Recommendations For Source Code Improvement

Project: RMC75E TEST BENCH

Author: Satchel Hamilton

Company: Delta Motion

Date: 8/11/2023

Last updated: August 11th, 2023

Here are some recommendations to improve the RMC75E source code:

1. **Modularization and Structuring:**
    - Break down the code into smaller, well-defined modules. This will improve readability, reusability, and maintenance.
    - Avoid “spaghetti code” or “ping-pong logic” that sends the developer back and forth between different process blocks when trying to follow the logical flow of the program.
2. **Consistent Naming Conventions:**
    - Follow consistent naming conventions for signals, variables, constants, and entities. This improves code readability and maintainability.
3. **Comments and Documentation:**
    - Add comments to describe the purpose and functionality of each module, process, and signal. This will make the code more understandable for those tasked with future maintenance.
4. **Use Enumerated Types:**
    - Instead of using constants for state encoding, consider using enumerated types. Enumerated types make the code more readable and help prevent errors related to state assignments.
5. **Signal and Variable Initialization:**
    - Initialize signals and variables where applicable, especially after a reset condition. This ensures predictable behavior during the initialization phase.
6. **Consolidate Similar Processes:**
    - In some cases, there are multiple processes that could be consolidated into one. This can simplify the code and make it easier to understand.
7. **Extract Common Logic:**
    - If one notices repeating logic patterns, consider extracting those into functions or procedures. This can reduce redundancies and make the code more efficient.
8. **Use Conditional Signal Assignments:**

- In cases where we have multiple concurrent signal assignments depending on conditions, consider using conditional signal assignments. This can make the code more concise and readable.

9. **Use Functions for Complex Expressions:**
    - If you have complex calculations or expressions, use functions to encapsulate them. This enhances readability and allows you to reuse those calculations if needed.
10. **Avoid Long Signal Assignments:**
    - If signal assignments become too long, it may be a sign that the code might be getting too complex. Consider breaking down complex operations into smaller steps or processes.
11. **Minimize the Use of "else" in Case Statements:**
    - Instead of using "else" in case statements, consider using "when others" and assigning a default value. This can make it easier to detect missing cases.
12. **Use VHDL Libraries Appropriately:**
    - Make sure to use the appropriate VHDL libraries for the functions and types being used. In the current code, there are references to **std_logic_unsigned** and **std_logic_arith**, which are not standard libraries in modern VHDL. Use **ieee.numeric_std** for arithmetic operations.
13. **Testbench Development:**
    - Continue developing testbenches to verify the functionality of the modules. Testbenches are crucial for catching errors and ensuring correct behavior.
14. **Check for Latches:**
    - Be cautious about unintentional latches. Ensure that all assignments are complete for all cases and that signals are not left unintentionally unchanged.
15. **Use Case Statements Instead of If-Else Chains:**
    - In situations where we have multiple conditions and actions to take, consider using case statements instead of long if-else chains. This can improve readability.
16. **Use Constants for Magic Numbers:**
    - Replace magic numbers in the code with named constants that describe their purpose. This makes the code more understandable and maintainable.

These recommendations are meant to improve the overall quality, readability, and maintainability of the RMC75E source code.