capability file, refer to the **.cycdx** file.

DMA Wizard features

The component also comes with a DMA cpabability file, which allows the component to take advantage of the DMA wizard tool. To use the DMA capabilities, place a DMA component in the schematic.

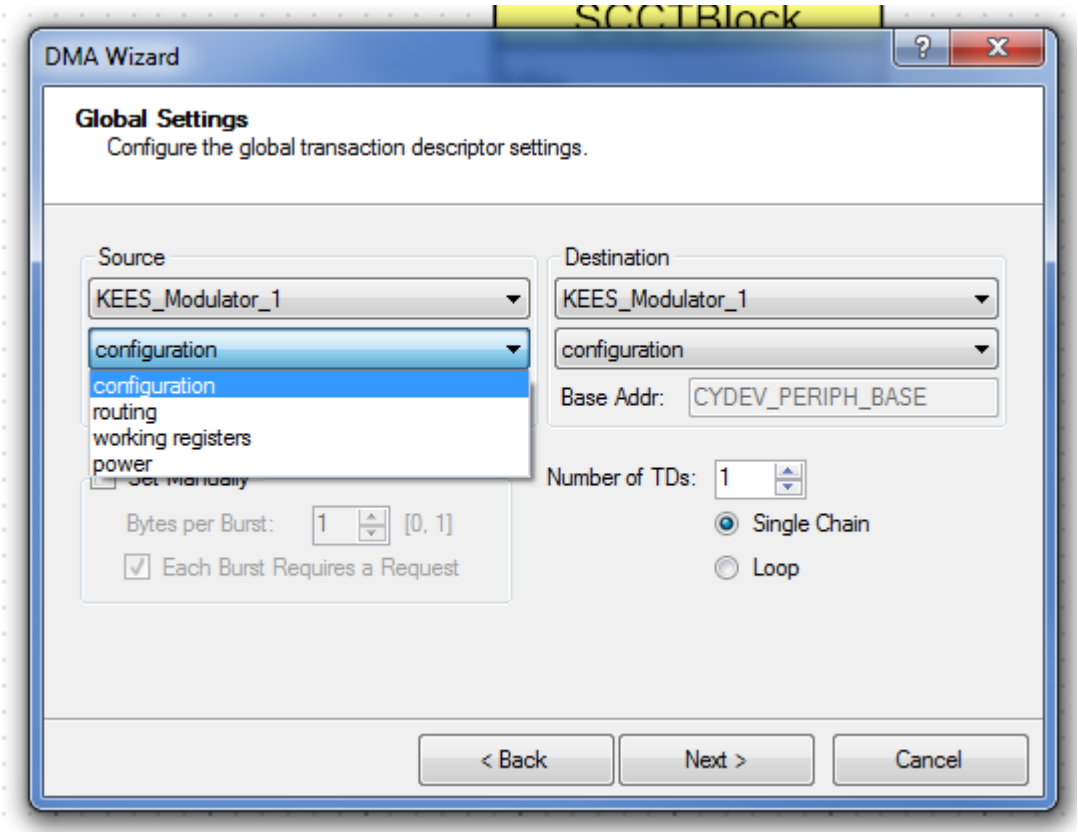


Open the DMA Wizard by selecting Tools->DMA Wizard

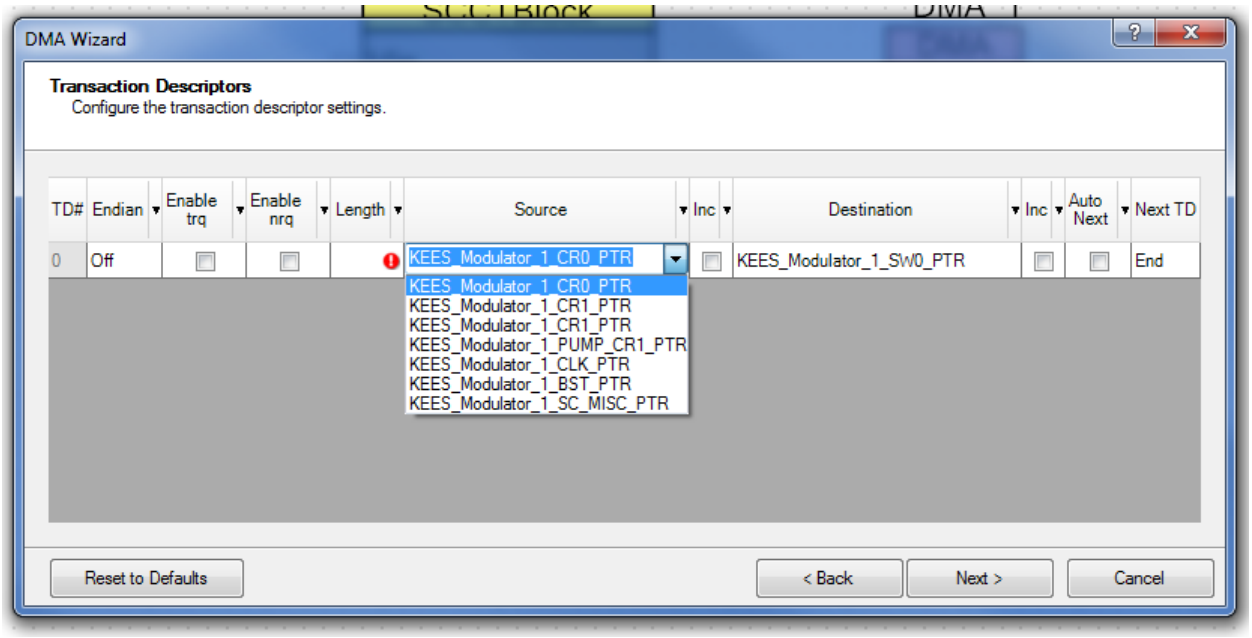


The DMA wizard will have register access to all the registers that the component has. These registers are grouped into 4 categories:

1.) Configuration (associated with configuring the options of the SC/CT block)
2.) Routing (analog routing switches)
3.) Working registers (Features the chip supports but Creator does not)
4.) Power (active and alt active configuration registers for the SC/CT blocks)



The registers can be either sources or destinations. Under each category, there are pointers to all the associated registers in the selected category.



If you are curious about the DMA capability file, look at the **.cydmacap** file. For more information on how to modify this file to fit the final needs of your component, take a look at KEES # 189.

.c and .h API files

The included .c and .h API files provide all the register, masks, modes and shift definitions for configuring the component in any way you see fit. The .h file has all the REG and PTR definitions at the top, followed by sets of masks, modes and shifts for each register grouped by register and functionality.

The .c file includes stub init, enable and start functions. You only really need to modify the init stub function by enabling the features and modes you desire for your component using the included definitions in the .h file. Under the init function is a commented out init function for enabling the DeltaSigma modulator functionality of the SC/CT blocks.

Using this component as a starting point, you can create ANY component you desire from the SC/CT blocks. Simply modify the component symbol to suit your needs, change the API init function to enable the features you want, and export your very own SC/CT block component. Feel free to modify the debugging or DMA capabilities files for you own requirements.