TN1252
Technical note

How to handle the execution of the on line MBIST

Introduction

The SPC58NEx implements the MBIST that verifies the integrity of the volatile memories. MBIST typically runs during the boot phases. SPC58NEx devices, however, offer an additional option. Indeed, the user can run the MBIST in on line mode while the application runs.
This is the on line mode of the MBIST.

The on line mode requires additional software. This software must configure MBIST, runs them, waits for its execution, and verifies the results. The STCU2 module offers the register interface to perform these operations.

Contents

1 How to proper exit the routine of the MBIST execution 5
1.1 Additional details 5
1.2 Root cause and solution 6

2 Conclusion 7

3 Revision history 8

List of tables

Table 1. Document revision history 8

List of figures

Figure 1. Example of wrong MBIST configuration 5
Figure 2. Correct MBIST configuration 6

1 How to properly exit the routine of the MBIST execution

The software starts the test by setting the RUNSW flag in the STCU_RUNSW register. In addition, the software reads back this flag to verify that the test ends.

But an unexpected effect occurs if the software reads this register while the MBIST runs. Indeed, the core gets stuck into an unexpected condition. As a result, the core stops executing any code and the application hangs.

The device embeds multiple cores. Only the core that accesses this STCU register goes into this unexpected state. Other cores execute their software without any issue.

1.1 Additional details

The software can poll the RUNSW bit to get the status of the ongoing test. The application has another option to verify the end of the test. Instead of polling the RUNSW bit it can handle the interrupt that the STCU triggers at the end of the ongoing test. In this case, no error occurs.

In addition, The UTEST miscellaneous DCF client has a correlation with the issue. If the user sets the Bit12 (CORE_CLK_ONLINE_SELF_TEST) to 0b, the issue doesn't occur.

If this bit is set to 0b, the hardware disables the clock to the cores during the on line self-test. In this case, the LBIST of partition 1-6 fails. This, however, is not a problem from a safety standpoint because the safety analysis assumes that the LBIST of these partitions only runr while the vehicle is in the garage.

The code shows in *Figure 1* an example of software implementation that leads to the issue.

Figure 1. Example of wrong MBIST configuration

```
void wrong_mbist_handling(void)
{
   /* STCU2 USER CONFIGURATION
    * Here the MBIST are configured via STCU2
    * This part of code is not in scope of this document     *
    * */

    /* Start the execution of configured MBIST in previous steps */
    STCU2.RUNSW.B.RUNSW = 1;

    /* Wait for completion of the MBIST being executed */
    while (1 == STCU2.RUNSW.B.RUNSW);

    /* !! This point will not be reached because of the issue !!
     * The core is currently stalled and the comparison of
     * 1 == STCU2.RUNSW.B.RUNSW is not physically executed
     * */
}
```

1.2 Root cause and solution

STCU2 itself initiates clock to the cores to be stopped tightly before starting MBIST execution. That means also the process of fetching via pipeline is stopped. At this point the instruction pipeline is stalled and doesn't resume after clock recovery.

The root cause of the issue is that the STCU2 interferes with the clock of the cores before starting the execution of the MBIST. As a result, the pipeline instruction of the core stalls and the software hangs.

To recover from this condition, the core must clean its instruction pipeline. The PPC instruction set includes different synchronization instructions that clean the pipeline, eg msync.

As a consequence, the solution is to insert two msync instructions before and after reading the RUNSW register.

In *Figure 2* an example of the code that avoids the occurrence of the issue.

Figure 2. Correct MBIST configuration

```
void correct_mbist_handling(void)
{
   /* STCU2 USER CONFIGURATION
    * Here the MBIST are configured via STCU2
    * This part of code is not in scope of this document
    * */

    /* Clean the pipeline of the executive core (2x msync) */
    asm("msync");
    asm("msync");

    /* Start the execution of configured MBIST in previous steps */
    STCU2.RUNSW.B.RUNSW = 1;

    /* Clean the pipeline of the executive core by (2x msync) */
    asm("msync");
    asm("msync");

    /* Wait for completion of the MBIST being executed */
    while (1 == STCU2.RUNSW.B.RUNSW);

    /* This point will be now successfully reached
     * and application can do the rest of code
     * */
}
```

2 Conclusion

Depending on the needs of the application, the user may run the MBIST in on line mode. In this mode, the software starts the MBIST and polls a flag to verify when it finishes. But in this case, the software can stall and stop executing any code.

To avoid this problem, the user should insert some msync instructions. This solution is compact, executes linearly, keeps the application functional, and doesn't require any additional system resources.

3 Revision history

Table 1. Document revision history

Date	Revision	Changes
10-Oct-2017	1	Initial release.