Application Note

Cortex-M33 Dual Core Lockstep

Version 1.0

Non-Confidential - Published

Cortex-M33 Dual Core Lockstep

Release Information

The following changes have been made to this Application Note.

Document History

Date	Issue	Confidentiality	Change
June 2017	**A**	Non-Confidential - Published	First release

ARM Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Product Status

The information in this document is Final, that is for a developed product.

Web Address

http://www.arm.com

Contents

Cortex-M33 Dual Core Lockstep

1 Conventions and Feedback

The following describes the typographical conventions and how to give feedback:

Typographical conventions

The following typographical conventions are used:

monospace denotes text that can be entered at the keyboard, such as commands, file and program names, and source code.

mono<u>space</u> denotes a permitted abbreviation for a command or option. The underlined text can be entered instead of the full command or option name.

monospace italic
denotes arguments to commands and functions where the argument is to be replaced by a specific value.

monospace bold
denotes language keywords when used outside example code.

italic highlights important notes, introduces special terminology, denotes internal cross-references, and citations.

bold highlights interface elements, such as menu names. Also used for emphasis in descriptive lists, where appropriate, and for ARM® processor signal names.

Feedback on this product

If you have any comments and suggestions about this product, contact your supplier and give:

- Your name and company.
- The serial number of the product.
- Details of the release you are using.
- Details of the platform you are using, such as the hardware platform, operating system type and version.
- A small standalone sample of code that reproduces the problem.
- A clear explanation of what you expected to happen, and what actually happened.
- The commands you used, including any command-line options.
- Sample output illustrating the problem.
- The version string of the tools, including the version number and build numbers.

Feedback on documentation

If you have comments on the documentation, e-mail errata@arm.com. Give:

- The title.
- The number, ARM-ECM-0690721, A.
- If viewing online, the topic names to which your comments apply.
- If viewing a PDF version of a document, the page numbers to which your comments apply.

- A concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

ARM periodically provides updates and corrections to its documentation on the ARM Information Center, together with knowledge articles and *Frequently Asked Questions* (FAQs).

Other information

- ARM Information Center, http://infocenter.arm.com/help/index.jsp.
- ARM Technical Support Knowledge Articles, http://infocenter.arm.com/help/topic/com.arm.doc.faqs/index.html.
- ARM Support and Maintenance, http://www.arm.com/support/services/support-maintenance.php.
- ARM Glossary, http://infocenter.arm.com/help/topic/com.arm.doc.aeg0014-/index.html.

2 Preface

This Application Note is indented for developers who deploy hardware for *Dual-redundant Core Lock-step* with the Cortex-M33 processor.

These topics support the following chapters:

2.1 References

[1] Design of SoC for High Reliability Systems with Embedded Processors (doc number)

[2] ARM® Cortex®-M33 Processor Integration and Implementation Manual (100323)

[3] ARM® Cortex®-M33 Processor Safety Manual (100601)

2.2 Terms and abbreviations

Abbreviations and terms used in this document are defined here.

DCLS	Dual-redundant Core Lock-step
MCU	Micro Controller Unit
RAR	Reset All Registers
RTL	Register Transfer Level

3 Introduction

3.1 Document purpose

Reliability is a critical concern in many embedded system applications such as industrial controller or automobile electronics. *Dual-redundant Core Lock-step* (DCLS) is one of many techniques to enhance the reliability of a *Micro Controller Unit* (MCU).

The purpose of this application note is to help hardware developers use and design a Cortex-M33 processor with DCLS in a MCU system.

ARM provides a:

- DCLS example, which is the instantiation of two Cortex-M33 processor instances that execute identical code in tandem
- Method to check equivalence of outputs. A mismatch in any output from either core must then be appropriately handled in system logic.

Knowledge of detailed hardware design in the processor is not required for reading this application note.

.

3.2 Document scope

This application note introduces the background knowledge for understanding DCLS, which covers:

- Why DCLS is good for reliability and how it .
- Typical considerations for DCLS.
- DCLS Example with Cortex-M33 processor.

The DCLS example is designed and verified on *Register Transfer Level* (RTL) logic. Some design considerations and implementation techniques of DCLS introduced in this application note might not be present in the example.

4 Why DCLS is good for reliability and how it works

This chapter introduces background knowledge related to DCLS.

It contains the following topics:

4.1 High reliability system

4.1.1 Requirements for a high reliability system

DCLS is a common technique for developing a high reliability system. It is helpful to understand the requirements for a high reliability system before we dive into DCLS. In [1], Joseph Yiu generalized the technical requirements for high reliability system into four areas as following:

- Reducing the possibility of failures.
- Detection of failures.
- Correction of failures.
- Robustness – single point failure could not lead to a complete system failure.

Note that some of them might be optional depending on the application and the property of potential failures.

4.1.2 Sources of failures

It is also crucial to look at what can cause failures at the processor level. As introduced in [1], these failures are categorized into the following:

- Memory: There is an accidental trigger that changes the memory state in the system. Common scenarios include a hit by a radiation particle, interference from RF transmitter, for example.
- Logic: A hardware failure exists in system internal logic. This category of failures can usually be detected by a scan test.
- Software: Mistakes from the software such as programming bugs. Incorrect setup for memory, for example, lead to unexpected system failures.

4.2 Why DCLS is good for reliability

A DCLS helps the system detect logic failures. When logic failures are detected the system chooses the appropriate action to take.

For example, aviation electronics usually have higher risk of being affected by alpha particles or cosmic rays. Failure detection and correction mechanisms are essential features for high reliability in such systems. DCLS is a common technique applied to the systems because of its failure detection capability.

4.3 How DCLS works

As revealed in its name, a system with DCLS deploys two identical processor cores inside. Two cores are initialized (reset) in the same states and fed with identical inputs. As a result, identical outputs from two cores should always be observed. A logic failure reaching the output in one of the cores can be detected by comparing the outputs of the two cores. After the detection of a failure, the system can choose various approaches to handle it depending on the application requirement.

Generally, one of cores is referred to as the *main core*, while the other one is referred to as the *redundant core*. The redundant core confirms the correctness of outputs from the main core but does not enhance the system performance since it takes identical instructions and data from the main core

5 Typical considerations for DCLS

This chapter introduces typical considerations for design a DCLS system.

It contains the following topics:

5.1 Reset all registers

A crucial requirement for DCLS is that both cores need to be initialized with the same state. In other words, all registers (flip-flops) in the processor should be reset to guarantee the same initial state.

In many processor designs, the hardware designer may deliberately keep the state of some registers, such as architectural registers, from reset to reduce the power consumption and silicon area. In DCLS, however, non-initialized values from non-reset registers might potentially propagate to outputs, causing false mismatch. To allow DCLS functioning correctly, resetting all registers in the processor by either hardware or software scheme is usually needed.

The Cortex-M33 processor provides RAR (*Reset all register*) configuration [2] that must be selected before implementation to ensure all registers are properly reset.

5.2 Avoid common mode failures

DCLS cannot detect potential failures that can occur at the same point in both cores since the failures do not cause any difference between their outputs. These failures are referred to as *common mode failures*, which cause false match in the DCLS system.

Several techniques have been proposed to address common mode failures. One of them is providing *temporal diversity* to two cores. A common approach to this is delaying the redundant core for few cycles by inserting shift registers into the inputs. With a temporal diversity of even a few cycles, it is less likely that an erroneous trigger occurs at the same point of two cores. Note that this approach requires resynchronization of outputs from two cores before comparisons.

Another technique to avoid common mode failures is implementing two cores in different ways. Hardware designers may choose different types of *arithmetic logic units* (ALU), block implementations or physical designs to implement the redundant core. This keeps the same functionality of two cores but reduce the risk of failures caused by the same erroneous transient pulse from power or signal interface.

5.3 Optimize the cost

For a DCLS processor, the timing on the redundant core can be relaxed because of the following reasons:

- The outputs of the processor are driven only by the main core. The redundant core usually does not directly connect to the system.
- Inputs to the redundant core are usually delayed by flip-flops to introduce temporal diversity between cores (See 5.2). These flip-flops can act as a method of *pipelining* to avoid critical paths that originate at core inputs.

Hardware designers may take advantage of the relaxed timing on the redundant core interface to optimize the cost in the redundant core.

On the other hand, it is recommended that two instances of comparator logic are implemented, and output mismatches from one or both qualify as a failure. This prevents a single stuck-at-fault in comparator logic from disabling the DCLS functionality.

6 DCLS Example with Cortex-M33 processor

This chapter introduces a DCLS example with the Cortex-M33 processor.

It contains the following topics:

6.1 Considerations before designing DCLS with the Cortex-M33 processor

Before designing DCLS with the Cortex-M33 processor, it is helpful to consider the following features from Cortex-M33 processor on the processor level [2]:

- *Reset all registers* (RAR) is an available configuration parameter.
- There is no internal memory (SRAM) inside.
- *Debug Access Port* (DAP) and *Trace Port Interface Unit* (TPIU) with asynchronous clock boundaries are already excluded from the processor level.
- There is a single clock source **CLKIN** for all internal logic.
- There are two reset signals, **nPORESET** and **nSYSRESET,** used for power-on reset (cold reset) and system reset (warm reset) respectively.

By these considerations, the RTL design of DCLS with Cortex-M33 processor can be very straightforward. We introduce the details in 6.2.

6.2 RTL design

ARM recommends that a new level of module on top of two cores and the DCLS logic should be added for the integration. Figure 6-1 shows an example Cortex-M33-based DCLS processor subsystem.

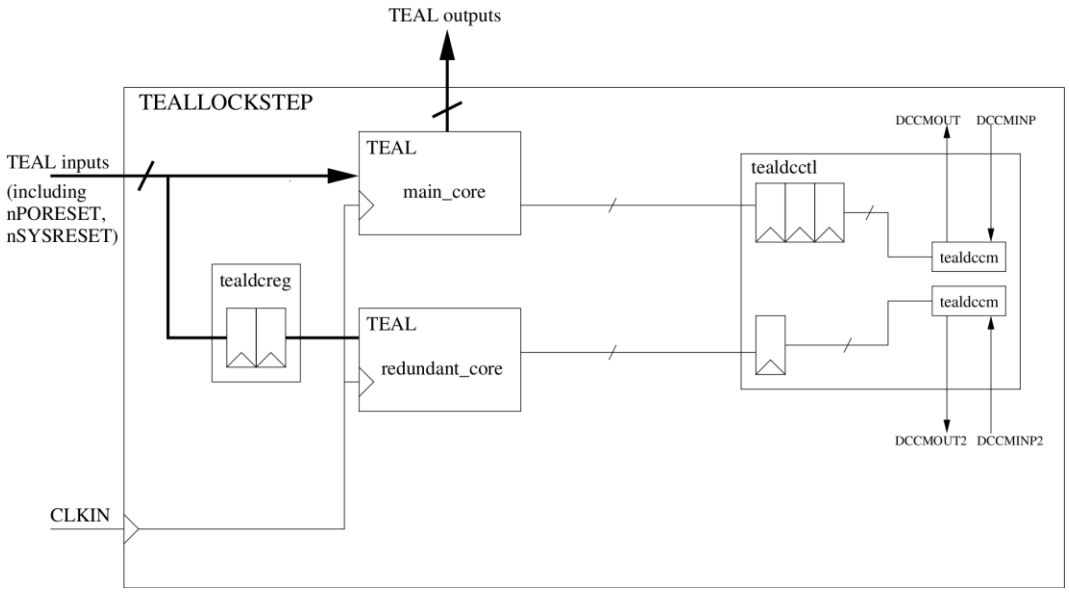

Figure 6-1 Cortex-M33-based DCLS processor

This DCLS processor subsystem (the TEALLOCKSTEP module) contains these sub-modules:

- TEAL: a single Cortex-M33 processor instance
- `tealdcctl`: the DCLS controller module and resynchronization flip-flops
- `tealdccm`: the comparator logic inside the DCLS controller
- `tealdcreg`: the redundant core input delay flip-flops

Important notes of the example are summarized as the followings:

- All logic of the M33 processor (TEAL) is duplicated, and no changes are required inside the M33 processor.
- All logic of the DCLS processor (TEALLOCKSTEP) shares the same external clock source **CLKIN**.
- All outputs from the DCLS processor subsystem are driven by the main core, and all inputs to the DCLS processor subsystem are input to the main core directly.
- All inputs are wired to the main core directly, and same signals are wired to the redundant core with two stages of back-to-back flip-flops except the **CLKIN**. This offers the DCLS processor *temporal diversity, which* reduces the risk of *common mode failures*.
- Before a comparison of outputs from two cores, they are resynchronized by the 3 stages and 1-stage flip-flops respectively. These flop banks also serve to isolate any critical output paths from the comparator logic.

- The comparator logic is instantiated twice to prevent a single stuck-at-fault in the comparator logic from disabling the fault detecting functionality.
- The pins **DCCMOUT[0]** and **DCCMOUT2[0]** are fault indicator signals from the main comparator and redundant comparator respectively. Each of them indicates that faults have been detected from the corresponding comparator when it is asserted.

The following items must be guaranteed when implementing the example:

- Both processor cores are configured with the RAR parameter set to 1. This can be done by overriding RAR parameter for both instantiations of M33. For example:

```
TEAL # ( .RAR  (1) )  main_core  (  …
TEAL # ( .RAR  (1) )  redundant_core (  …
```

- The external reset signals **nPORESET** and **nSYSRESET** are synchronized to **CLKIN**.

6.3 DCLS controller and comparators

The DCLS controller passes output signals from two cores to the comparators, and controls the qualification signal to the fault indicator signals. Fault indicator signals are sticky, which means only the power-on reset or signals from the DCLS controller can clear them.

The DCLS controller provides the interface to access the comparators. The fault indicator signals drive the controller pins **DCCMOUT[0]** and **DCCMOUT2[0]** that can be cleared by de-asserting the controller pins **DCCMINP[0]** and **DCCMINP2[0]** (clear fault indicator signals on negative).

Every time DCLS exits from the power-on reset, there are three cycles of bogus data from the re-alignment flip-flops sent to the comparators. To avoid false detection of failures caused by the bogus data, the DCLS controller uses a qualification signal to guarantee that the comparators never fire in first three cycles after the power-on reset.

6.4 Verification methodology

Customers may select to validate the DCLS logic with the combination of methodologies (simulation and formal) that meets their requirements. ARM recommends that if properties are used, the following three properties should be written to ensure the DCLS logic is functionally correct:

1. The **DCCMOUT[0]** should never be asserted unless we insert a fault deliberately.
2. Any fault causing output difference between two cores should cause corresponding **DCCMOUT[0]** asserted.
3. Once **DCCMOUT[0]** is asserted, it should remain asserted until power-on reset or **DCCMINP[0]** is de-asserted.

ARM recommends that customers add fault injection logic similar to the example shown within the dotted line of Figure 6-2 into the original RTL design.

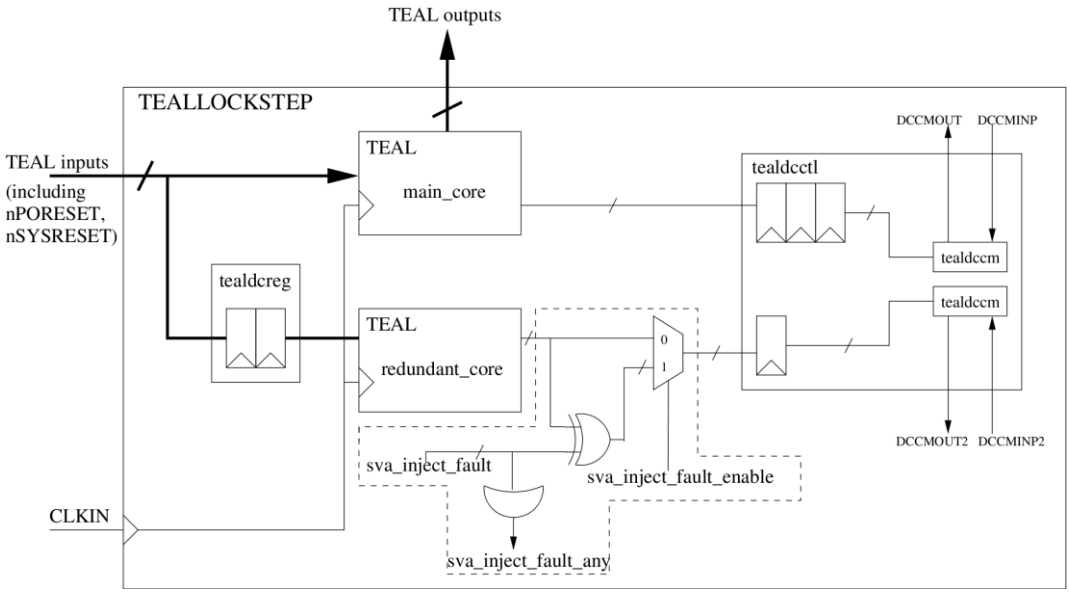

Figure 6-2 DCLS processor with fault injection logic

The validation environment can invert any bits of the redundant core outputs by setting **sva_inject_fault_enable** (fault injection qualifier) and any bit of **sva_inject_fault[1118:0]** (fault bits selection) to emulate faults that have reached the redundant core outputs. Injecting faults on the outputs instead of the internal register state provides better controllability of fault injection, as it is not always possible to figure out when an injected fault of the internal memory will cause difference to the outputs.

6.5 External logic requirements

The reaction to any failures is not implemented in this example. It is an integration decision on how to recover the system from failure upon fault detection for IP integrators.

This application note only demonstrates the DCLS example on *Register Transfer Level* (RTL). However, you might want to apply other implementation techniques after the RTL design stage. For example, you might want to use different types of ALU or floorplans to implement the redundant logic to reduce the risk of common mode failures.

It is also feasible to incorporate DCLS with other fault detection mechanisms to further increase the fault detection rate. Several configurable features of the Cortex-M33 processor at synthesis time are applicable to fault detection. For more details, please refer to [3].