- **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.

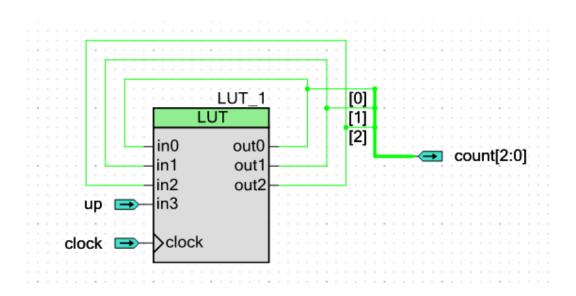
If one of the following is a characteristic of an IP, then that IP should not be encapsulated to a component:

- 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

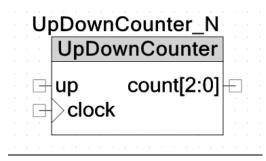
If IP violates any of the rules listed above and still it is encapsulated into a component, then relevant limitations and issues should be communicated to the user as soon as they make an attempt to use them.

- **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. <u>Schematic based component:</u> 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. <u>PSoC Creator 110: Schematic Components</u> <u>PSoC Creator 111: Component Parameters</u> PSoC Creator 112: Introduction to Component API Generation

2. <u>Verilog based component:</u> 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:

	<u>E</u> dit ⊻iew <u>H</u> elp											
Configu	ration: UART_TX_SHIFTE	R_DP_CONF	G 🗸									
FGRA	м											
Reset	Reg Binary Value	FUNC	SRCA	SRCB	SHIFT	A0 WR A' SRC SI	1 WR CFB	EN CI SEL	SI SEL	CMP SEL	Comment	
	Reg0 00000000   000000	00 PASS	A0	DO	PASS	NONE N	DNE DSBI	L CFGA	CFGA	CFGA	IDLE	
	Reg1 00000000   000000	00 PASS	A0	DO	PASS	NONE N	DNE DSBI	L CFGA	CFGA	CFGA	IDLE	
	Reg2 00000000   000000	00 PASS	A0	DO	PASS	NONE N	DNE DSBI	L CFGA	CFGA	CFGA	IDLE	
	Reg3 00000000   110000	00 PASS	A0	DO	PASS	FO N	DNE DSBI	L CFGA	CFGA	CFGA	SEND START	
	Reg4 00000000   000000		A0	DO	PASS	NONE N	DNE DSBI	L CFGA	CFGA	CFGA	IDLE	
	Reg5 00000010   010000	DO PASS	A0	DO	SR	ALU N	DNE DSBI	L CFGA	CFGA	CFGA	SEND DATA (SF	a)
	Reg6 00000000   000000	00 PASS	A0	DO	PASS	NONE N	DNE DSBI	L CFGA	CFGA	CFGA	IDLE	
	Reg7 00000000   000000	00 PASS	A0	DO	PASS	NONE N	DNE DSBI	L CFGA	CFGA	CFGA	SEND LAST BIT	(PARITY/STOP1/2)
Reset	AMASK A A Value [7] [6]			A [0] Unused								
CFG9 Reset	AMASK A A Value [7] [6] FF 1 1	A A / [5] [4] [ 1 1										
Reset	Value (7) [6] FF 1 1 10	1 1	1 1	[0] Unused	10							
Reset	Value [7] [6] FF 1 1 10	1 1	1 1	[0] Unused		CO CO CO [7] [6] [5]	C0 C0 C0 (4) [3] [2]	CO CO Co	omment			
Reset CFG11- Reset	Value (7) (6) FF 1 1 10 CMASK1 CMASK0	1 1	1 1	[0] Unused 1 0000000 1 C1 C1 1 [3] [2]	10	C0 C0 C0 (7) [6] [5] 1 1 1 1	C0 C0 C0 [4] [3] [2] 1 1 1 1	C0 C0 C0 [1] [0] Cc	omment			
Reset FG11- Reset	Value         [7]         [6]           FF         1         1           10         CMASK1         CMASK0           Value         Value         Value           FF         FF         FF	1 1		[0] Unused 1 0000000 1 C1 C1 1 [3] [2]	C1 C1 [1] [0]			CO CO Co (1) [O] Co	omment			
Reset CFG11- Reset	Value         [7]         [6]           FF         1         1           10         CMASK1         CMASK0           Value         Value         Value           FF         FF         12	1 1 C1 (7) 1		(0) Unused 1 0000000 1 C1 C1 1 [3] [2] 1 1 1	0 C1 C1 [1] [0] 1 1	1 1 1	1 1 1	1 1				
Reset CFG11- Reset	Value         [7]         [6]           FF         1         1         1           10         CMASK1         CMASK2         CMASK2           Value         Value         FF         FF           12         Binary Value         SE         SE	1 1 C1 (7) 1 P LB SE		[0] Unused 1 0000000 1 C1 C1 3 [3] [2] 1 1 1 ELB CI SE	C1 C1 (1) [0] 1 1 LA CMASI EN	1 1 1 1 CMASKO	1 1 1 A MSK EN	DEF SI SI	I SELB SI S		nent	
Reset CFG11- Reset	Value         [7]         [6]           FF         1         1           10         CMASK1         CMASK0           Value         Value         Value           FF         FF         1           12         D         D	1 1 C1 (7) 1 P LB SE		[0] Unused 1 0000000 1 C1 C1 3 [3] [2] 1 1 1 ELB CI SE	C1 C1 (1) [0] 1 1 LA CMASI EN	1 1 1 (1 CMASKO	1 1 1	DEF SI SI			nent	
Reset CFG11- Reset	Value         [7]         [6]           FF         1         1         1           10         CMASK1         CMASK1         Value           Value         Value         Value         1           12         Binary Value         0         0         55           000000000000000000000000000000000000	1 1 C1 (7) 1 P LB SE		[0] Unused 1 0000000 1 C1 C1 3 [3] [2] 1 1 1 ELB CI SE	C1 C1 (1) [0] 1 1 LA CMASI EN	1 1 1 1 CMASKO	1 1 1 A MSK EN	DEF SI SI	I SELB SI S		nent	
Reset CFG11- Reset CFG13- Reset	Value         [7]         [6]           FF         1         1           10         CMASK1         CMASK0           Value         FF         FF           12         Binary Value         CS           00000000   00000000   00000000   11         A1	1 1 C1 (7) 1 P LB SE	C1 C1 C [5] [5] [7 1 1 1 C1 C1 C [5] [7] [7 [7] [7] [7] [7] [7] [7] [7] [7] [7] [7]	(0) Unused 1 0000000 1 C1 C1 1 [3] [2] 1 1 1 ELB CI SE FH ABIT	0 C1 C1 (1) [0] 1 1 LA CMASI EN 4 DSBL	1 1 1 CMASKO EN DSBL	A MSK EN DSBL	1 1 DEF SI SI DEF_0 DE	I SELB SI S EFSI DEF			Comment

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 $\rightarrow$ 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

New Project		? 🗙
Design 0	ther	۹ ۵
PSoC Creato	r Installed Templates	
PSoC Librar	y 🔟 Empty Workspace	
0		
Creates a Library (	project to create C libraries and/or components.	
Name:	NewComponent	
Location:	C:V	
	Dependency of:	
<b>—</b> • • •		
+ Advanced -		
	OK Can	el .:

2. Create a component/symbol

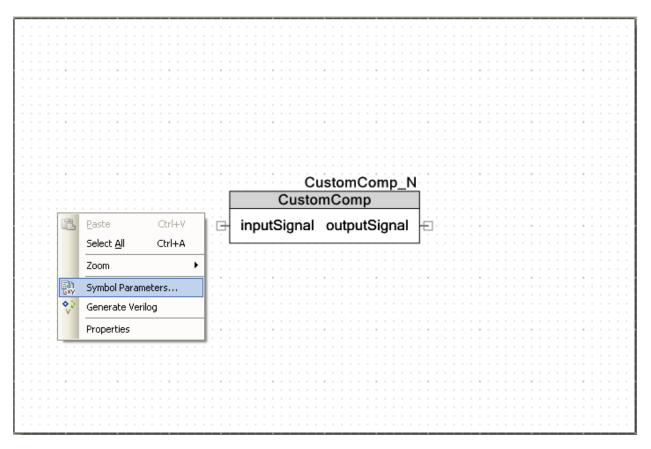
Workspace Explorer	🗸 🕂 🗙 🖌 Start Page		
· 🖫 🕿			
Workspace 'NewComponent'	(1 Projects) PSoC® Cr		
Project 'NewCompo	Add Component Item		
	Import Component		
	Update Components		
<b>***</b>	Build NewComponent		
	Clean NewComponent		
11 A	Clean and Build NewComponent		
E	Copy Ctrl+C		
	Paste Ctrl+V		
	E <u>x</u> clude		
	Rename F2		
	Unload/Reload Project		
	Dependencies		
	Build <u>O</u> rder		
	Archive Workspace/Project		
	Build Settings		
	Properties		

Add Component Item			? 🗙
Templates:			
Symbol			
🍖 Empty Symbol	🌆 Symbol Wiza	rd	=
Implementation			
🔣 Schematic 🎬 Verilog File	🚯 Schematic M	lacro	
			<u> </u>
Creates a symbol using a wizard.			
Target		Item name:	CustomComp 🖌
Family:	~	Component name:	CustomComp
Device:	~	Destination:	NewCompon 🔽
		Create New 🔻	Cancel

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

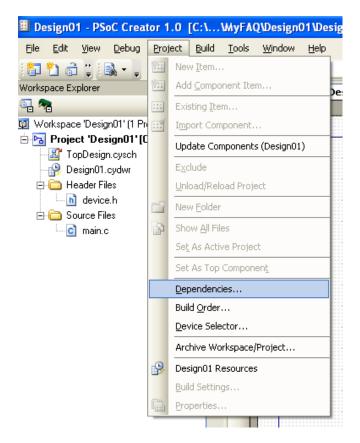
Sy	mbol C	reation Wiz	ard				? 🛛
	-Add Ne	ew Terminals —					Symbol Preview
		Name	Туре		Directio	n	
		inputSignal	DIGITAL	*	INPUT	*	•
	۱.	outputSignal	DIGITAL	~	OUTPUT	~	
	*			*		¥	
							□ inputSignal outputSignal □
l							
ſ	Help	the row cell to a			L_		Number of Terminals Added
							Inputs : 1
	Double	Click on a row h	neader to de	elete	a terminal.		Outputs : 1
l							InOuts : 0
	l					OK Cancel	

## **3.** Define symbol parameters



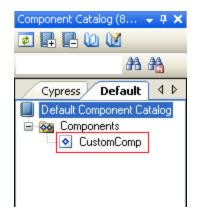
P	Parameters Definition						
		Name	Туре				
		CY_REMOVE	bool	false	🗆 Misc		
		CY_SUPPRESS_API_GEN	bool	false	Category Misc Check Ra True		
		INSTANCE_NAME	string	`=GetSh	Descriptic		
	•	inputParameter	int	C	DisplayOr False		
	*				Hardware False Read Onl False		
					Tab Basic		
					Validators <b>0 Validator</b> Visible True		
	<			>	VISIDIE TTUE		
		rmals Locals		4 ۵			
	Name: Type: i Value:				<b>Category</b> Parameter Category within a Tab		
	Types OK Cancel						

- 4. Create the implementation
  - i. Implement using the schematic based components. Click <u>here</u> for more details.
  - **ii.** Implementation completely in Verilog. Click <u>here</u> for more details.
- 5. Create API files Please click <u>here</u> for more details
- **6.** Customize the Component click <u>here</u> 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:	n: 🗀 NewComponent.cylib 🔹 🕜 🏂 🗁 🖽 -	
My Recent Documents	CustomComp NewComponent.cyprj	
My Documents		
My Computer		
	File name: NewComponent.cyprj	Open
My Network	Files of type:         Project Files (*.cyprj)	Cancel

Dependencies		? 🗙
Dependencies Build Order Bootloader		
Projects Design01		*
System Dependencies		
Project	Components	Code
CyPrimitives		
CyComponentLibrary		
User Dependencies		X 🖡 🕈
Project	Components	Code
NewComponent		<ul><li>✓</li></ul>
OK		Cancel



For more details about how to add dependencies please click <u>here</u>.

- **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.

Configure 'CapSense_CSD'	?×
Name: CapSense	
General Widgets Config Scan Order Advanced Tune Helper Built-in	4 Þ
Enable Tune Helper	
Instance name for EzI2C component: EZI2C	
Please open EzI2C component customizer and assign these properties:	
Sub-address size: 16 🖌 bit	
Data Sheet OK Apply Cance	a

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.

Configure 'USBFS'		? ×
Name: USBUART_1		
Name: USBUART_1	Audio Descriptor CDC Descriptor Advanced Built-In Configuration Attributes Configuration Strift Cypress Semiconduc Max Power (mA) 100 Vevice Power Self Powered Remote Wakeup Disabled Vevice Power Self Powered	4 4
Datasheet	OK Apply Cancel	

**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.