|  |
| --- |
| **I2C Pattern Description** |
| **Version 1.0.0** |
| **Hau Sy Le** |
| **FED1/PER/TIMER** |

# Table of contents

[Table of contents 2](#_Toc532729683)

[1. Introduction 3](#_Toc532729684)

[2. Startup package 4](#_Toc532729685)

[2.1. Startup sequence for ARM processor 4](#_Toc532729686)

[2.2. Individual ARM exception vectors table 4](#_Toc532729687)

[2.3. Exception handlers 5](#_Toc532729688)

[3. Software driver package 6](#_Toc532729689)

[3.1. Heading 6](#_Toc532729690)

[3.2. Heading 6](#_Toc532729691)

[3.3. Heading 6](#_Toc532729692)

[4. Heading 7](#_Toc532729693)

[4.1. Heading 7](#_Toc532729694)

[4.2. Heading 7](#_Toc532729695)

[4.3. Heading 7](#_Toc532729696)

[5. Reference documents 8](#_Toc532729697)

# Introduction

I2C has a functional verification package. This package contains 4 sub-packages as listed as below.

* **Startup package**

This package contains a startup sequence and an individual ARM exception vectors table written in ARM assembly for ARM processor and exceptions handler written in C.

* **Software driver package**

This package contains software driver for I2C and common modules such as PFC, CPG, GIC and DMAC.

* **Test patterns**

This package contains all I2C test patterns.

* **Scatter files**

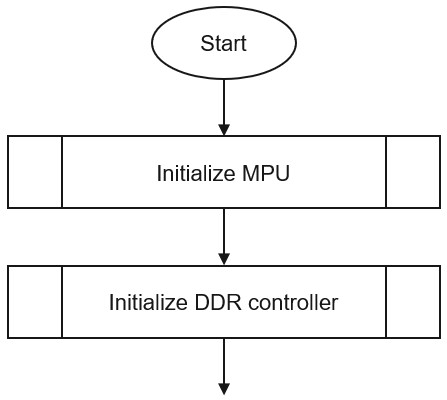
This package contains scatter files necessary for the compilation step.

This document describes all the detail of this package.

# Startup package

## Startup sequence for ARM processor

The startup sequence for ARM processor is necessary to make the ARM processor to operate correctly. Each ARM processor version and architecture has its own startup sequence. The startup sequence for ARMv8-R 32-bits architecture is demonstrate as shown as below.



After the initialization of the DDR controller, the next sequence will be the C library initialization and the main function will be called after this sequence complete.

## Individual ARM exception vectors table

The ARM exception vectors table is copied to an individual source file and it will be placed in either internal SRAM memory or the DDR memory.

During execution, if there is any exception occurs, the ARM processor cannot access to the original exception vectors table which is located at the top of the BSC region because some of LSI pins are multiplex between I2C and LBSC, they will not available for LBSC to access to BSC region while I2C is in-use.

Therefore, an individual ARM exception vectors table located in a memory region outside of the BSC region is necessary for I2C patterns.

## Exception handlers

Exception handlers for ARM processor contains necessary action for each exception. Currently, only IRQ exception will be handled, the other exceptions will lead the processor to an infinity loop to prevent for incorrect result.

# Software driver package

This package contains software drivers for I2C and some common module such as PFC, CPG, DMAC and GIC. It has been written in C with object-oriented method to help developing the test patterns more easily and well organized.

This document only describes in detail for I2C software driver. I2C software driver supplies a set of functions to help to quickly configure an I2C channel to a desired operation mode, to send and receive data, get status and manage interrupts for the I2C.

## Master operation

## Slave operation

# Test patterns

This package contains a set of individual patterns written in C. Each pattern targeting to a specific test item in the CT check list.

All patterns use the software checking method that is comparing the test result with a golden result right in the code.

## Common check

### Register attributes check

This pattern checks the I2C’s registers attributes to see if they are consistency with the hardware manual. The expected results are:

* Registers address is correct
* Registers can be accessed
* Qualification (RW/RO/WO) is correct

The test procedure is shown as below.

Dump FAIL

result != golden?

Dump PASS

N

Y

REG 🡨 0xFFFFFFFF

result(1) 🡨 REG

REG 🡨 0x00000000

result 🡨 REG

Check register

Repeat for all registers

|  |  |
| --- | --- |
| **Note** | (1) **result** is an array containing read back data. It will be compared with the **golden** array element-by-element at the end of the test. |

### Module-stop check

This pattern checks the effect of a module-stop does to the I2C. The expected results are:

* I2C can be accessed normally before a module-stop.
* I2C cannot be accessed during a module-stop.
* Registers value are retained after a module-stop.
* I2C can be accessed normally after a module-stop.

The test procedure is shown as below.

Dump FAIL

result != golden?

Dump PASS

N

Y

Check register

REG 🡨 0xFFFFFFFF

Assert module-stop

Check register

Deassert module-stop

Result 🡨 REG

Check register

### Software reset check

This pattern checks the effect of a software reset does to the I2C. The expected results are:

* I2C can be accessed normally before a software reset.
* I2C cannot be accessed during a software reset.
* Registers value are reset after a software reset.
* I2C can be accessed normally after a software reset.

The test procedure is shown as below.

Dump FAIL

result != golden?

Dump PASS

N

Y

Check register

REG 🡨 0xFFFFFFFF

Assert software reset

Check register

Deassert software reset

Result 🡨 REG

Check register

### Pad connection check

### Interrupt check

## Master operation check

## Slave operation check

# Appendix A

**Assembly pattern vs. C pattern comparision**

This appendix describes the different between assembly and C and the reason why we should switch to use C in CT verification.

Generally, people select the language at the start of a project. The selection is based on the following points.

* **Portability**

Portability is the most important factor in language selection. If the source code is not portable then we have to pay a lot of time to either rewrite or modify it when the design changes.

* **Maintainability**

Programing language should be easy to understand and does not require a lot of time to study. Otherwise, it will take a lot of time for small changes.

* **Availability**

Compiler and may be an IDE should be easily available and ready to use.

* **Efficiency**

Language should have a good efficiency and bug rate should be less.

* **Development time**

Language should take less amount of time to develop the project.

Every programing language has its own advantage and disadvantages. We cannot say any language good or bad at all. It depends on the situation. In CT verification, there are some peace of code need to be written in assembly language because simply C cannot, but the rest of code should be written in C because of its advantages comparing with assembly language.

The different between C and assembly are described below.

* **Portability**

The C language provides portability and does not depend on a specific platform. This is the biggest advantage of. The code which was written in C could be easily reused on a different platform. Besides, assembly does not provide the portability and source code specific to a processor because assembly instruction depends on the processor architecture. For example, in R-CarV3U, the C code can be used for both ARM Cortex-R52 processor and ARM Cortex-A76 processor even they are different in architecture. With assembly, all the code must be rewritten.

* **Maintainability**

It is hard to maintain assembly code because it is a unique language. Beside of understanding the syntax of the language, we must understand the function of the code. Not like assembly, C is an easy to understand programing language. It does not cost a lot of time to understand the code and maintain it.

* **Availability**

In Renesas, assembly and C has the same availability since assembly code and C code can be compiled using ARM Compiler 6.9 package.

* **Efficiency**

We cannot deny the fact that well manual coded assembly code is more efficiency than other language. But the fact is, not much human can code well in assembly. Now a day, many compilers handles the optimization a lot better than human. Besides, compilers give a lot of options to balance the output between code size and code efficiency.

* **Development time**

Start from the beginning of a project, assembly requires a huge of time to develop while C only cost a small amount of time. Besides, C code can be used in other projects without any modification leading to reduce a lot of development time.

While C has a lot of advantages, some cases requires assembly such as the ARM exception vectors table, GIC registers access. Therefore, the ARM startup sequence still written in assembly.

# Reference documents

[1] Author, *Name*, Year.