

Atmel ClassB Library for SAMC2x/D2x

User Manual

Example Application with Atmel ClassB Library 2.1

RELEASED

Software User Manual

040103, 1.4, 11-3-2016

Edition 2-23-2015

Published by:
Hitex GmbH
Greschbachstr. 12, 76229 Karlsruhe, Germany
© 2016 Hitex GmbH
All Rights Reserved.

Legal Disclaimer

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics. With respect to any examples or hints given herein, any typical values stated herein and/or any information regarding the application of the product, Hitex Development Tools GmbH hereby disclaims any and all warranties and liabilities of any kind, including without limitation, warranties of non-infringement of intellectual property rights of any third party.

Information

For further information on technology, delivery terms and conditions and prices, please contact the nearest Hitex Office (www.hitex.com).

Document Change History

Date	Version	Changed By	Change Description
2-23-2015	0.1	Grohmann Stefan (HTX)	initial document creation, draft
2-25-2015	0.2	Grohmann Stefan (HTX)	Measurements added
3-11-2015	0.3	Timo Löffler (HTX)	Measurements added and tables corrected
3-11-2015	1.0	S. Grohmann	Released
6-02-2015	1.0.4	S. Grohmann	Typos
7-03-2015	1.0.5	S. Grohmann	Export to pdf
7-22-2015	1.1	S. Grohmann	SAM-C21 Support added
9-22-2015	1.2	S. Grohmann	Limitation for Core Test (BIST) added
04-15-2016	1.3	S. Grohmann	Update for SAMC20 derivative compliance
11-03-2016	1.4	S. Grohmann	Update for minor adaptations and changes Library Version 2.1

We Listen to Your Comments

Is there any information in this document that you feel is wrong, unclear or missing?

Your feedback will help us to continuously improve the quality of this document.

Please send your comments (including a reference to this document) to:

application@hitex.de



Table of Contents

1	Abbreviations, definitions and document scope.....	8
1.1	Abbreviations	8
1.2	Definitions.....	8
1.3	Document scope	8
1.4	Related documents	8
2	Introduction	9
2.1	Purpose of the document	9
2.2	Overview	9
3	Device specifications.....	10
3.1	Device Overview	10
3.1.1	Generic features (relevant for ClassB library).....	10
4	Software Architecture	12
4.1	IEC60730 Requirements Coverage	12
4.2	System Design and Design Method.....	14
4.3	Application Framework and Example	14
4.4	Safety Period Monitoring.....	15
4.5	CPU Register Test	15
4.6	Program Counter Test.....	15
4.7	Interrupt Logic Test	15
4.8	Address Decoder Test	15
4.9	Memory Test	16
4.9.1	Volatile Memory Test	16
4.9.1	Non-Volatile Memory Test.....	16
4.10	Clock Test	17
4.11	Watchdog Timer Test.....	17
5	Software API	18
5.1	API Functions Overview.....	18
5.2	Pre Operational Test (POST).....	19
5.2.1	WatchdogTest (ClassB_WDTest_POST)	21
5.2.2	Interrupt Logic Test (ClassB_IRQTest_POST).....	21
5.2.3	Clock Test (ClassB_CLKTest_POST)	22
5.2.4	CPU Test (ClassB_CPUTest_POST)	22
5.2.5	Program Counter Test (ClassB_PCTest_POST).....	22
5.2.6	Address Decoder Test (ClassB_ADDRDECTest_POST).....	22
5.2.7	Volatile Memory Test (ClassB_RAMTest_POST)	22
5.2.8	Flash Content Test (ClassB_FLASHTest_POST)	22
5.2.9	Hardware CRC functional Test (configurable)	22
5.3	Runtime Tests (BIST).....	23
5.3.1	Controller Hardware Tests	23
5.3.1.1	Core Test (ClassB_CPUTest_BIST)	23
5.3.1.2	PC Test (ClassB_PCTest_BIST)	23
5.3.1.3	Interrupt Handler	23
5.3.2	Functional and Occurrence Tests	24
5.3.2.1	Clock Test	24
5.3.2.2	FLASH Test.....	25
5.3.2.3	RAM Test	25
5.3.3	Safety Process Monitoring Test	26
5.3.4	Supply Functions.....	27
5.3.4.1	Fail Safe State Example (FailSafe)	27
5.3.4.2	SPMon Call Back Example (CheckSPMonCallBack)	27
5.3.4.3	SysTick Interrupt Handler Example (SysTick_Handler).....	27
5.4	Configuration API Definitions Overview	28
5.4.1	CPU Parameters	31
5.4.2	Clock Source Configuration	31

CONFIDENTIAL

5.4.3	Flash Test Configuration	31
5.4.4	RAM Test Configuration	32
5.4.5	POST Test Configuration	32
5.4.6	BIST Test Configuration	32
5.5	API Types Overview	32
5.5.1	ClassB_Callback_t	33
5.5.2	ClassB_testResult_t	33
5.5.3	ClassB_Proc_ID_t	33
5.5.4	ClassB_FlashSign_t	33
5.5.5	Critical Data	33
5.6	SP Monitoring	34
5.6.1	Definition of the Safety Period	34
5.6.2	SpMon Test Implementation	34
5.6.3	Example	34
5.6.4	Timing considerations	34
6	Environment, Usage and Limits	36
6.1	Application Environment	36
6.2	Example Applications	36
6.3	Application Keil_SAMD20_ClassB	37
6.4	Memory Usage	37
6.5	Runtime Requirements	38
6.6	Safety Period	39
6.7	Limitations	39
6.8	Interaction with other tools like code generators	39
6.9	Trouble Shooting	40

List of Figures

Figure 3.1	Overview	11
Figure 4.1	Software Architecture / Software Layer.....	12
Figure 4.2	V-Model	14
Figure 4.3	March algorithm.....	16
Figure 4.4	MarchC coverage and execution time.....	16
Figure 4.5	Flash Memory Mapping Example SAMD20J180	17
Figure 5.1	POST flow	20
Figure 5.2	Configuration API defines	28
Figure 5.3	ARM-Keil MDK Configuration Wizard Example	31
Figure 5.4	API Types Overview.....	32
Figure 6.1	SAMD20 Xplained evaluation board	37

List of Tables

Table 1.1	Abbreviations.....	8
Table 1.2	Definitions.....	8
Table 4.1	Functional References by Table H.1.....	12
Table 4.2	Library Functions Referenced to ClassB Requirements	13
Table 4.3	Library Functions to Control States and Results.....	13
Table 5.1	API overview	18
Table 5.2	POST Test Overview	21
Table 5.3	Interrupt Test CallBack Overview.....	21
Table 5.3	Category 1 Controller Hardware Tests	23
Table 5.4	Category 2 Functional/Occurrence Tests	24
Table 5.5	Category 3 Safety Process Monitoring.....	26
Table 5.6	Supply Functions.....	27
Table 6.1	Memory Usage of the Modules	38
Table 6.2	Time Consumption of the Modules	38
Table 6.3	Trouble Shooting.....	40

1 Abbreviations, definitions and document scope

1.1 Abbreviations

Table 1.1 Abbreviations

Abbreviation	Comment
BIST	Build In Self-Test (Runtime Tests)
DAP	Device Access Port
ECU	Electronic Control Unit
IO	Input/Output
JTAG	Joint Test Action Group
LED	Light Emitting Diode
PCB	Printed Circuit Board
POST	Post Operational Self-Test (Startup Tests)
USB	Universal Serial Bus

1.2 Definitions

Table 1.2 Definitions

Definition	Comment
-	-

1.3 Document scope

This document contains the User Manual for the Example Application with Atmel ClassB Library 2.1 of Atmel ClassB Library for SAMC2x/D2x.

1.4 Related documents

Changes to all of the following documents can have impact on this document:

- [REF_001] Software requirements specification
- [REF_002] Software architectural and interface design specification

Note that changes in this document can have impact on the following documents:

-

Referenced documents (for information only, no requirements)

-

2 Introduction

This chapter will provide an introduction to the user manual to the project Atmel ClassB Library for SAMC2x/D2x.

2.1 Purpose of the document

The purpose of this document is to describe the decomposition of the Atmel ClassB Library for SAMC2x/D2x software system to integrate it into user defined software modules, setup and test.

2.2 Overview

- Chapter 3 contains the Device Specification.
- Chapter 4 contains the Software Architecture.
- Chapter 5 contains the Software API.
- Chapter 6 contains the Environment, Usage and Limits.

3 Device specifications

Atmel provides ARM Cortex M0+ based devices in the SAM D series. With SAMC20 ,SAMC21, SAMD20 and SAMD21 two series of Cortex M0+ devices are specified containing a rich set of peripherals but based on the same core technology.

Device Overview:

The SAMC20,SAMC21, SAMD20 and SAMD21 series devices combine the performance and energy efficiency of an ARM Cortex-M0+ based MCU with an optimized architecture and peripheral set. The following key features are available in the SAMC2x/SAMD2x series devices:

CPU Subsystem

- CPU Core
 - High Performance 32-bit Cortex-M0+ CPU
 - ARM 16-bit and subset of 32-bit Thumb2 instruction set
 - High code density with 32-bit performance
 - Single cycle 32-bit hardware multiplier
 - System timer (SysTick) for Operating System support
 - Ultra low power consumption
- Nested Vectored Interrupt Controller (NVIC)
- Event System Unit (EVSYS) for direct peripheral-to-peripheral communication and signaling
- CRC32 hardware unit

On-Chip Memories

- 32 kbytes on-chip SRAM
- up to 256 kbytes on-chip Flash program and data memory

Packages

- TQFP, QFN 64 pin
- TQFP, QFN 48 pin
- TQFP, QFN 32 pin

3.1 Device Overview

3.1.1 Generic features (relevant for ClassB library)

The implementation of ARM Cortex M0+ CPUs provides common system architecture features.

- Cores, Bus
 - Core Registers and Stack
 - ARM Cortex AMBA and AHB-Lite Bus System
 - Peripheral Bus System
 - Trap generation
 - NVIC: core attached vector interrupt controller
- Memory
 - SRAM without parity
 - Flash without ECC or parity, build in and self-programmable
 - Flash locking mechanism
- System Peripherals
 - WatchDog Timer
 - Reset structure
 - Internal Oscillators, Digital PLL
 - Brown out detection
 - Hardware CRC32 unit

The common architectural items are the reason to enable a common ClassB library with little differences in the assembler coding and little differences in the interrupt vectors for several SAMC2x/SAMD2x derivatives.

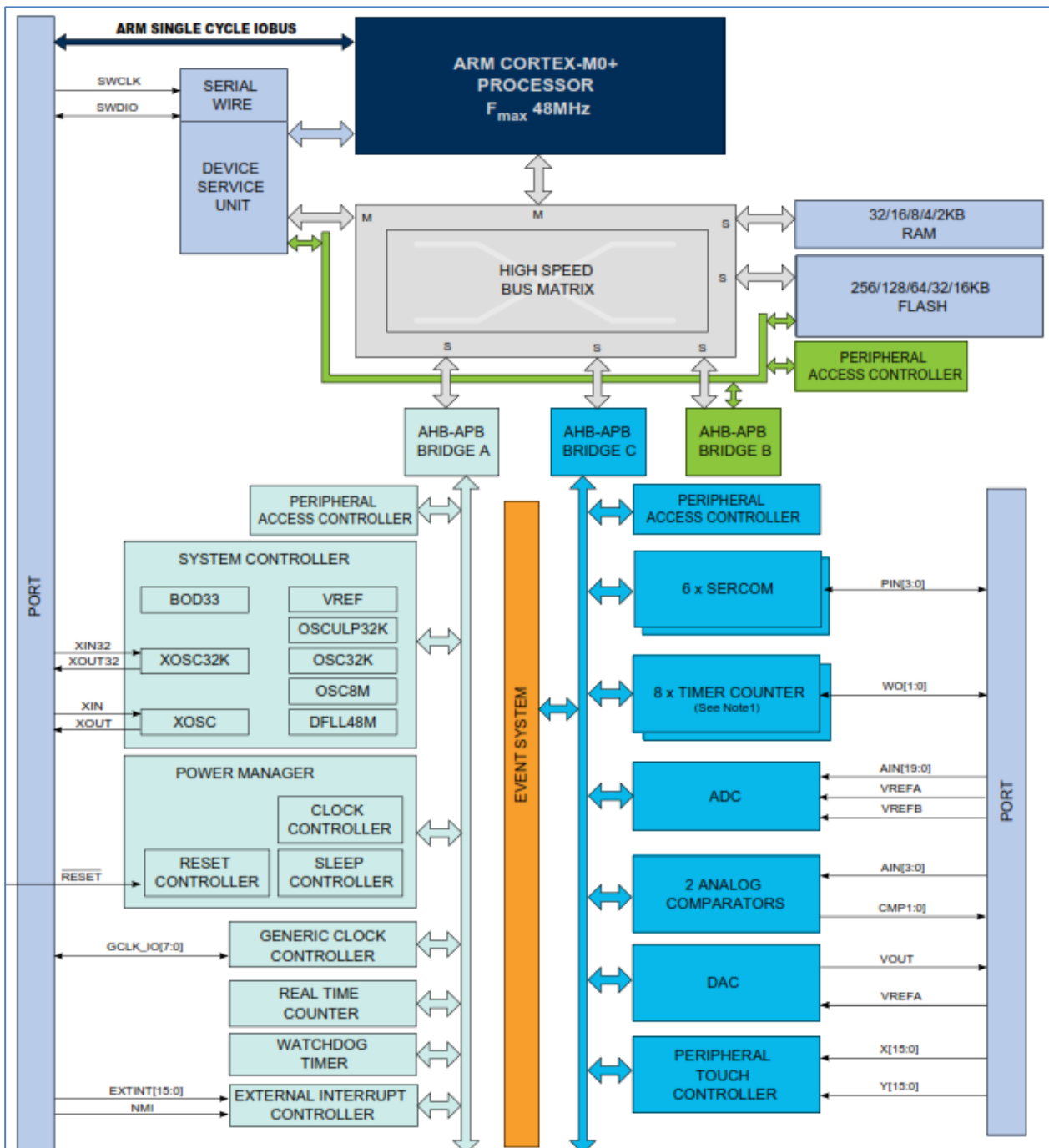


Figure 3.1 Overview

4 Software Architecture

The architecture of the ClassB library is mainly the same as the architecture of a hardware library. The most processes can be called under control of the above layer (application or task) which is responsible for the safety monitoring. In parallel the hardware layer can be accessed with no major inter-effects.

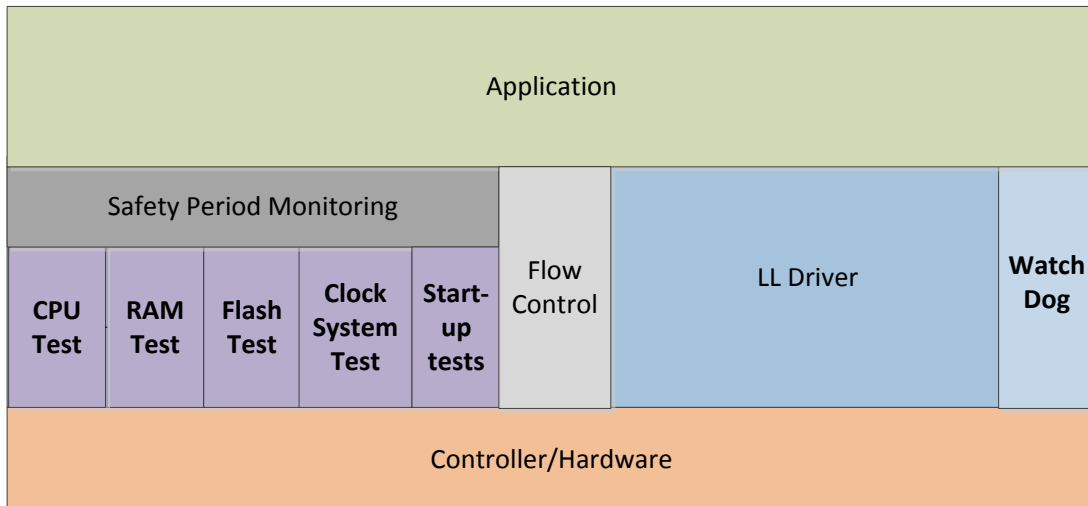


Figure 4.1 Software Architecture / Software Layer

4.1 IEC60730 Requirements Coverage

The key requirements are defined in the IEC 60730-1 Annex H standard called “requirements for electronics control” from 1/2010. This defines that home appliances, have to comply with the ICE 60730 and IEC 60335. For MCU systems the IEC 60730 Annex H defines the details of tests and diagnostics and measures to provide proper and safe operations of embedded software and hardware in household appliances.

The structure of controls is specified with

- Functional test structure on a single channel system
- Periodic self-test, periodic check of critical functions non interfering the applications operation on a single channel system

The software is classified in Annex H into 3 classes of automatic controls

Class A: Intended not to rely on safety of the equipment

Class B: Intended to prevent unsafe operations of the equipment

Class C: Intended to prevent special hazardous functions of the equipment

For this device being used in home appliance the Class B requirements are the target.

The standard defines 10 of the possible 14 components which have to be tested in single MCU chip (single channel) devices as relevant. The library covers 10 system comprehensive components in its implementation.

Table 4.1 Functional References by Table H.1

#	Test	IEC ref. (Annex H)	Covered Error
1	Register R0...R13	1.1	Stuck at fault
2	Program Counter (PC)	1.3	Stuck at fault
3	Stack Pointer (SP)	1.1	Stuck at fault
4	Address Decoding	1.4	Internal error detection
5	Interrupt Operation	2	Functional or timeslot test
6	CPU clock	3	Frequency match
6	Invariable Memory	4.1	Single Bit fault, multibit faults
7	Variable Memory	4.2	Coupling fault
8	Critical Data	4.2	Coupling fault
9	Race condition and Addressing	4.3	Program execution
10	Internal Data	5.1	Data redundancy

The Atmel ClassB Library provides routines which can be obtained by the user application to check the system at startup or periodically during runtime. The requirements references of the test routines are summarized in the following table:

Table 4.2 Library Functions Referenced to ClassB Requirements

IEC60730 Table H1	Affected Component	Test Routine	Description
1.1	CPU, registers	ClassB_CPUtest_POST	POST test of all CPU registers
1.1	CPU, registers	ClassB_CPUtest_BIST	BIST test of all CPU registers
1.2	CPU, decoding	HardFault_Handler	Decoding and opcode test
1.3	CPU, PC	ClassB_PCtest_POST	POST test of CPU Program Counter
1.3	CPU, PC	ClassB_PCtest_BIST	BIST test of CPU Program Counter
1.3	CPU, PC	ClassB_WDTest_POST	POST test of the watchdog unit
1.3	CPU, PC	ClassB_SPMon_BIST	BIST time slot monitoring
1.4	CPU, Address Decoder	ClassB_ADDRDECTest_POST	BIST test of CPU address decoder
2	Interrupt	ClassB_IRQTest_POST	POST test on interrupt function and priority logic
3	ClockSystem	ClassB_CLKTest_POST	POST test of Clock system and PLL
3	ClockSystem	ClassB_CLKTest_RTCHandler	BIST test of Clock system and Clock monitoring called in an interrupt
4.1	Invariable Memory	ClassB_FLASHTest_POST	BIST test on single bit faults
4.1	Invariable Memory	ClassB_FLASHTest_BIST	POST test on single bit faults
4.2	Variable Memory	ClassB_RAMTest_POST	POST test on DC fault, dynamic and crosslink
4.2	Variable Memory	ClassB_RAMTest_BIST	BIST test on DC fault, dynamic and crosslink
5.1	Internal Data Path	ClassB_CriticalDataValidate	Word protection with multibit redundancy
5.1	Internal Data Path	ClassB_CriticalDataPop	Word protection with multibit redundancy
5.1	Internal Data Path	ClassB_CriticalDataPush	Word protection with multibit redundancy
5.2	Internal Data, Addressing	ClassB_ADDRDECTest_POST	POST test of the address decoder

The library provides measures for most requirements in the table H.1. According to the needs on integrating the library to user applications sometimes more than one measure is offered. To determine a maximum of integration the usage of dedicated hardware can be omitted (e.g. CRC unit).

Table 4.3 Library Functions to Control States and Results

Routine	Affected Test Item	Description
ClassB_CLKTest_Init	Clock System Test	Initialize the BIST Clock test with parameter
ClassB_CLKTest_Status	Clock System Test	Request the test result
ClassB_CLKTest_Reset	Clock System Test	Reset the test result
ClassB_FLASHTest_Status	Flash Content Test	Request the test result
ClassB_FLASHTest_Reset	Flash Content Test	Reset the test result
ClassB_RAMTest_Init	RAM Cell Test	Initialize the BIST RAM test with parameter
ClassB_RAMTest_Status	RAM Cell Test	Request the test result
ClassB_RAMTest_Reset	RAM Cell Test	Reset the test result
ClassB_FLOWTest_Init	Flow Control	Initialize the BIST Flow Control with parameter
ClassB_FLOWTest_Enter	Flow Control	Enter a controlled function
ClassB_FLOWTest_Exit	Flow Control	Exit a controlled function
ClassB_FLOWTest_Probe	Flow Control	Check current Flow Control status
ClassB_FLOWTest_TimerHandler	Flow Control	Check current Flow Control status from a Timer Interrupt
ClassB_POST	Startup Tests	Call of all configured on startup

CheckSPMonCallBack	Safety Period	Callback function checking the BIST test results from a SysTick interrupt
ClassB_FailSafe	Error detection	Example for a safe state routine
HardFault_Handler	System Fault Handler	HardFault System Vector
TC3_Handler	Interrupt Logic Test	Test Vector for Interrupt Logic Test
TC4_Handler	Interrupt Logic Test	Test Vector for Interrupt Logic Test
TC5_Handler	Interrupt Logic Test	Test Vector for Interrupt Logic Test
RTC_Handler	Safety Period	Test Vector for Interrupt Clock System Test

4.2 System Design and Design Method

Traceability of the requirements of the implemented functions are described above, where traceability means the continuous process of referencing requirements to the state and flow of development from the specification over verification and validation to the production test defined by the V-model.

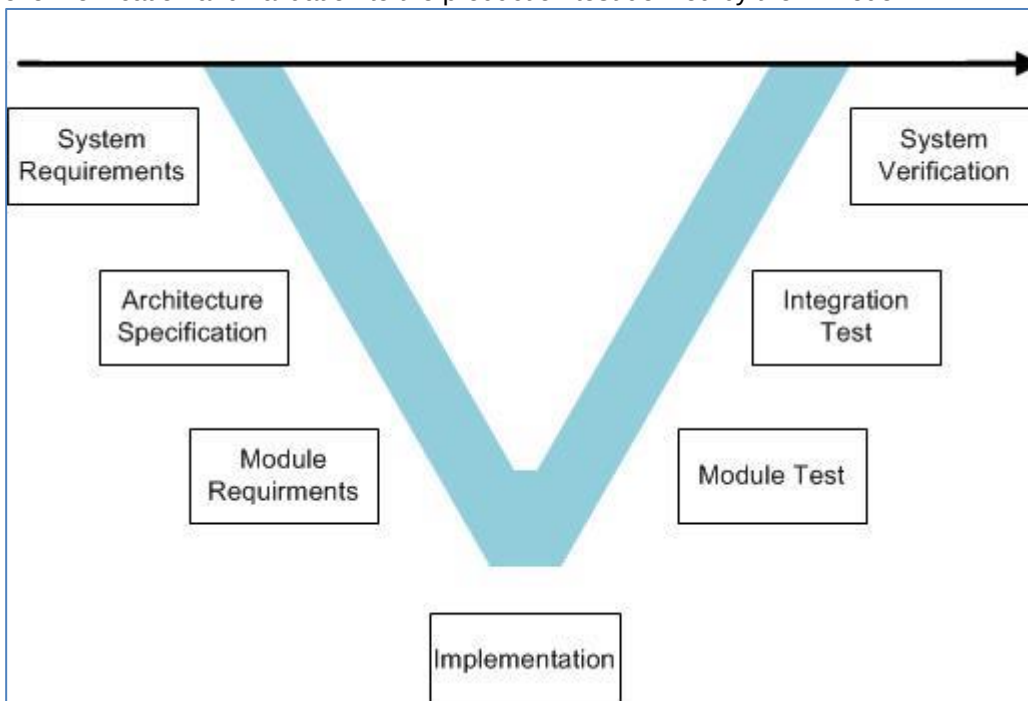


Figure 4.2 V-Model

Related documents are the

- System Requirement Specification (SysRS)
- Architecture Design Specification (ADS)
- Software Requirement Specification (SRS)
- Detailed Design Specifications (DDS)
- Unit Test Specification (UTS)
- Integration Test Specification (ITS)
- Unit Test Documents (generated with TESSY)
- Traceability Matrix (TM)

4.3 Application Framework and Example

The example applications are part of the framework to demonstrate the capabilities of the library and allow testing of the functionality in parts and detail.

Modified Start-up code and all test features are also part of the library to test several controller hardware parts on the uninitialized controller.

4.4 Safety Period Monitoring

An above or so called “super layer” of the ClassB library is the safety period monitoring feature. The main task of this monitor is to ensure that all safety related tests are performed completely in a predefined period called “safety period”. For this feature all BIST test functions are designed to generate a long term status output.

4.5 CPU Register Test

The CPU of the ARM Cortex™-M0+ has a number of registers used during program execution. Nineteen (19) of these registers are read and writeable. Since these registers are all used during program execution in the various core operation modes and states, they need to be tested for “stuck-at” faults and “direct coupling” faults using a pattern test.

These tests are provided for POST and BIST.

POST testing is a destroying test, so the CPU registers will not be retained. Since the POST CPU register tests don't retain register data, it is mandatory to execute this test before the CPU is initialized. All tests are executed in one routine, which allows the fastest test execution.

BIST testing the CPU registers must not corrupt the data of the CPU registers, so all data are restored after testing. To decrease test time and therefore CPU resources, the CPU register BIST testing is partitioned in four separate tests. The first two test steps are testing the general purpose registers. The third test tests the stack pointers. To prevent the system from crashing, all interrupts and exceptions are disabled while running this part of the BIST CPU register test. The last test step is testing the remaining special function CPU registers.

4.6 Program Counter Test

To check the variability of the PC against register stuck the PC is initialized with several addresses from a ROM based array. Each address represents a small separate sub function which performs small code to enable the check of a return value. Each sub function is located to a separate Flash section to force the compiler to generate branches from the main caller function. After loading the PC with the address the sub routine is returning a return value from an also ROM based array. Additional a function global struct is modified. This forces the controller to load the return value. Checking the values against the expected values from the array the function returns passed or failed status.

4.7 Interrupt Logic Test

To check the functionality of the NVIC module of the core a POST test of the interrupt logic is checking interrupt priority, interrupt enabling and interrupt set pending with 3 selected interrupt sources. The interrupts are activated in the same cycle. According to the priority a global variable is filled with individual values after shifting the content. The test routine waits for a maximum time and verifies the content of the variable value against the expected value. A wrong execution order or missing execution will lead to a mismatch. The function returns passed or failed status of the verification.

4.8 Address Decoder Test

The address decoder is tested to provide full access to all volatile memory locations. Checking the hardware will address a specified RAM cell and store a pattern created of the address of the cell. The addresses are selected using all address lines and cover the RAM address space. Reading the RAM cells back and comparison of the content with the address will finalize the test. This test is designed destructive and can only run in the POST stage before the RAM is initialized for the application.

4.9 Memory Test

4.9.1 Volatile Memory Test

The volatile memory tests are focussed on the SRAM of the devices. The derivatives differ in size and amount of volatile memory. The library provides configurations to enable or disable non used parts and adjust the total memory size which is tested check.

The RAM cell check is performing a MarchC- test using the algorithm defined as:

■ **March C:** $< (w0); < (r0, w1); < (r1, w0); < (r0);$
 $> (r0, w1); > (r1, w0); > (r0);$

Figure 4.3 March algorithm

The estimated runtime of the test itself is calculated with:

Algorithm	Fault Coverage								Operation count
	SAF	AF	TF	CF _{in}	CF _{id}	CF _{dyn}	CF _{st}		
March C-	All	All	All	All	All	All	All		10.n

Figure 4.4 MarchC coverage and execution time

The coverage of the MarchC algorithm provides a medium coverage for linked errors.

4.9.1 Non-Volatile Memory Test

The Non-volatile memory test is focussed on the Flash content with the principle of modified checksums. The reference is stored to a part of the flash which is not involved in the checking. The test functions are located in a dedicated area, with the possibility to identify this part.

Depending on the derivative the total size of the Flash memory may be different. To take care on this the library is configurable for the total amount of flash to test and total amount of Flash available in independent defines.

The test mainly uses the on chip CRC engine if configured.

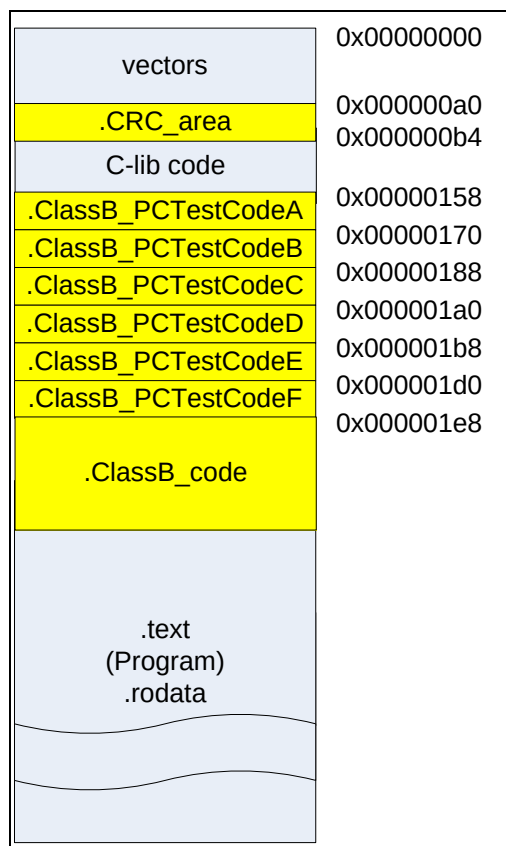


Figure 4.5 Flash Memory Mapping Example SAMD20J180

4.10 Clock Test

The ClassB Library provides a POST function to test the internal clock sources. Additionally, the locking of the PLL is tested and the status register is tested. During runtime, a test of the system clock in relation to the RTC is provided. The RTC is used as the reference clock. The SysTick timer is running with a fraction of the core clock. So the SysTick timer can run in comparison to the RTC with a defined number of timer ticks. The BIST Clock test is calculating the System Tick Timer events and compares this to the defined number with a configured tolerance.

4.11 Watchdog Timer Test

The Watchdog timer provides a reset function if it is not served by the application in the defined time. The ClassB Library provides a test of the watchdog timer and the reset feature. Checking the reset cause, the watchdog timer is initialized, and the function waits for the reset or detects the watchdog reset flag as the reset cause and states the test passed.

5 Software API

5.1 API Functions Overview

All public functions of the ClassB Library are listed here with their parameter and return types.

Table 5.1 API overview

Returnvalue	Function	Parameter	
Address Decoder Test			
ClassB_testResult_t	ClassB_ADDRDECTest_POST	void	
Clock test			
ClassB_testResult_t	ClassB_CLKTest_POST	void	
ClassB_testResult_t	ClassB_CLKTest_Init	uint32_t ratio	uint32_t tolerance
ClassB_testResult_t	ClassB_CLKTest_Status	void	
void	ClassB_CLKTest_Reset	void	
void	ClassB_CLKTest_RTCHandler	void	
void	ClassB_CLKTest_TimerHandler	void	
Core Test			
ClassB_testResult_t	ClassB_CPUPTest_POST	void	
ClassB_testResult_t	ClassB_CPUPTest_BIST	void	
ClassB_testResult_t	ClassB_PCTest_POST	void	
ClassB_testResult_t	ClassB_PCTest_BIST	void	
Flash Test			
ClassB_testResult_t	ClassB_FLASHtest_POST	void	
ClassB_testResult_t	ClassB_FLASHTest_BIST	uint8_t StartMode	
ClassB_testResult_t	ClassB_FLASHTest_Status	void	
void	ClassB_FLASHTest_Reset	void	
Interrupt Test			
ClassB_testResult_t	ClassB_IRQTest_POST	void	
POST			
void	ClassB_POST	void	
RAM test			
ClassB_testResult_t	ClassB_RAMTest_POST	void	
ClassB_testResult_t	ClassB_RAMTest_Init	uint32_t startAddr	uint32_t length
ClassB_testResult_t	ClassB_RAMTest_BIST	void	
ClassB_testResult_t	ClassB_RAMTest_Status	void	
void	ClassB_RAMTest_Reset	void	
Safety Process Monitor			

void	ClassB_SPMon_Enter	ClassB_Proc_ID_t ID	
void	ClassB_SPMon_Exit	ClassB_Proc_ID_t ID	
ClassB_testResult_t	ClassB_SPMon_Init	ClassB_Callback_t CallBack	
void	ClassB_SPMon_TimerHandler	void	
ClassB_testResult_t	ClassB_SPMon_BIST	void	
WDT Test			
ClassB_testResult_t	ClassB_WDTest_POST	void	
Flow Control			
void	ClassB_FLOWTest_Enter	uint8_t ID	
void	ClassB_FLOWTest_Exit	uint8_t ID	
void	ClassB_FLOWTest_Init	ClassB_Callback_tt CallBack	
ClassB_testResult_t	ClassB_FLOWTest_Probe	uint8_t ID	
void	ClassB_FLOWTest_TimerHandler	void	
Interrupt Handler provided in Interrupt Test			
void	NMI_Handler	void	
void	HardFault_Handler	void	
void	RTC_Handler	void	
void	TC3_Handler	void	
Void	TC4_Handler	void	
void	TC5_Handler	Void	
Application			
int	main	void	
void	CheckSPMonCallBack	ClassB_testResult_t SPMon	
Interrupt in the application			
void	SysTick_Handler	void	

5.2 Pre Operational Test (POST)

Before the controller is initialized the POST tests must be performed to test the basic functions of the chip. Most of the tests are held small and less resource consuming which leads to the fact that the tests are most using own setting and do not care for any memory content etc. because the chip will be initialized afterwards. Basic information like the reset cause is checked and there is a second reset performed by the WDT test. The POST function is called from the assembler startup code before any other initialization. POST will restart the controller and this leads to a total destruction of the volatile memory content.

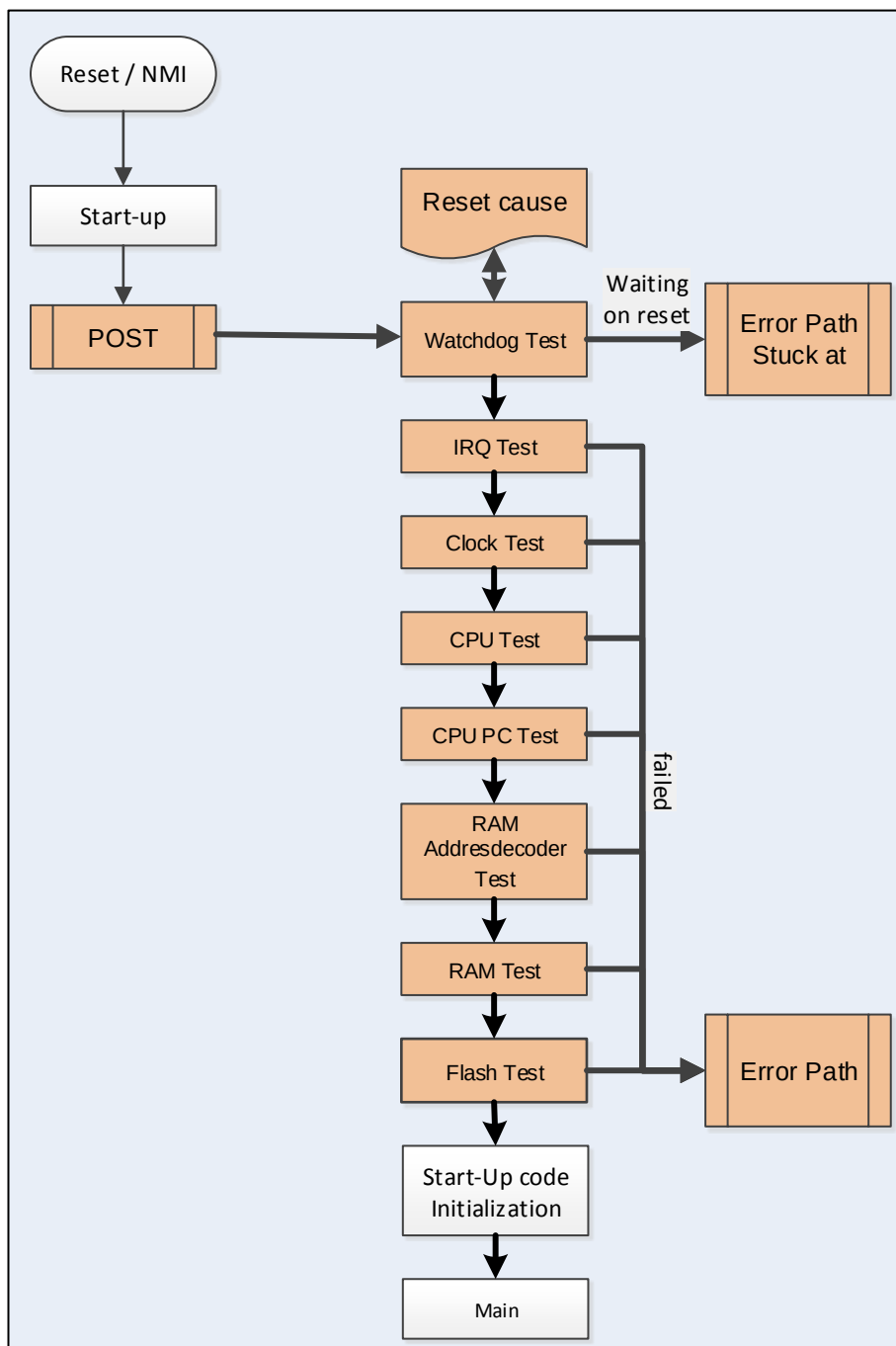


Figure 5.1 POST flow

The POST function runs all POST related tests in a sequence as described below. If any of these tests fails the POST function will not return to its caller function but will stay in an endless loop.

Table 5.2 POST Test Overview

Returnvalue	Function	Parameter	
WDT Test			
ClassB_testResult_t	ClassB_WDCTest_POST	void	
IRQ test			
ClassB_testResult_t	ClassB_IRQTest_POST	void	
Clock test			
ClassB_testResult_t	ClassB_CLKTest_POST	void	
Core Test			
ClassB_testResult_t	ClassB_CPUPTest_POST	void	
ClassB_testResult_t	ClassB_PCTest_POST	void	
Address Decoder Test			
ClassB_testResult_t	ClassB_ADDRDECTest_POST	void	
RAM test			
ClassB_testResult_t	ClassB_RAMTest_POST	void	
Flash Test			
ClassB_testResult_t	ClassB_FLASHTest_POST	void	
Interrupt Handler provided in Interrupt Test			
void	TC3_Handler	void	Optional use see chapter 5.2.2
void	TC4_Handler	void	
void	TC5_Handler	void	

5.2.1 WatchdogTest (ClassB_WDCTest_POST)

As a first test the watchdog timer is tested by configuring it and then waiting for the reset to test the reset mechanism. Only if the startup was caused by this reset (Watch dog Timer reset flag) the test was successful and the function is left with this result.

5.2.2 Interrupt Logic Test (ClassB_IRQTest_POST)

The IRQ tests checks the basic function of the NVIC. 3 interrupts are configured with different priorities and enabled. Then all interrupts are activated at the same time. If the NVIC processes the interrupt vectors in the specified order (according to the priorities) the function returns with a passed result. The Interrupt logic test enables and activates real Interrupt vectors. For this purpose the examples provide a file "interrupts.c" to demonstrate the functionality. The example demonstrates how to implement the Test routine calls.

Table 5.3 Interrupt Test Callback Overview

Interrupt Callback routines instrumentation for Interrupt Test			
void	ClassB_TC3Handler_TestCallback	void	Optional use see for NON – RAM Based interrupt test
void	ClassB_TC4Handler_TestCallback	void	
void	ClassB_TC5Handler_TestCallback	void	

Additional the option ClassB_TEST_POST_IRQ_RAM is available to run the interrupt test with a RAM based Vector Table. This requires a little more RAM but enables the user to leave the interrupt service routines unchanged. The routine uses direct vectors to the test callback routines in a temporary copy of the vector table. The test is identical to the classic test and the result is provided as return value from the test.

5.2.3 Clock Test (ClassB_CLKTest_POST)

To provide functional sources the Clock test checks the functionality of the PLL, the function of the internal and external oscillators. After the test the PLL is set to the defaults and the controller still runs with the internal oscillator (as at the startup) without any speed enhancement or optimization. If the tests are made the function returns with the result.

5.2.4 CPU Test (ClassB_CPUPTest_POST)

The CPU test tests all register in the CPU on stuck and transition faults. The test keeps the CPU running and enables the return to the caller address. All other register content is destroyed after the test. If all test are made the function returns with the result.

5.2.5 Program Counter Test (ClassB_PCTest_POST)

The PC test checks the addressing mechanism of the core. For this test several sub functions are called referenced by a table and verified if executed by a specified return code and a global counter. The sub functions are located in different Flash sections individually specified. If all tests are made the function returns with the result.

5.2.6 Address Decoder Test (ClassB_ADDRDECTest_POST)

The Address Decoder Test checks the addressing of the RAM. For every address line a pattern is generated and the corresponding RAM location is filled with a unique number. This covers the complete internal RAM address space but not every RAM element.. Afterwards the numbers are verified in a separated read action. The RAM content is not restored afterwards. If all tests are made the function returns with the result.

5.2.7 Volatile Memory Test (ClassB_RAMTest_POST)

The RAM test calls an optimized MarchC- test algorithm for the configured RAM. The RAM content is totally destroyed and will not be restored. After the MarchC- test a special adaption of the test is started to test the stack region. This RAM content is restored afterwards to enable the controller continuing its program flow. If all tests are made the function returns with the result.

5.2.8 Flash Content Test (ClassB_FLASHTest_POST)

The Flash test recalculates a CRC value build over the configured Flash content and compares the result with the separate stored modified checksum. This reference checksum is located in a separate flash location which is excluded from the CRC recalculation. If all tests are made the function returns with the result.

5.2.9 Hardware CRC functional Test (configurable)

The ClassB_config.h file provide a define CRC_CHECK_HARDWARE to enable a test of the hardware CRC. The test requires usage of the hardware CRC unit configured for Flash tests. Running the test provides a independent check of the hardware in comparison to a software defined CRC algorithm over a short define Byte array. The results of both tests are compared. The result is indirectly provided with the result of the POST Flash content test (Chapter 5.2.8 Flash Content Test (ClassB_FLASHTest_POST)).

5.3 Runtime Tests (BIST)

During the normal program execution three sets of tests are available.

- 1) Controller Hardware Tests
- 2) Functional and Occurrence Tests
- 3) Safety Period Monitoring Test

Additional some supplying functions are defined.

5.3.1 Controller Hardware Tests

The first category covers all tests which can be performed during the runtime by interrupting the normal program and run a small test function testing a dedicated controller feature

Table 5.4 Category 1 Controller Hardware Tests

Returnvalue	Function	Parameter	
Core Test			
ClassB_testResult_t	ClassB_CPUtest_BIST	void	
ClassB_testResult_t	ClassB_PCTest_BIST	void	
Safe State			
void	ClassB_FailSafe	uint32_t param	
Interrupt Handler provided for Clock Tests			
void	SysTick_Handler	void	
void	RTC_Handler	void	
Interrupt Handler provided for generic use			
void	NMI_Handler	void	
void	HardFault_Handler	void	

5.3.1.1 Core Test (ClassB_CPUtest_BIST)

The core test provides core register (CPU) and program counter (PC) test during the runtime.

ClassB_CPUtest_BIST

The CPU BIST test function tests all CPU register on stuck and transition faults with final recovery of the content. The function is defined as “critical section” which means that the interrupt processing is deactivated during the execution. The function provides tests of the CPU registers, stack pointer register and all status and mode register. If one test fails the function is terminated and returns the result immediately. If all tests are made the function returns with the result. Limitation: The function must not be called from an Interrupt service routine because PSP is not accessible then.

5.3.1.2 PC Test (ClassB_PCTest_BIST)

The PC BIST function tests the functionality of the program counter during the runtime of the application. If one test fails the function is terminated and returns the result immediately. If all tests are made the function returns with the result.

5.3.1.3 Interrupt Handler

The interrupt **NMI_Handler** is used on the POST watchdog reset and on the POST clock test function to detect oscillator faults

The interrupt **RTC_Handler** provides RTC interrupt processing used in the BIST Clock Test.

The interrupt **HardFault_Handler** provides trap features of the ARM core.

Additional the hardware interrupts for the POST Interrupt logic test are provided with the examples. According to the derivative the used IRQs may differ.

5.3.2 Functional and Occurrence Tests

The second category covers tests which may take too long to run them in one call like the RAM test or tests which are depending on the occurrence of events in relation to other software or hardware events e.g. interrupt occurrences.

Table 5.5 Category 2 Functional/Occurrence Tests

Returnvalue	Function	Parameter	
Clock test			
ClassB_testResult_t	ClassB_CLKTest_Init	uint32_t ratio	uint32_t tolerance
ClassB_testResult_t	ClassB_CLKTest_BIST	void	
void	ClassB_CLKTest_RTCHandler	void	
void	ClassB_CLKTest_TimerHandler	void	
void	ClassB_CLKTest_Status	void	
void	ClassB_CLKTest_Reset	void	
Flash Test			
ClassB_testResult_t	ClassB_FLASHTest_BIST	uint8_t StartMode	
ClassB_testResult_t	ClassB_FLASHTest_Status	void	
void	ClassB_FLASHTest_Reset	void	
RAM test			
ClassB_testResult_t	ClassB_RAMTest_Init	uint32_t startAddress	uint32_t length
ClassB_testResult_t	ClassB_RAMTest_BIST	void	
void	ClassB_RAMTest_Status	void	
void	ClassB_RAMTest_Reset	void	
Interrupt Handler provided for Clock Test			
void	SysTick_Handler	void	
void	RTC_Handler	void	
Interrupt Handler provided for generic use			
void	NMI_Handler	void	
void	HardFault_Handler	void	

5.3.2.1 Clock Test

The clock test provides runtime test of the occurrence of timer events compared to a second time base. In the standard configuration the SysTick timer is monitored in comparison to the real time clock (RTC). This provides a continuous test of the clock source and PLL functions because the RTC is driven by a separate clock domain.

ClassB_CLKTest_Init

The function is called with the parameter ratio and tolerance. Where "ratio" the number of events divided by a single event from the reference clock is. If the timer is expected to run with 1000 events per second the ratio should be set to 1000. The parameter "tolerance" is the percentage the ratio can vary before an error is stated by the BIST function. The initialization checks the timer running and enables them if necessary. Additional the configuraton file allows to ignore a number of cycles during the startup phase using the define ClassB_BIST_IGNORE_CLOCKCYCLES. If all initializations are made the function returns with the result.

ClassB_CLKTest_TimerHandler

The function must be called in a cyclic timer interrupt or event. It is defined as inline function to reduce CPU load and interrupt duration. The function does not return any result.

ClassB_CLKTest_RTCHandler

The function must be called in a cyclic RTC interrupt or event. The function starts the evaluation of the occurrences of the ClassB_CLKTest_TimerHandler executions in relation to the RTC interval. The function does not return anything.

ClassB_CLKTest_Status

The clock test status function provides runtime test over the already passed and finalized clock tests. The function should be called once in a safety period. The function returns with the result.

ClassB_CLKTest_Reset

The clock test reset function resets the result of the last clock test. The function should be called once in a safety period after a ClassB_CLKTest_Status call. The function returns nothing.

5.3.2.2 FLASH Test

The flash test provides flash content check. To reduce time consumption the tests can be configured to a size of flash to test per call. If not all flash is selected the function must be called several times to check all the Flash configured with a defined size of memory. The Flash memory tests are monitored to be finalized in the safety period by the process monitoring feature. The Flash test must be initialized with the ClassB_FLASHTest_BIST function and the parameter FLASH_CRC_Restart. If a block tests is made the function returns with the status. The overall result can be checked with the status request (ClassB_FLASHTest_Status).

ClassB_FLASHTest_BIST

The FLASH test provides complete portion wise Flash content test during runtime. The portion tested is configured in the ClassB_config.h file with the define CRC_block_size. If all configured flash is tested the function verified the calculated CRC with the predefine CRC (ClassB_Ref_ChkSum) and returns with the result. The result over the complete Flash area is returned by the function ClassB_FLASHTest_Status. If the function is called again after a complete test it will restart with the testing from the beginning of the configured flash area.

ClassB_FLASHTest_Status

The FLASH test status function provides runtime test over the already passed and finalized Flash test. The function should be called once in a safety period. The function returns with the result.

ClassB_FLASHTest_Reset

The Flash test reset function resets the result of the last full Flash test. The function should be called once in a safety period after a ClassB_FLASHTest_Status call. The function returns nothing.

5.3.2.3 RAM Test

ClassB_RAMTest_Init

The function initializes the BIST RAM test function with its parameter startAddress to start the test from and the size parameter to specify the RAM size which must be tested. If the initialization was successful the function returns the result.

ClassB_RAMTest_BIST

The RAM test provides portion wise RAM test during runtime. The portion tested is directly depending on the buffer size (ClassB_RAM_buffersize) defined in the file ClassB_config.h. If all tests are made the function returns with the result. The result over the complete RAM area is returned by the function ClassB_RAMTest_Status. Integrated Stack-Overflow Test can be configured in the ClassB_config header.

ClassB_RAMTest_Status

The RAM test status function provides runtime test over the already passed and finalized RAM test. The function should be called once in a safety period. The function returns with the result.

ClassB_RAMTest_Reset

The clock test reset function resets the result of the last full RAM test.
The function should be called once in a safety period after a ClassB_RAMTest_Status call. The function returns nothing.

5.3.3 Safety Process Monitoring Test

The third category is a test ensuring the hardware features are tested within a defined period called safety period. This defines the maximum time in which the system must react with the fault detection.

Table 5.6 Category 3 Safety Process Monitoring

Returnvalue	Function	Parameter	
Safety Process Monitor			
void	ClassB_SPMon_Enter	ClassB_Proc_ID_t ID	
void	ClassB_SPMon_Exit	ClassB_Proc_ID_t ID	
enum testResult_t	ClassB_SPMon_Init	ClassB_Callback_t CallBack	
void	ClassB_SPMon_TimerHandler	void	
enum testResult_t	ClassB_SPMon_BIST	void	

Safety Process Tracking (ClassB_SPMon_Enter, ClassB_SPMon_Exit)

Clock Test, Core Test, Flash test, interrupt test and RAM test functions are calling the tracking functions of the Safety Process Monitor. The functions are called with individual IDs.

Safety Process Monitor Initialization (ClassB_SPMon_Init)

According to the defined SafetyPeriod (ClassB_config.h) the configured timer is initialized and a callback function is registered to be called if the timer reaches the interval SafetyPeriod. The function returns with an inProgress result if the initialization was successful.

Safety Process Monitor Handler (ClassB_SPMon_TimerHandler)

The handler function is called by the configured cyclic timer handler. E.g. the SysTick interrupt handler. This function also calls the callback function. If the Safety Monitor Callback is called the Monitor data are reset for the next safety period.

Safety Process Monitor (ClassB_SPMon_BIST)

The function must be called to validate the safety process monitor result. The result states the information if all test routines have been called in the safety period and if the long term tests (like RAM test) are fully performed at minimum one time completely.

CallBack function

The user application has to provide a callback function to evaluate the Safety Period Monitor. The function is called with the SPMon result as a parameter.

5.3.4 Supply Functions

The Supply functions provide general features useful or necessary for running and evaluating the tests and their results.

Table 5.7 Supply Functions

Returnvalue	Function	Parameter	
Fail State Action			
void	FailSafe	void	
Application			
int	CheckSPMonCallBack	testResult_t	
Interrupt in the application			
void	SysTick_Handler	void	

5.3.4.1 Fail Safe State Example (FailSafe)

A simple routine called if any error is detected by the ClassB library. The further processing is not provided in this function.

5.3.4.2 SPMon Call Back Example (CheckSPMonCallBack)

Call back function registered for the SPMon functions and called by the interrupt matching the Safety Period. The function is called with the result of the safety process monitor.

5.3.4.3 SysTick Interrupt Handler Example (SysTick_Handler)

The SysTick handler is a CMSIS defined standard interrupt handler called from the SysTick Timer interrupt. This function provides some simple counter with flags on multiples. The Timer is basically initialized to provide milliseconds timing.

5.4 Configuration API Definitions Overview

To configure the Atmel ClassB library it is not necessary to edit the code. All modules are individual configurable in a central file: ClassB_config.h.

The configuration of the ClassB library provides a set of easy to use possibilities for enabling or disabling features. Also some detailed configurations are provided in this central place to specify particular behavior on certain features.

This chapter provides an overview to these configurations.

Figure 5.2 Configuration API defines

Parameter	Default Value	Description
CPU Parameter		
CPU_SELECTION	0	CPU SAMD20, SAMD21 or SAMC20,SAMC21 selection (0=SAMD20, 1=SAMD21, 2=SAMC20, 3=SAMC21)
SAM_Y2x_DERIVATE	18	Derivative (memory size)
Clock Sources		
ClassB_CLOCK_XOSC_ENABLE	0	External high speed clock enabled
ClassB_CLOCK_XOSC_DISABLE_AGAIN	1	Disable auto gain settings during setup
ClassB_CLOCK_XOSC_SPEED	4	1= 2MHz, 2= 4MHz, 3= 8MHz, 4= 16MHz, 5= 30MHz
ClassB_CLOCK_XOSC32K_ENABLE	1	External low speed clock enabled
SysTickTickSet	1000	1ms ticks for 1 MHz frequency
Flash Test Configuration		
CRC_FLASHSIZE	0	0=used Flash is checked
ClassB_Ref_ChkSum	32 bit number	CRC32Val
CRC_block_size	1024	defines the maximum portion of a flash test block in BIST
CRC_USE_HARDWARE	1	Use the CRC engine
CRC_CHECK_HARDWARE	1	Enable a POST check of the CRC engine
CRC Configuration		
ClassB_FLASH_CRC_REFERENCE	ClassB_Ref_ChkSum	CRC32Val
	CRC_FLASHSIZE	MemLenght
	(uint32_t)__STEXT	StartAddress from linker
	(uint32_t)__ETEXT	End Address from linker
	FLASH_CRC_Valid	Status
POST test Configuration		
ClassB_RAM_buffersize	40	Buffer size for RAM backup
POST test configuration		
ClassB_TEST_POST_WDT	1	WatchDog timer test (1=Enabled, 0=Disabled)
ClassB_TEST_POST_WDT_PERIOD	11	8192 clock cycles for timeout and reset
ClassB_TEST_POST_IRQ	1	IRQ Logic test (1=Enabled, 0=Disabled)
ClassB_TEST_POST_IRQ_RAM	1	Use RAM based Vector Table (1=Enabled, 0=Disabled)
ClassB_TEST_POST_CLOCK	1	Clock test (1=Enabled, 0=Disabled)
ClassB_TEST_POST_CPUREG	1	Register test (1=Enabled, 0=Disabled)
ClassB_TEST_POST_PC	1	Program Counter test (1=Enabled, 0=Disabled d)

ClassB_TEST_POST_ADDR_DECODER	1	RAM address decoder test (1=Enabled, 0=Disabled)
Configuration of the RAM tests		
ClassB_TEST_POST_RAM	1	RAM test (1=Enabled, 0=Disabled)
ClassB_TEST_POST_STACK	1	RAM test on Stack area (1=Enabled, 0=Disabled)
configuration of the FLASH tests		
ClassB_TEST_POST_FLASH	1	FLASH test (1=Enabled, 0=Disabled)
BIST test configuration		
Configuration of the flash content tests		
ClassB_TEST_BIST_FLASH	1	BIST FLASH test (1=Enabled, 0=Disabled)
Configuration of the RAM tests		
ClassB_TEST_BIST_RAM	1	BIST RAM test (1=Enabled, 0=Disabled)
ClassB_TEST_BIST_CONSECUTE_RAM	0	RAM test continuously in BIST (1=Enabled, 0=Disabled)
ClassB_TEST_BIST_BEHINDSTACK_RAM	0	RAM test behind stack in BIST (1=Enabled, 0=Disabled)
Configuration of the core tests		
ClassB_TEST_BIST_CPUREG	1	BIST Register test (1=Enabled, 0=Disabled)
ClassB_TEST_BIST_PC	1	BIST Program Counter test (1=Enabled, 0=Disabled)
Configuration of the clock tests		
ClassB_TEST_BIST_CLOCK	1	BIST Clock test (1=Enabled, 0=Disabled)
ClassB_USE_SYSTICK	1	Specifies timer test with systick (1=Enabled, 0=Disabled)
ClassB_USE_TIMERx	0	Specifies timer test with TIMERx (TCx) not implemented!
ClassB_BIST_IGNORE_CLOCKCYCLES	4	cycles the clock test is ignored in BIST
Configure Safety Process Monitor		
ClassB_TEST_SPMON	1	Enables the sequence monitoring feature (1=Enabled, 0=Disabled)
ClassB_SPMon_SafetyPeriod	5500	[ms] Safety Period in milliseconds
ClassB_SPMon_Timer	1	Specify the timer (1=SysTick, 2=RTC)
Configure flow control		
ClassB_FlowControl	1	Enables the flow control (1=Enabled, 0=Disabled)
ClassB_FLOW_USE_CALLBACK	0	Use the callback function (1=Enabled, 0=Disabled)
ClassB_FLOW_NOTIFY	0	Immediate notification (1=Enabled, 0=Disabled)
ClassB_FLOW_TIMERCHECK	1	Internal timer comparison (1=Enabled, 0=Disabled)
ClassB_FlowTest_Period	100000	Period to check the flow control
user defined functions and replacements		
RTC_Init()	ClassB_RTC_Init()	Use the build in function
Critical Section Mcro		
EnterCriticalSection()	__disable_irq();\n__DMB()	enable atomic instructions
ExitCriticalSection()	__DMB(); \n__enable_irq()	disable atomic instructions

port IO defines for LED access on Xplained SAM-D20 board		
CLASSB_DBG_GPIO		GPIO for debugging purpose
__DBG_PIN_OUTPUT ()	PORT->Group[0].DIRSET.reg = PORT_PB14	Setup Pin for LED
__DBG_GET_IO0	(PORT->Group[0].OUT.reg & PORT_PB14)	mask LED pin value
__DBG_SET_IO0 ()	PORT->Group[0].OUTCLR.reg = PORT_PB14	set LED
__DBG_RESET_IO0 ()	PORT->Group[0].OUTSET.reg = PORT_PB14	reset LED
__DBG_SET_IO1 ()		Not used
__DBG_RESET_IO1 ()		Not used
__DBG_WAIT(x)		Wait for x nop instructions
port IO defines for LED access on Xplained SAM-D21 board		
__DBG_PIN_OUTPUT ()	PORT->Group[1].DIRSET.reg = PORT_PB30	Setup Pin for LED
__DBG_GET_IO0	(PORT->Group[1].OUT.reg & PORT_PB30)	mask LED pin value
__DBG_SET_IO0 ()	PORT->Group[1].OUTCLR.reg = PORT_PB30	set LED
__DBG_RESET_IO0 ()	PORT->Group[1].OUTSET.reg = PORT_PB30	reset LED
port IO defines for LED access on Xplained SAM-C20 board		
__DBG_PIN_OUTPUT()	PORT->Group[0].DIRSET.reg = PORT_PA15	Setup Pin for LED
__DBG_GET_IO0()	!(PORT->Group[0].OUT.reg & PORT_PA15)	mask LED pin value
__DBG_SET_IO0()	PORT->Group[0].OUT.reg = PORT_PA15	reset LED
__DBG_RESET_IO0()	PORT->Group[0].OUT.reg &= ~PORT_PA15	set LED
__DBG_SET_IO1()		macro defined empty
__DBG_RESET_IO1()		macro defined empty
__DBG_WAIT(x)	{uint32_t i; for (i=0; i<x; i++) __NOP();}	
port IO defines for LED access on Xplained SAM-C21 board		
__DBG_PIN_OUTPUT ()	PORT->Group[1].DIRSET.reg = PORT_PB05	Setup Pin for LED
__DBG_GET_IO0	!(PORT->Group[1].OUT.reg & PORT_PB05)	mask LED pin value
__DBG_SET_IO0 ()	PORT->Group[1].OUTCLR.reg = PORT_PB05	set LED
__DBG_RESET_IO0 ()	PORT->Group[1].OUTSET.reg = PORT_PB05	reset LED

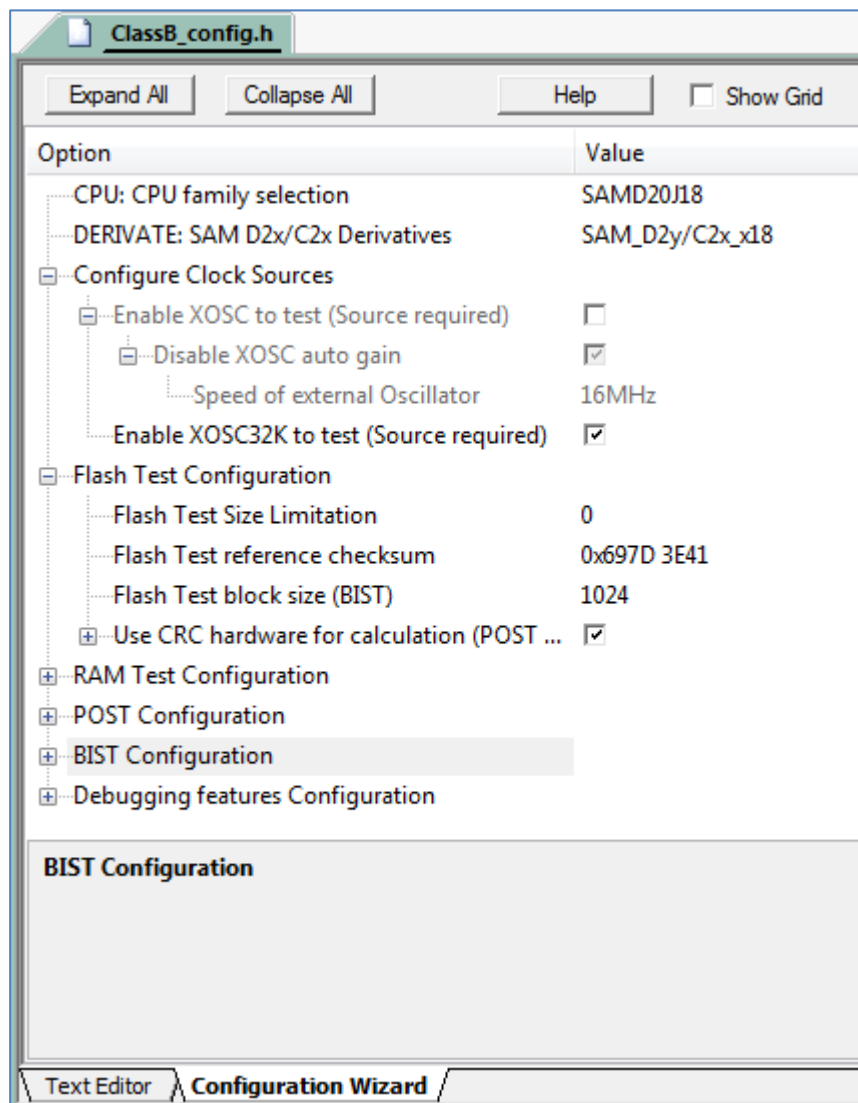


Figure 5.3 ARM-Keil MDK Configuration Wizard Example

5.4.1 CPU Parameters

The System Parameters are specifications of the controller. RAM and Flash allocation and size are specified and the CPU clock of the application is defined. Additionally, the port IO is defined with very low level functions.

5.4.2 Clock Source Configuration

The clock source configuration enables to select between the internal and external clock sources. This directly impacts the clock tests in POST and BIST.

5.4.3 Flash Test Configuration

The Flash test configuration enables to select between hardware and software CRC calculation. If the hardware is selected, a CRC engine check can be enabled. The Flash size can be limited to a section less than the amount used by the program and a referencing CRC value can be configured.

5.4.4 RAM Test Configuration

The RAM test configuration enables to specify the buffer in the stack used for RAM backup during the BIST RAM test.

5.4.5 POST Test Configuration

In this chapter the available POST test functions can be enabled or disabled.

ClassB_Disabled (0) means disabled and ClassB_Enabled (1) means enabled. The features are:

- 1) Watchdog Test and a timeout value
- 2) Interrupt Logic Test
 - a. Enable or disable RAM based Vector Table test
- 3) Clock Test
- 4) CPU Register Test
- 5) Program Counter Test
- 6) Address Decoder Test
- 7) RAM Tests
 - a. Enable or disable the test of the stack section
- 8) Flash Tests
 - a. CRC reference
 - b. Hardware/software method

5.4.6 BIST Test Configuration

In this chapter the available BIST test functions can be enabled or disabled.

ClassB_Disabled (0) means disabled and ClassB_Enabled (1) means enabled. The features are:

- 1) Flash Tests
 - a. CRC reference
 - b. Hardware/software method
 - c. Slice size
- 2) RAM Test
 - a. Run continuously or stop after finishing for the safety period
 - b. Test RAM area behind the stack
- 3) Core Tests
- 4) Program Counter Tests
- 5) Clock Tests
 - a. Detailed configuration available
- 6) Safety Process Monitor
 - a. SysTicks for the Safety Period
 - b. Timer selection
- 7) Flow Control
 - a. Detailed configuration available
- 8) Debugging features
 - a. Pin IO routines

5.5 API Types Overview

The types used in the ClassB library provide a set of easy to use structs and type definitions. This chapter provides an overview to these types.

Figure 5.4 API Types Overview

Name	Type	Description
ClassB_Callback_t	typedef	generic call back function pointer
ClassB_ClockTest_t	typedef	clock test

ClassB_CPUtest_t	typedef	CPU register test
ClassB_Data_t	typedef	Program Counter test
ClassB_Exec_t	typedef	ptr to function returning enum ClassB_ReturnValues
ClassB_FlashSign_t	typedef	Flash test data
ClassB_IRQTest_Data_t	typedef struct	type structure for interrupt testing
ClassB_IRQTest_Matrix_t	typedef struct	Structure for interrupt test configuration
ClassB_PCTest_ReturnValues	typedef	type declaration fo the return values
ClassB_Proc_ID_t	typedef	error messages and codes
ClassB_RAMTest_Data_t	typedef struct	Process parameter for the RAM test
ClassB_RAMTest_Mode_t	typedef	RAM test modes
Classb_Stub_DataType_t	typedef	Report/stub global data type
Classb_Stub_Result_t	typedef	Report/stub global data type
ClassB_testResult_t	typedef	generic type for ClassB functional results
ClassB_testState_t	typedef	
critical_double	typedef	typedef for a critical floating point variable with 64 bits
critical_float	typedef	typedef for a critical floating point variable with 32 bits
critical_int64_t	typedef	typedef for a critical signed int variable with 64 bits
critical_int32_t	typedef	typedef for a critical signed int variable with 32 bits
critical_int16_t	typedef	typedef for a critical signed int variable with 16 bits
critical_int8_t	typedef	typedef for a critical signed int variable with 8 bits
critical_uint64_t	typedef struct	typedef for a critical unsigned int variable with 64 bits
critical_uint32_t	typedef struct	typedef for a critical unsigned int variable with 32 bits
critical_uint16_t	typedef struct	typedef for a critical unsigned int variable with 16 bits
critical_uint8_t	typedef	typedef for a critical unsigned int variable with 8 bits

5.5.1 ClassB_Callback_t

The Call back types are used to specify the function passed to SPMon and Flow Control.

5.5.2 ClassB_testResult_t

This is the generic return type of all ClassB functions.

5.5.3 ClassB_Proc_ID_t

Individual process IDs are specified for the BIST and some POST processes.

5.5.4 ClassB_FlashSign_t

The Flash test requires some reference data to verify and evaluate. The data are stored in as struct of this type

5.5.5 Critical Data

The critical data types are available with macros to push, pop and evaluate. The library provides a set of predefined variable types from 64 bit over floating types to integers of 32 to 8 bit in signed and unsigned versions.

5.6 SP Monitoring

5.6.1 Definition of the Safety Period

The safety period is defined by the maximum duration which is required by the safety concept to detect any internal error in the system. According to IEC60730 this is the “Software Fault/Error Detection Time”. As a consequence all required tests have to be executed in the safety period.

The SP Monitoring provides a diagnostic test (“ClassB_SPMon”) which could help to supervise if the requirement mentioned above is fulfilled.

The SP Monitoring features is NOT designed to provide a fast reaction on any applicational fault states. The adequate reaction on any safety critical occurrences driven by the application and the appropriate reaction is explicitly not covered by the SP Monitoring. Also the reaction of the system must be implemented by the user according the safety concept of the system. The function “safeState” can give an impression to handle the safe state of the system but is only sufficient for internal errors within the Demo applications.

5.6.2 SpMon Test Implementation

Despite to the other BIST tests the SP-Monitoring-Test runs in the background and is triggered through a timer. The timer value is configured with the safety period value to produce a trigger output when it expires and calls the Callback routine in which the SP-Monitoring-Test is executed. This ensures to execute the SP-Monitoring-Test function each time when the safety period is over.

In the SP-Monitoring-Test the state and the result of all long time configured tests are checked if they have been completely and successfully executed during the last interval. A result is produced accordingly.

5.6.3 Example

This may be seen as a proposal how the period for the SP Monitor can be determined:

- Specify the call time of all enabled and required long-time-tests.
Long-time-tests are RAM-Test, Flash-Test, Clock-Test. Even new defined applicational tests may be added. All split or partitioned tests can be added to the list.
- Estimate a summary of the longest lasting test to proceed completely
by multiplying the calling period with the number required to complete: e.g. Test of 0x10000 bytes of Flash: portion is 0x400 bytes, call period is 100ms: $0x10000/400=40$; $40*100ms=4000ms$.
In this case the runtime (<2ms) is much smaller than the call period so it can be disregarded here.
- Specify the safety period according the longest period of completion in your application.
#define ClassB_SPMon_SafetyPeriod XXXX (ms) e.g. 4500
- Enable the test in the ClassB_config.h
#define ClassB_TEST_SPMON” 1 1=enable
- It is necessary to select which timer is used for the monitoring
#define ClassB_SPMon_Timer X e.g. 1 for the default SysTick

5.6.4 Timing considerations

Most BIST tests finish with their diagnosis in one procedure but some of them are split into several parts. The reason for that are timing constraints since these tests need too much time to proceed in one process.

The tests are:

ClassB_RAM_Test_BIST()

ClassB_FLASH_Test_BIST()

A special function is also the

ClassB_ClkTest running with a RTC defined period of 1 second

Besides the calling frequency for these tests a factor (S) can be configured which determines the number of portions to be executed for one complete test.

The factor configuration for ClassB_RAM_Test_BIST(): ClassB_RAM_buffersize

The factor configuration for ClassB_FLASH_Test_BIST(): CRC_block_size

The Clock test can not be configured by a number but by the definition of a RTC configuration procedure. It is up to the user to specify any different functions which may result in a shorter period than 1 second of the RTC.

The timing calculation shall be demonstrated by means of 2 scenarios:

1. All configured tests proceed in one call.

$$D(SP) > P(CBIST) = \sum T(test) + T(RBIST) + T(FBIST)$$

The duration of the safety period has to be significantly longer than the sum of the execution times of all enabled tests. For the calculation all timings from "Table 6.2 Time Consumption of the Modules" must be considered with special regard of the conditions in the table.

2. RAM and FLASH BIST are enabled by periodic calls

$$D(SP) > P(CBIST) > \sum T(test) + (S(RBIST) * P(RBIST)) + (S(FBIST) * P(FBIST))$$

$D(SP)$:	Duration of safety period
$T(test)$:	Execution time of all simple diagnostic test
$S(RBIST)$:	Split factor for the RAM BIST
$T(RBIST)$:	Execution time of all RAM BIST portion
$P(RBIST)$:	Execution period of a RAM BIST portion
$S(FBIST)$:	Split factor for the Flash BIST
$T(FBIST)$:	Execution time of all Flash BIST portion
$P(FBIST)$:	Execution period of a Flash BIST portion
$P(CBIST)$:	Execution period of a Clock BIST test = 1Second

Note that the duration $D(SP)$ is expected to be significantly longer than at least calculated with the equation above since the application itself needs time to proceed.

Using the parameters to configure the test will have dramatically effects on the number of calls. This will change the calculation for the second scenario. E.g. the RAM test shall test a RAM portion of 0x4000 bytes. With the default configuration (0x28 bytes per call) this will be completed with 200 calls. Double the RAM buffer will decrease this number to the half of 100 calls. By calling the function every 10ms, as in the example, this will take 200calls x 10ms = 2000ms or 1000ms with changed configuration. It has to be mentioned, that in this case the RAM test will run more than double as long: about 12000 CPU cycles instead of about 5100. In this time no other CPU action is possible. Also stack usage will increase.

It shall be stated here again that the value for $D(SP)$ is determined through the requirements of the safety application and not by the implementation.

6 Environment, Usage and Limits

The workspace provides two example applications to demonstrate and test the functionality of the ClassB Library. For each controller family a single example is implemented. It is loaded into the Atmel "SAM D20 Xplained Pro" board the Keil-MDK environment.

6.1 Application Environment

The usage of the library is strongly restricted to the following condition and settings:

- 1) Use of the library with the appropriate derivatives
 - a. SAM-C20 (all devices)
 - b. SAM-C21 (all devices)
 - c. SAM-D20 (all devices)
 - d. SAM-D21 (all devices)
- 2) The Toolchain is restricted to the tested versions of
 - a. ARM/Keil MDK v.5.13
 - b. IAR Embedded Workbench (7.40.3)
- 3) The Optimization is set to its minimum (-O0)
- 4) Linker uses the provided linker description file according the desired Flash memory setup
- 5) Application use the provided startup files
 - a. Watchdog test runs first
 - b. Post memory test are used before memory is initialized

6.2 Example Applications

The application is compiled and linked especially for the SAMD20J18 derivative.
The application

Example Application are available:

Example Keil_SAMC20, Keil_SAMC21, Keil_SAMD21, Keil_SAMD20 for ARM/Keil Toolchain 5.13 and later

Example IAR_SAMC20, IAR_SAMC21, IAR_SAMD21, IAR_SAMD20 for IAR Toolchain 7.40 and later

Testing the application will enable the user to modify the ClassB configuration according to the individual needs.

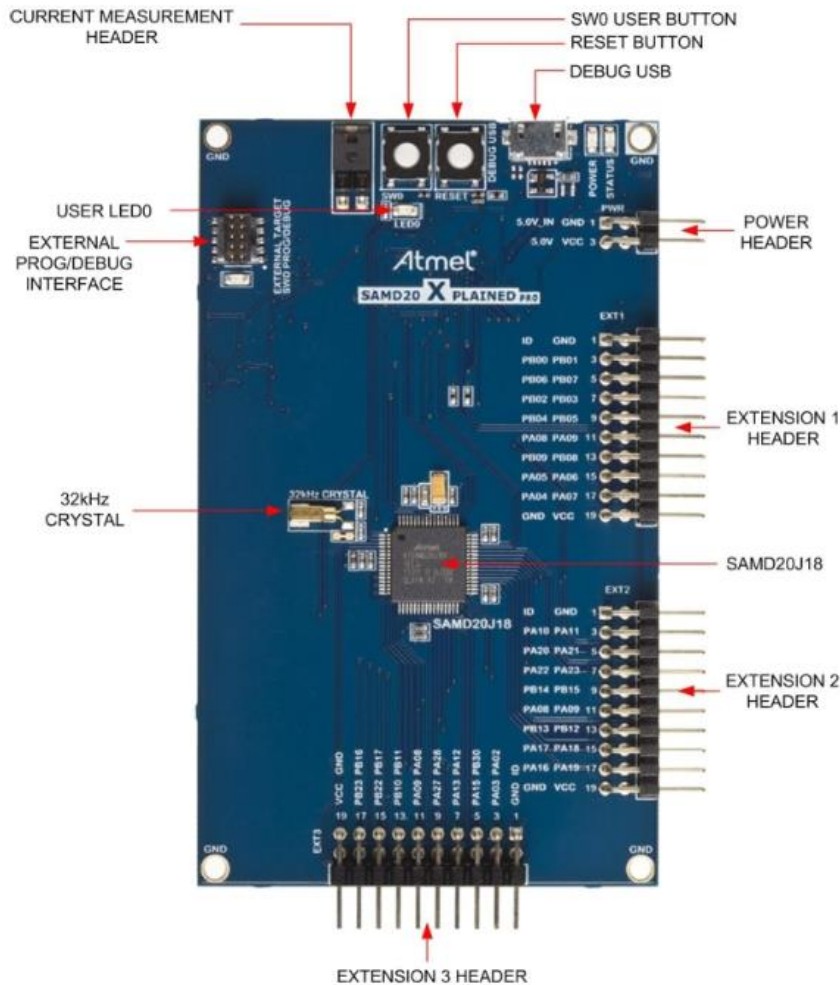


Figure 6.1 SAMD20 Xplained evaluation board

6.3 Application Keil_SAMD20_ClassB

The application is compiled and linked especially for the SAMD20J18 derivative.

The application runs a fully featured POST test and continues with the BIST test in main. All features except the flow control is applied.

The POST executes tests on: the Watchdog with reset, IRQ, Clock module, CPU register and PC, Address decoder, RAM cell test and flash content test. The flash test is configured to run with hardware calculated CRC values.

The main function initializes the safety period monitoring, RAM test, Flash test and the Clock test.

With an interrupt driven from the SysTick Timer a scheduling is provided. In a higher frequency the function RAMtest_BIST and Flashtest_BIST are called to check the memory. CPU test and PC test called less often. Additional a LED is toggled to show heard beats.

The SysTick interrupt also calls the handler functions for safety period monitoring and clock test. The SPMon_Callback function is automatically called if the SPMonitor detects the period over. In this function the long term status information of the long term test are checked. This is Flash test, RAM test and Clock test. In the super loop an LED is toggled if none of the tests failed.

Linker settings: special linker settings are required to generate a layout as required by the ClassB library. Therefor a linker command file is delivered with the library.

6.4 Memory Usage

The test library's memory usage depends on the configuration. The chapter provides an overview to the modules without any optimization switched on and without special conditions (no optimization).

Table 6.1 Memory Usage of the Modules

Module	ROM-Size [bytes]	RAM-Size [bytes]
FlashCRC	76	20
ClassB_CLKTest	1048	0
ClassB_CPUPTest	772	0
asmCPUPTest_BIST	956	0
asmCPUPTest_POST	952	0
ClassB_FailSafe	32	0
ClassB_FLASHTest	680	20
ClassB_SPMon	480	0
ClassB_IRQTest_POST	896	32
ClassB_POST	220	0
ClassB_RAMTest	3636	0
PCTest_Codes	360	0
HardFault_Handler	12	0
RO-data	656	-
bss	98	-

6.5 Runtime Requirements

The modules of the Atmel Classb library are designed to run as separate entities. To fit the library into a given design the runtime consumption of the periodical tests are required.

Table 6.2 Time Consumption of the Modules

Test Function	Cycles (decimal)		Description / conditions
	SAMD20J18	SAMD21J18	
ClassB_WDCTest_POST	350	Na	+ watchdog timeout
ClassB_hFlashcrc32 POST (CRC only)	Na	Na	Length = 10924d
ClassB_sFlashcrc32 POST (CRC only)	Na	Na	Length = 10924d
ClassB_PCTest_POST()	27930	Na	
ClassB_ADDRDECTest_POST()	28950	Na	
ClassB_RAMTest_POST()	416100	Na	
ClassB_FLASHTest_POST()	40040	Na	
ClassB_CLKTest_POST()	27760	Na	
ClassB_CPUPTest_POST()	27610	Na	
ClassB_IRQTest_POST	33000	Na	
ClassB_RAMTest_BIST()	5100	Na	total length 0x400, portion 0x40
ClassB_CPUPTest_BIST()	756	Na	
ClassB_PCTest_BIST()	141	Na	
ClassB_FLASHTest_BIST	208	Na	Length = 10924d, Portion = 1024d (Hardware CRC used for calculation)

6.6 Safety Period

The safety period specifies the time which is required by the safety concept to detect and react on any error in the system.

Depending on the safety period the time to test all internal functions is specified. The selection of measures to test the controller is specified by the individual safety concept which is covering the FMEA. On one hand to select or deselect tests the safety design must be defined. On the other hand the selected measures must be configured to fit to the safety period.

Example scenario1:

A safety period of 5.5 seconds is predefined. The safety design specifies a full flash test at moving checksum and a full RAM test. Additional the CPU is tested and the watchdog is enabled.

To specify the overall time which is required to finish all tests some details are essential:

The RAM and Flash test is parted into smaller portions to avoid a long time blocking of the application.

Assuming that the application periodically calls the test increasing the portions will increase the time consumption of the single call with the effect that the number of periods required for a full test will decrease. Since there is a CRC engine for the Flash test available the effect will be less significant than calculation with software is configured.

6.7 Limitations

ClassB_CPUtest_BIST: CPU Register Test (BIST):

Limitation: The function must not be called from an Interrupt service routine because PSP is not accessible then. The cortex controller will change to MSP stack pointer by hardware when interrupt mode is active.

Reset Cause

The reset cause is a register of the chip stating the reason the last reset was caused from. This register is used for the watchdog test to determine the WDT functionality and the reset mechanism. After a watchdog timer reset it is only stating the WDT cause flag. So a previous reset is not detected. The controller provides no additional memory that keeps content after any reset condition. So it is impossible to get any information except the test passes with a WDT reset.

6.8 Interaction with other tools like code generators

To minimize the effects on automated build code like interrupt service routines the library is designed to run in encapsulated units with less than possible interaction. E.g. the IRQ test can run without any code modifications in the interrupt service routines if the option "ClassB_TEST_POST_IRQ_RAM" is selected.

6.9 Trouble Shooting

The table below lists some common problems encountered when using the Application.

Table 6.3 **Trouble Shooting**

Symptom	Cause/Workaround
Flash Tests fails	Enable a breakpoint in the file flash_test.c line 389. Restart the application and run to this location. Check the next lines in the code with single steps. If the verification went wrong the CRC of the reference (file ClassB_config.h line 198) must be adjusted. Use the CRCValue from the variable TestCRC struct.
WatchDog Test is not enabled	The watchdog test causes a reset after startup. If the program should be debugged, the watch dog tests reset may cause a disconnection from the debugger or the debugger reset may cause the Watchdog test function to stuck in a failed loop. Disabling the Watchdog reset will prevent from this trap during development.
SPMon is not responding "passed" immediately	The SPMon requires some runtime to evaluate the test features monitored as passed. This is because some processes need some time to process completely. To avoid any "failed" states at the startup a process counter can be implemented to delay the evaluation effects once.
Interrupt Logic Test stuck at startup_xxx.s	The interrupt logic test requires 3 interrupt service routines calling the test callback functions. For SAMD devices TC3, TC4 and TC5 are used. For SAMC2x devices use TC2 instead of TC5. The Interrupt service routines are located in the files interrupt.c in the sources folder of the examples. If they are not implemented the default vectors from startup.s are used.
Clock Test (POST) sticks or fails	The clock test requires, if configured for the test of external clock sources, external oscillators. The oscillator must be functional. The clock test will wait on the external source to be ready if the tests are enabled. If a test is intended and the clock source fails the test will state a valid failure because the hardware does not work correctly.

www.hitex.com

Published by Hitex Development Tools GmbH