Q. What are the factors that decide whether an IP is to be converted to a component or not?

A. There are certain guidelines that are used to decide whether or not an IP should be converted to a component or not. When an IP implements a specific function and has a small set of inputs, outputs and resources; it can be converted to a component.

If one of the following is a characteristic of an IP, then that IP should not be encapsulated to a component:

1. Single Instance fixed function blocks like DelSig ADC is used
2. Too much usage of abundant resources like UDBs, DAC
3. Too much usage of less obvious resources like Flash, Analog routing
4. Operates only under certain conditions like CPU or Bus clock speed
5. Multiple instances of IP cannot be implemented

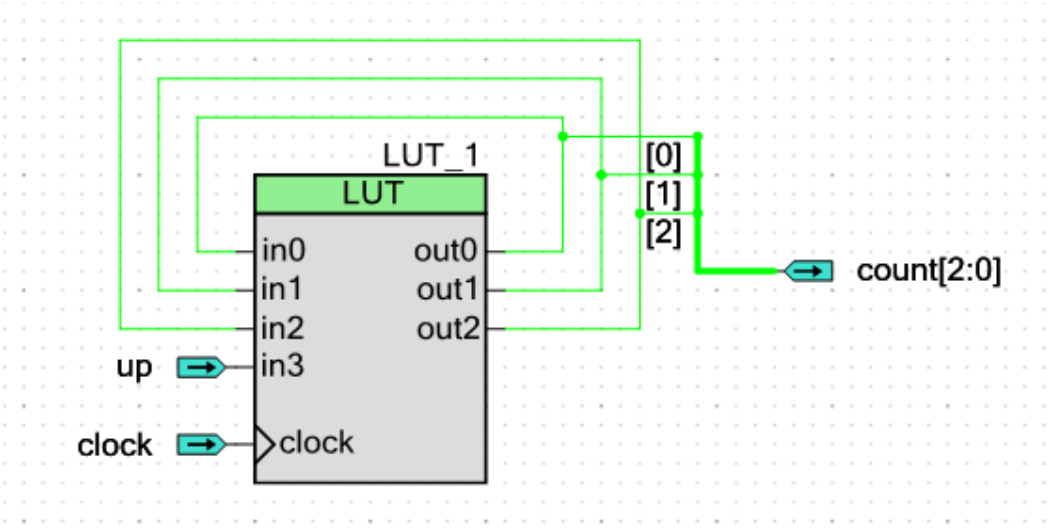
If IP violates any of the rules listed above and still it is encapsulated into a component, then relevant limitations and issues should be communicated to the user as soon as they make an attempt to use them.

Q. What are the different options for creating a component and which one should be used?

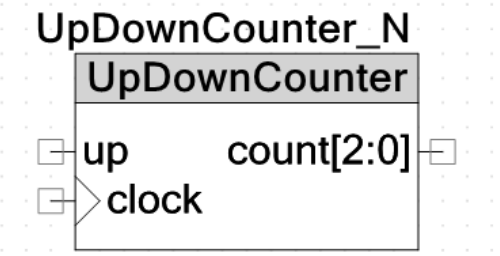
A. There are primarily two options for creating a component in PSoC Creator:

1. Schematic based component: Schematic based components are created by using existing components in PSoC Creator. You may add more than one component into a schematic and then set them to function combined together as a component. You can also create APIs associated with the new component. This method is very useful when a part of the functionality of a component to be created can be achieved by existing component.

   Following is a screenshot of a simple schematic based component. This component uses already existing look up table component along with two input and one output terminal to implement a custom up-down counter.

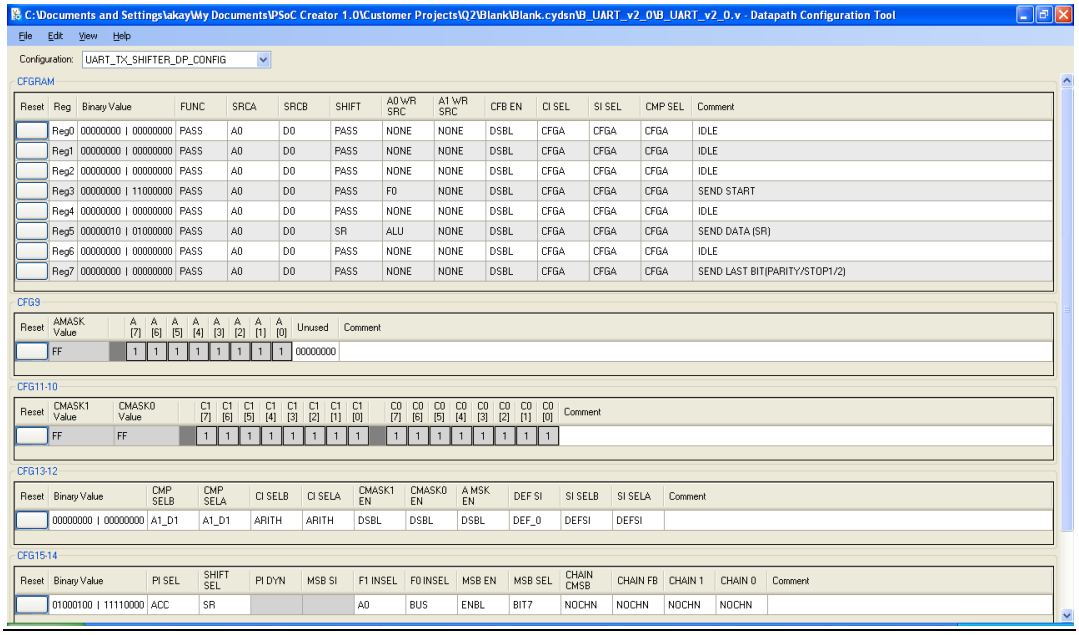


There is a symbol generated for the same component which looks as follows:



Refer to following training modules to learn more about schematic based components.
PSoC Creator 110: Schematic Components
PSoC Creator 111: Component Parameters
PSoC Creator 112: Introduction to Component API Generation

2. Verilog based component: These components are written by user in verilog. These components generally are implemented in the PLDs within the universal digital blocks (UDBs) of PSoC 3/5. User can optionally use the datapaths of UDBs as a part of these components. In order to use the datapath in verilog based components, a separate tool called 'Datapath Configuration Tool' is provided with PSoC Creator. This tool is accessible from the Start menu of Windows at Start → All Programs → Cypress → PSoC Creator 1.0 → Component Development Kit → Datapath Configuration Tool. Following is the screenshot of this tool when a file is opened in it:



Verilog based components offer more flexibility and scope to optimize the resource utilization. But, they demand considerably more rigorous tests as the whole functionality is designed by the user.

For details about verilog synthesis, refer to “Warp Verilog reference guide” available in the Help→Documentation menu of PSoC Creator. For additional information regarding use of datapath in the verilog based components, refer to following training modules.

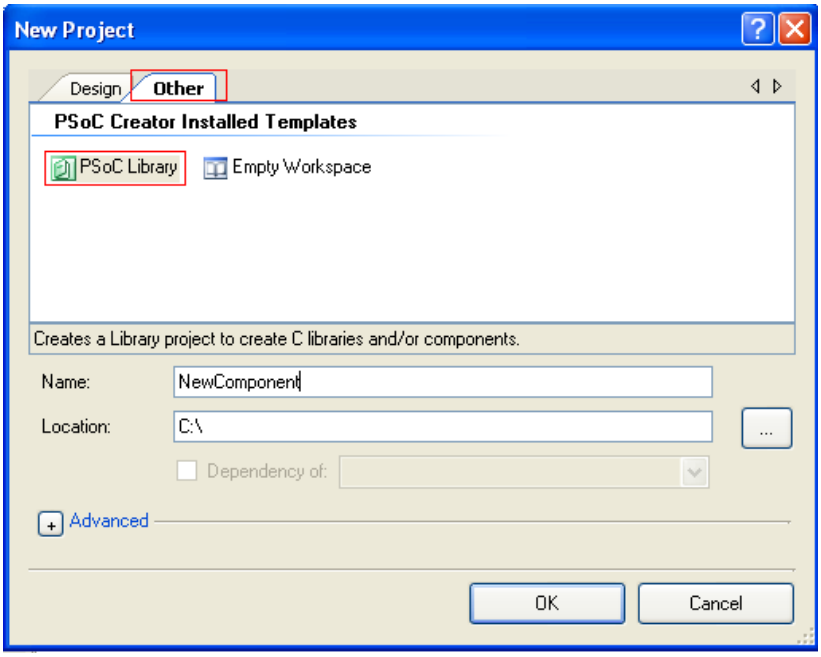
PSoC Creator 113: PLD Based Verilog Components
PSoC Creator 210: Intro to Datapath Components
PSoC Creator 211: Datapath Computation
PSoC Creator 212: Datapath FIFOs
PSoC Creator 213: Multi-Byte Datapath Components
PSoC Creator 214: Datapath API Generation

The two types of component creation methods described above can always be used together to as per requirement. There is no restriction to stick to only one of these methods while developing a component.

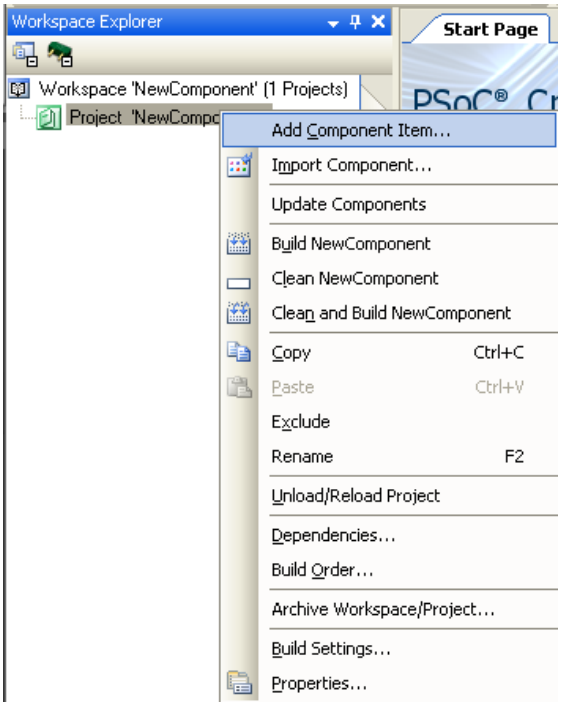
Q. What is the design flow of component development?

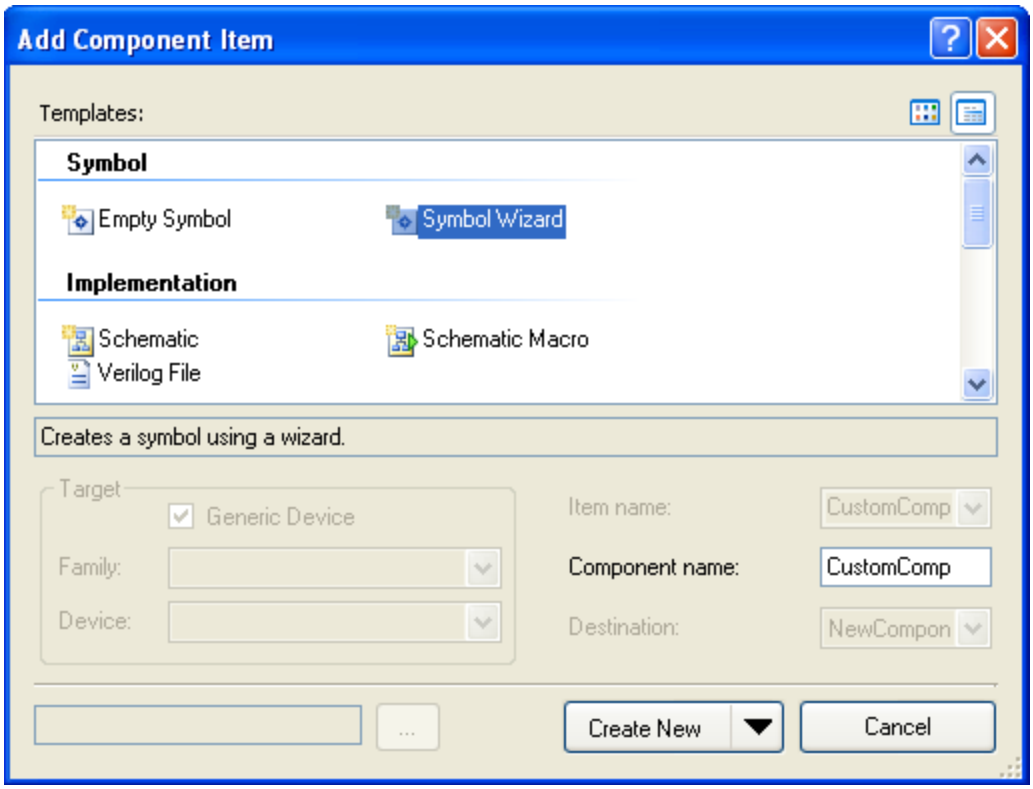
A. A typical design flow for component development is as follows:

1. Create a library project

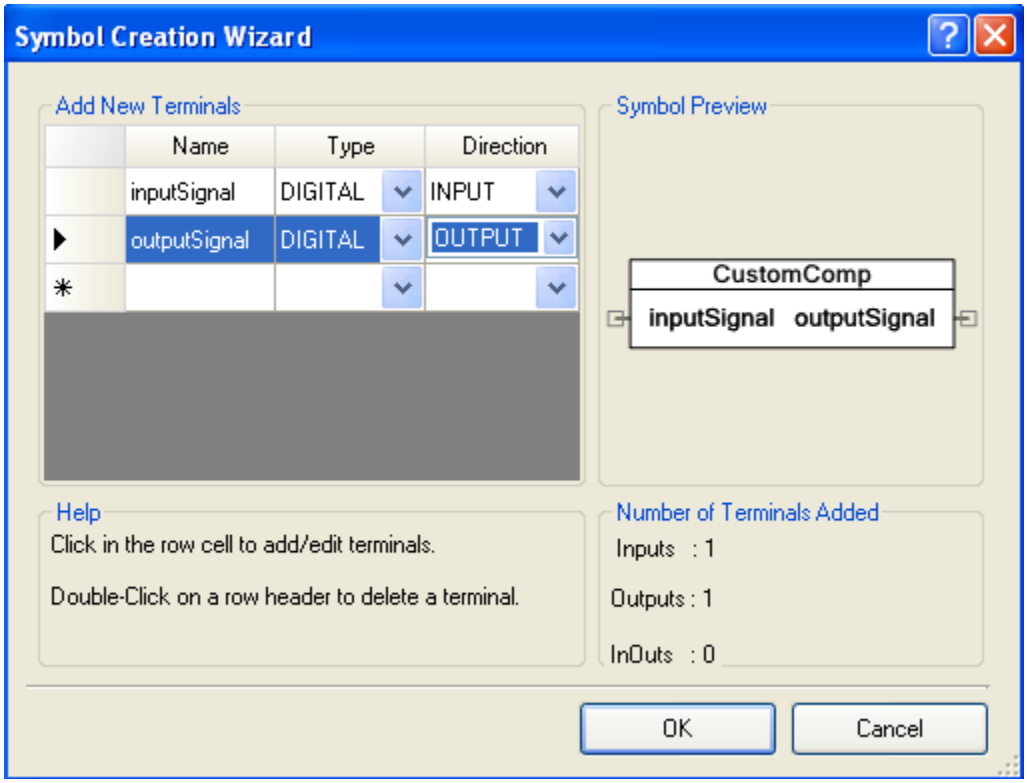


2. Create a component/symbol

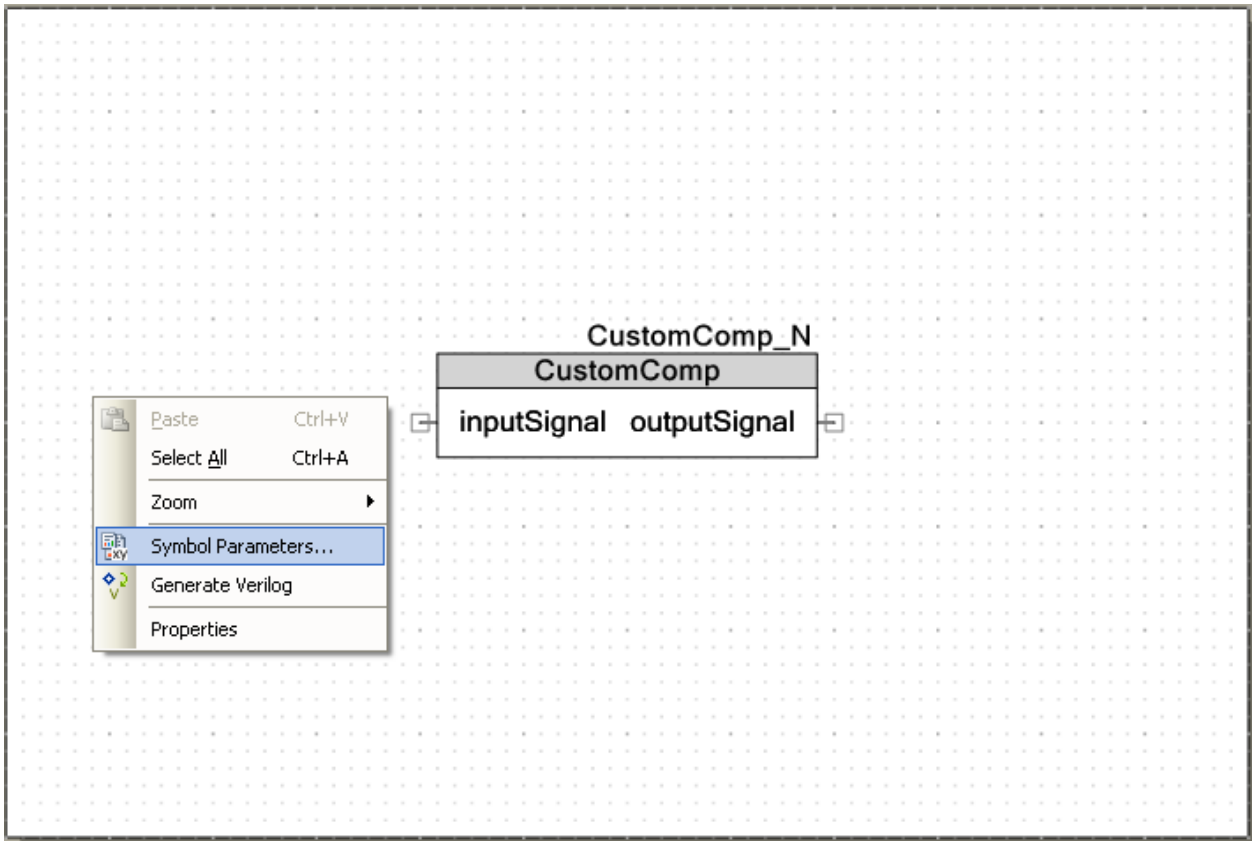


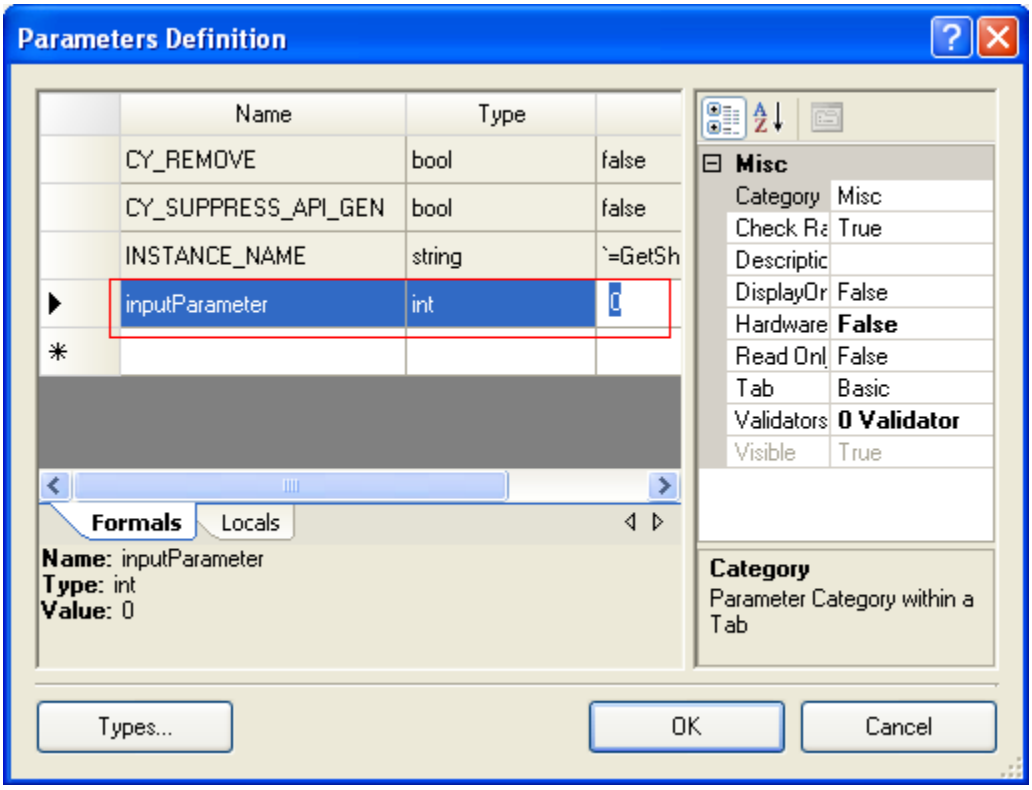


From the symbol wizard, select the input and output terminal which are to be used in the design.



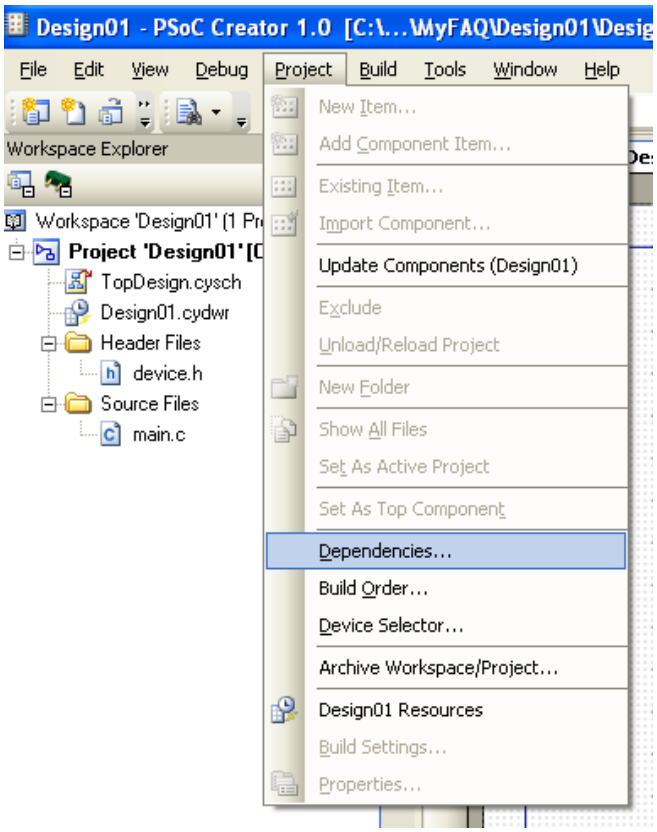
3. Define symbol parameters

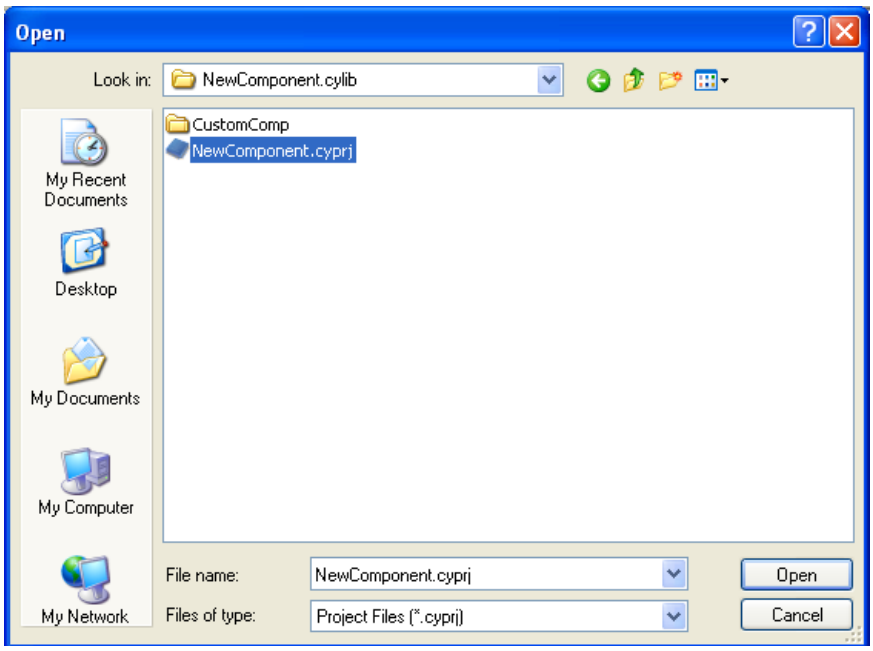


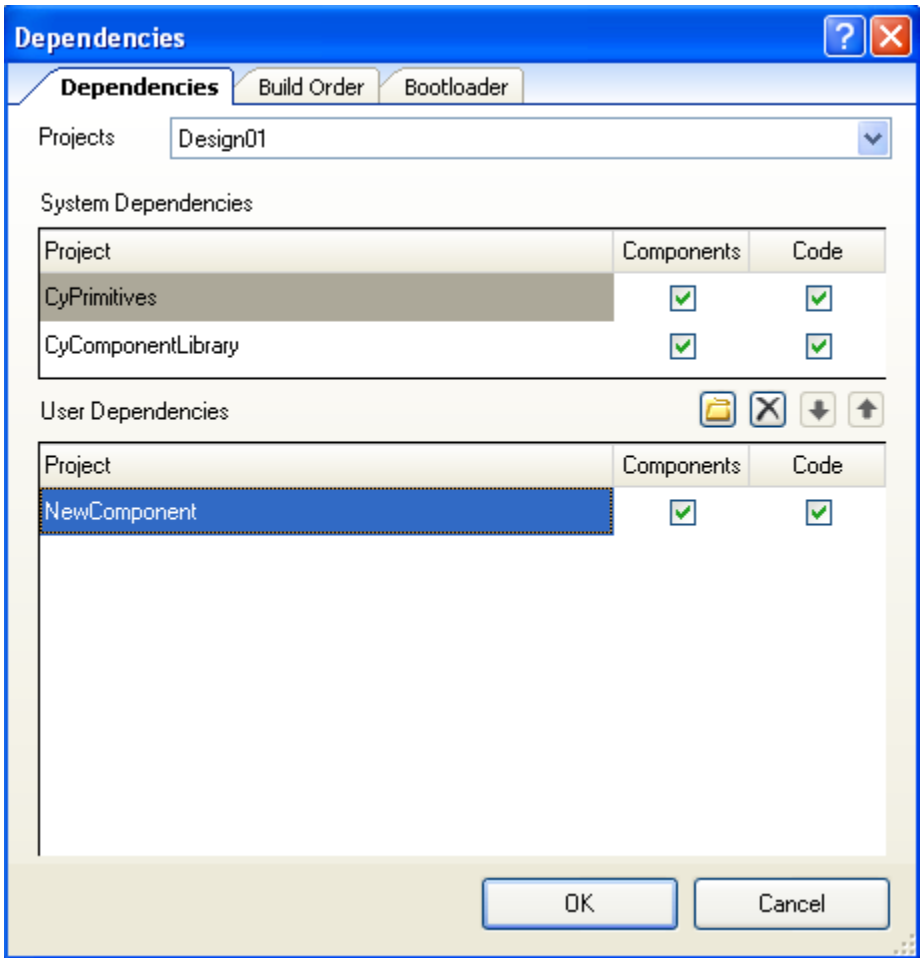


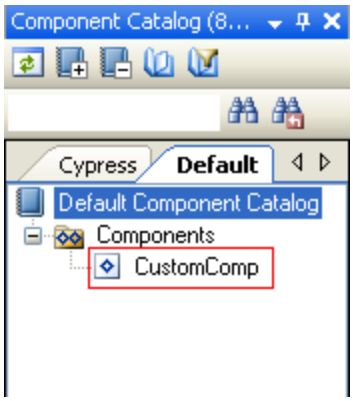
4. Create the implementation
    i. Implement using the schematic based components. Click here for more details.
    ii. Implementation completely in Verilog. Click here for more details.

5. Create API files – Please click here for more details
6. Customize the Component – click here to understand the benefits of customizer.
7. Add Bootloader support (if needed)
8. Create datasheet for the component
9. Do tests and add the specifications in Datasheet
10. Include the component library in a project, Build & test the component




Open
Look in:
NewComponent.cylib
CustomComp
NewComponent.cyprj
My Recent Documents
Desktop
My Documents
My Computer
My Network
File name:
NewComponent.cyprj
Files of type:
Project Files (*.cyprj)
Open
Cancel


Dependencies
Dependencies
Build Order
Bootloader
Projects
Design01
System Dependencies
Project
Components
Code
CyPrimitives
CyComponentLibrary
User Dependencies
Project
Components
Code
NewComponent
OK
Cancel

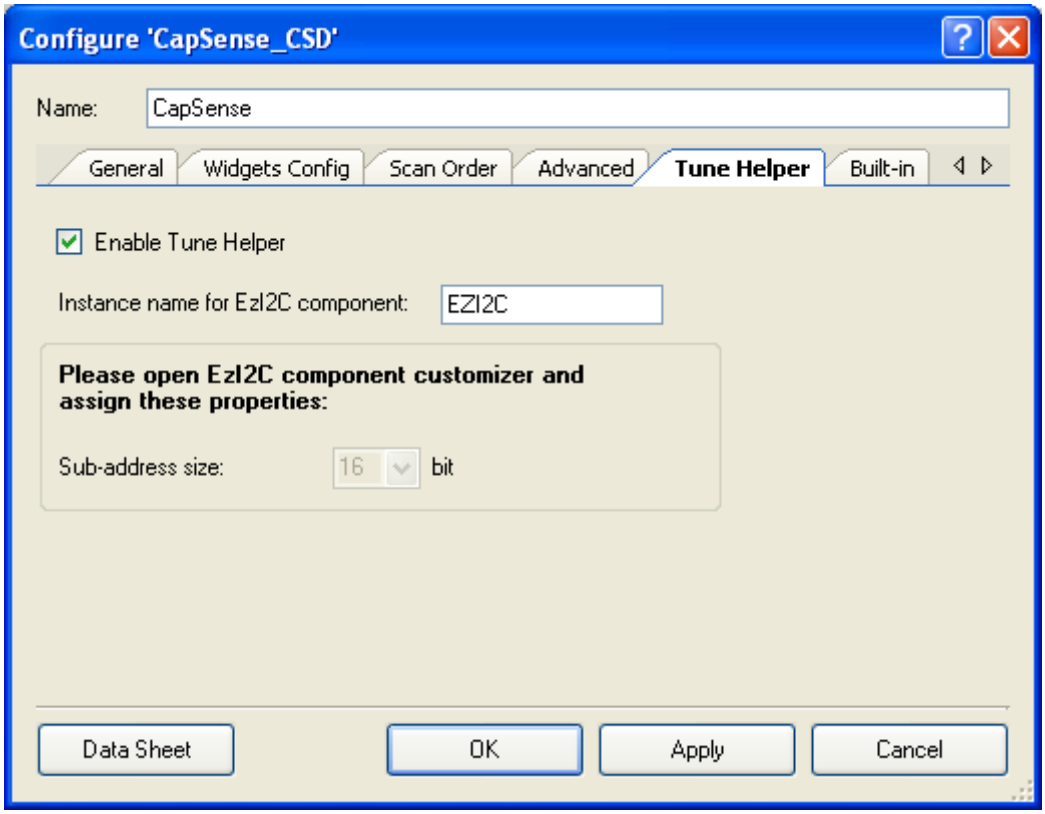


For more details about how to add dependencies please click here.

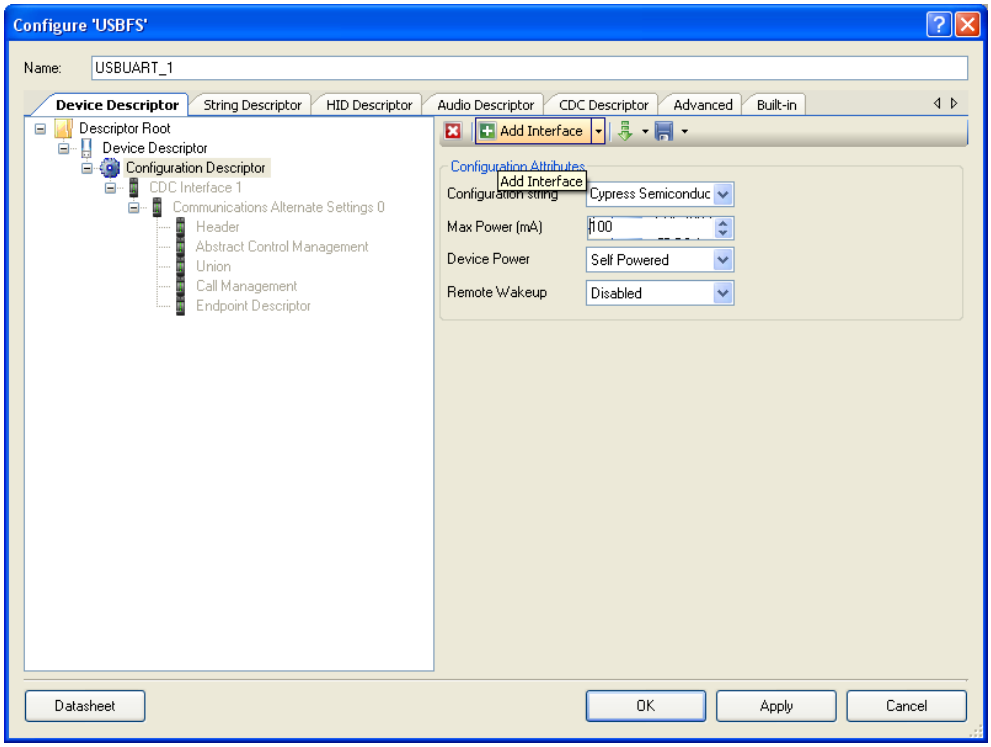
Q. What are the guidelines when it comes to using a single instance fixed function block as a part of the component?

A. Single-instance fixed-function blocks such as DelSig ADC, CAN, USB, and I2C should not be used as a part of the component.

1. Recommended way is to provide a notification to the USER in the configuration window that he needs to place the specified fixed function block with the given name and configuration parameters. Below screen shot provides a similar implementation when EZI2C is required for the operation.



2. If option 1 cannot be used, then an option to add multiple configurations should be provided in the component configuration window to ensure that component should not block the usage of that particular module. For example, below snap shot shows support for adding additional interface for USB when one interface is used for USBUART block.



3. If neither Option 1 nor Option 2 can be used, then the limitations shall be clearly communicated in the PSoC Creator component such that the user becomes aware of those issues as soon as an attempt is made to use the component.