Q. What is meant by component customization and how it is advantageous?

A. Customizing components refers to the mechanism of allowing custom code (C#) to modify the default behavior of an instantiated component within PSoC Creator. This custom code can customize the configure dialogue, customize symbol shapes, terminal names, counts or configuration based on parameter values. It can generate custom verilog, C or assembly code and interact with the clocking system.

The icyinstancecustomizer.cs C# source file provides the definitive parameters and return values for the methods provided in the customizer interfaces. The cydsextensions project contains the necessary source code for customization. This project is included with PSoC Creator, and the documentation is located in the PSoC Creator Customization API Reference Guide.

Q. How to estimate the resources like macro cells, status registers etc. used by a particular component?

A. With the current version of PSoC Creator, calculation of resource usage has to be done manually. Following steps need to be performed in order to estimate the resource usage:

1. <u>For resource usage like: Macro cells, status registers etc.</u>
   - **i.** Create a blank project and build the same to get a note of the resources used.
   - **ii.** Go to the Results tab in workspace explorer and open <project_name>.rpt file.
   - **iii.** Take a note of resources used from the Technology mapping summary as shown below:

```
<CYPRESSTAG name="Technology mapping summary" expanded>

-------------------------------------------------------------
Technology mapping summary
-------------------------------------------------------------

                Resource Type : Used : Free :  Max :  % Used
=============================================================
Digital domain clock dividers :    0 :    8 :    8 :   0.00%
 Analog domain clock dividers :    0 :    4 :    4 :   0.00%
                         Pins :    3 :   69 :   72 :   4.17%
                   Macrocells :    0 :  192 :  192 :   0.00%
                Unique Pterms :    0 :  384 :  384 :   0.00%
              Datapath Cells  :    0 :   24 :   24 :   0.00%
                Status Cells  :    0 :   24 :   24 :   0.00%
         Control/Count7 Cells :    0 :   24 :   24 :   0.00%
                   Sync Cells :    0 :   96 :   96 :   0.00%
                         Drqs :    0 :   24 :   24 :   0.00%
                   Interrupts :    0 :   32 :   32 :   0.00%
            DSM Fixed Blocks  :    0 :    1 :    1 :   0.00%
```

```
        VIDAC Fixed Blocks :    0 :    4 :    4 :   0.00%
           SC Fixed Blocks :    0 :    4 :    4 :   0.00%
  Comparator Fixed Blocks :    0 :    4 :    4 :   0.00%
       Opamp Fixed Blocks :    0 :    4 :    4 :   0.00%
         CapSense Buffers :    0 :    2 :    2 :   0.00%
         CAN Fixed Blocks :    0 :    1 :    1 :   0.00%
   Decimator Fixed Blocks :    0 :    1 :    1 :   0.00%
         I2C Fixed Blocks :    0 :    1 :    1 :   0.00%
      Timer Fixed Blocks :    0 :    4 :    4 :   0.00%
         DFB Fixed Blocks :    0 :    1 :    1 :   0.00%
         USB Fixed Blocks :    0 :    1 :    1 :   0.00%
         LCD Fixed Blocks :    0 :    1 :    1 :   0.00%
        EMIF Fixed Blocks :    0 :    1 :    1 :   0.00%
   LPF Fixed Blocks :    0 :    2 :    2 :   0.00%
```

iv. Place the component and build the project. Make sure that you resolve all the errors before proceeding to next step.

v. Once this is done again go to the same rpt file and take a note of the Technology mapping summary. Difference of the two indicates the resources used by your component.

```
<CYPRESSTAG name="Technology mapping summary" expanded>

-----------------------------------------------------------
Technology mapping summary
-----------------------------------------------------------

                Resource Type : Used : Free :  Max :  % Used
===========================================================
Digital domain clock dividers :    3 :    5 :    8 :  37.50%
 Analog domain clock dividers :    0 :    4 :    4 :   0.00%
                         Pins :   15 :   57 :   72 :  20.83%
                   Macrocells :   34 :  158 :  192 :  17.71%
                Unique Pterms :   62 :  322 :  384 :  16.15%
                  Total Pterms :   77 :      :      :
               Datapath Cells :    4 :   20 :   24 :  16.67%
                 Status Cells :    4 :   20 :   24 :  16.67%
         Control/Count7 Cells :    8 :   16 :   24 :  33.33%
                   Sync Cells :    0 :   80 :   80 :   0.00%
                        Drqs :    0 :   24 :   24 :   0.00%
                  Interrupts :    7 :   25 :   32 :  21.88%
            DSM Fixed Blocks :    0 :    1 :    1 :   0.00%
          VIDAC Fixed Blocks :    0 :    4 :    4 :   0.00%
             SC Fixed Blocks :    0 :    4 :    4 :   0.00%
      Comparator Fixed Blocks :    0 :    4 :    4 :   0.00%
           Opamp Fixed Blocks :    0 :    4 :    4 :   0.00%
             CapSense Buffers :    0 :    2 :    2 :   0.00%
             CAN Fixed Blocks :    0 :    1 :    1 :   0.00%
       Decimator Fixed Blocks :    0 :    1 :    1 :   0.00%
             I2C Fixed Blocks :    0 :    1 :    1 :   0.00%
          Timer Fixed Blocks :    1 :    3 :    4 :  25.00%
             DFB Fixed Blocks :    0 :    1 :    1 :   0.00%
             USB Fixed Blocks :    0 :    1 :    1 :   0.00%
             LCD Fixed Blocks :    0 :    1 :    1 :   0.00%
```

```
EMIF Fixed Blocks :    0 :    1 :    1 :   0.00%
 LPF Fixed Blocks :    0 :    2 :    2 :   0.00%
 SAR Fixed Blocks :    0 :    2 :    2 :   0.00%
```

2. For memory usage

- **i.** Compile an empty project and take a note of the memory used.
- **ii.** Place the component and call the required APIs and check the memory usage.
- **iii.** Difference of the two is the memory used by the component for the APIs you have called.

P.S.: Memory usage will vary based on the APIs called because unused APIs will not be allocated any space in the memory. It'll also depend on the optimization level used for the compilation.

Q. How to test a component?

A. Reusable designs are encapsulated as components in a PSoC Creator library project. However neither a library project nor a component by itself is very useful. Library projects cannot be built, programmed or tested. So one or more standard projects should be developed along with the component, for the purpose of testing or demonstrating the functionality of the reusable design in that component.

The reference component should be included with each of its parameters set to as many different settings as possible. Note that in order to do this either multiple instances of the component may need to be used or multiple test / demo projects may need to be created. All functions in the component's API should be called at least once. All macros should be used at least once. As much as possible, the test projects should support both PSoC 3 and PSoC 5, in various configurations. For example, standard and bootloadable, debug and release, different PSoC 5 compilers, and different compiler optimization settings.