1. What are the three kinds of parameters?
    - Parameters: Fixed constants in a module that can be overridden during instantiation.
    - Local Parameters (localparam): Constants that cannot be overridden during instantiation.
    - Spec Parameters (specparam): Special parameters used inside specify blocks for timing checks.
2. How can I override a module's parameter values during instantiation?
   Parameters can be overridden in two ways:
    - Named parameter override: module_name #(.PARAM1(value1), .PARAM2(value2)) instance_name(...);
    - Positional parameter override:module_name #(value1, value2) instance_name(...);
3. What are the rules governing parameter assignments?
    - Parameters must be assigned constant values (no real-time calculations).
    - Parameters can be assigned expressions, but they should be constant at compile time.
    - Parameters can be overridden during instantiation unless they are declared as localparam.
    - Overriding parameters during instantiation does not affect the original module definition.
4. How do I prevent selected parameters of a module from being overridden during instantiation?
   Use localparam instead of parameter. localparam creates a constant that cannot be changed during module instantiation
5. What are the differences between using 'define and using either parameter or defparam for specifying variables?

- define: A preprocessor directive used to define macros. It's global, and once defined, it can be used throughout the file or included files.
    - Pros: Simple to use, no scope restrictions.
    - Cons: Can lead to unexpected results since it's global and can be redefined elsewhere in the code.
- parameter/defparam: Used for declaring constants within the scope of a module. parameter values can be overridden at instantiation, while defparam explicitly modifies a parameter value at runtime.
    - Pros: Scoped to the module, more predictable than define.
    - Cons: defparam can be hard to trace when used in large designs.

6. What are the pros and cons of specifying the parameters using defparam construct vs specifying during instantiation?
   defparam:

    - Pros: Provides a way to override parameters without modifying the instantiation code.
    - Cons: Can be hard to trace because it changes parameters in a separate section from the instantiation.

   Instantiation-based parameter specification:

    - Pros: Clear, and easy to understand, as all parameters are specified directly during instantiation.
    - Cons: May require changes to multiple instances if you want to override a parameter in many places.
7. What is the difference between specparam and parameter constructs?
    - parameter: Used to define constants within a module, typically for functionality or configuration.
    - specparam: Used inside specify blocks, specifically for timing checks or delay values in ASICs or FPGAs. It's used only for timing and electrical constraint specification.
8. What are derived parameters? When are derived parameters useful, and what are their limitations?
   Derived parameters are parameters calculated from other parameters. They help ensure that values remain consistent across a design.
   Derived parameters are very useful for:

- Consistency Across Designs: Derived parameters ensure consistency between related values across a design. If a base parameter is changed (e.g., WIDTH), all derived parameters update automatically, avoiding manual recalculations and reducing errors.
- Easier Code Maintenance: By using derived parameters, you reduce redundancy in parameter declarations. Changes in one place will propagate throughout the module, making the code easier to manage and maintain.
- Customization and Flexibility: They enable flexibility in the design, allowing parameters to be adjusted based on high-level configuration, while derived values are automatically updated.
- Avoid Hardcoding: Derived parameters allow more abstract design by avoiding hardcoded values in multiple places.

Limitations:

- Limited Override Capabilities: Derived parameters can only be overridden by changing the base parameters. You cannot directly override derived parameters during instantiation.
- Complexity in Large Designs: If the derived parameter expressions become too complex, it may become harder to track how a particular value is calculated, making the design less readable and harder to debug.
- Interdependencies: Derived parameters introduce dependencies between parameters. A change in one base parameter might have unintended effects on other parts of the module if not carefully designed.
- Static Nature: Derived parameters are evaluated at compile-time. You cannot modify derived parameters dynamically during simulation, limiting flexibility in certain dynamic use cases.

9. Explain the overriding of parameters using defparam with an example.

```
module my_module #(parameter SIZE = 8)(...);
  ...
endmodule

module top;
  my_module my_instance(...);
  defparam my_instance.SIZE = 16; // Overrides SIZE to 16 for my_instance
endmodule
```

10. Explain the meaning of the following code snippets:
    a.
    ```
    specify
            Specparam t_rise=200,t_fal=150
    endspecify
    ```
    This snippet defines timing parameters t_rise (rise time) and t_fal (fall time) used for timing checks in ASIC designs.
    b. Parameter BUS_WIDTH=32,DATA_WIDTH=64
    This defines two parameters, BUS_WIDTH and DATA_WIDTH, with values 32 and 64 respectively. These parameters are typically used to configure the size of data buses or data paths in a design.