**Module Design Document**

**For**

**MicroCtrlrSuprt**

**7/19/17**

**Prepared For:**

**Software Engineering**

**Nexteer Automotive,**

**Saginaw, MI, USA**

**Prepared By:**

**Software Group,**

**Nexteer Automotive,**

**Saginaw, MI, USAChange History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Description** | **Author** | **Version** | **Date** |
| Initial Version | Lucas Wendling | 1 | 7/19/17 |

**Table of Contents**

[1 Introduction 5](#_Toc488222083)

[1.1 Purpose 5](#_Toc488222084)

[1.2 Scope 5](#_Toc488222085)

[2 High-Level Description 6](#_Toc488222086)

[3 Design details of software module 7](#_Toc488222087)

[3.1 Graphical representation of NxtrOsErrHndlg (Expected External Intefaces) 7](#_Toc488222088)

[3.2 Data Flow Diagram 7](#_Toc488222089)

[3.2.1 Component level DFD 7](#_Toc488222090)

[3.2.2 Function level DFD 7](#_Toc488222091)

[4 Constant Data Dictionary 8](#_Toc488222092)

[4.1 Program (fixed) Constants 8](#_Toc488222093)

[4.1.1 Embedded Constants 8](#_Toc488222094)

[5 Software Component Implementation 9](#_Toc488222095)

[5.1 Sub-Module Functions 9](#_Toc488222096)

[5.1.1 Init: 9](#_Toc488222097)

[5.1.2 Per: 9](#_Toc488222098)

[5.2 Server Runables 9](#_Toc488222099)

[5.2.1 NxtrOsErrHndlg 9](#_Toc488222100)

[5.2.1.1 Design Rationale 9](#_Toc488222101)

[5.2.1.2 Processing 9](#_Toc488222102)

[5.3 Interrupt Functions 13](#_Toc488222103)

[5.4 Module Internal (Local) Functions 13](#_Toc488222104)

[5.4.1 Local Function #1 13](#_Toc488222105)

[5.4.1.1 Design Rationale 13](#_Toc488222106)

[5.4.1.2 Processing 13](#_Toc488222107)

[5.5 GLOBAL Function/Macro Definitions 13](#_Toc488222108)

[5.5.1 GLOBAL Function #1 13](#_Toc488222109)

[5.5.1.1 Design Rationale 13](#_Toc488222110)

[5.5.1.2 Processing 13](#_Toc488222111)

[6 Known Limitations with Design 14](#_Toc488222112)

[7 UNIT TEST CONSIDERATION 15](#_Toc488222113)

[Appendix A Abbreviations and Acronyms 16](#_Toc488222114)

[Appendix B Glossary 17](#_Toc488222115)

[Appendix C References 18](#_Toc488222116)

# Introduction

## Purpose

This design document will capture the design of the Nexteer Mcu Support Library (NxtrMcuSuprtLib) functionality. This is the only portion of this component that is designed by Nexteer rather than generated by a 3rd party tool.

## Scope

The following definitions are used throughout this document:

* **Shall**: indicates a mandatory requirement without exception in compliance.
* **Should**: indicates a mandatory requirement; exceptions allowed only with documented justification.
* **May**: indicates an optional action.

# High-Level Description

The Nexteer designed portions of this component include the definition of some inline functions for supporting the Renesas microcontroller. These are described in detail in the following sections. Note that this component supports multiple files to support the different microcontroller variants used by Nexteer.

# Design details of software module

## Graphical representation of Protected Register Write Functions

In general, the design of all protected write functions contained in this module follow the following high-level flow:



## Graphical representation of Nexteer Software Reset Function



## Graphical representation of Nexteer Software Reset From Exception Function



# Constant Data Dictionary

## Program (fixed) Constants

### Embedded Constants

#### Local Constants

|  |  |  |  |
| --- | --- | --- | --- |
| Constant Name | Resolution | Units | Value |
| None |  |  |  |

# Software Component Implementation

## Sub-Module Functions

## Init:

*None*

## Per:

*None*

## Library/Server Runables

## P1M Micro Variants

#### Protected Write APIs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Name** | WrProtdRegPortJ\_u32 | Type | Min | Max |
| **Arguments Passed** | WrVal\_Arg | uint32 | Full | Full |
|  | WrAddr\_Arg | pointer to volatile uint32 | Valid values:  0xFFC24014  0xFFC24018  0xFFC24028 | |
| **Return Value** | N/A (void) |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Name** | WrProtdRegPort0\_u32 | Type | Min | Max |
| **Arguments Passed** | WrVal\_Arg | uint32 | Full | Full |
|  | WrAddr\_Arg | pointer to volatile uint32 | Valid values:  0xFFC14014  0xFFC14018  0xFFC1403C  0xFFC14028  0xFFC10030 | |
| **Return Value** | N/A (void) |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Name** | WrProtdRegPort1\_u32 | Type | Min | Max |
| **Arguments Passed** | WrVal\_Arg | uint32 | Full | Full |
|  | WrAddr\_Arg | pointer to volatile uint32 | Valid values:  0xFFC14054  0xFFC14058  0xFFC1407C  0xFFC14068  0xFFC10070 | |
| **Return Value** | N/A (void) |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Name** | WrProtdRegPort2\_u32 | Type | Min | Max |
| **Arguments Passed** | WrVal\_Arg | uint32 | Full | Full |
|  | WrAddr\_Arg | pointer to volatile uint32 | Valid values:  0xFFC14094  0xFFC14098  0xFFC140BC  0xFFC140A8  0xFFC100B0 | |
| **Return Value** | N/A (void) |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Name** | WrProtdRegPort3\_u32 | Type | Min | Max |
| **Arguments Passed** | WrVal\_Arg | uint32 | Full | Full |
|  | WrAddr\_Arg | pointer to volatile uint32 | Valid values:  0xFFC140D4  0xFFC140D8  0xFFC140FC  0xFFC140E8  0xFFC100F0 | |
| **Return Value** | N/A (void) |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Name** | WrProtdRegPort4\_u32 | Type | Min | Max |
| **Arguments Passed** | WrVal\_Arg | uint32 | Full | Full |
|  | WrAddr\_Arg | pointer to volatile uint32 | Valid values:  0xFFC14114  0xFFC14118  0xFFC1413C  0xFFC14128  0xFFC10130 | |
| **Return Value** | N/A (void) |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Name** | WrProtdRegPort5\_u32 | Type | Min | Max |
| **Arguments Passed** | WrVal\_Arg | uint32 | Full | Full |
|  | WrAddr\_Arg | pointer to volatile uint32 | Valid values:  0xFFC14154  0xFFC14158  0xFFC1417C  0xFFC14168  0xFFC10170 | |
| **Return Value** | N/A (void) |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Name** | WrProtdRegSys\_u08 | Type | Min | Max |
| **Arguments Passed** | WrVal\_Arg | uint8 | Full | Full |
|  | WrAddr\_Arg | pointer to volatile uint8 | Valid values:  0xFFF82838  0xFFF82830  0xFFF8282C  0xFFF8283C | |
| **Return Value** | N/A (void) |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Name** | WrProtdRegSys\_u32 | Type | Min | Max |
| **Arguments Passed** | WrVal\_Arg | uint32 | Full | Full |
|  | WrAddr\_Arg | pointer to volatile uint32 | Valid values:  0xFFF82840 | |
| **Return Value** | N/A (void) |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Name** | WrProtdRegSysClmac\_u32 | Type | Min | Max |
| **Arguments Passed** | WrVal\_Arg | uint32 | Full | Full |
|  | WrAddr\_Arg | pointer to volatile uint32 | Valid values:  0xFFF82C00  0xFFF8AC18  0xFFF89080  0xFFF890C0  0xFFF89200  0xFFF8A440  0xFFF88204 | |
| **Return Value** | N/A (void) |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Name** | WrProtdRegClma0\_u08 | Type | Min | Max |
| **Arguments Passed** | WrVal\_Arg | uint8 | Full | Full |
|  | WrAddr\_Arg | pointer to volatile uint8 | Valid values:  0xFFF88400 | |
| **Return Value** | N/A (void) |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Name** | WrProtdRegClma1\_u08 | Type | Min | Max |
| **Arguments Passed** | WrVal\_Arg | uint8 | Full | Full |
|  | WrAddr\_Arg | pointer to volatile uint8 | Valid values:  0xFFF88420 | |
| **Return Value** | N/A (void) |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Name** | WrProtdRegClma2\_u08 | Type | Min | Max |
| **Arguments Passed** | WrVal\_Arg | uint8 | Full | Full |
|  | WrAddr\_Arg | pointer to volatile uint8 | Valid values:  0xFFF88440 | |
| **Return Value** | N/A (void) |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Name** | WrProtdRegClma3\_u08 | Type | Min | Max |
| **Arguments Passed** | WrVal\_Arg | uint8 | Full | Full |
|  | WrAddr\_Arg | pointer to volatile uint8 | Valid values:  0xFFF88460 | |
| **Return Value** | N/A (void) |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Name** | WrProtdRegEcmm\_u08 | Type | Min | Max |
| **Arguments Passed** | WrVal\_Arg | uint8 | Full | Full |
|  | WrAddr\_Arg | pointer to volatile uint8 | Valid values:  0xFFD60000  0xFFD60004 | |
| **Return Value** | N/A (void) |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Name** | WrProtdRegEcmc\_u08 | Type | Min | Max |
| **Arguments Passed** | WrVal\_Arg | uint8 | Full | Full |
|  | WrAddr\_Arg | pointer to volatile uint8 | Valid values:  0xFFD61000  0xFFD61004 | |
| **Return Value** | N/A (void) |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Name** | WrProtdRegEcm\_u08 | Type | Min | Max |
| **Arguments Passed** | WrVal\_Arg | uint8 | Full | Full |
|  | WrAddr\_Arg | pointer to volatile uint8 | Valid values:  0xFFD62000  0xFFD6203C | |
| **Return Value** | N/A (void) |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Name** | WrProtdRegEcm\_u16 | Type | Min | Max |
| **Arguments Passed** | WrVal\_Arg | uint16 | Full | Full |
|  | WrAddr\_Arg | pointer to volatile uint16 | Valid values:  0xFFD62044 | |
| **Return Value** | N/A (void) |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Name** | WrProtdRegEcm\_u32 | Type | Min | Max |
| **Arguments Passed** | WrVal\_Arg | uint32 | Full | Full |
|  | WrAddr\_Arg | pointer to volatile uint32 | Valid values:  0xFFD62004  0xFFD62008  0xFFD6200C  0xFFD62010  0xFFD62014  0xFFD62018  0xFFD6201C  0xFFD62020  0xFFD62024  0xFFD62028  0xFFD62034  0xFFD62038  0xFFD62048  0xFFD6204C  0xFFD62050  0xFFD62054 | |
| **Return Value** | N/A (void) |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Name** | WrProtdRegFlmd\_u32 | Type | Min | Max |
| **Arguments Passed** | WrVal\_Arg | uint32 | Full | Full |
|  | WrAddr\_Arg | pointer to volatile uint32 | Valid values:  0xFFA00000 | |
| **Return Value** | N/A (void) |  |  |  |

##### Design Rationale

*These functions will perform the correct sequence of writes to protected registers. These are designed such that the function attempts the write sequence up to 3 times with increasing levels of interrupt disabling for each attempt (no interrupts disabled->Os interrupts disabled->All interrupts disabled). Since these functions are broken out based on the peripheral register set to be written and width of register write, DET error checking is done to ensure the implementer is using the correct API. A DET error is also set if for some reason the 3 write attempts all fail.*

##### Processing

*See source code for implementation.*

#### Nexteer Software Reset

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Name** | NxtrSwRst | Type | Min | Max |
| **Arguments Passed** | McuDiagcData0\_Arg | McuDiagc1 | Full | Full |
|  | McuDiagcData1\_Arg | uint32 | Full | Full |
| **Return Value** | N/A (void) |  |  |  |

##### Design Rationale

*This function exists to ensure that before calling a reset, the caller is properly indicating what the source of the reset is and that this type of reset is part of the known list of reset causes. Additionally, the second parameter is able to store more information along with the reset. This processing is done in a separate component, but the interface is the “SetMcuDiagcIdnData” API. Additionally, the Renesas SAN indicates that before any software reset, the register containing the reset cause flags shall be cleared. The Mcu\_PerformReset() function is then called to perform the actual reset. In the event that there is an issue where this function doesn’t actually perform a reset as expected, a while loop is entered at the end of this function, which could likely lead to a hardware watchdog timeout if this loop is entered.*

##### Processing

*See source code for implementation.*

#### Nexteer Software Reset From Exception

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Name** | NxtrSwRstFromExcpn | Type | Min | Max |
| **Arguments Passed** | McuDiagcData0\_Arg | McuDiagc1 | Full | Full |
|  | McuDiagcData1\_Arg | uint32 | Full | Full |
| **Return Value** | N/A (void) |  |  |  |

##### Design Rationale

*This function exists to ensure that before calling a reset from a hardware exception, the caller is properly indicating what the source of the reset is and that this type of reset is part of the known list of reset causes. This function will clear all ECM status registers prior to issuing a reset to ensure a known state of these registers after the reset. Additionally, the second parameter to this function is used to store more information along with the reset.*

*In an effort to try to capture useful data relating to the exception, the internal logic of this function will attempt to at least store the register value that contains the program counter of when the exception occurred. In order for this to work, a value of “0x0000000” must be passed to the “McuDiagcData1\_Arg” argument in the case of an FE exception, and an value of “0xFFFFFFFF” must be passed to the “McuDiagcData1\_Arg” argument in the case of an EI exception. In case of any other values in the “McuDiagcData1\_Arg” , this function assumes the caller of this function has already setup data that is desired to be stored in this argument, and therefore leaves it unmodified.*

*This storage of this reset information is done in a separate component, but the interface is the “SetMcuDiagcIdnData” API.*

*This function also intentionally attempts to write the error output pin to an “error” state as a redundant mechanisms for putting the system into a safe state in the event that the software reset (which also should drive the system to a safe state) doesn’t properly execute.*

*Additionally, the Renesas SAN indicates that before any software reset, the register containing the reset cause flags shall be cleared. The Mcu\_PerformReset() function is then called to perform the actual reset. In the event that there is an issue where this function doesn’t actually perform a reset as expected, a while loop is entered at the end of this function, which could likely lead to a hardware watchdog timeout if this loop is entered.*

##### Processing

*See source code for implementation.*

## P1XC Micro Variants

#### Protected Write APIs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Name** | WrProtdRegEcmm0\_u32 | Type | Min | Max |
| **Arguments Passed** | WrVal\_Arg | uint32 | Full | Full |
|  | WrAddr\_Arg | pointer to volatile uint32 | Valid values:  0xFFD60000  0xFFD60004 | |
| **Return Value** | N/A (void) |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Name** | WrProtdRegEcmc0\_u32 | Type | Min | Max |
| **Arguments Passed** | WrVal\_Arg | uint32 | Full | Full |
|  | WrAddr\_Arg | pointer to volatile uint32 | Valid values:  0xFFD61000  0xFFD61004 | |
| **Return Value** | N/A (void) |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Name** | WrProtdRegEcm0\_u32 | Type | Min | Max |
| **Arguments Passed** | WrVal\_Arg | uint32 | Full | Full |
|  | WrAddr\_Arg | pointer to volatile uint32 | Valid values:  0xFFD62000  0xFFD62004  0xFFD62008  0xFFD6200C  0xFFD62010  0xFFD62014  0xFFD62018  0xFFD6201C  0xFFD62020  0xFFD62024  0xFFD62028  0xFFD6202C  0xFFD62030  0xFFD62034  0xFFD62038  0xFFD6203C  0xFFD62048  0xFFD6204C  0xFFD62050  0xFFD62054  0xFFD6205C  0xFFD62060  0xFFD62064  0xFFD62068  0xFFD6206C  0xFFD62070  0xFFD62074  0xFFD62078 | |
| **Return Value** | N/A (void) |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Name** | WrProtdRegFlmd\_u32 | Type | Min | Max |
| **Arguments Passed** | WrVal\_Arg | uint32 | Full | Full |
|  | WrAddr\_Arg | pointer to volatile uint32 | Valid values:  0xFFA00000 | |
| **Return Value** | N/A (void) |  |  |  |

##### Design Rationale

*These functions will perform the correct sequence of writes to protected registers. These are designed such that the function attempts the write sequence up to 3 times with increasing levels of interrupt disabling for each attempt (no interrupts disabled->Os interrupts disabled->All interrupts disabled). Since these functions are broken out based on the peripheral register set to be written and width of register write, DET error checking is done to ensure the implementer is using the correct API. A DET error is also set if for some reason the 3 write attempts all fail.*

##### Processing

*See source code for implementation.*

#### Nexteer Software Reset

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Name** | NxtrSwRst | Type | Min | Max |
| **Arguments Passed** | McuDiagcData0\_Arg | P1mcDiagc1 | Full | Full |
|  | McuDiagcData1\_Arg | uint32 | Full | Full |
| **Return Value** | N/A (void) |  |  |  |

##### Design Rationale

*This function exists to ensure that before calling a reset, the caller is properly indicating what the source of the reset is and that this type of reset is part of the known list of reset causes. Additionally, the second parameter is able to store more information along with the reset. This processing is done in a separate component, but the interface is the “SetMcuDiagcIdnData” API. Additionally, the Renesas SAN indicates that before any software reset, the register containing the reset cause flags shall be cleared. The Mcu\_PerformReset() function is then called to perform the actual reset. In the event that there is an issue where this function doesn’t actually perform a reset as expected, a while loop is entered at the end of this function, which could likely lead to a hardware watchdog timeout if this loop is entered.*

##### Processing

*See source code for implementation.*

#### Nexteer Software Reset From Exception

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Name** | NxtrSwRstFromExcpn | Type | Min | Max |
| **Arguments Passed** | McuDiagcData0\_Arg | P1mcDiagc1 | Full | Full |
|  | McuDiagcData1\_Arg | uint32 | Full | Full |
| **Return Value** | N/A (void) |  |  |  |

##### Design Rationale

*This function exists to ensure that before calling a reset from a hardware exception, the caller is properly indicating what the source of the reset is and that this type of reset is part of the known list of reset causes. This function will clear all ECM status registers prior to issuing a reset to ensure a known state of these registers after the reset. Additionally, the second parameter to this function is used to store more information along with the reset.*

*In an effort to try to capture useful data relating to the exception, the internal logic of this function will attempt to at least store the register value that contains the program counter of when the exception occurred. In order for this to work, a value of “0x0000000” must be passed to the “McuDiagcData1\_Arg” argument in the case of an FE exception, and an value of “0xFFFFFFFF” must be passed to the “McuDiagcData1\_Arg” argument in the case of an EI exception. In case of any other values in the “McuDiagcData1\_Arg” , this function assumes the caller of this function has already setup data that is desired to be stored in this argument, and therefore leaves it unmodified.*

*This storage of this reset information is done in a separate component, but the interface is the “SetMcuDiagcIdnData” API.*

*This function also intentionally attempts to write the error output pin to an “error” state as a redundant mechanisms for putting the system into a safe state in the event that the software reset (which also should drive the system to a safe state) doesn’t properly execute.*

*Additionally, the Renesas SAN indicates that before any software reset, the register containing the reset cause flags shall be cleared. The Mcu\_PerformReset() function is then called to perform the actual reset. In the event that there is an issue where this function doesn’t actually perform a reset as expected, a while loop is entered at the end of this function, which could likely lead to a hardware watchdog timeout if this loop is entered.*

##### Processing

*See source code for implementation.*

## Interrupt Functions

*None*

## Module Internal (Local) Functions

## Local Function #1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Name** |  | Type | Min | Max |
| **Arguments Passed** |  |  |  |  |
|  |  |  |  |  |
| **Return Value** |  |  |  |  |

## Design Rationale

## Processing

## GLOBAL Function/Macro Definitions

## GLOBAL Function #1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function Name** |  | Type | Min | Max |
| **Arguments Passed** |  |  |  |  |
|  |  |  |  |  |
| **Return Value** |  |  |  |  |

## Design Rationale

## Processing

# Known Limitations with Design

Functionality for P1XC devices that is currently supported by this component is only targeted for P1MC variants (not designed for P1HC variants).

# UNIT TEST CONSIDERATION

Abbreviations and Acronyms

| **Abbreviation or Acronym** | **Description** |
| --- | --- |
|  |  |
|  |  |

Glossary

**Note**: Terms and definitions from the source “Nexteer Automotive” take precedence over all other definitions of the same term. Terms and definitions from the source “Nexteer Automotive” are formulated from multiple sources, including the following:

* ISO 9000
* ISO/IEC 12207
* ISO/IEC 15504
* Automotive SPICE® Process Reference Model (PRM)
* Automotive SPICE® Process Assessment Model (PAM)
* ISO/IEC 15288
* ISO 26262
* IEEE Standards
* SWEBOK
* PMBOK
* Existing Nexteer Automotive documentation

| **Term** | **Definition** | **Source** |
| --- | --- | --- |
| MDD | Module Design Document |  |
| DFD | Data Flow Diagram |  |

References

| **Ref. #** | **Title** | **Version** |
| --- | --- | --- |
| 1 | AUTOSAR Specification of Memory Mapping (Link:[AUTOSAR\_SWS\_MemoryMapping.pdf](http://www.autosar.org/download/R4.0/AUTOSAR_SWS_MemoryMapping.pdf)) | v1.3.0 R4.0 Rev 2 |
| 2 | MDD Guideline EA4 01.00.01.docx | EA4 01.00.01 |
| 3 | [Software Naming Conventions.doc](http://misagweb01.nexteer.com/eRoomReq/Files/erooms8/NextGeneration/0_fc55f/Software%20Naming%20Conventions%2003x(In%20Work).doc) | 1.0 |
| 4 | EA4 Software Naming Conventions 01.01.00.docx | 01.01.00 |