

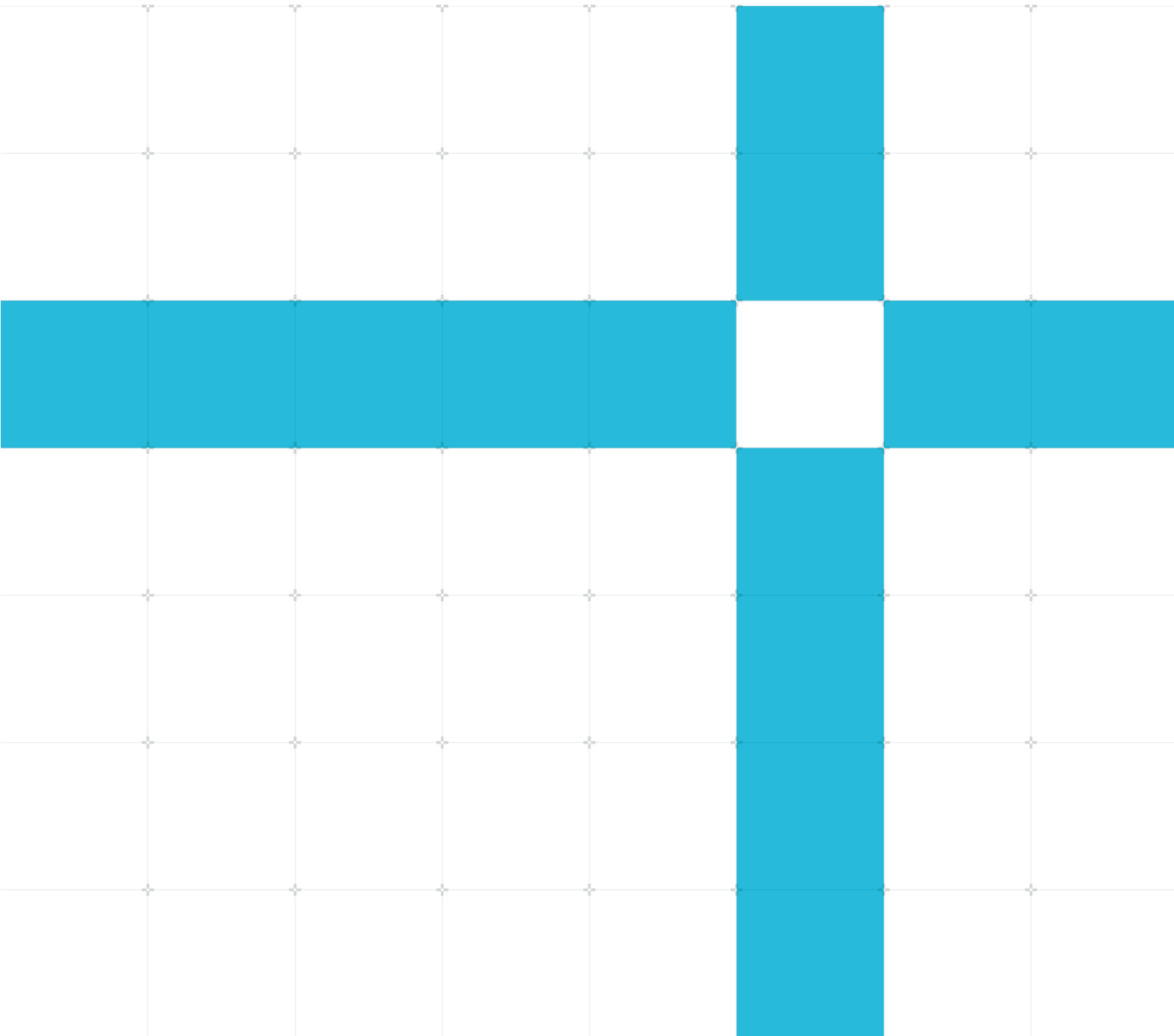


Cortex M23 Dual Core Lockstep

Application Note

Non-confidential
Copyright © 2023 Arm Limited (or its affiliates).
All rights reserved.

Document Issue: 1
Document ID: 107936



Release information

Document history

| Issue | Date | Confidentiality | Change |
|---------|-------------------------------|------------------|---------------|
| 0200-01 | 27 th January 2023 | Non-Confidential | First release |

Non-Confidential Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, has undertaken no analysis to identify or understand the scope and content of, patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word "partner" in reference to Arm's customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its affiliates) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm's trademark usage guidelines at <https://www.arm.com/company/policies/trademarks>.

Copyright © 2023 Arm Limited (or its affiliates). All rights reserved.

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.

Unrestricted Access is an Arm internal classification.

Feedback

Arm welcomes feedback on its products and documentation. To provide feedback on a product, create a ticket on <https://support.developer.arm.com>.

To provide feedback on this document, fill the following survey:
<https://developer.arm.com/documentation-feedback-survey>.

Arm periodically provides updates and corrections to its technical content and Frequently Asked Questions (FAQs) on [Arm Developer](#).

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

To report offensive language in this document, email terms@arm.com.

Contents

| | |
|--|-----------|
| 1. Introduction..... | 5 |
| 1.1. Intended audience..... | 5 |
| 1.2. Conventions..... | 5 |
| 1.3. Useful resources | 6 |
| 2. Introduction..... | 8 |
| 2.1. Document purpose | 8 |
| 2.2. Document scope..... | 8 |
| 3. Why DCLS is good for reliability and how it works. | 9 |
| 3.1. High reliability system..... | 9 |
| 3.1.1. Requirements for a high reliability system..... | 9 |
| 3.1.2. Sources of failures | 9 |
| 3.2. Why DCLS is good for reliability..... | 10 |
| 3.3. How DCLS works..... | 10 |
| 4. Typical considerations for DCLS | 11 |
| 4.1. Reset all registers..... | 11 |
| 4.2. Avoid common mode failures | 11 |
| 4.3. Optimize the cost..... | 12 |
| 5. Using Cortex-M23 to create a DCLS processor..... | 13 |
| 5.1. Considerations before designing DCLS with the Cortex-M23 processor..... | 13 |
| 5.2. RTL design..... | 13 |
| 5.3. DCLS controller and comparators..... | 15 |
| 5.4. Verification methodology | 16 |
| 5.5. External logic requirements | 16 |

1. Introduction

1.1. Intended audience

This application note is intended for hardware developers using a Cortex-M23 processor when designing a MCU system.

1.2. Conventions

The following subsections describe conventions used in Arm documents.

Glossary

The Arm Glossary is a list of terms used in Arm documentation, together with definitions for those terms. The Arm Glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the Arm Glossary for more information: <https://developer.arm.com/glossary>.







This document uses the following terms and abbreviations.

Terms and abbreviations

| Term | Meaning |
|------|-------------------------|
| DCLS | Dual Core Lock-Step |
| MCU | Micro Controller Unit |
| RAR | Reset All Registers |
| RTL | Register Transfer Level |

Typographical conventions

| Convention | Use |
|----------------------------|---|
| <i>italic</i> | Citations. |
| bold | Interface elements, such as menu names. Terms in descriptive lists, where appropriate. |
| monospace | Text that you can enter at the keyboard, such as commands, file and program names, and source code. |
| monospace bold | Language keywords when used outside example code. |
| monospace <u>underline</u> | A permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name. |

| Convention | Use |
|---|---|
| <and> | Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: <code>MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2></code> |
| SMALL CAPITALS | Terms that have specific technical meanings as defined in the Arm® Glossary. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE. |
|  Caution | Recommendations. Not following these recommendations might lead to system failure or damage. |
|  Warning | Requirements for the system. Not following these requirements might result in system failure or damage. |
|  Danger | Requirements for the system. Not following these requirements will result in system failure or damage. |
|  Note | An important piece of information that needs your attention. |
|  Tip | A useful tip that might make it easier, better, or faster to perform a task. |
|  Remember | A reminder of something important that relates to the information you are reading. |

1.3. Useful resources

This document contains information that is specific to this product. See the following resources for other relevant information.

- Arm Non-Confidential documents are available on developer.arm.com/documentation. Each document link in the tables below provides direct access to the online version of the document.
- Arm Confidential documents are available to licensees only through the product package.

| Arm products | Document ID | Confidentiality |
|---|-------------------|-----------------|
| Arm ® Cortex®-M23 Processor Integration and Implementation Manual | DIT 0062C_0200_en | Confidential |

| Arm products | Document ID | Confidentiality |
|--|-------------------|---------------------|
| Arm® Cortex®-M23 Processor Safety Manual | 100988_0200_03_en | <i>Confidential</i> |

| Arm architecture and specifications | Document ID | Confidentiality |
|---|-------------|-------------------------|
| Arm® v8-M Architecture Reference Manual | DDI0553B | <i>Non-Confidential</i> |

| Non-Arm resources | Document ID | Organization |
|---|-------------|--------------|
| https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/embedded-world-2015---design-of-soc-for-high-reliability-systems-with-embedded-processors | | |



Arm tests its PDFs only in Adobe Acrobat and Acrobat Reader. Arm cannot guarantee the quality of its documents when used with any other PDF reader.

Adobe PDF reader products can be downloaded at <http://www.adobe.com>.

2. Introduction

2.1. Document purpose

Reliability is a critical concern in many embedded system applications such as industrial control or automobile electronics. Dual 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-M23 processor with DCLS in a MCU system.

This document describes:

- The instantiation of two Cortex-M23 processor instances that execute identical code in tandem to form a DCLS processor.
- A method to check equivalence of processor outputs. A mismatch in any output from either core must then be propagated to the system logic to be appropriately handled.

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

2.2. Document scope

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

- Why DCLS is good for reliability and how it works.
- Typical considerations for DCLS.
- Designing a DCLS processor using the Cortex-M23.

The application note assumes the DCLS processor will be designed and verified using Register Transfer Level (RTL) logic.

3. Why DCLS is good for reliability and how it works.

This chapter introduces background knowledge related to DCLS.

It contains the following topics:

- *High reliability system* page 9.
- *Why DCLS is good for reliability* on page 10.
- *How DCLS works* on page 10.

3.1. High reliability system

3.1.1. 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.

3.1.2. 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. There is an accidental trigger that changes the value held in a register. Common scenarios include a hit by a radiation particle, interference from RF transmitter, for example.
- Software: Mistakes from the software such as programming bugs. Incorrect setup for memory, for example, lead to unexpected system failures.

3.2. 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.

3.3. 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 the cores is referred to as the *main core*, while the other one is referred to as the *redundant core*. The dual cores allow detection of errors within the processor but do not enhance performance over a single core system since both cores execute exactly the same program stream.

4. Typical considerations for DCLS

This chapter introduces typical considerations for designing a DCLS system.

It contains the following topics:

- Reset all registers on page 11.
- Avoid common mode failures on page 11.
- Optimize the cost on page 12.

4.1. 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 a false mismatch. To allow DCLS to function correctly, resetting all registers in the processor by either hardware or software scheme is usually needed.

The Cortex-M23 processor provides RAR (Reset all register) configuration [see Arm® Cortex®-M23 Processor Integration and Implementation Manual] that must be selected before implementation to ensure all registers are properly reset.

4.2. 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 matches in the DCLS system.

Several techniques have been proposed to address common mode failures. One of them is providing temporal diversity between the cores. A common approach to this is delaying the execution of 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 in both cores. Note that this approach requires resynchronization of outputs from two cores before any comparisons are made.

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

4.3. 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 4.2). 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.

5. Using Cortex-M23 to create a DCLS processor

This chapter shows how to create a DCLS processor using Cortex-M23.

It contains the following topics:

- Considerations before designing DCLS with the Cortex-M23 processor on page 13.
- RTL design on page 13.
- DCLS controller and comparators on page 15.
- Verification methodology on page 16.
- External logic requirements on page 16.

5.1. Considerations before designing DCLS with the Cortex-M23 processor

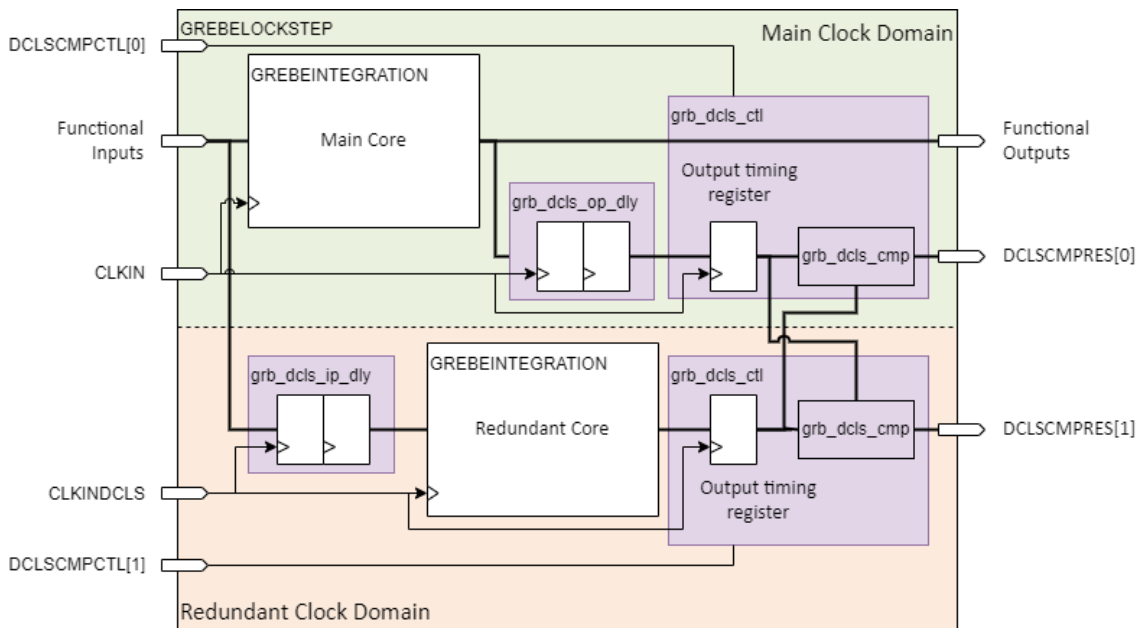
Before designing DCLS with the Cortex-M23 processor, it is helpful to consider the following features from Cortex-M23 processor on the processor level [2]:

- 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 taking these considerations into account, the RTL design of DCLS with Cortex-M23 processor can be very straightforward. We introduce the details in 5.2.

5.2. RTL design

Arm recommends that a new top-level module is created that integrates the two Cortex-M23 cores and the DCLS logic. Figure 5-1 shows an example Cortex-M23-based DCLS processor subsystem.

Figure 5-1: Cortex-M23-based DCLS Processor Subsystem

In this example, the DCLS processor subsystem (the GREBELOCKSTEP module) should contain:

- GREBEINTEGRATION : two instances of a Cortex-M23 processor.
- grb_dcls_ctl : the DCLS controller module and resynchronization flip-flops
- grb_dcls_cmp : the comparator logic inside the DCLS controller
- grb_dcls_ip_dly : the redundant core input delay flip-flops(*)
- grb_dcls_op_dly : the main core output delay flip-flops(*)

(*) These modules should include any register enable logic required to meet interface specifications.

Important requirements of the example system are summarized here:

- All logic of the M23 processor (GREBEINTEGRATION) is duplicated, and no changes are required inside the M23 processor.
- All main logic of the DCLS processor (GREBELOCKSTEP) should share the same external clock source CLKIN. All redundant logic should share the same external clock source CLKINDCLS. These clocks should be in phase and of the same frequency.
- All outputs from the DCLS processor subsystem should be driven by the main core, and all inputs to the DCLS processor subsystem should be input to the main core directly.
- All inputs should be wired to the redundant core with two stages of back-to-back flip-flops. These flip-flops must adhere to the interface specification for the signal being delayed just as the M23 processor (GREBEINTEGRATION) does. This offers the DCLS processor temporal diversity, which reduces the risk of common mode failures.

- Before a comparison of outputs from two cores, the main core outputs are aligned by the output delay block and then both outputs are delayed by a further stage of flip-flops. These flip-flops must adhere to the interface specification for the signal being delayed just as the M23 processor (GREBEINTEGRATION) does. These flop banks also serve to isolate any critical output paths from the comparator logic.
- The specifications for all signals must be followed precisely when designing the control and clocking of the registers in the input and output delay blocks. These flip-flops must only capture the signal when it is specified to be valid (as Cortex-M23 is designed to) or it may adversely affect the timing closure of the DCLS processor subsystem and potentially the whole design. Further details on all signals can be found in Arm® Cortex®-M23 Processor Integration and Implementation Manual, section 5.
- The comparator logic should be instantiated twice to prevent a single stuck-at-fault in the comparator logic from disabling the fault detecting functionality.
- Any asynchronous outputs should be checked and delayed using a free-running version of CLKIN/CLKINDCLS to avoid false fault reporting when turning on the clocks from a low power mode.
- The pins DCLSCMPRES[0] and DCLSCMPRES[1] 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 M23. For example:

```
GREBEINTEGRATION # ( .RAR (1) ) main_core ( ...
GREBEINTEGRATION # ( .RAR (1) ) redundant_core ( ...
```
- The external reset signals nPORESET and nSYSRESET are not asserted or deasserted when CLKIN/CLKINDCLS are running. For an increase in the fault coverage, independent resets for the redundant core and logic can be used which must be in step with the main resets.

5.3. DCLS controller and comparators

The DCLS controller must pass output signals from two cores to the comparators and control 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 must provide an interface to access the comparators. The fault indicator signals should drive the output pins DCLSCMPRES[0] and DCLSCMPRES[1] that can be cleared by de-asserting the controller pins DCLSCMPCTL[0] and DCLSCMPCTL[1].

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 must generate a qualification signal to disable the comparators during the first three cycles after the power-on reset.

5.4. Verification methodology

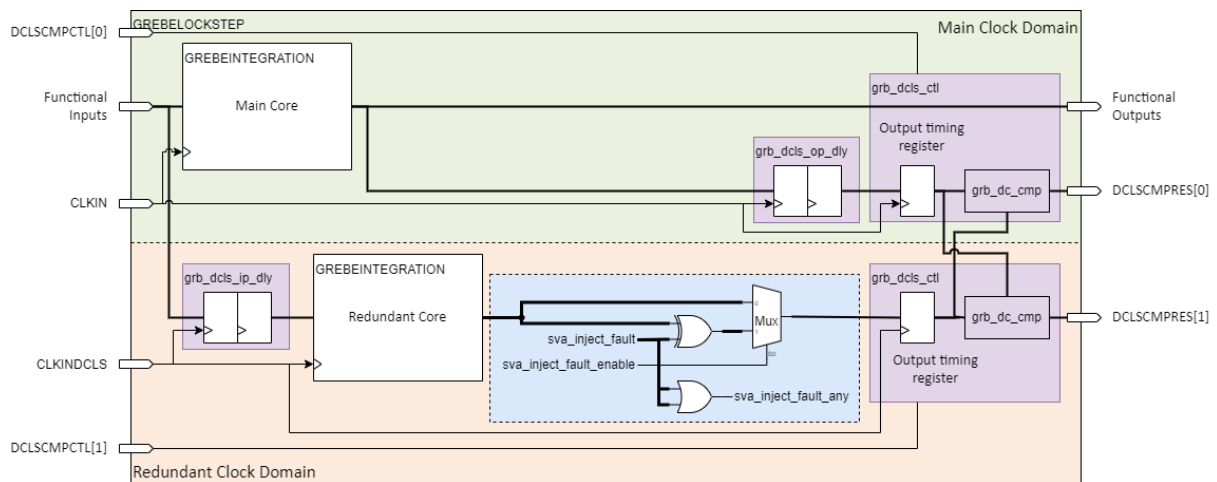
Customers may select to validate the DCLS logic with the combination of methodologies (simulation and formal) that meets their requirements. ARM recommends that if properties are used, the following three properties should be written to ensure the DCLS logic is functionally correct:

- The `DCLSCMPRES[x]` should never be asserted unless we insert a fault deliberately.
- Any fault causing output difference between two cores should cause corresponding `DCLSCMPRES[x]` asserted.
- Once `DCLSCMPRES[x]` is asserted, it should remain asserted until power-on reset or `DCLSCMPCTL[x]` is de-asserted.

Where x in $[0, 1]$

ARM recommends that customers add fault injection logic similar to the example shown within the dotted line of Figure 5-2 into the original RTL design.

Figure 5-2: 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` (fault bits selection) to emulate faults at 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 determine when a fault injected into inputs of the redundant core will cause a difference to be visible at the outputs.

5.5. External logic requirements

This application note does not describe system recovery after a DCLS failure. This is for the IP integrator to implement.

Any asynchronous inputs to `GREBEINTEGRATION` should be synchronized to `CLKIN` externally to the DCLS processor subsystem.

Any external state change request should only occur when the system is quiescent and should be done individually. For example, the system should not attempt to access the system Q-Channel and the DBG Q-Channel at the same time. The operations should be done sequentially and must not overlap. In addition, any operation on these channels should add a period of 4 `CLKIN` cycles after the last transition to allow for the redundant logic to complete its operation and any fault that occurred to be reported.

This application note only describes an implementation of a DCLS processor for a Register Transfer Level methodology (RTL). However, an IP integrator might want to apply other implementation techniques after the RTL design stage. For example, a different floorplan for 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-M23 processor at synthesis time are applicable to fault detection, e.g., bus protection.

`FLOPPARITY` is not intended to work within a DCLS processor and should not be configured for either Cortex M23 processing element.

For more details, please refer to Arm® Cortex®-M23 Processor Integration and Implementation Manual.