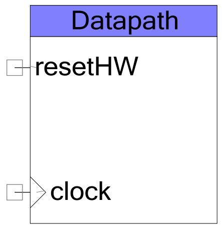
# Datapath Primitives

**DatapathComponent v1.0**

## Features



- Readymade component symbol
- A0, A1, D0, D1 initializations in the Component customizer
- Easy access to the FIFO single buffer mode, configuration of status signals
- API files with Datapath register pointers, masks, shifts, Start function
- Skeleton Verilog with pre-chained Datapath ready to configure in DP Tool, and a State machine to control it
- Datapath instantiation in merge region of Verilog file – regenerate the Verilog file with peace
- Component debug capability file gives easy access to the Datapath working registers during debug
- DMA capability file simplifies DMA wizard access to all your important registers
- Reference HTML file provided – contains all variants of the built-in Verilog instances of UDB Clock Enable, Control and Status registers

## General Description

These datapath primitives allow you to start designing with an 8, 16, 24 or 32 bit datapath, preconfigured and bristling with features to help you try out new ideas and debug when things go wrong.

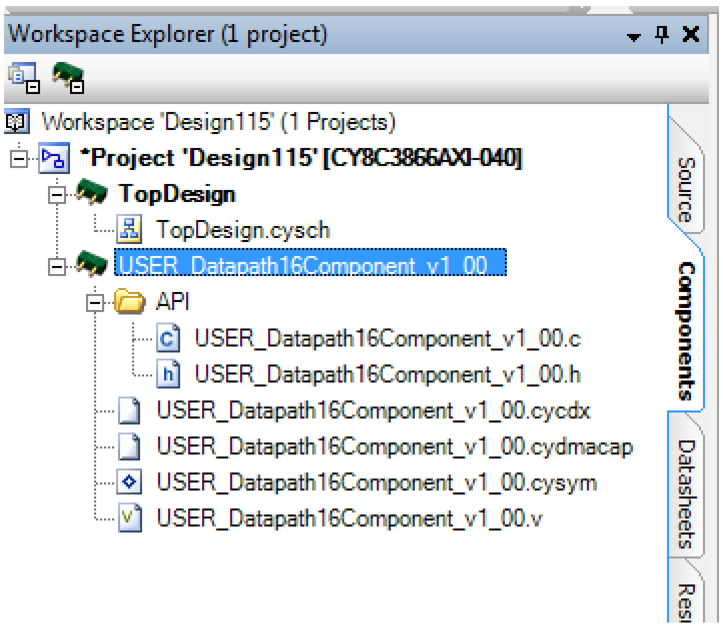
With these components, all you need to do to start using a datapath is import the desired super primitive, point the Datapath configuration tool at the premade Verilog file, configure your datapath and write your Verilog to control it and GO!

## Getting Started with the Datapath primitive

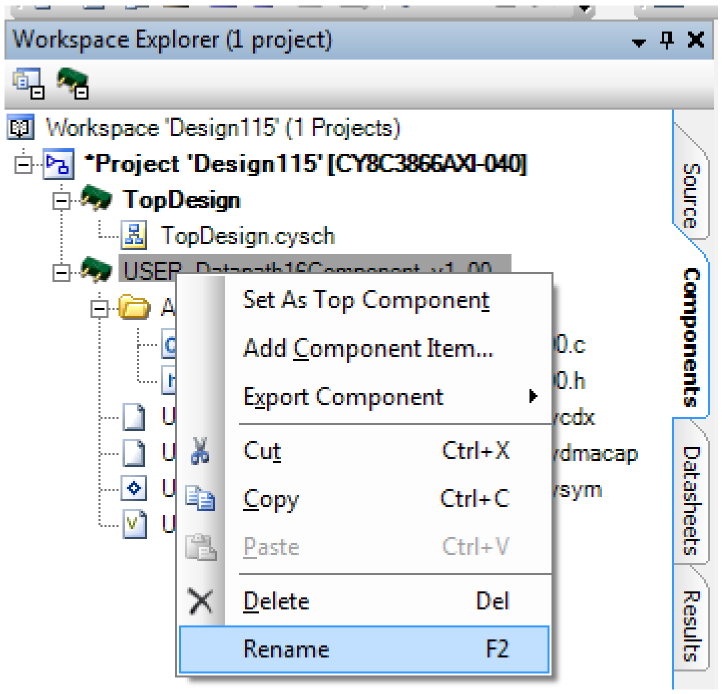
In this datasheet, the Component is assumed to have the name:
USER_Datapath16Component_v1_00

This section discusses the steps in getting familiar and running with the Component.

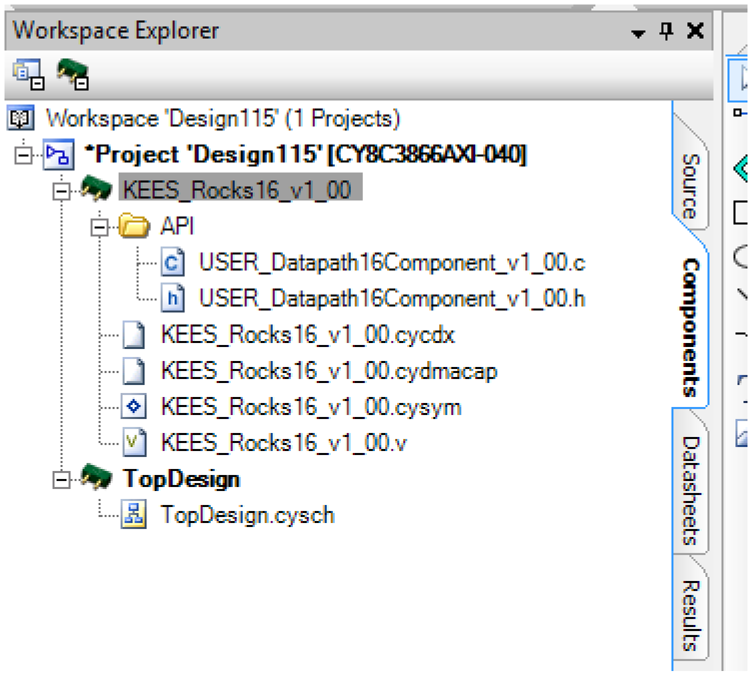
## Essential First Steps


Workspace Explorer (1 project)
Workspace 'Design115' (1 Projects)
*Project 'Design115' [CY8C3866AXI-040]
TopDesign
TopDesign.cysch
USER_Datapath16Component_v1_00
API
USER_Datapath16Component_v1_00.c
USER_Datapath16Component_v1_00.h
USER_Datapath16Component_v1_00.cycdx
USER_Datapath16Component_v1_00.cydmacap
USER_Datapath16Component_v1_00.cysym
USER_Datapath16Component_v1_00.v
Source
Components
Datasheets
Res

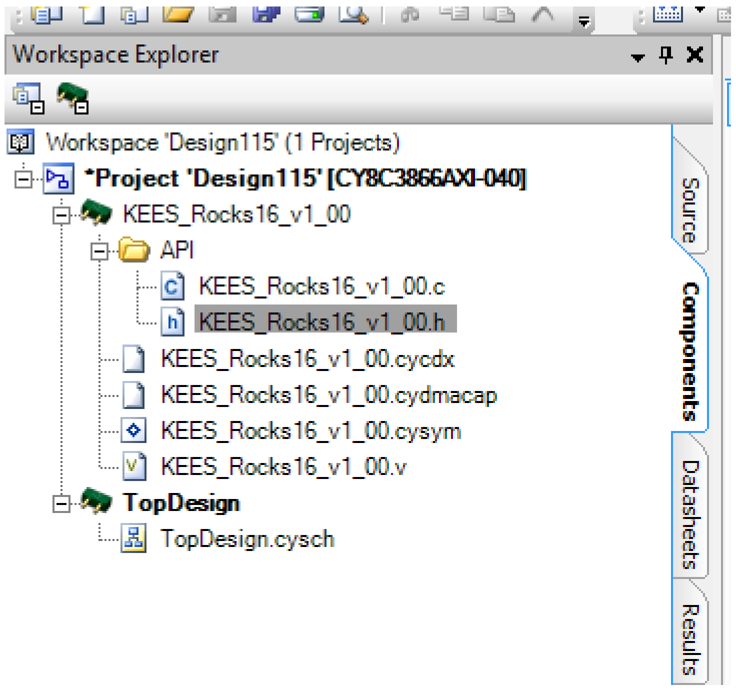
First thing you should do is rename the component by right clicking on the component and selecting “Rename.”



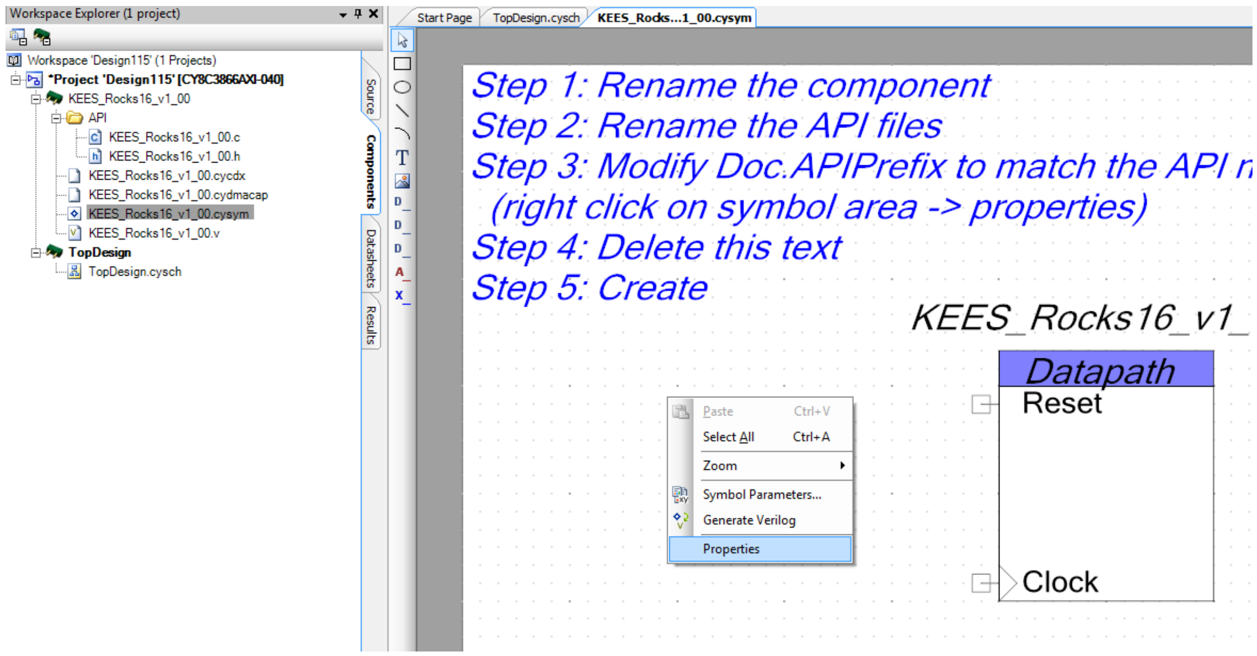
Rename the Component to whatever you want. This Datasheet uses KEES_Rocks16_v1_00. Make sure to leave the _v1_00 appended to the end so that you can make new versions later if you choose.


Workspace Explorer
Workspace 'Design115' (1 Projects)
*Project 'Design115' [CY8C3866AXI-040]
KEES_Rocks16_v1_00
API
USER_Datapath16Component_v1_00.c
USER_Datapath16Component_v1_00.h
KEES_Rocks16_v1_00.cycdx
KEES_Rocks16_v1_00.cydmacap
KEES_Rocks16_v1_00.cysym
KEES_Rocks16_v1_00.v
TopDesign
TopDesign.cysch

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"

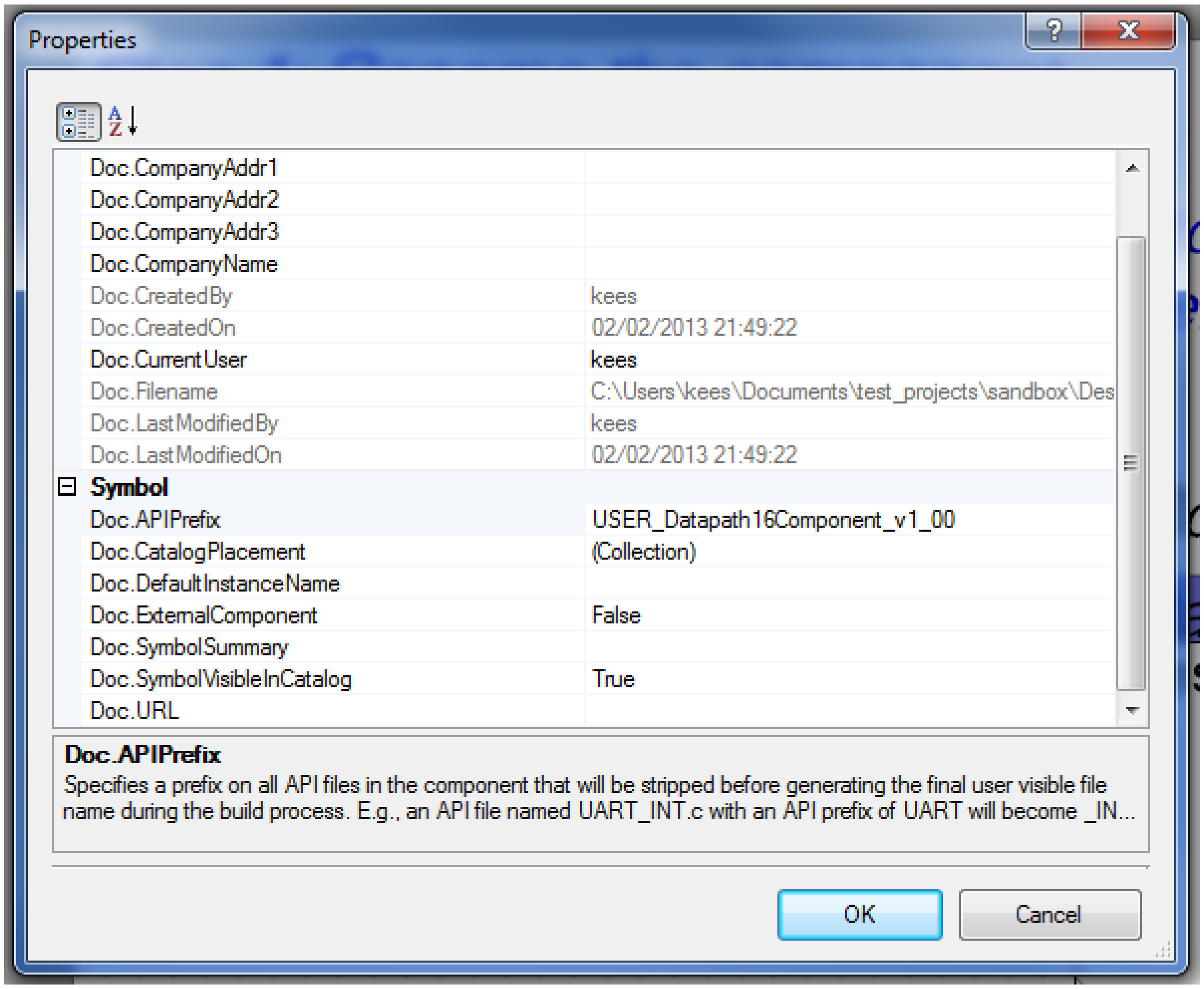

Workspace Explorer
Workspace 'Design115' (1 Projects)
*Project 'Design115' [CY8C3866AXI-040]
KEES_Rocks16_v1_00
API
KEES_Rocks16_v1_00.c
KEES_Rocks16_v1_00.h
KEES_Rocks16_v1_00.cycdx
KEES_Rocks16_v1_00.cydmacap
KEES_Rocks16_v1_00.cysym
KEES_Rocks16_v1_00.v
TopDesign
TopDesign.cysch

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



*Step 1: Rename the component*
*Step 2: Rename the API files*
*Step 3: Modify Doc.APIPrefix to match the API*
*(right click on symbol area -> properties)*
*Step 4: Delete this text*
*Step 5: Create*

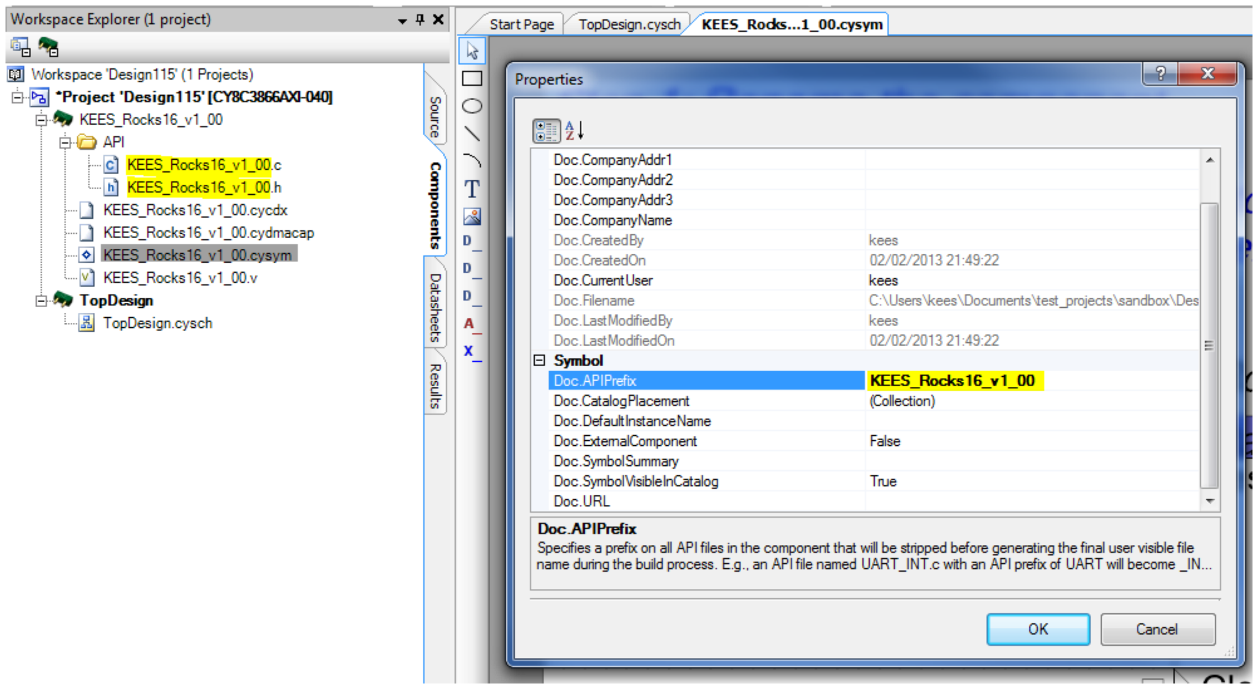
*(The component symbols also include text to remind you of these steps. This text can be deleted at any time)*



| | |
|---|---|
| Doc.CompanyAddr1 | |
| Doc.CompanyAddr2 | |
| Doc.CompanyAddr3 | |
| Doc.CompanyName | |
| Doc.CreatedBy | kees |
| Doc.CreatedOn | 02/02/2013 21:49:22 |
| Doc.CurrentUser | kees |
| Doc.Filename | C:\Users\kees\Documents\test_projects\sandbox\Des |
| Doc.LastModifiedBy | kees |
| Doc.LastModifiedOn | 02/02/2013 21:49:22 |
| **Symbol** | |
| Doc.APIPrefix | USER_Datapath16Component_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 _IN...

Change the Doc.APIPrefix to match the name of the renamed API files.

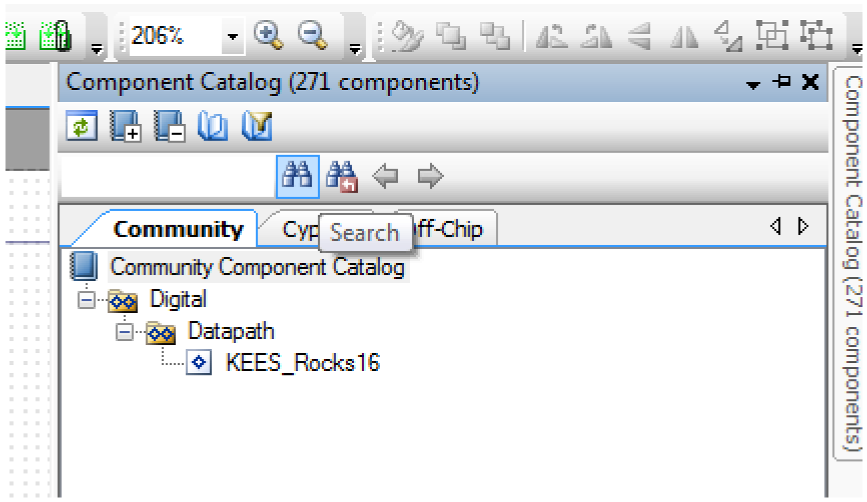


Workspace 'Design115' (1 Projects)
*Project 'Design115' [CY8C3866AXI-040]
KEES_Rocks16_v1_00
API
KEES_Rocks16_v1_00.c
KEES_Rocks16_v1_00.h
KEES_Rocks16_v1_00.cydx
KEES_Rocks16_v1_00.cydmacap
KEES_Rocks16_v1_00.cysym
KEES_Rocks16_v1_00.v
TopDesign
TopDesign.cysch

| | |
|---|---|
| Doc.CompanyAddr1 | |
| Doc.CompanyAddr2 | |
| Doc.CompanyAddr3 | |
| Doc.CompanyName | |
| Doc.CreatedBy | kees |
| Doc.CreatedOn | 02/02/2013 21:49:22 |
| Doc.CurrentUser | kees |
| Doc.Filename | C:\Users\kees\Documents\test_projects\sandbox\Des |
| Doc.LastModifiedBy | kees |
| Doc.LastModifiedOn | 02/02/2013 21:49:22 |
| **Symbol** | |
| Doc.APIPrefix | KEES_Rocks16_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 _IN...

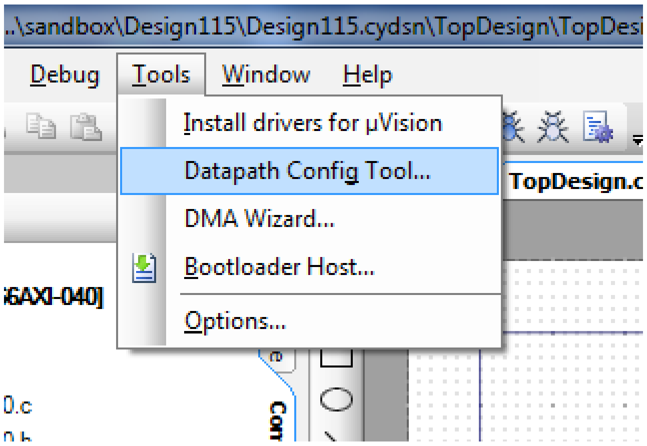
And that concludes all the component customization that you MUST do. Any further customization is optional. At this point, your component will be located in the “Community/Digital/Datapath/…” library tab.


206%
Component Catalog (271 components)
Community
Cyp
Search
ff-Chip
Community Component Catalog
Digital
Datapath
KEES_Rocks16
Component Catalog (271 components)

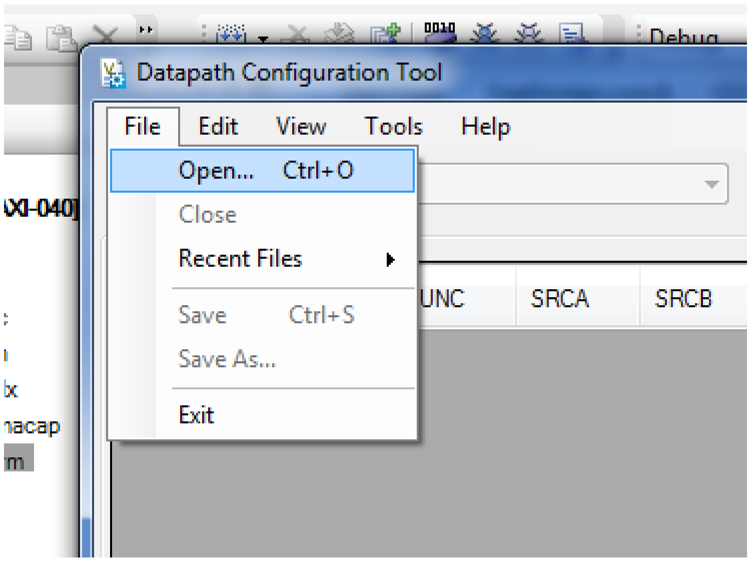
## Datapath Primitive and the Datapath Configuration Tool

You can start to experiment with the datapath. Depending on your work flow with datapaths, you may dive right into the verilog file, or you may fire up the datapath configuration tool and set up your datapath instructions. For the purpose of this memo, I’m going to fire up the datapath configuration tool, and walk you through finalizing the setup of your new datapath component.

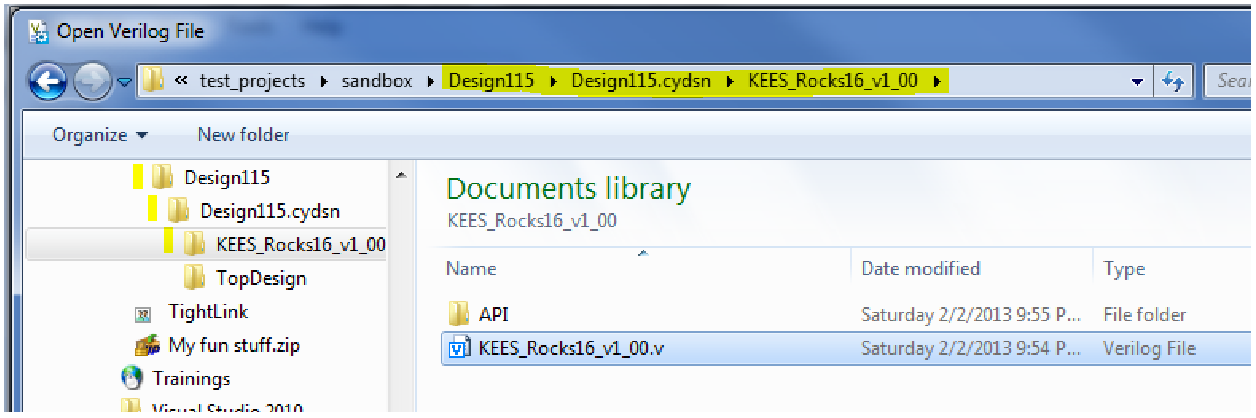
Go to Tools->Datapath configuration tool



When the tool opens, select File->Open



Navigate to your project directory. In you project design folder, there will be another folder with your newly imported component name. Go into that folder, and select the Verilog file.



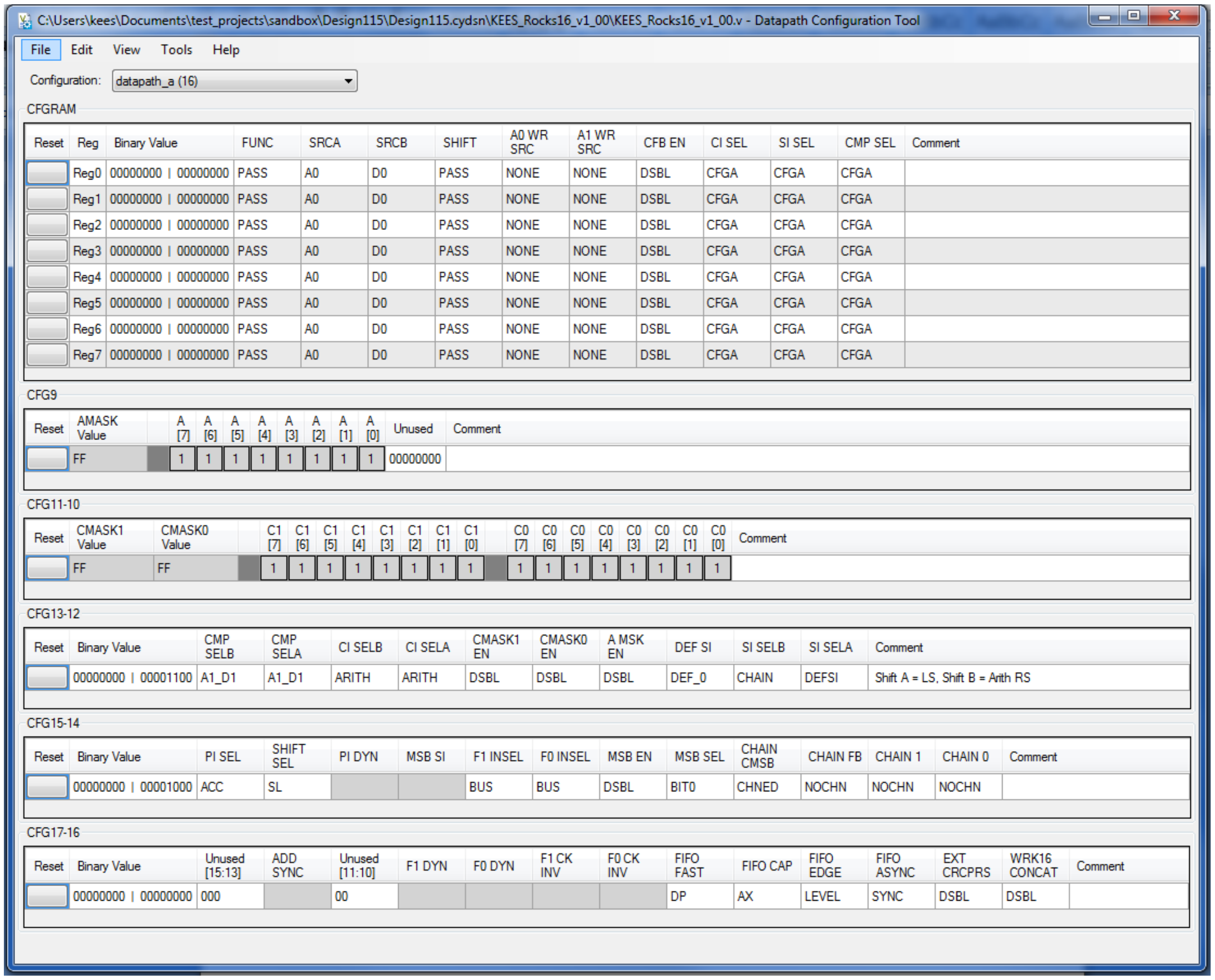
Open Verilog File

« test_projects ▸ sandbox ▸ Design115 ▸ Design115.cydsn ▸ KEES_Rocks16_v1_00 ▸

Documents library
KEES_Rocks16_v1_00

| Name | Date modified | Type |
|---|---|---|
| API | Saturday 2/2/2013 9:55 P... | File folder |
| KEES_Rocks16_v1_00.v | Saturday 2/2/2013 9:54 P... | Verilog File |

The Datapath configuration tool will update with the preconfigured datapath already in the file.



C:\Users\kees\Documents\test_projects\sandbox\Design115\Design115.cydsn\KEES_Rocks16_v1_00\KEES_Rocks16_v1_00.v - Datapath Configuration Tool

Configuration: datapath_a (16)

CFGRAM

| Reset | Reg | Binary Value | FUNC | SRCA | SRCB | SHIFT | A0 WR SRC | A1 WR SRC | CFB EN | CI SEL | SI SEL | CMP SEL | Comment |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Reg0 | 00000000 \| 00000000 | PASS | A0 | D0 | PASS | NONE | NONE | DSBL | CFGA | CFGA | CFGA | |
| | Reg1 | 00000000 \| 00000000 | PASS | A0 | D0 | PASS | NONE | NONE | DSBL | CFGA | CFGA | CFGA | |
| | Reg2 | 00000000 \| 00000000 | PASS | A0 | D0 | PASS | NONE | NONE | DSBL | CFGA | CFGA | CFGA | |
| | Reg3 | 00000000 \| 00000000 | PASS | A0 | D0 | PASS | NONE | NONE | DSBL | CFGA | CFGA | CFGA | |
| | Reg4 | 00000000 \| 00000000 | PASS | A0 | D0 | PASS | NONE | NONE | DSBL | CFGA | CFGA | CFGA | |
| | Reg5 | 00000000 \| 00000000 | PASS | A0 | D0 | PASS | NONE | NONE | DSBL | CFGA | CFGA | CFGA | |
| | Reg6 | 00000000 \| 00000000 | PASS | A0 | D0 | PASS | NONE | NONE | DSBL | CFGA | CFGA | CFGA | |
| | Reg7 | 00000000 \| 00000000 | PASS | A0 | D0 | PASS | NONE | NONE | DSBL | CFGA | CFGA | CFGA | |

CFG9

| Reset | AMASK Value | A [7] | A [6] | A [5] | A [4] | A [3] | A [2] | A [1] | A [0] | Unused | Comment |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | FF | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 00000000 | |

CFG11-10

| Reset | CMASK1 Value | CMASK0 Value | C1 [7] | C1 [6] | C1 [5] | C1 [4] | C1 [3] | C1 [2] | C1 [1] | C1 [0] | C0 [7] | C0 [6] | C0 [5] | C0 [4] | C0 [3] | C0 [2] | C0 [1] | C0 [0] | Comment |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | FF | FF | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | |

CFG13-12

| Reset | Binary Value | CMP SELB | CMP SELA | CI SELB | CI SELA | CMASK1 EN | CMASK0 EN | A MSK EN | DEF SI | SI SELB | SI SELA | Comment |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 00000000 \| 00001100 | A1_D1 | A1_D1 | ARITH | ARITH | DSBL | DSBL | DSBL | DEF_0 | CHAIN | DEFSI | Shift A = LS, Shift B = Arith RS |

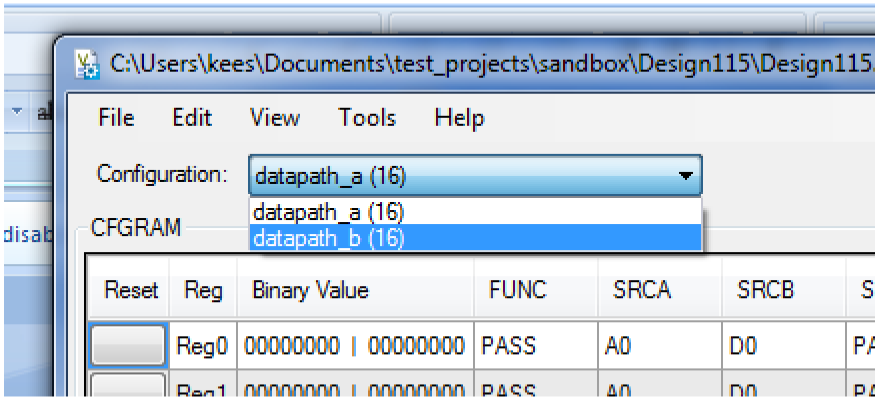
CFG15-14

| Reset | Binary Value | PI SEL | SHIFT SEL | PI DYN | MSB SI | F1 INSEL | F0 INSEL | MSB EN | MSB SEL | CHAIN CMSB | CHAIN FB | CHAIN 1 | CHAIN 0 | Comment |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 00000000 \| 00001000 | ACC | SL | | | BUS | BUS | DSBL | BIT0 | CHNED | NOCHN | NOCHN | NOCHN | |

CFG17-16

| Reset | Binary Value | Unused [15:13] | ADD SYNC | Unused [11:10] | F1 DYN | F0 DYN | F1 CK INV | F0 CK INV | FIFO FAST | FIFO CAP | FIFO EDGE | FIFO ASYNC | EXT CRCPRS | WRK16 CONCAT | Comment |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 00000000 \| 00000000 | 000 | | 00 | | | | | DP | AX | LEVEL | SYNC | DSBL | DSBL | |

If you select a 16, 24, or 32 bit datapath, there will be a "datapath_a/b/c/d" representing the 8-15, 16-23, and 24-31st bit datapaths. In this example, I imported a 16 bit datapath, so I have a datapath_a (LSB) and a datapth_B (MSB)


C:\Users\kees\Documents\test_projects\sandbox\Design115\Design115
File
Edit
View
Tools
Help
disab

Configuration: datapath_a (16)

datapath_a (16)
datapath_b (16)

CFGRAM

| Reset | Reg | Binary Value | FUNC | SRCA | SRCB | S |
|---|---|---|---|---|---|---|
|  | Reg0 | 00000000 \| 00000000 | PASS | A0 | D0 | PA |
|  | Reg1 | 00000000 \| 00000000 | PASS | A0 | D0 | PA |

For reference, datapath_a is always the lowest byte, and the increasing letters correspond to the increasing bit order of the datapath.

datapath_a: bits 0-7

datapath_b: bits 8-15

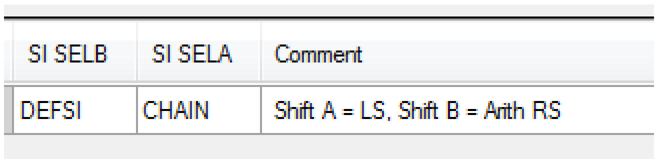
datapath_c: bits 16-23

datapath_d: bits 24-31

The datapaths have already been chained for you so that they "just work" for arithmetic, comparisons, and shifting. For a cheat sheet on how to chain datapaths, refer to KBA86838 Datapath Configuration Tool Cheat Sheet.

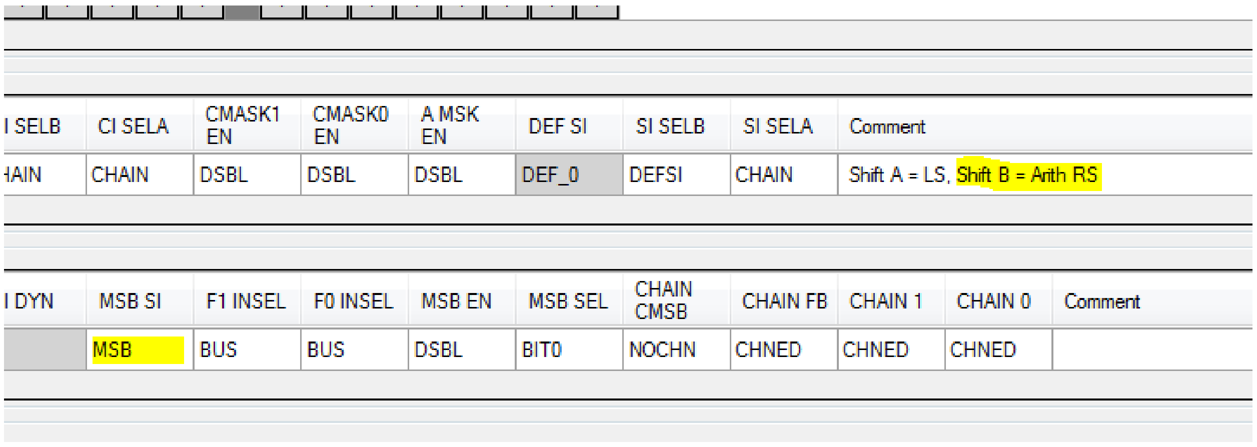
**A note on shifting**:

**The preconfigured datapaths have been set up to perform Left Shifts (zero shifted in) when shift configuration A is selected, and ARITHmetic Right Shifts when shift configuration B is selected. I felt that these modes would be the most useful by default.**



| SI SELB | SI SELA | Comment |
|---|---|---|
| DEFSI | CHAIN | Shift A = LS, Shift B = Arith RS |

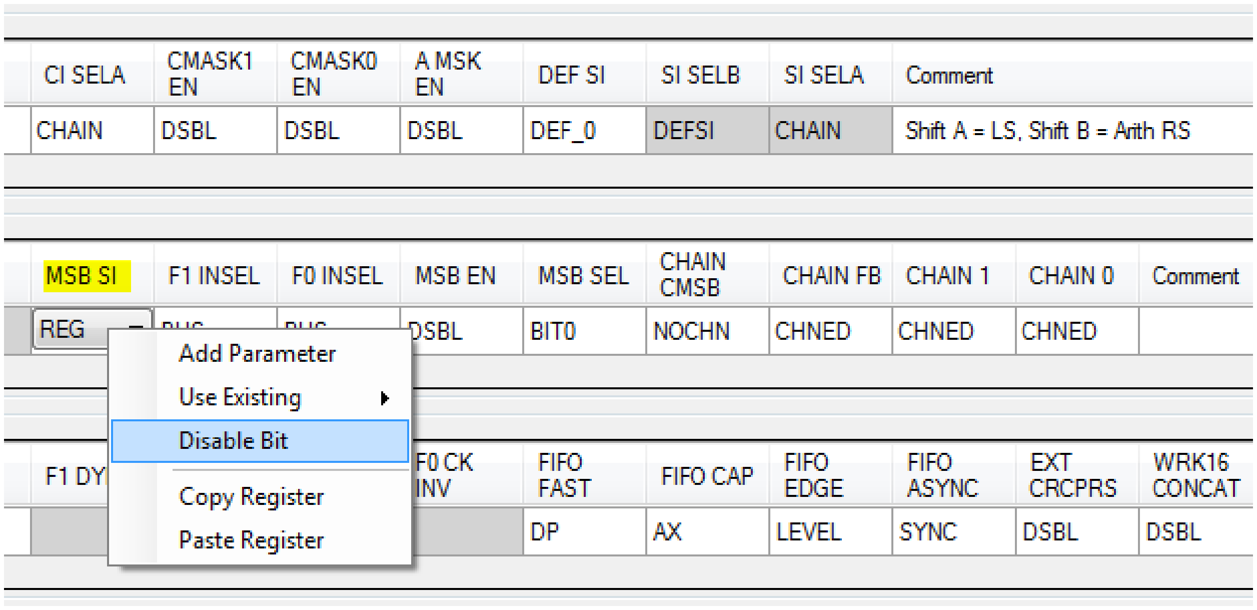
***To change the behavior of the right shift from arithmetic* to a normal right shift, shifting in zeros into the MSB, make the following change on the <u>most significant byte datapath</u>**



| I SELB | CI SELA | CMASK1 EN | CMASK0 EN | A MSK EN | DEF SI | SI SELB | SI SELA | Comment |
|---|---|---|---|---|---|---|---|---|
| HAIN | CHAIN | DSBL | DSBL | DSBL | DEF_0 | DEFSI | CHAIN | Shift A = LS, Shift B = Arith RS |

| I DYN | MSB SI | F1 INSEL | F0 INSEL | MSB EN | MSB SEL | CHAIN CMSB | CHAIN FB | CHAIN 1 | CHAIN 0 | Comment |
|---|---|---|---|---|---|---|---|---|---|---|
| | MSB | BUS | BUS | DSBL | BIT0 | NOCHN | CHNED | CHNED | CHNED | |

**Right click on the "MSB SI" bit field, and select "Disable Bit"**

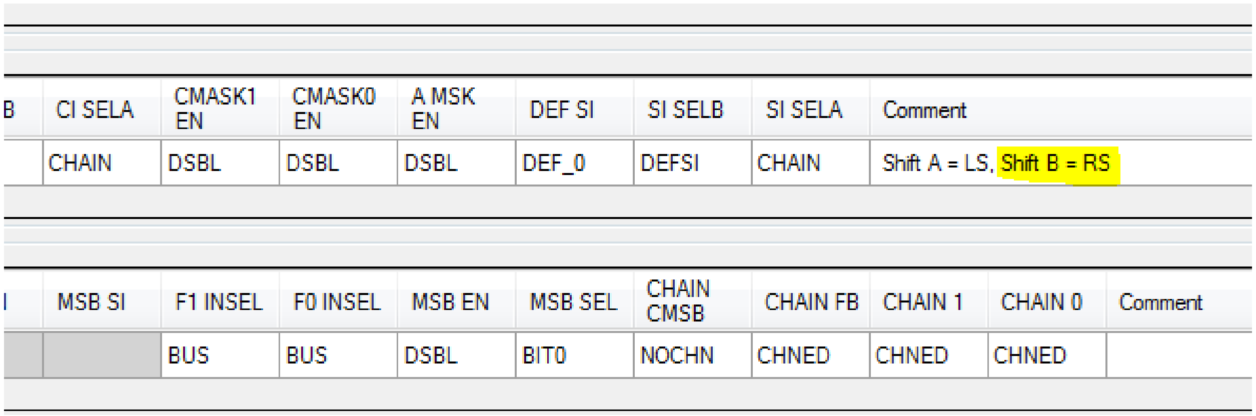


| CI SELA | CMASK1 EN | CMASK0 EN | A MSK EN | DEF SI | SI SELB | SI SELA | Comment |
|---|---|---|---|---|---|---|---|
| CHAIN | DSBL | DSBL | DSBL | DEF_0 | DEFSI | CHAIN | Shift A = LS, Shift B = Arith RS |

| MSB SI | F1 INSEL | F0 INSEL | MSB EN | MSB SEL | CHAIN CMSB | CHAIN FB | CHAIN 1 | CHAIN 0 | Comment |
|---|---|---|---|---|---|---|---|---|---|
| REG | BUS | BUS | DSBL | BIT0 | NOCHN | CHNED | CHNED | CHNED | |

| F1 DY | | F0 CK INV | FIFO FAST | FIFO CAP | FIFO EDGE | FIFO ASYNC | EXT CRCPRS | WRK16 CONCAT |
|---|---|---|---|---|---|---|---|---|
| | | | DP | AX | LEVEL | SYNC | DSBL | DSBL |

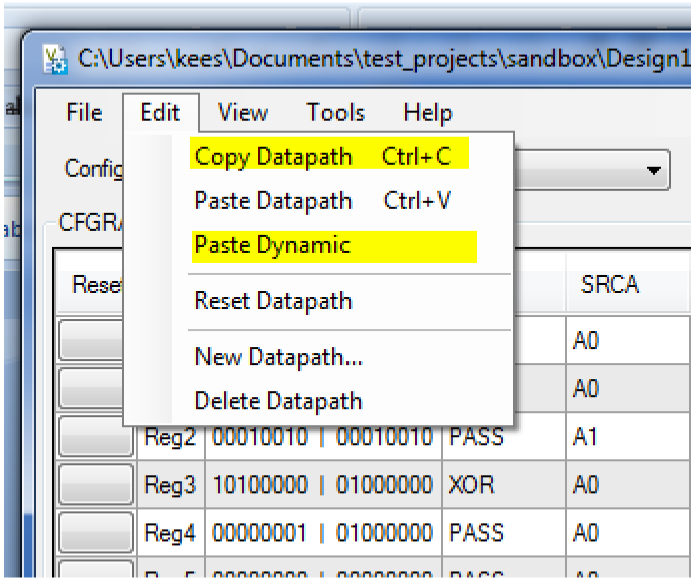
***The datapath is now configured to do a non-arithmetic right shift in shift configuration B.***



| B | CI SELA | CMASK1 EN | CMASK0 EN | A MSK EN | DEF SI | SI SELB | SI SELA | Comment |
|---|---|---|---|---|---|---|---|---|
| | CHAIN | DSBL | DSBL | DSBL | DEF_0 | DEFSI | CHAIN | Shift A = LS, Shift B = RS |

| I | MSB SI | F1 INSEL | F0 INSEL | MSB EN | MSB SEL | CHAIN CMSB | CHAIN FB | CHAIN 1 | CHAIN 0 | Comment |
|---|---|---|---|---|---|---|---|---|---|---|
| | | BUS | BUS | DSBL | BIT0 | NOCHN | CHNED | CHNED | CHNED | |

Now configure your datapath! Set up the instructions to do as you please (make sure to configure all bytes of the datapaths the same way! You can use Edit->Copy Datapath, then move to another byte of the datapath and Edit->Paste Dynamic)

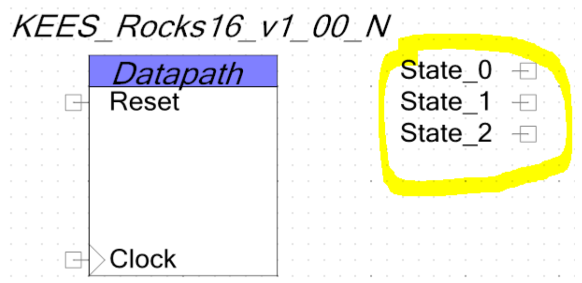


| | | | | |
|---|---|---|---|---|
| | | | | A0 |
| | | | | A0 |
| Reg2 | 00010010 | 00010010 | PASS | A1 |
| Reg3 | 10100000 | 01000000 | XOR | A0 |
| Reg4 | 00000001 | 01000000 | PASS | A0 |

Be wary of “Paste Datapath” since this will paste the entire datapath, messing up the chaining.

Save your datapath configuration and close the tool. Closing the tool is a hassle when you want to make quick edits to your datapath, but leaving the tool open can become problematic when you forget to save the Verilog file or the datapath configuration and have 2 versions of the Verilog file out of sync with each other. Take my advice, only edit one at a time, and close the datapath configuration tool when you are done using it. It will save you from many headaches.

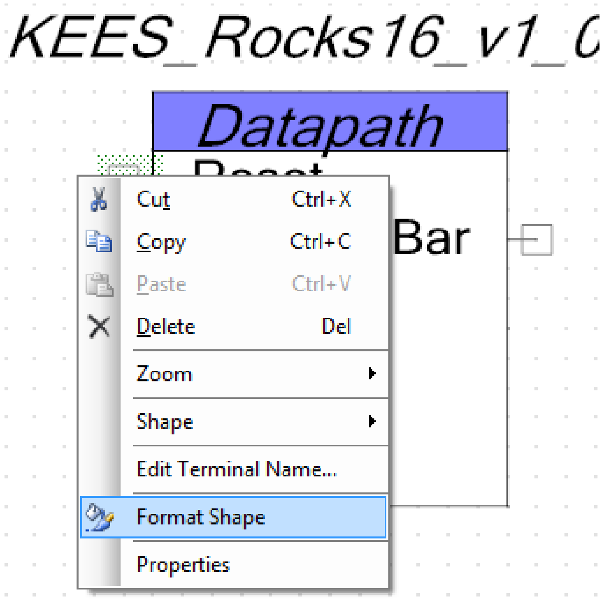
When you are done editing your datapath, you may want to add some input or outputs to your component. You can open the schematic symbol and add your inputs and outputs there. I am going to assume you know how to add inputs and outputs to the symbol so I will leave those instructions out. I will however point out the set of debugging pins included with the symbol.

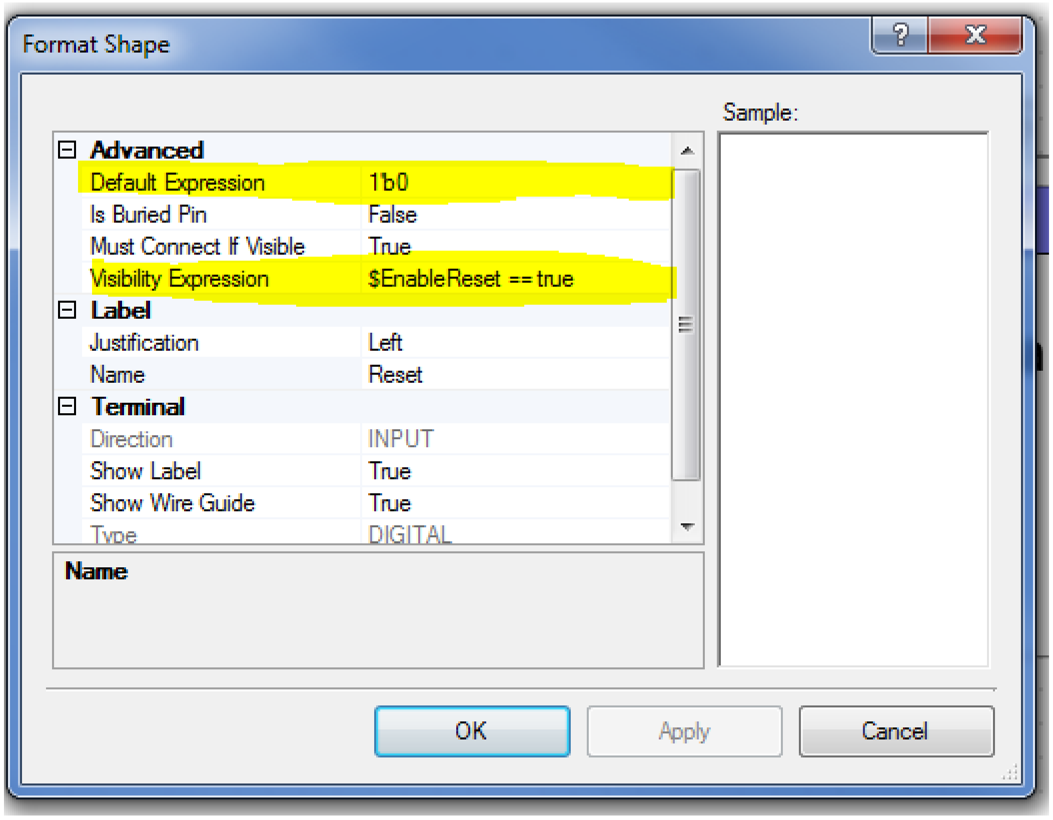


These pins are set to be invisible when the '$Debug' parameter is set to 'false'. I recommend adding useful signals to this set of debugging pins and setting them to hide when debugging is disabled for your component. When you don't need them, they are out of the way and not bothering anything. But when you do need them, they can provide critical insight into how your datapath is behaving. The easiest way to add debugging specific pins is to just copy one of the State_x pins. The shape formatting that hides the pin when debugging is disabled will be copied along with it. Just change the name to suit your needs.

A tip for adding digital inputs to your design that may be hidden based on a component parameter, make sure you set the default value along with the visibility expression.

To access the visibility expression, right click on the pin and select "Format Shape"



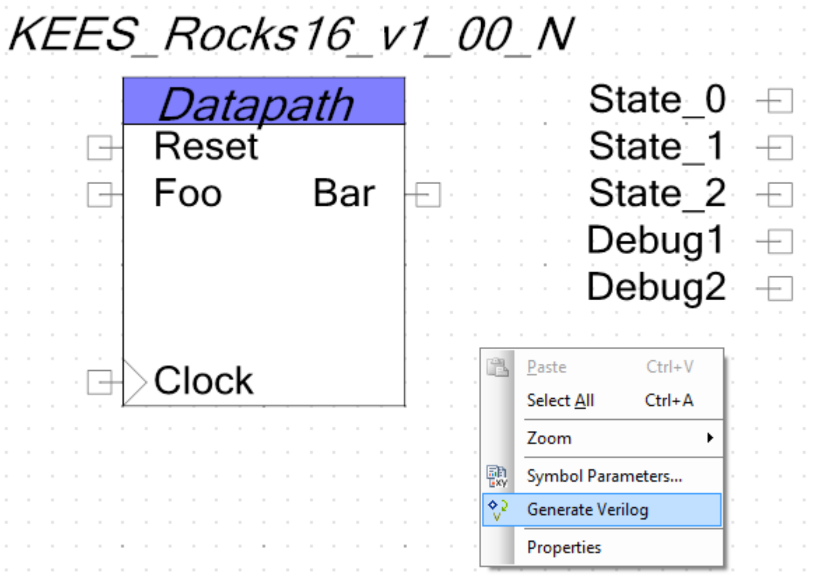


| | |
|---|---|
| **Advanced** | |
| Default Expression | 1'b0 |
| Is Buried Pin | False |
| Must Connect If Visible | True |
| Visibility Expression | $EnableReset == true |
| **Label** | |
| Justification | Left |
| Name | Reset |
| **Terminal** | |
| Direction | INPUT |
| Show Label | True |
| Show Wire Guide | True |
| Type | DIGITAL |

**Name**

The default expression is the logic value that this input pin will be driven to when it's hidden.

After you have added whatever signals you desire, you can right click on the symbols empty space and select "Generate Verilog."



Click “Generate” on the dialog box that shows up, and then click “OK” when it warns that you are going to overwrite an existing file. All of the preconfigured verilog and datapath information is placed inside a merge region, meaning it will be untouched when the verilog is updated.

Before:

```
`include "cypress.v"
//`#end` -- edit above this line, do not edit this line
// Generated on 02/02/2013 at 22:49
// Component: KEES_Rocks16_v1_00
module KEES_Rocks16_v1_00 (
      output  State_0,
      output  State_1,
      output  State_2,
      input   Clock,
      input   Reset
);
      parameter A0_init_a = 0;
      parameter A0_init_b = 0;
…
```

After:

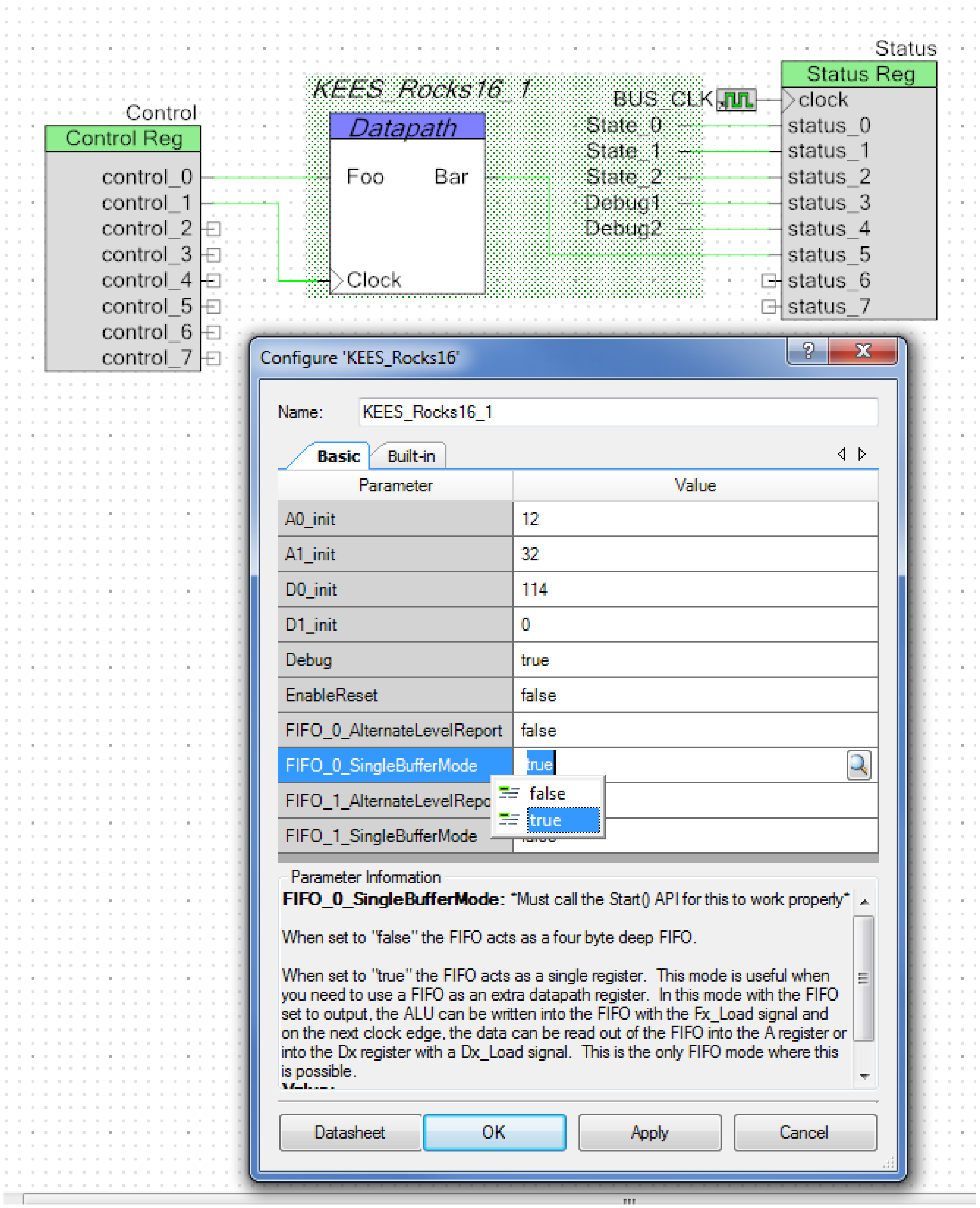
```
`include "cypress.v"
//`#end` -- edit above this line, do not edit this line
// Generated on 02/02/2013 at 22:51
// Component: KEES_Rocks16_v1_00
module KEES_Rocks16_v1_00 (
      output  Bar,
      output  Debug1,
      output  Debug2,
      output  State_0,
      output  State_1,
      output  State_2,
      input   Clock,
      input   Foo,
      input   Reset
);

      parameter A0_init_a = 0;
      parameter A0_init_b = 0;
…
```

You can use the same technique to add new parameters, or remove parameters you don't want.

From here, take a quick look at the skeleton code provided in the verilog file. It has parameters passed in from the customizer to initialize the A0, A1, D0 and D1 registers, a 3 bit register called "State" that controls the state machine as well as provides the address for the datapath instruction. The skeleton state machine is set up with a synchronous reset (if used) as well as 8 empty states for you to write your Verilog. Once your Verilog is written, it's off to the debugger!

## Component Debug features

Add one of your components to your design schematic, and add a Start() API call in main. Hook up your necessary signals, configure the parameters offered in the customizer and start a debugging session.



Name: KEES_Rocks16_1

| Parameter | Value |
|---|---|
| A0_init | 12 |
| A1_init | 32 |
| D0_init | 114 |
| D1_init | 0 |
| Debug | true |
| EnableReset | false |
| FIFO_0_AlternateLevelReport | false |
| FIFO_0_SingleBufferMode | true |
| FIFO_1_AlternateLevelRepo | |
| FIFO_1_SingleBufferMode | false |

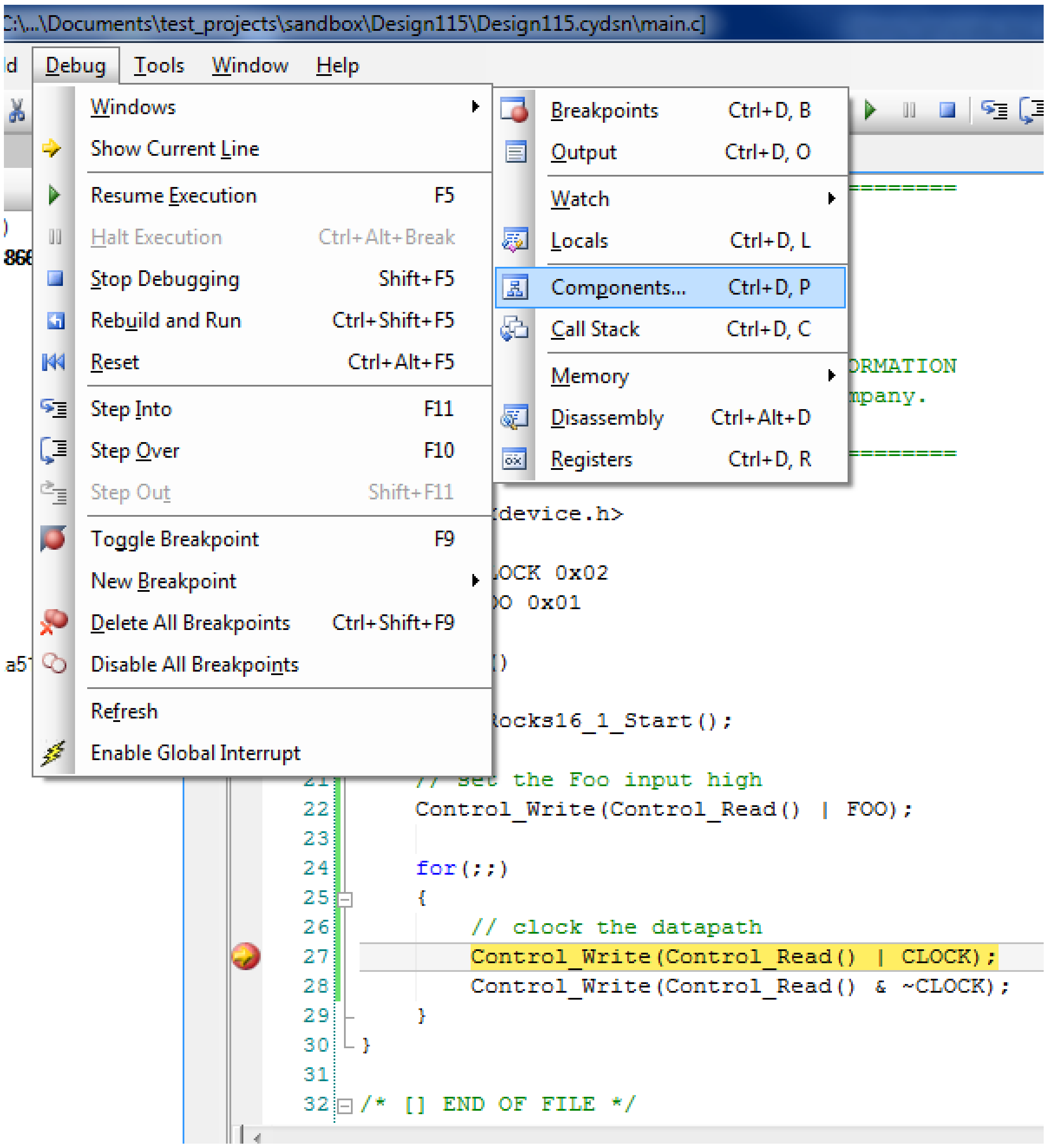
Parameter Information

**FIFO_0_SingleBufferMode**: *Must call the Start() API for this to work properly*

When set to "false" the FIFO acts as a four byte deep FIFO.

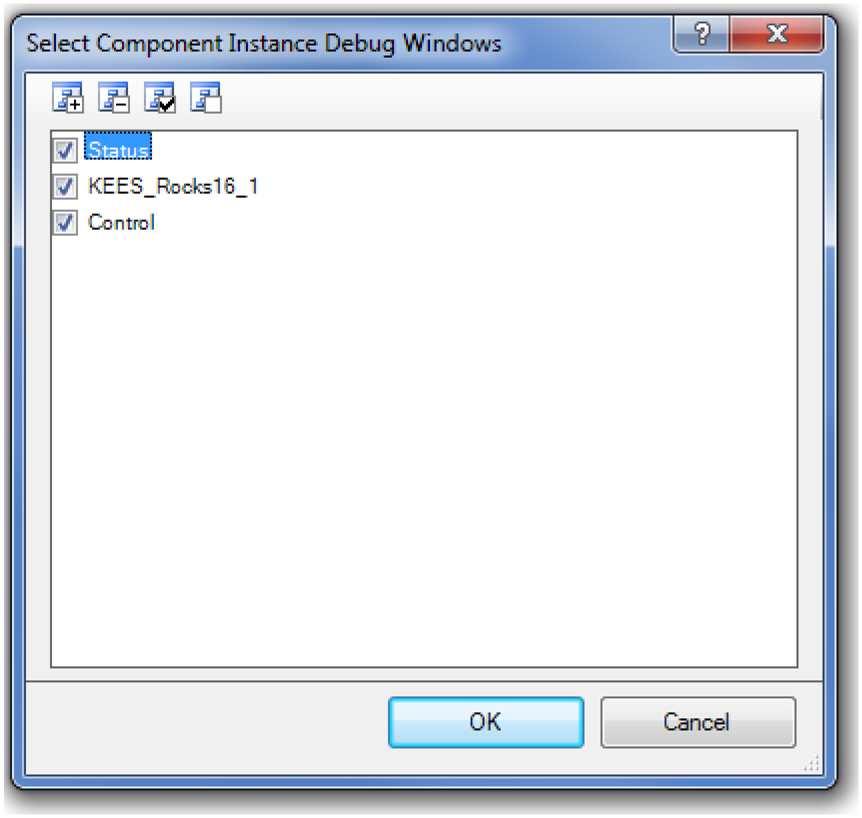
When set to "true" the FIFO acts as a single register. This mode is useful when you need to use a FIFO as an extra datapath register. In this mode with the FIFO set to output, the ALU can be written into the FIFO with the Fx_Load signal and on the next clock edge, the data can be read out of the FIFO into the A register or into the Dx register with a Dx_Load signal. This is the only FIFO mode where this is possible.

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

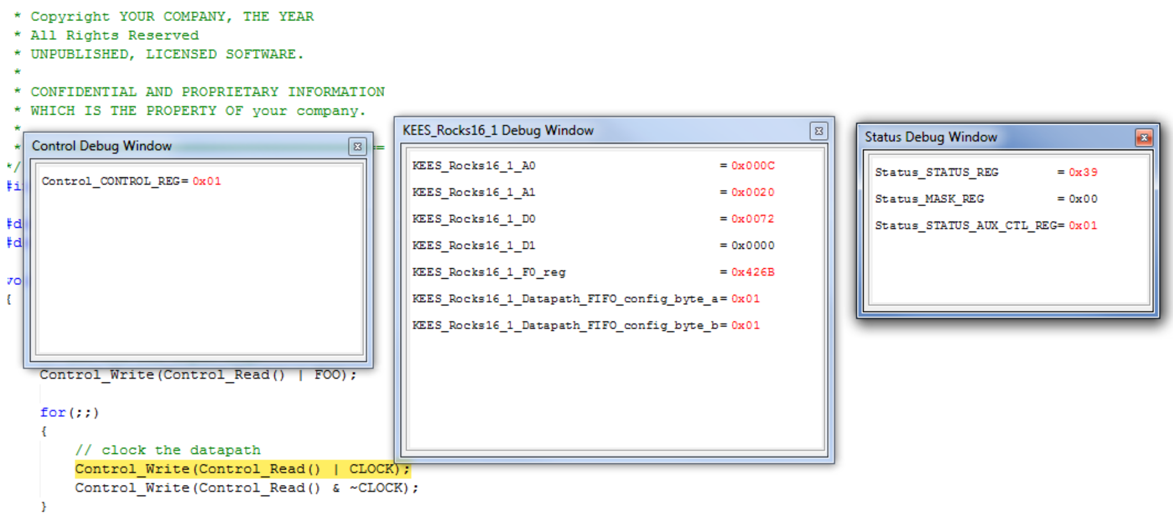


```
========
ORMATION
mpany.
========
<device.h>

OCK 0x02
O 0x01

)

ocks16_1_Start();
    // set the Foo input high
    Control_Write(Control_Read() | FOO);

    for(;;)
    {
        // clock the datapath
        Control_Write(Control_Read() | CLOCK);
        Control_Write(Control_Read() & ~CLOCK);
    }
}

/* [] END OF FILE */
```

This will bring up the component debug selection window. Check the box next to your component and click OK. I also have a control register and a status register attached to my component, allowing me to stimulate the inputs and check the outputs right in the debugger.



A new window in the debugger will appear filled with all of the registers associated with your datapath.



```
 * Copyright YOUR COMPANY, THE YEAR
 * All Rights Reserved
 * UNPUBLISHED, LICENSED SOFTWARE.
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION
 * WHICH IS THE PROPERTY OF your company.
```

Control Debug Window

```
Control_CONTROL_REG= 0x01
```

KEES_Rocks16_1 Debug Window

```
KEES_Rocks16_1_A0                         = 0x000C
KEES_Rocks16_1_A1                         = 0x0020
KEES_Rocks16_1_D0                         = 0x0072
KEES_Rocks16_1_D1                         = 0x0000
KEES_Rocks16_1_F0_reg                     = 0x426B
KEES_Rocks16_1_Datapath_FIFO_config_byte_a= 0x01
KEES_Rocks16_1_Datapath_FIFO_config_byte_b= 0x01
```

Status Debug Window

```
Status_STATUS_REG        = 0x39
Status_MASK_REG          = 0x00
Status_STATUS_AUX_CTL_REG= 0x01
```

```
    Control_Write(Control_Read() | FOO);

    for(;;)
    {
        // clock the datapath
        Control_Write(Control_Read() | CLOCK);
        Control_Write(Control_Read() & ~CLOCK);
    }
```

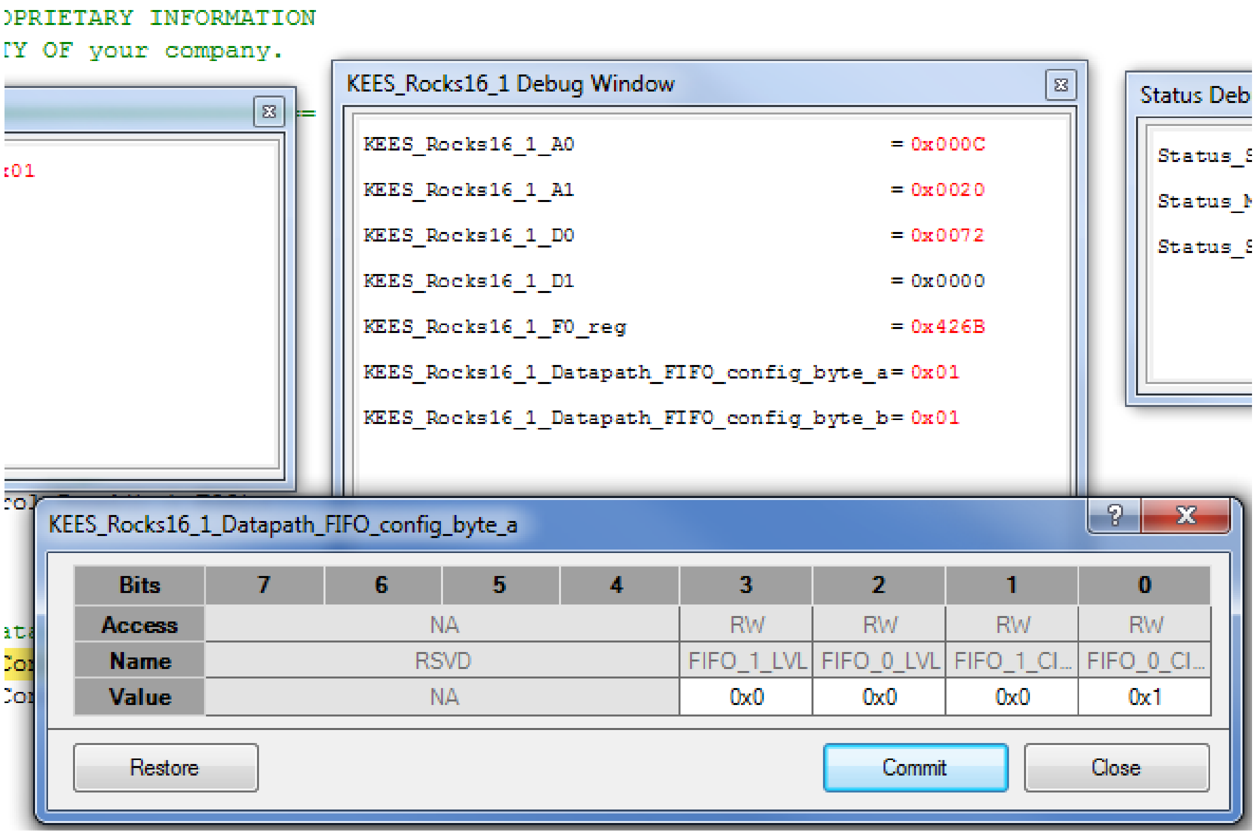
***Note! There a few bugs in Creator 2.2 related to the component debugger.***

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.***

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


KEES_Rocks16_1 Debug Window
Status Deb
KEES_Rocks16_1_Datapath_FIFO_config_byte_a
Restore
Commit
Close

```
KEES_Rocks16_1_A0                          = 0x000C
KEES_Rocks16_1_A1                          = 0x0020
KEES_Rocks16_1_D0                          = 0x0072
KEES_Rocks16_1_D1                          = 0x0000
KEES_Rocks16_1_F0_reg                      = 0x426B
KEES_Rocks16_1_Datapath_FIFO_config_byte_a= 0x01
KEES_Rocks16_1_Datapath_FIFO_config_byte_b= 0x01
```

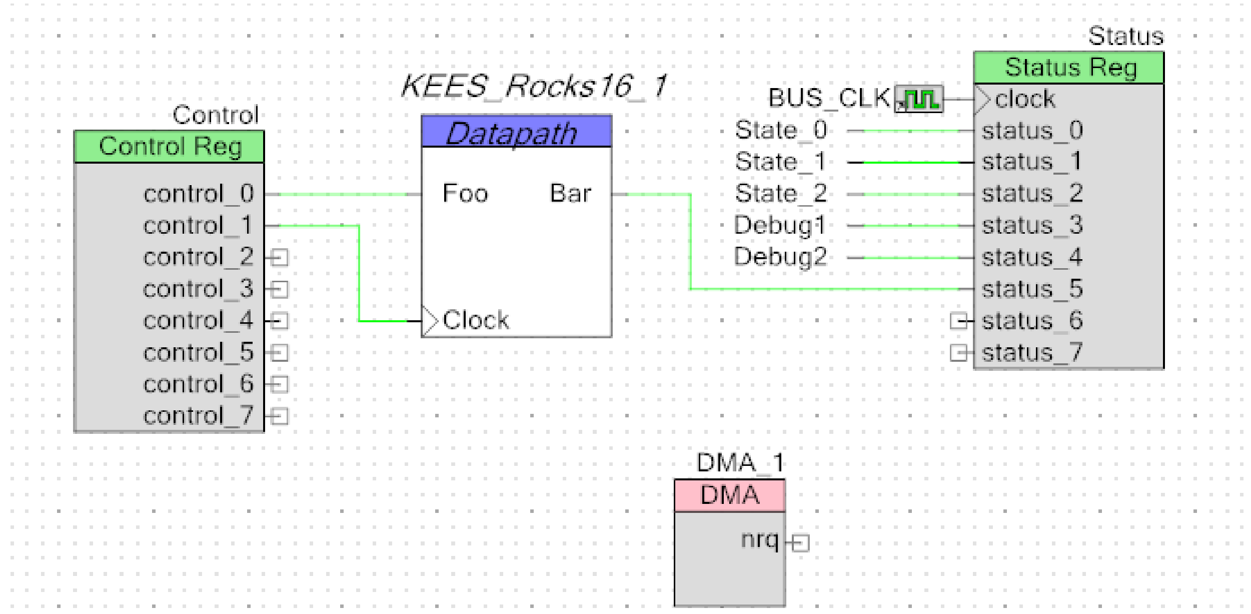
| Bits | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Access | NA | | | | RW | RW | RW | RW |
| Name | RSVD | | | | FIFO_1_LVL | FIFO_0_LVL | FIFO_1_Cl... | FIFO_0_Cl... |
| Value | NA | | | | 0x0 | 0x0 | 0x0 | 0x1 |

Hovering your cursor over a value will show all the options. You can modify the registers directly within these detailed views. Simply enter your new desired value and click Commit.

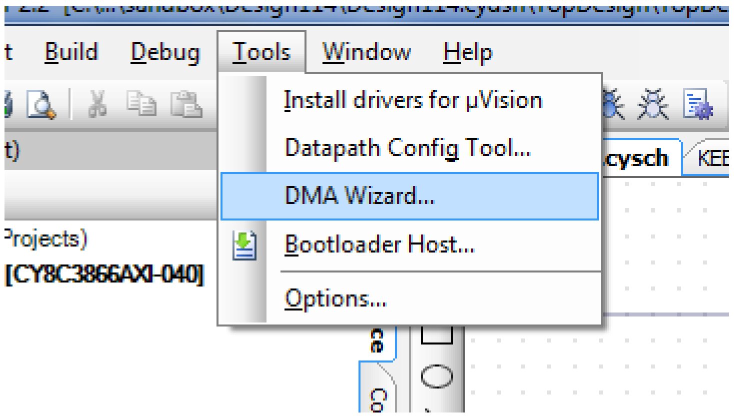
The contents of the A0, A1, D0 and D1 registers can be easily read and modified. Step through your datapath's operation! Debug your design and learn about the capabilities of the part!

## DMA Wizard features

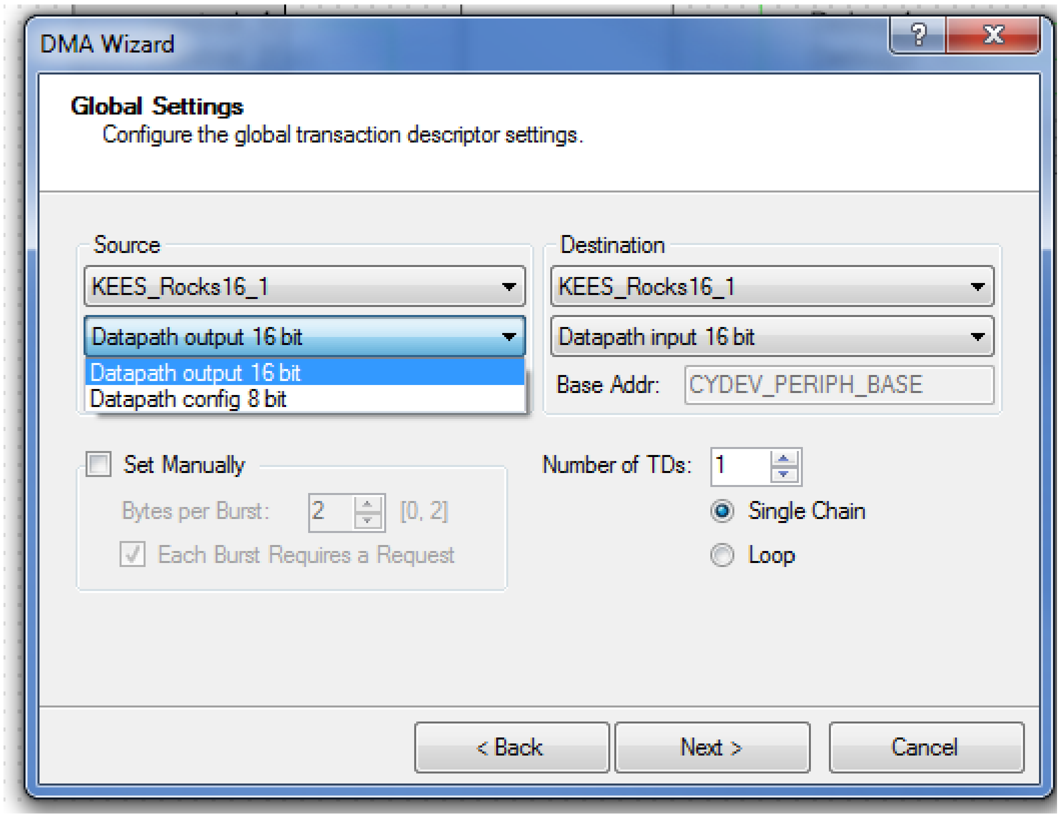
The component also comes with a DMA cpabability file, which allows the component to take advantage of the DMA wizard tool. To use the DMA capabilities, place a DMA component in the schematic.



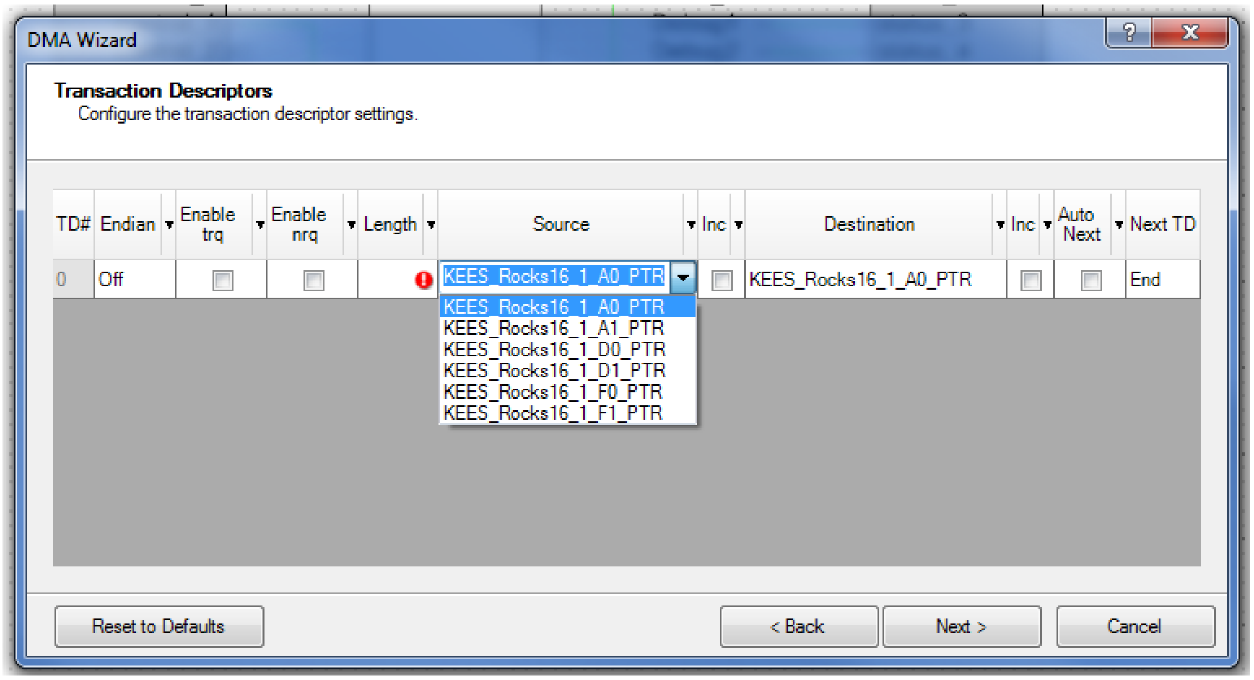
Open the DMA Wizard by selecting Tools->DMA Wizard



The DMA wizard will have register access to all the registers that the component has. There are multiple categories including input and output registers (A0, A1, D0, D1, F0, F1) as well as the FIFO configuration registers for each datapath.



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.

### .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 a stub start function that configures the registers not configured by the datapath configuration tool. Using this component as a starting point, you can create ANY component you desire from the datapaths. Simply modify the component to suit your needs.

Happy Creating!

## Support

PSoC Creator Community Components are developed and supported by the Cypress Developer Community. Go to www.cypress.com/CommunityComponents to discuss this and other Community Components.