Energy Micro EFM32G

IEC60335 Class B

Firmware Library

User Documentation

Revision History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Issue** | **Status** | **Date** | **Author** | **Changes** |
| 0.1 | Draft | 12-01-2011 | S. Grohmann | Initial Revision |
| 0.2 | External Review | 11-02-2011 | S. A. Olsen | Core Documentation |
| 0.3 | Internal Review | 12-5-2011 | S. Giesbrecht | Rework internal review remarks |
| 1.0 | Released | 12.07.2011 | S. Grohmann | released |

Content

[1. The EFM32G Cortex-M3 Microcontroller family 6](#_Toc299609341)

[1.1. The EFM32G890F Cortex-M3 microcontrollers 6](#_Toc299609342)

[1.1.1. The ARM Cortex-M3 Core 6](#_Toc299609343)

[1.1.2. The EnergyMicro ARM Cortex-M3 Product Options 7](#_Toc299609344)

[1.1.3. EFM32G890F 7](#_Toc299609345)

[1.2. Detailed Library Tested Peripherals Description 9](#_Toc299609346)

[1.2.1. CPU, Cortex-M3 9](#_Toc299609347)

[1.2.2. Interrupt Operation 12](#_Toc299609348)

[1.2.3. Clock Management Unit 13](#_Toc299609349)

[1.2.4. Memory 15](#_Toc299609350)

[2. The Class B Self-Test Library 21](#_Toc299609351)

[2.1. Library Architecture 21](#_Toc299609352)

[2.1.1. Target and System Requirements 21](#_Toc299609353)

[2.1.2. System Overview 21](#_Toc299609354)

[2.1.3. System Design and Design Method 21](#_Toc299609355)

[2.1.4. Logical Presentation, Decomposition 22](#_Toc299609356)

[2.1.5. Requirements and Traceability 23](#_Toc299609357)

[2.1.6. Module Requirements 24](#_Toc299609358)

[3. Library Description and Usage 26](#_Toc299609359)

[3.1. Framework application 26](#_Toc299609360)

[3.2. Library Memory Requirements 26](#_Toc299609361)

[3.3. The Library Functions Specification 27](#_Toc299609362)

[3.3.1. POST and BIST 27](#_Toc299609363)

[3.3.2. POST Tests 28](#_Toc299609364)

[3.3.3. CPU Register Test (1.1) 29](#_Toc299609365)

[3.3.4. Program Counter Test (1.2) 43](#_Toc299609366)

[3.3.5. Interrupt Handling and Execution Test (2) 46](#_Toc299609367)

[3.3.6. Clock System Test (3) 51](#_Toc299609368)

[3.3.7. Watchdog Timer Test (3.1) 54](#_Toc299609369)

[3.3.8. Memory Tests and Safety Functions (4) 57](#_Toc299609370)

[3.3.9. Test Execution Time 66](#_Toc299609371)

[3.4. Library Usage Description 67](#_Toc299609372)

[3.4.1. CPU Register Test Usage 67](#_Toc299609373)

[3.4.2. Program Counter Test Usage 67](#_Toc299609374)

[3.4.3. Interrupt Operation Test Usage 67](#_Toc299609375)

[3.4.4. Clock System Test Usage 67](#_Toc299609376)

[3.4.5. Memory Test and Safety Functions Usage 68](#_Toc299609377)

[3.4.6. FLASH Test Usage 69](#_Toc299609378)

[3.4.7. RAM Test Usage 70](#_Toc299609379)

[3.4.8. Watchdog Timer Test Usage 70](#_Toc299609380)

[3.5. The Development Environment 71](#_Toc299609381)

[3.6. List of tools 72](#_Toc299609382)

[4. Sources 73](#_Toc299609383)

[5. Bibliography 74](#_Toc299609384)

[6. List of Images 74](#_Toc299609385)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

NOTE!

All marked is still in work

All Comments must be processed

All tbd must be processed

Abbreviations

|  |  |
| --- | --- |
| ADC | Analog-Digital Converter |
| AHB | Advanced High Speed Bus |
| AMBA | Advanced Microcontroller Bus Architecture |
| APB | Advanced Peripheral Bus |
| APSR | Application Program Status Register |
| ARM | ARM Ltd, Cambridge, UK |
| AUXHFRCO | auxiliary 14 MHz RC oscillator |
| BGA | Ball Grid Array |
| BIST | Build-In System Test |
| CPU | Central Processing Unit |
| DAC | Digital-Analog Converter |
| DMA | Direct Memory Access |
| DWT | Data Watchpoint and Trace |
| EM | Energy Micros AS., Oslo, Norway |
| EPSR | Execution Program Status Register |
| ETM | Embedded Trace Macro Cell |
| HFCORECLK | Frequency Core Clocks |
| HFPERCLK | Peripheral Clocks |
| HFRCO | High Frequency RC Oscillator |
| HFXO | High Frequency Crystal Oscillator 4-32 MHz |
| I²C | Inter IC Communication |
| IPSR | Interrupt Program Status Register |
| JTAG | Joint Test Action Group (defined interface for debugging) |
| LCD | Liquid Cristal Display |
| LFRCO | Low Frequency RC Oscillator 32.768 kHz |
| LFXO | Low Frequency Crystal Oscillator 32.768 kHz |
| MISRA | Motor Industry Software Reliability Association |
| MPU | Memory Processing Unit |
| NVIC | Nested Vectored Interrupt controller |
| POST | Pre Operation System Test |
| PSR | Program Status Register |
| PWM | Pulse-Width Modulation |
| RAM | Random Access Memory |
| ROM | Read-Only Memory |
| RTC | Real-Time Clock |
| SP | Stack Pointer |
| TCM | Tightly Coupled Memory |
| UART | Universal Asynchronous Receive and Transmit |
|  |  |
|  |  |
|  |  |

# The EFM32G Cortex-M3 Microcontroller family

This chapter gives a general description on the EFM32G families for which the IEC60335 Class B self-test libraries are written.

* 1. The EFM32G890F Cortex-M3 microcontrollers

The EFM32G890F family is an ARM Cortex™-M3 (1) based microcontroller for embedded applications requiring a high level of integration and low power dissipation. The ARM Cortex-M3 is a next generation core that offers system enhancements such as modernized debugging features and a higher level of support block integration.

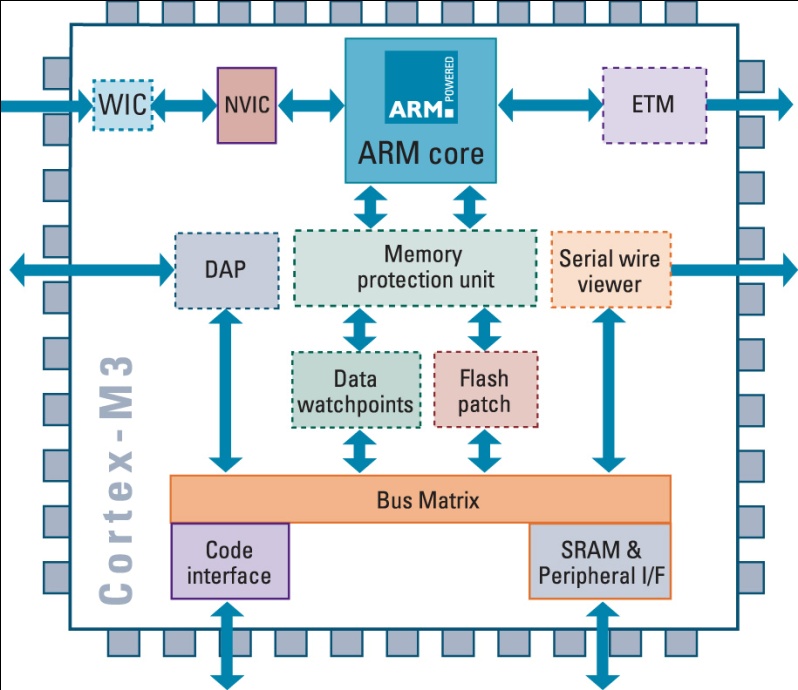
The EFM32G890F family operates at up to a 32 MHz CPU. The ARM Cortex™-M3 CPU incorporates a 3-stage pipeline and uses Harvard architecture with separate local instruction and data buses as well as a third bus for peripherals. The ARM Cortex™-M3 CPU also includes an internal prefetch unit that supports speculative branches.

### The ARM Cortex-M3 Core

The ARM Cortex™-M3 32-bit processor has been specifically developed to provide a high-performance, low-cost platform for a broad range of applications including microcontrollers, automotive body systems, industrial control systems and wireless networking. The Cortex™-M3 processor provides outstanding computational performance and exceptional system response to interrupts while meeting low cost requirements through small core footprint, industry leading code density enabling smaller memories, reduced pin count and low power consumption.

The central core of the Cortex™-M3 processor, based on a 3-stage pipeline Harvard bus architecture, incorporates advanced features including single cycle multiply and hardware divide to deliver an outstanding efficiency of 1.25 DMIPS/MHz. The Cortex™-M3 processor also implements the   
Thumb®-2 instruction set architecture, which when combined with features such as unaligned data storage and atomic bit manipulation delivers 32-bit performance at a cost equivalent to modern 8- and 16-bit devices.

The EFM32G890F family uses the r2p0 version of the Cortex™-M3 core.

Image Cortex™-M3 core architecture

### The EnergyMicro ARM Cortex-M3 Product Options

The EFM32G is available in various configurations regarding memory sizes, packages and peripherals.

Table 1 shows the variety of available products.

### EFM32G890F

The EFM32 MCUs are the world’s most energy friendly microcontrollers. With a unique combination of the powerful 32-bit ARM Cortex™-M3, innovative low energy techniques, short wake-up time from energy saving modes, and a wide selection of peripherals, the EFM32G microcontroller is well suited for any battery operated application as well as other systems requiring high performance and low-energy consumption. This section gives a short introduction to each of the modules in general terms and also and shows a summary of the configuration for the EFM32G890 devices. For a complete feature set and in-depth information on the modules, the reader is referred to the EFM32G Reference Manual.(2)

The peripheral complement of the EFM32G family includes up to

128 Kbytes of internal flash memory, 16 Kbytes of data memory, Energy management unit, Watch dog timer, 8-channel general purpose DMA controller, 3 USARTs, UART, 2 Low Energy UARTs, I2C-bus interface, 8-channel 12-bit ADC, 2 channel 12-bit DAC, 3 general purpose 16-bit timers with PWM function, Pulse Counter with quadrature decoder interface, Low Energy general purpose timer, 8-channel analogue comparator, ultra-low power Real-Time Clock (RTC), external memory interface, LCD interface and up to 90 general purpose I/O pins.

The EFM32G devices are available in a BGA112 package.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EFM32G Part # | Flash | RAM | GPIO(pins) | LCD | USART+UART | LEUART | I²C | Timer (PWM) | LETIMER | RTC | PCNT | Watchdog | ADC(pins) | DAC(pins) | ACMP(pins) | AES | EBI |
| 840F128 | 128 | 16 | 56 | 4x24 | 3 | 2 | 1 | 3 (9) | 1 | 1 | 3 | 1 | 1 (8) | 2 (2) | 2 (8) | Y | - |
| 880F32 | 32 | 8 | 85 | 4x40 | 3+1 | 2 | 1 | 3 (9) | 1 | 1 | 3 | 1 | 1 (8) | 2 (2) | 2 (16) | Y | Y1 |
| 880F64 | 64 | 16 | 85 | 4x40 | 3+1 | 2 | 1 | 3 (9) | 1 | 1 | 3 | 1 | 1 (8) | 2 (2) | 2 (16) | Y | Y1 |
| 880F128 | 128 | 16 | 85 | 4x40 | 3+1 | 2 | 1 | 3 (9) | 1 | 1 | 3 | 1 | 1 (8) | 2 (2) | 2 (16) | Y | Y1 |
| 890F32 | 32 | 8 | 90 | 4x40 | 3+1 | 2 | 1 | 3 (9) | 1 | 1 | 3 | 1 | 1 (8) | 2 (2) | 2 (16) | Y | Y1 |
| 890F64 | 64 | 16 | 90 | 4x40 | 3+1 | 2 | 1 | 3 (9) | 1 | 1 | 3 | 1 | 1 (8) | 2 (2) | 2 (16) | Y | Y1 |
| 890F128 | 128 | 16 | 90 | 4x40 | 3+1 | 2 | 1 | 3 (9) | 1 | 1 | 3 | 1 | 1 (8) | 2 (2) | 2 (16) | Y | Y1 |

Table Derivative overview

The ARM Cortex™-M3 includes three AHB-Lite buses, one system bus and the I-code and D-code buses which are faster and are used similarly to TCM interfaces: one bus dedicated for instruction fetch (I-code) and one bus for data access (D-code). The use of two core buses allows for simultaneous operations if concurrent operations target different devices.

The EFM32G uses an energy optimized multi-layer AMBA AHB matrix to connect the Cortex™-M3 buses and other bus masters to peripherals in a flexible manner that optimizes performance by allowing peripherals on different slaves ports of the matrix to be accessed simultaneously by different bus masters.

APB peripherals are connected to the CPU via two APB busses using separate slave ports from the multilayer AHB matrix. This allows for better performance by reducing collisions between the CPU and the DMA controller. The APB bus bridges are configured to buffer writes so that the CPU or DMA controller can write to APB devices without always waiting for APB write completion.

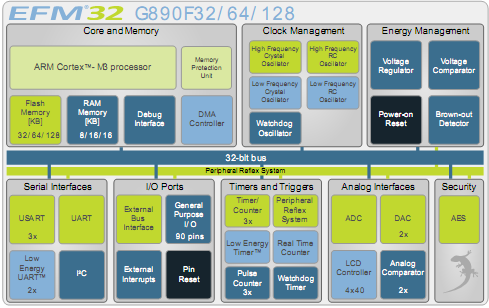


Image Structural overview

The controller combines the CortexTM–M3 core with dedicated peripheral blocks:

The “Energy Management Unit” (EMU)manages all the low energy modes (EM) in EFM32G microcontrollers. The EMU can also be used to turn off the power to unused SRAM blocks. The “Clock Management Unit” (CMU) is responsible for controlling the oscillators and clocks on-board the EFM32G. The CMU provides the capability to turn on and off the clock on an individual basis to all peripheral modules in addition to enable/disable and configure the available oscillators. The high degree of flexibility enables software to minimize energy consumption in any specific application by not wasting power on peripherals and oscillators that are inactive. The “Serial Interfaces” provide common and energy aware standard serial communication ports. The “I/O Ports” block provides access to individual I/O configuration on dedicated functions like Interrupt or module interfaces. “Timers and Triggers” combines counter, timers, real time clock and watchdog timer with the “Peripheral Reflex System” which is used for autonomous peripheral interaction. The “Analog Interfaces” with comparator, DAC and ADC is also capable to drive LCD controller. A “Security” block includes an AES encryption/decryption module for encrypted communication.

* 1. Detailed Library Tested Peripherals Description

This chapter gives a detailed description of the components in the EFM32G ARM Cortex™-M3 families that should be tested according the CEI/IEC 60335-1 Class B and CEI/IEC 60730 specification.

The components that are mandatory for the Class B certification are: (see (3), table H.11.12.7, see (4))

1) CPU

1.1) CPU registers

1.2) Program Counter

2) Interrupt Operations

3) Clock Structure

3.1) Watchdog Timer

4) Memory

4.1) Invariable Memory

4.2) Variable Memory

4.3) Critical Data

### CPU, Cortex-M3

The processor or central processing unit (CPU) of the EFM32G Cortex™-M3 microcontrollers uses the ARM Cortex™-M3 version r2p0 core, which is an implementation of the ARMv7-M architecture, developed by ARM Ltd (1).

This processor core incorporates (5):

Processor Core. A low gate count core, with low latency interrupt processing that features:

* ARMv7-M. A Thumb®-2 *Instruction Set Architecture* (ISA) subset, consisting of all base Thumb®-2 instructions, 16-bit and 32-bit, and excluding blocks for media, *Single Instruction Multiple Data* (SIMD), enhanced *Digital Signal Processor* (DSP) instructions (E variants), and ARM system access.
* Banked *Stack Pointer* (SP) only.
* Hardware divider instructions, SDIV and UDIV (Thumb®-2 instructions).
* Handler and Thread modes.
* Thumb and Debug states.
* Interruptible-continued LDM/STM, PUSH/POP for low interrupt latency.
* Automatic processor state saving and restoration for low latency *Interrupt* *Service Routine* (ISR) entry and exit.
* ARM architecture v6 style BE8/LE support.
* ARMv6 unaligned accesses.

*Nested Vectored Interrupt Controller* (NVIC) closely integrated with the processor core to achieve low latency interrupt processing. Features:

* External interrupts of 1 to 240 configurable in size.
* Bits of priority of 3 to 8 configurable size.
* Dynamic reprioritization of interrupts.
* Priority grouping. This enables selection of pre-empting interrupt levels and non pre-empting interrupt levels.
* Support for tail-chaining and late arrival of interrupts. This enables back-to-back interrupt processing without the overhead of state saving and restoration between interrupts.
* Processor state automatically saved on interrupt entry, and restored on interrupt exit, with no instruction overhead.
* Memory Protection Unit (MPU):
* Eight memory regions.
* *Sub Region Disable* (SRD), enabling efficient use of memory regions.
* Background region that implements the default memory map attributes.

Bus interfaces:

* *Advanced High-performance Bus-Lite* (AHB-Lite) ICode, DCode and System bus interfaces.
* Advanced Peripheral Bus (APB) and Private Peripheral Bus (PPB) Interface.
* Bit band support that includes atomic bit band write and read operations.
* Memory access alignment.
* Write buffer for buffering of write data.

Low-cost debug solution that features:

* Debug access to all memory and registers in the system, including Cortex™-M3 register bank when the core is running, halted, or held in reset.
* Serial Wire Debug Port (SW-DP) or Serial Wire JTAG Debug Port (SWJ-DP) debug access, or both.
* *Flash Patch and Breakpoint* (FPB) unit for implementing breakpoints and code patches.
* *Data Watchpoint and Trace* (DWT) unit for implementing watchpoints, data tracing, and system profiling.
* *Instrumentation Trace Macrocell* (ITM) for support of code instrumented debugging.
* Trace Port Interface Unit (TPIU) for bridging to a Trace Port Analyzer (TPA).

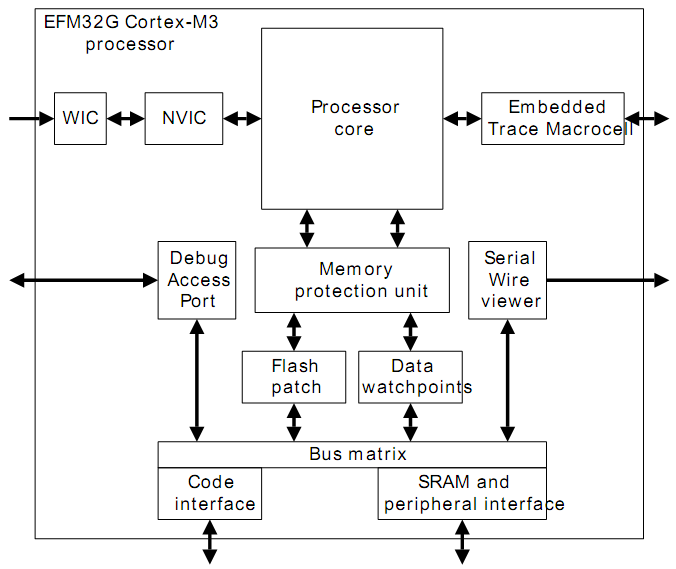


Image EFM32G Cortex™-M3 Implementation

#### CPU Registers And Program Counter

The Cortex™-M3 r2p0 core has 13 32-bit wide general-purpose registers [R0-R12], which can be divided by two sets of registers the low and high registers. The low registers are accessible by all instructions that specify a general-purpose register and the high registers are only accessible by 32-bit instructions.

Besides the general-purpose registers, r13-r15 has some special functions. The Stack Pointer (SP) is the register R13. In Thread mode, bit [1] of the CONTROL register stack pointer to use:

* 0 = Main Stack Pointer (MSP). This is the reset value.
* 1 = Process Stack Pointer (PSP).

On reset, the processor loads the MSP with the value from address 0x00000000.

The Link Register (LR) is the register R14. It stores the return information for subroutines, function calls, and exceptions. Function calls are *branch and link* (BL) or *branch and link with exchange* (BLX) instruction. Register R14 can also be used as a general-purpose register.

The Program Counter (PC) is the register R15. It contains the current program address. Bit [0] is always 0 because instruction fetches must be halfword aligned. On reset, the processor loads the PC with the value of the reset vector, which is at address 0x00000004..

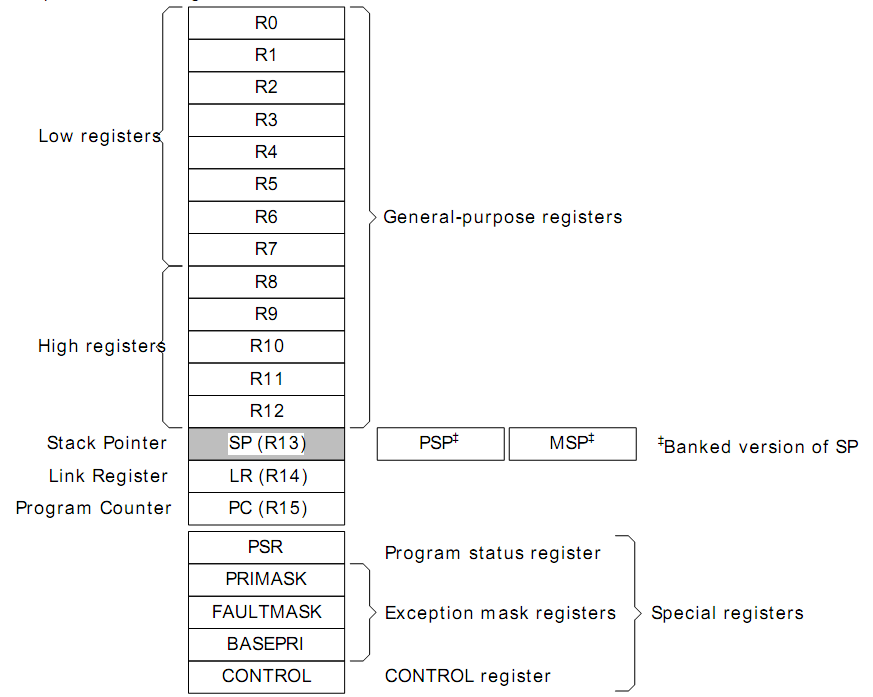


Image Processor Registers

The processor also has some status registers that can be divided in three categories at system level.

The Program Status Register (PSR) combines:

* Application Program Status Register (APSR)
* Interrupt Program Status Register (IPSR)
* Execution Program Status Register (EPSR).

For a detailed description see the *Cortex-M3* r2p0 *Technical Reference Manual* (1).

### Interrupt Operation

The Cortex™-M3 processor supports interrupts and system exceptions. The processor and the Nested Vectored Interrupt Controller (NVIC) prioritize and handle all exceptions. An exception changes the normal flow of software control. The processor uses handler mode to handle all exceptions except for reset.

The NVIC supports:

* 1 to 30 interrupts.
* Programmable priority levels of 0-7 for each interrupt. A higher level corresponds to a lower priority, so level 0 is the highest interrupt priority.
* Level detection of interrupt signals.
* Dynamic reprioritization of interrupts.
* Grouping of priority values into group priority and subpriority fields.
* Interrupt tail-chaining.

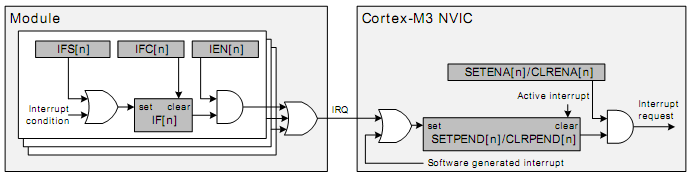


Image Interrupt Operation

Each of the interrupt lines (Image 5 Interrupt Operation) is connected to one or more interrupt flags in one or more modules. The interrupt flags are set by hardware on an interrupt condition. It is also possible to set/clear the interrupt flags through the IFS/IFC registers. Each interrupt flag is then qualified with its own interrupt enable bit (IEN register), before being OR'ed with the other interrupt flags to generate the IRQ. (Image 5 Interrupt Operation) illustrates the interrupt system. For more information on how the interrupts are handled inside the Cortex™-M3, see the EFM32G Cortex™-M3 Reference Manual or the Cortex™-M3 r2p0 Technical Reference Manual, Chapter 8 “Nested Vectored Interrupt controller”. (5)

### Clock Management Unit

This chapter describes the clock management in the EFM32G Cortex™-M3 family.

The Clock Management Unit (CMU) controls oscillators and clocks. EFM32G supports four different oscillators with minimized power consumption and short start-up time. An additional and separate RC oscillator is used for flash programming and debug trace. The CMU also has HW support for calibration of RC oscillators.

#### Clock Sources And Oscillators Support

The EFM32G device offers a clock structure where multiple clock sources are available:

* 1-28 MHz High Frequency RC Oscillator (HFRCO)
* 4-32 MHz High Frequency Crystal Oscillator (HFXO)
* 32.768 kHz Low Frequency RC Oscillator (LFRCO)
* 32.768 kHz Low Frequency Crystal Oscillator (LFXO)

The clock structure offers support for low power oscillators, low start-up times, separate prescaler for High Frequency Core Clocks (HFCORECLK) and Peripheral Clocks (HFPERCLK), Individual clock prescaler selection for each Low Energy Peripheral and Clock Gating on an individual basis to core modules and all peripherals

Selectable clocks can be output on two pins for use externally.

The auxiliary 14 MHz RC-Oscillator (AUXHFRCO) is supported for flash programming and debug trace.

#### High Frequency System Clock

HFCLK (High Frequency Clock) is the selected High Frequency Clock. This clock is used by the CMU and drives the two prescaler that generate HFCORECLK and HFPERCLK. The HFCLK can be driven by a high-frequency oscillator (HFRCO or HFXO) or one of the low-frequency oscillators (LFRCO or LFXO). By default the HFRCO is selected. In most applications, one of the high frequency oscillators will be the preferred choice.

HFCORECLK is a prescaled version of HFCLK. This clock drives the Core Modules, which consists of the CPU and modules that are tightly coupled to the CPU, e.g. MSC, DMA etc. This also includes the interface to the Low Energy Peripherals. Some of the modules that are driven by this clock can be clock gated completely when not in use.

Like HFCORECLK, HFPERCLK is also a potentially prescaled version of HFCLK. This clock drives the High-Frequency Peripherals. All the peripherals that are driven by this clock can be clock gated completely when not in use.

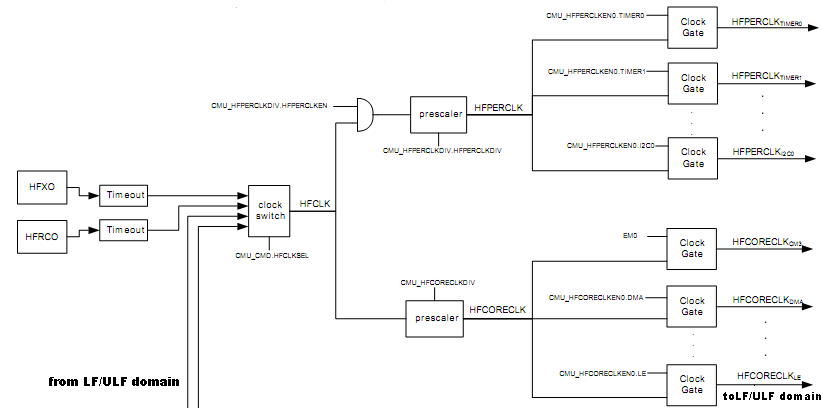


Image High Frequency Domain

#### Low Frequency A and B Clock, Watchdog Clock

LFACLK is the selected clock for the Low Energy A Peripherals. There are three selectable sources for LFACLK: LFRCO, LFXO and HFCORECLKLE/2. In addition, the LFACLK can be disabled. From reset, the LFACLK source is set to LFRCO. The LFRCO is disabled from reset.

LFBCLK is the selected clock for the Low Energy B Peripherals. There are three selectable sources for LFBCLK: LFRCO, LFXO and HFCORECLKLE/2. In addition, the LFBCLK can be disabled. From reset, the LFBCLK source is set to LFRCO. The LFRCO is disabled from reset.

Each available pulse counter is driven by its own clock PCNTnCLK where n is the pulse counter instance number.

The Watchdog Timer (WDOG) can be configured to use one of three different clock sources: LFRCO, LFXO or ULFRCO. ULFRCO (Ultra Low Frequency RC Oscillator) is a separate 1 kHz RC oscillator that also runs in low energy modes.

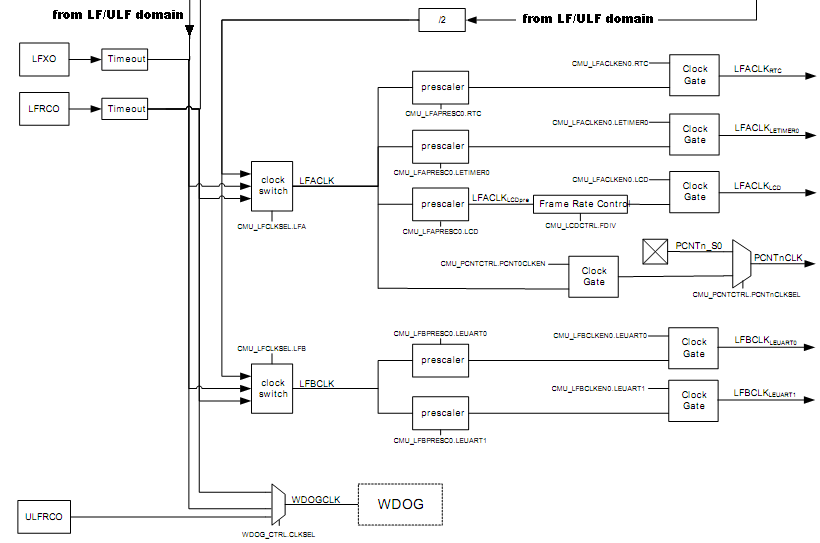


Image Low Frequency Domain

#### Auxiliary Clock

AUXCLK is a 14 MHz clock driven by a separate RC oscillator, AUXHFRCO. The AUXCLK clock is used for flash programming and debugs trace. During flash programming, this clock will be active. If the AUXHFRCO has not been enabled explicitly by software, the MSC module will automatically start and stop it. Explicit enabling is required when debug trace is used.

### Memory

This chapter describes the memory in the EFM32G Cortex™-M3 family.

The memory size for both variable and invariable memory depends on the family member selected. The memory sizes can be found in the EFM32G reference guide.

#### ARM Cortex-M3 Memory Map

The ARM Cortex™-M3 processor memory architecture is different from the traditional ARM7/ARM9 processors.

The following features are implemented:

Predefined memory map providing a 4GB addressable memory area.

The ARM Cortex™-M3 memory map specifies which bus interface is to be used when a memory location is accessed.(1) , (5)

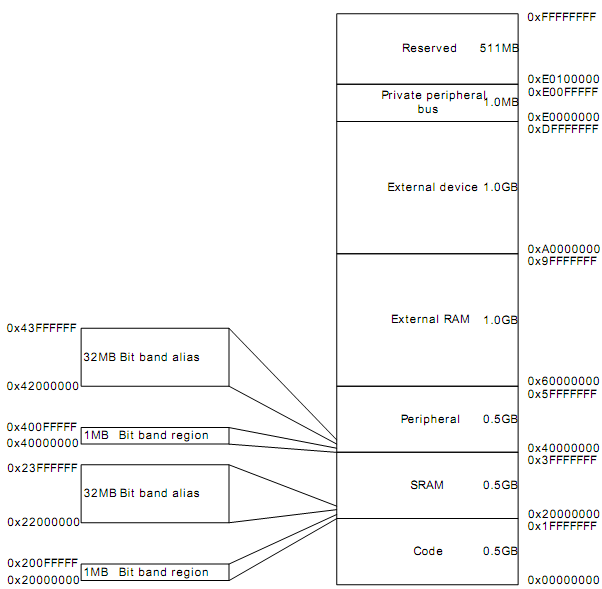


Image Cortex-M3 Memory Model

The regions for SRAM and peripherals include bit-band regions. Bit-banding provides atomic operations.

The processor reserves regions of the Private peripheral bus (PPB) address range for core peripheral registers.

The EFM32G uses a little-endian format, in which the processor stores the least significant byte of a word at the lowest-numbered byte, and the most significant byte at the highest-numbered byte.

For detailed information on the ARM Cortex™-M3 memory architecture and model please see the Cortex™-M3 r2p0 Technical Reference Manual, Chapter 4 “Memory Map” (5) or the ARMv7-M Architecture Application Level Reference Manual, Chapter A3 “ARM Architecture Memory Model” (6)

#### EFM32G Cortex-M3 Memory Map

The Cortex-M3 architecture supports both little- and big-endian modes of operation. The EFM32G device will always operate in little-endian mode. For a detailed overview of the Cortex-M3 Memory Map, see the Cortex-M3 Technical Reference Manual,(1) chapter 4.

4 main memory segments are present, which can be accessed by the Cortex-M3 or the DMA controller:

RAM

Flash

Peripherals

External Bus Interface (EBI)

The memory segments are mapped together with the internal segments of the Cortex-M3 into the system memory map as shown by the figure below:

MemoryMap.emf

Image Detailed Memory Map

#### SRAM

The embedded SRAM is located at address 0x2000 0000 in the memory map of the EFM32G. When running code located in SRAM starting at this address, the Cortex-M3 uses the System bus interface to fetch instructions. SRAM is also mapped in the code space at address 0x1000 0000. When running code from this space, the Cortex-M3 fetches instructions through the I/D-Code bus interface.

#### Bit-banding

The SRAM bit-band alias and peripheral bit-band alias regions are located at 0x2200 0000 and 0x4200 0000 respectively. Read and write operations to these regions are converted into masked single-bit reads and atomic single-bit writes to the embedded SRAM and peripherals of the EFM32G.

#### Peripherals

The EFM32G contains both generic (Peripherals) and Cortex-M3 private peripherals (noted as CM3 Peripherals in the memory map). The Cortex-M3 private peripherals include the Debug, MPU, NVIC interrupt controller, and the ROM table and other peripherals tightly coupled to the Cortex-M3 core. These are mapped in address space going from 0xE000 0000 to 0xE00F FFFF.

The device specific peripherals are all other peripherals, mapped in the address range from 0x4000 0000 to 0x41FF FFFF.

#### Device specific memory configurations

The following list defines the Flash and SRAM configuration of the various parts in the EFM32G series:

|  |  |  |  |
| --- | --- | --- | --- |
| Part Number | Flash (KB) | SRAM (KB) | Package |
| [EFM32G890F128](http://www.energymicro.com/products/efm32g890f128-efm32g890f64-efm32g890f32) | 128 | 16 | BGA112 |
| [EFM32G890F64](http://www.energymicro.com/products/efm32g890f128-efm32g890f64-efm32g890f32) | 64 | 16 | BGA112 |
| [EFM32G890F32](http://www.energymicro.com/products/efm32g890f128-efm32g890f64-efm32g890f32) | 32 | 8 | BGA112 |
| [EFM32G880F128](http://www.energymicro.com/products/efm32g880f128-efm32g880f64-efm32g880f32) | 128 | 16 | QFP100 |
| [EFM32G880F64](http://www.energymicro.com/products/efm32g880f128-efm32g880f64-efm32g880f32) | 64 | 16 | QFP100 |
| [EFM32G880F32](http://www.energymicro.com/products/efm32g880f128-efm32g880f64-efm32g880f32) | 32 | 8 | QFP100 |
| [EFM32G840F128](http://www.energymicro.com/products/efm32g840f128-efm32g840f64-efm32g840f32) | 128 | 16 | QFN64 |
| [EFM32G840F64](http://www.energymicro.com/products/efm32g840f128-efm32g840f64-efm32g840f32) | 64 | 16 | QFN64 |
| [EFM32G840F32](http://www.energymicro.com/products/efm32g840f128-efm32g840f64-efm32g840f32) | 32 | 8 | QFN64 |
| [EFM32G290F128](http://www.energymicro.com/products/efm32g290f128-efm32g290f64-efm32g290f32) | 128 | 16 | BGA112 |
| [EFM32G290F64](http://www.energymicro.com/products/efm32g290f128-efm32g290f64-efm32g290f32) | 64 | 16 | BGA112 |
| [EFM32G290F32](http://www.energymicro.com/products/efm32g290f128-efm32g290f64-efm32g290f32) | 32 | 8 | BGA112 |
| [EFM32G280F128](http://www.energymicro.com/products/efm32g280f128-efm32g280f64-efm32g280f32) | 128 | 16 | QFP100 |
| [EFM32G280F64](http://www.energymicro.com/products/efm32g280f128-efm32g280f64-efm32g280f32) | 64 | 16 | QFP100 |
| [EFM32G280F32](http://www.energymicro.com/products/efm32g280f128-efm32g280f64-efm32g280f32) | 32 | 8 | QFP100 |
| [EFM32G230F128](http://www.energymicro.com/products/efm32g230f128-efm32g230f64-efm32g230f32) | 128 | 16 | QFN64 |
| [EFM32G230F64](http://www.energymicro.com/products/efm32g230f128-efm32g230f64-efm32g230f32) | 64 | 16 | QFN64 |
| [EFM32G230F32](http://www.energymicro.com/products/efm32g230f128-efm32g230f64-efm32g230f32) | 32 | 8 | QFN64 |
| [EFM32G210F128](http://www.energymicro.com/products/efm32g210f128) | 128 | 16 | QFN32 |
| [EFM32G200F64](http://www.energymicro.com/products/efm32g200f64-efm32g200f32-efm32g200f16) | 64 | 16 | QFN32 |
| [EFM32G200F32](http://www.energymicro.com/products/efm32g200f64-efm32g200f32-efm32g200f16) | 32 | 8 | QFN32 |
| [EFM32G200F16](http://www.energymicro.com/products/efm32g200f64-efm32g200f32-efm32g200f16) | 16 | 8 | QFN32 |

Table Memory on Derivatives available

Flash address space is always linear, and starts at address 0x0000 0000 and upwards.

SRAM address space is also linear, and starts at address 0x2000 0000 and upwards.

#### Bus Matrix

The Bus Matrix connects the memory segments to the bus masters.

Code: CPU instruction or data fetches from the code space

System: CPU read and write to the SRAM, EBI and peripherals

DMA: Access to code space, SRAM, EBI and peripherals

The Bus Matrix uses a round-robin arbitration algorithm which enables high throughput and low latency while starvation of simultaneous accesses to the same bus slave are eliminated. Round-robin does not assign a fixed priority to each bus master. The arbiter does not insert any bus wait-states.

The Bus Matrix is a multi-layer energy optimized AMBA AHB compliant bus with an internal bandwidth of 4x a single AHB interface. The Bus Matrix accepts new transfers to be initiated by each master in each cycle without inserting any wait-states.

It has 3 slave inputs and 4 master outputs:

Slaves:

idcodebus (Unified DCode and ICode interface)

systembus (System interface)

dmabus (DMA interface)

Masters:

imembus (Flash)

dmembus (RAM)

perbus (Peripherals)

ebibus (External Bus Interface)

Image  Bus Matrix

#### Invariable Memory

The Flash retains data in any state and typically stores the application code, special user data and security information. The Flash memory can be programmed through the debug interface, but can also be erased and written to from software.

Feature and organization details

128 KB of memory (max)

512 bytes page size (minimum erase unit)

Minimum 20K erase cycles endurance

Greater than 10 years data retention at 85°C

Lock-bits for memory protection

Data retention in any state

32-bit word size for programming

Organization 32768-words x 32-bits

55ns min cycle time

12mA (max) write operation

230uA (1MHz) read operation

7uA (max) standby at 105ºC, 4uA (max) at 85ºC

#### Variable Memory

Two main options for using variable memory is present, the on-board SRAM and through the External Bus Interface (EBI). SRAM can be used for variables and code, while EBI can only be used for storing data.

#### External Bus Interface

The External Bus Interface provides access to external parallel interface devices such as SRAM, FLASH, ADCs and LCDs. The interface is memory mapped into the address bus of the Cortex-M3. This enables seamless access from software without manually manipulating the IO settings each time a read or write is performed. The data and address lines are multiplexed in order to reduce the number of pins required to interface the external devices.

The timing is adjustable to meet specifications of the external devices. The interface is limited to asynchronous devices.

Features include

Multiplexed data and address lines for reduced pin count

Up to 24-bit address width

Up to 16-bit data width

8-bit true parallel operation

4 memory bank regions with individual chip select lines (EBI\_CSn)

Accurate control of setup, strobe, hold and turn-around timing

Individual active high / active low setting of interface control signals

Slave read/write cycle extension

An overview of the External Bus Interface is shown in the following figure.



Image External Bus Interface

#### SRAM Architecture

The primary task of the SRAM memory is to store application data. Additionally, it is possible to execute instructions from SRAM, and the DMA may be set up to transfer data between the SRAM, Flash and peripherals.

Feature and organization details

Up to 16 KB memory

Bit-band access support

4KB blocks may be individually powered down when not in use

Data retention of the entire memory in EM0 to EM3

Physically, the SRAM memory is organized in

4 blocks (of 4KB each)

1024 words per block

32 bits per word

EFM32G devices with 8KB of SRAM, has only 2 blocks available, while EFM32G devices with 16KB of SRAM (see table above) has all 4 blocks available.

# The Class B Self-Test Library

* 1. Library Architecture

In this section, the software architecture of the Class B Test Library is described. All modules to be implemented will be designed in detail. This document describes the decomposition of the system and module requirements allocation on the software units. When needed interaction between the software units will also be described.

### Target and System Requirements

The target for the Class B library is

Build reusable software modules of the EFM32G Class B library

Sufficient coverage of the IEC standard if IEC 60730 edition 4

The key requirements are defined in the IEC 60730-1 Annex H standard called “requirements for electronics control” from 10/2007. This defines that home appliances, sold in Europe, have to comply to the ICE 60730 and IEC 60335. For MCU systems the IEC 60730 Annex H defines the details of tests and diagnostics 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 single a channel

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

The software is classified in Annex H to 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.

### System Overview

The library modules will be allocated to the system Flash together with drivers and user application. The system RAM is applicable for all data. A first part is the POST test which runs in uninitialized mode on the system. The second part is the application or BIST part of the library which allows testing during runtime controlled by an application. If the application uses functions of the library the application has to take care of the environment, setting up required hardware and peripherals, terminate and check the results of the library function.

### System Design and Design Method

Due to memory restrictions and compactness of the software units developers will only be using flow chart diagrams to detail the processing of their modules. Within this flow chart all the condition and activity nodes will be described. Also all control paths shall have an activity, for all none standard control paths an error response will be described.

More in general it will be that the library will be equipped with the following software protection mechanisms:

1. Non volatile memory access will be restricted to valid address range and checksum controlled. The checksum and its control data are outside the controlled memory range.
2. Volatile memory access will be restricted to address ranges outside the stack memory. Illegal addresses or address ranges will be reduced to sufficient values.
3. Ranges and plausibility check on every interface handling with data parameter will be implemented. In case of non valid addresses and data are passed the library function should return the common error codes.
4. Global data is reduced to a minimum. This data is protected against influence by declaring them module static.

### Logical Presentation, Decomposition

Image logical Library Representation

User Applikation

EFM32G Class B Library

EFM32G Hardware

CPU reg

Flash

RAM

Clock / R TC/WDT

IRQ

Data

The IEC60730 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 XX system comprehensive components in it’s implementation.

|  |  |  |  |
| --- | --- | --- | --- |
| #r | 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 | StackPointer (SP) | 1.1 | Stuck at fault |
| 4 | Interrupt Operation | 2 | Miss or ignorance |
| 5 | CPU clock | 3 | Frequency match |
| 6 | Invariable Memory | 4.1 | Bit fault |
| 7 | Variable Memory | 4.2 | Coupling fault |
| 8 | Critical Data | 4.2 | Coupling fault |
| 9 | Race condition and Addressing | 4.3 | Program execution |

Table Library Components Test Implementation

### Requirements and Traceability

In this section, the traceability of the requirements of the implemented functions is described, 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.

System-verification

Integration

testing

Module-

testing

System Requirements Specification

Architecture Specification

Module Requirements

Specification Verification

Time

Definition Implementation Verification

Image V-model

The V-model defines several fixed identifiers:

Requirements: visible and traceable feature in a product. (stakeholder features, product requirements and module requirements)

Refinements: parent requirement to add additional information to other requirements

Derived Requirements: feature defined in secondary order not related to mean requirements

Coverage: reference to requirement or refinements to avoid duplications in implementation and verification

The V-model specifies the specification of requirements in every step of the development. This specification can be scaled to the individual depth of the part of actual targeted development.

The EFM32G Class B library is not defined as product but is intended to be part of a product; this means the library is defined as module.

For every function in the module a specification or requirement is necessary to enable verification of the related functions.

### Module Requirements

The EFM32G Class B library provides routines which can be obtained by the user to check the system at startup or periodically during runtime. The requirements of the test routines can be summarized in the following table:

|  |  |  |  |
| --- | --- | --- | --- |
| # | Affected Component | Test Routine | Description |
| 1 | CPU registers | IEC60335\_ClassB\_CPUregTest\_POST() | POST test of all CPU registers |
| 1.1 |  | IEC60335\_ClassB\_CPUregTest\_BIST() | BIST test of all CPU registers |
| 1.2 |  | \_CPUregTestLOW() | BIST test of CPU registers R0-R7 |
| 1.3 |  | \_CPUregTestMID() | BIST test of CPU registers R4-R10 |
| 1.4 |  | \_CPUregTestHIGH() | BIST test of CPU registers R8-R12 |
| 1.5 |  | \_CPUregTestSP() | BIST test of CPU registers SP, MSR, MRS |
| 1.6 |  | \_CPUregTestSPEC() | BIST test of CPU registers LR |
| 2.0 | ClockSystem | IEC60335\_ClassB\_initClockTest() | Init BIST test for clock and RTC |
| 2.1 |  | IEC60335\_ClassB\_Clocktest\_PollHandler() | BIST test for continues timer test |
| 2.2 |  | IEC60335\_ClassB\_Clocktest\_TimerHandler() |  |
| 2.3 |  | IEC60335\_ClassB\_Clocktest\_RTCHandler() | BIST test for continues RTC test |
| 3.0 | WDT, PC stuck | IEC60335\_ClassB\_initWDT() | Init BIST test for WDT |
| 3.1 |  | IEC60335\_ClassB\_Refresh\_WDT() | Refresh WDT |
| 3.2 |  | IEC60335\_ClassB\_ForceWDTreset() | Test to force reset function of WDT |
| 4.0 | RAM | IEC60335\_ClassB\_RAMtest\_POST | POST test of the complete RAM |
| 4.1 |  | IEC60335\_ClassB\_RAMtest\_BIST |  |
| 5.0 | Flash | IEC60335\_ClassB\_FLASHtest\_POST() | POST test of Flash content |
|  |  | IEC60335\_ClassB\_FLASHtest\_BIST() | BIST consecutive test of Flash content |
| 6 | Interrupt | \_\_call\_Vect() | Subsystem call the original vector |
|  |  | IEC60335\_IRQReplacementHandler() | Interrupt replacement handler |
|  |  | IEC60335\_ClassB\_InitInterruptTest() | Initialization of a specified interrupt test |
|  |  | IEC60335\_ClassB\_InterruptCheck() | Interrupt test check routine |

Table Requirements Description

The document to specify the requirements for the EFM32G Class B library is this document. The chapter is specifying the test specification is this document.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Requirement** | **Reference** | **Coverage** | **Textual coverage** |
| 1 | IEC60730 Annex H compliance | this | this | CPU Register: stuck at fault |
| 2 |  |  |  | CPU registers are tested: R0..R12, PC, LR, PSP, MSP, PSR, APSR |
| 3 |  |  |  | PC stuck at fault |
| 4 |  |  |  | PC stuck and WDT overflow causes a reset |
| 5 |  | this | this | RAM marching tests on bit stuck and DC |
| 6 |  | this | this | FLASH test on data retention or change |
| 7 | IEC60730 Annex H compliance | this | this | Timer reliability |
| 7a | IEC60730 Annex H compliance | this |  | Watchdog Timer reliability |
| 8 | IEC60730 Annex H compliance | this | this | IRQ response |

Table Coverage Points

Architectural requirements specifications are not implemented and mentioned here because of the library character of the sources. In this case there is no necessary content described in which the library is used. This is up to the final user and developer. The frame which is running the library is described in its purpose for exemplary usage of the library in chapter 3.4: Library Usage Description.

|  |  |  |  |
| --- | --- | --- | --- |
| # | Textual coverage | Module Requirements Chapter | Test Specification Chapter |
| 1 | CPU Register: stuck at fault | 3.3.3 | 0 |
| 2 | CPU registers are tested: R0..R12, PC, LR, PSP, MSP, PSR, APSR | 3.3.3, | 0, 0 |
| 3 | PC stuck at fault | 3.3.4 | 0, 0 |
| 4 | PC stuck and WDT overflow causes a reset | 3.3.7 | 0, 0 |
| 5 | RAM marching tests on bit stuck and DC | 3.3.8.3, 3.3.8.4 | 0, 0 |
| 6 | FLASH test on data retention or change | 3.3.8.1, 3.3.8.2 | 0, 0 |
| 7 | Timer reliability | 3.3.6 | 0 |
| 7a | Watchdog Timer reliability | 3.3.7 | 0 |
| 8 | IRQ response | 3.3.5 | 0, 0 |

Table Requirements and References

# Library Description and Usage

The V-model also defines the methods of testing to be specified. Testing means the functional and formal verification of behavior according the specification of each process. Functional and formal test description is documented in this document for each single function. Practical usage and descriptions to run the tests will follow in the chapter 3.4 Library Usage Description.

* 1. Framework application

The library includes an example project for all supported compilers (see chapter 4.3) to show how the POST and BIST tests can be used effectively. The application is not doing anything but testing the chip by using the given IEC60335 library.

Also functional verification and check can be evaluated for adaption to special needs.

* 1. Library Memory Requirements

The library is designed to run on Energy Micro Cortex-M3 systems with defined system memory mappings defined in the ARMv7 Architecture Reference Manual. Especially the System Address Mapping described in chapter B3 is basic requirement for the library’s design.

The fact that the library performs special routines which can possibly corrupt volatile memory and other routines referencing on fixed locations the library requires some fixed basic memory mapping constrains.

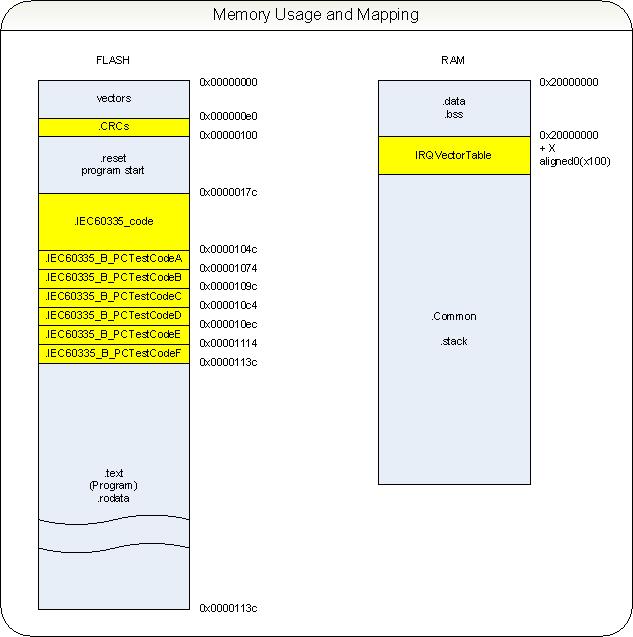


Image Memory Mapping

Special Flash related areas are CRC and the separated code sections IEC60335\_code and IEC60335\_B\_PCTestCodeA to IEC60335\_B\_PCTestCodeF. IEC60335\_code contains all test code. IEC60335\_B\_PCTestCodeA contains the TestCodeA of program counter test routines. The section .CRCs contains the CRC test struct “IEC60335\_Flash\_CRC\_REF” for the Flash test routines. This section is located outside the .reset and .text sections to allow changes without affecting the programs CRC value.

A Special RAM related area is the section IRQVectorTable. This section must be aligned to an address multiple of 0x100. The section contains the copy of the vector table in case of an interrupt test activated.

The library is trimmed to use a very short amount of RAM. If possible the data are defines either const (uses Flash) or local static which is using data or bss section. The library is not intended to run with dynamic memory usage.

The example program requires for all sections

.text 16544 bytes

.data 28 bytes

.bss 2912 bytes

* 1. The Library Functions Specification

In this section, the functionality of the test function will be explained and who they are implemented.

### POST and BIST

POST (Pre Operation System Test) means the testing as part of the start-up procedure. This test can be destroyable, which means, that the data contents are not restored after executing the test. Also, in this state of application, there are normally no interrupts active.

Note, at start-up each test must be driven! CPU registers, PC, RAM and ROM. For this reason, there are special POST functions. To speed up this procedure, the tests are destroyable, and monolithic. Monolithic means for example the whole used flash memory in one shot.

For testing during runtime, the test functions must not be destroyable. As a result they must not be monolithic; this could be dangerous in time critical applications.

For this reason, there are the same test functions implemented as BIST (Build-In System Test), which should be used after the application is started. Also these functions allow only testing smaller parts of big blocks (only a section of RAM instead of the whole RAM in one shot for example).

### POST Tests

POST tests are collected in a functional order to enable the user to have low changes in the very first startup code provided by libraries like CMSIS. For this purpose the function IEC60335\_ClassB\_POST() is created.

|  |  |
| --- | --- |
| Functions specification |  |
| Function Name | IEC60335\_ClassB\_POST |
| File (module) | iec60335\_class\_b\_post.c |
| Parameter, type | None, - |
| Return value, type | None, - |
| Affected variables | Result, testResult\_t |
| conditions | Compiler specific files with different mnemonics are called for CPU-test and compiler specific code for PC test |

This function is called before the Data and BSS sections are initialized so that no data returned from the function can remain in the memory. For this reason the function will stuck in an endless “while” loop if any test fails.

Test case IEC60335\_ClassB\_POST\_01

This test case checks the function returning with no error condition in the called sub functions.

### CPU Register Test ()

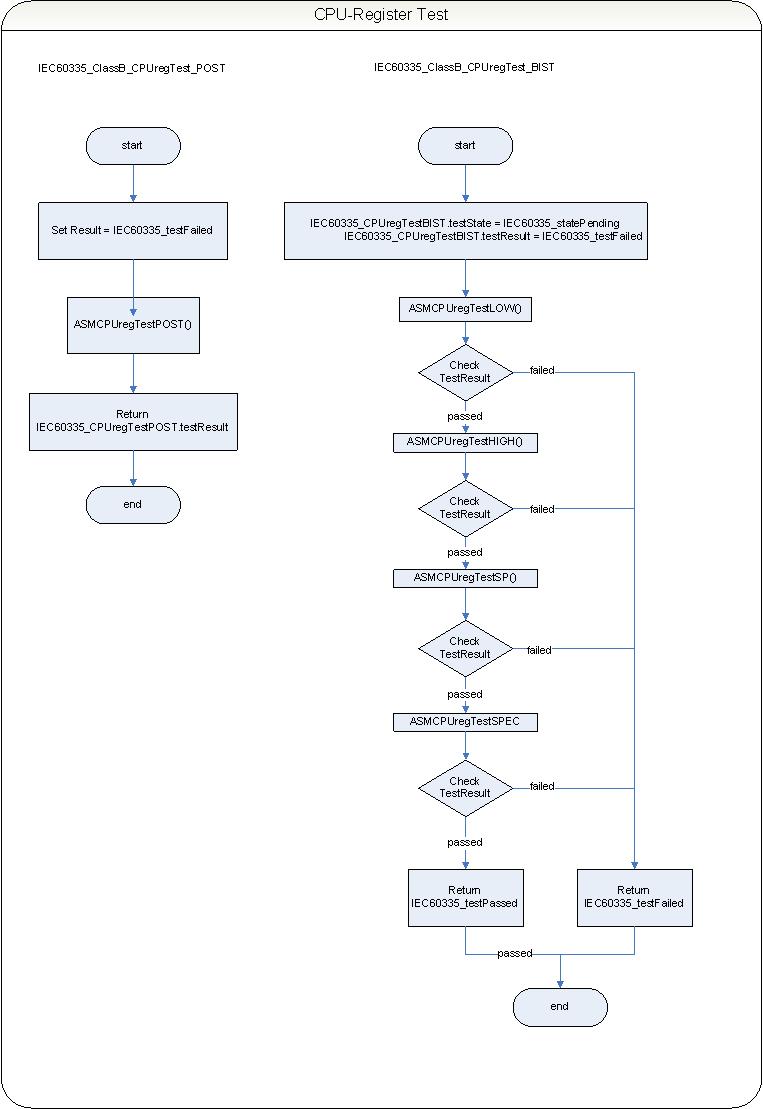
As described in chapter “1.2.1.1 CPU Registers And Program Counter“ the ARM Cortex™-M3 core has a number of registers used during program execution. Nineteen of these registers are read and writeable.

Image CPU-Register Test

Since these registers are all used during program execution in the various core operation modes, they need to be tested for “stuck-at” faults and “direct coupling” faults.

These tests are to be executed as POST and BIST.

POST testing is a destroyable test, so the CPU registers will not be retained. Since the POST CPU register tests doesn’t retain register data, it is mandatory to execute this test prior to the branch to main, this will be explained in the test usage description”3.4.1 CPU Register Test Usage”. All tests are executed in one routine, which allows the quickest test completion.

BIST testing the CPU registers must not to be destroyable, so all data must be restored after testing. To decrease test time and therefore CPU resources, the CPU register BIST testing is parted in four separate tests. The first three tests the general purpose registers. The fourth test tests the stack pointer. To prevent the system from crashing, all interrupts and exceptions are disabled while running this part of the BIST CPU register test.

The fifth and last BIST test is testing the other special registers.

|  |  |
| --- | --- |
| Functions specification |  |
| Function Name | IEC60335\_ClassB\_CPUregTest\_POST |
| File (module) | iec60335\_class\_b\_cpureg\_test.c |
| Parameter (type) | None, - |
| Return value type | Result (testResult\_t) |
| Affected variables | IEC60335\_CPUregTestPOST (testState, testResult) |
| conditions | Compiler specific files with different mnemonics are called |

Test description (POST) from file iec60335\_class\_b\_cpureg\_test\_post\_gcc.asm:

To trace the execution and fails each test adds a bit to the result status variable in register r8 to identify the last performed test or overall good condition. This is no mentioned in the following.

Starting the register test in the POST test the registers are saved, especially the link register because the assembler coded tests are called from a C-file. Even is the stack area is uninitialized the push and pop calls are relevant for returning to the caller function. The result status register is cleared and set to the first (r0\_test) status.

Initially the register r0 is tested:

First a pattern (0xAA000000) is moved in the register. Second the register r0 is shifted left by 24 (3 bytes) which lead to the pattern residing in the lowest byte of the register. Third the pattern is compared with a literal 0xAA. If the content is correct the next step is performed.

A pattern (0x00AA0000) is moved in the register. Second the register r0 is shifted left by 16 (2 bytes) which lead to the pattern residing in the lowest byte of the register. Third the pattern is compared with a literal 0xAA. If the content is correct the next step is performed.

A pattern (0x0000AA00) is moved in the register. Second the register r0 is shifted left by 8 (1 byte) which lead to the pattern residing in the lowest byte of the register. Third the pattern is compared with a literal 0xAA. If the content is correct the next step is performed.

A pattern (0x000000AA) is moved in the register. Second the pattern is compared with a literal 0xAA. If the content is correct the next step is performed.

Fail condition will branch to the label Test\_r0\_Exit, pass condition will continue with next test.

Register r1 to r7 test:

All registers a loaded with the pattern1 from the constants list and each register is compared against a literal pattern1. This test is repeated with the pattern3 from the constants list. Fail condition will branch to the label Test\_r1\_r7\_Exit, pass condition will continue with next test.

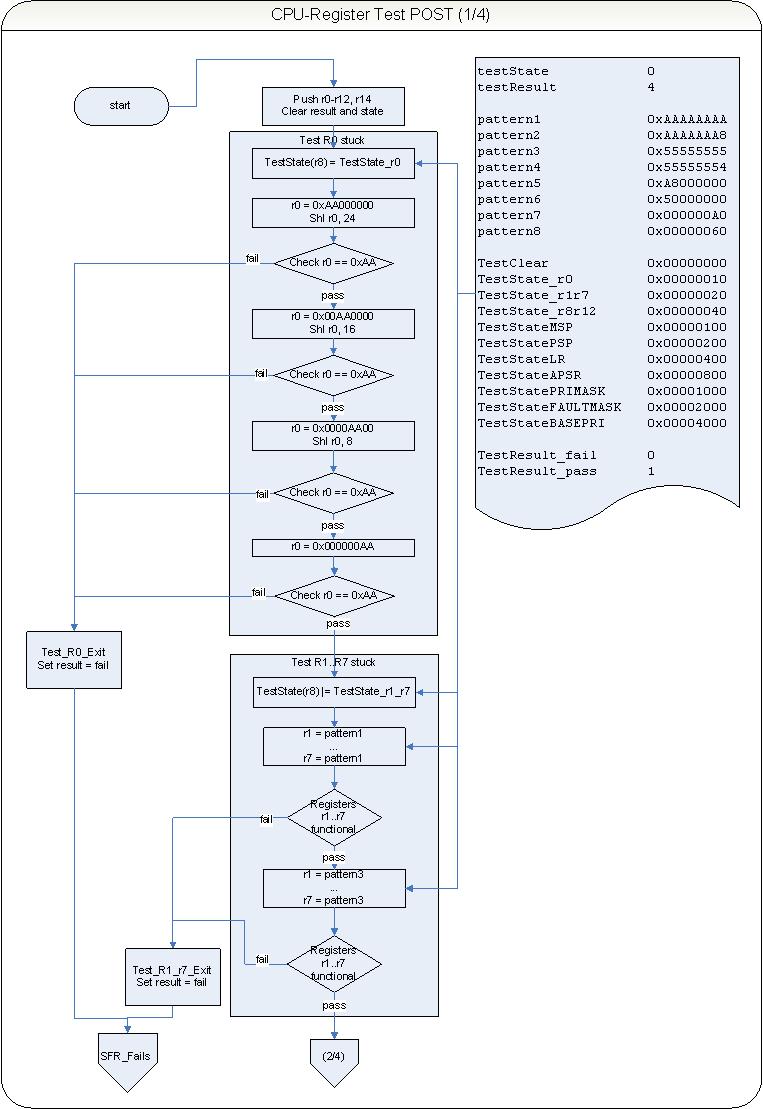


Image CPU Register Test LOW

Register r8 to r12 test:  
All registers a loaded with the pattern1 from the constants list and each register is compared against a literal pattern1. This test is repeated with the pattern3 from the constants list. Fail condition will branch to the label Test\_r8\_r12\_Exit, pass condition will continue with next test.

Register MSP test:  
The MSP register is accessible only in certain control states. So the MSP is set and verified to be the current and accessible stack pointer by clearing and reading the control register. In this case the register is also accessible via register r13 Because of the fact that MSP bit1 and 0 are always read as zero the MSP is filled with pattern2 and pattern4 from the constants list.   
Accessing the SMP register is possible with the commands mrs (move register from special register) and msr (move special register from register) or simple mov command. All combinations are used with the both pattern. After the register r0 with the constant pattern the MSP (r13) is loaded with a mov command. The content is checked by comparing the source r0 with r13. Writing the MSP with the msr command software is enabled to read the content back with the mrs command to register r1. An indirect comparison of the source register r0 and the MSP registers content in register r1 is made. Fail condition will branch to the label Test\_MSP\_Exit, pass condition will continue with next test.

Register PSP test:  
Accessing the PSP requires other state options set in the control register. Setting the control register to 0x02 enables the access. Because of the fact that PSP bit1 and 0 are always read as zero the PSP is filled with pattern2 and pattern4 from the constants list.   
Accessing the SMP register is possible with the commands mrs (move register from special register) and msr (move special register from register) or simple mov command. All combinations are used with the both pattern. After the register r0 with the constant pattern the MSP (r13) is loaded with a mov command. The content is checked by comparing the source r0 with r13. Writing the MSP with the msr command software is enabled to read the content back with the mrs command to register r1. An indirect comparison of the source register r0 and the MSP registers content in register r1 is made. Fail condition will branch to the label Test\_PSP\_Exit, pass condition will continue with next test.

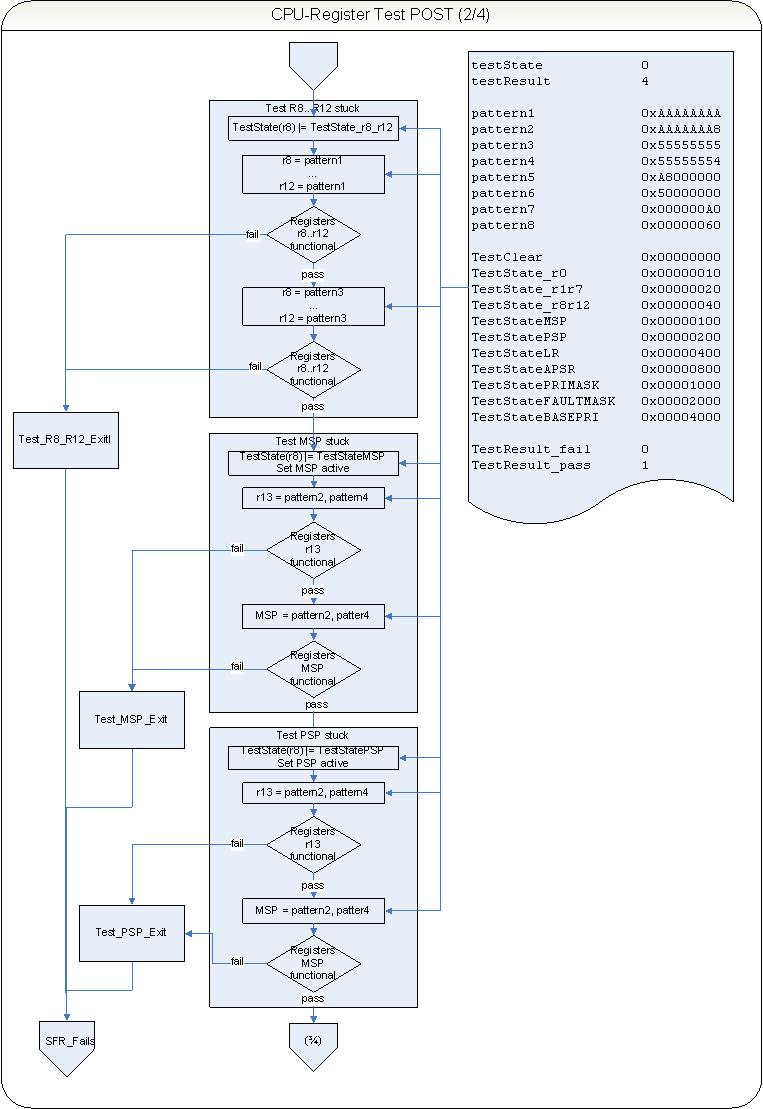


Image CPU Register Test High, MSP, PSP

Register LR test:  
The link register must be saved before testing and restored after the test .The link register is filled with pattern1 and with pattern3 from the constant list only with mov commands. Each load action is verified by comparing the register content with the source register r0. Fail condition will branch to the label Test\_LR\_Exit, pass condition will continue with next test.

Register APSR test:  
The APSR can only be accessed with msr and mrs command indirect from other registers. APSR is filled with pattern5 and pattern6 from the constant list and read back to register r1 to be compared with. Fail condition will branch to the label Test\_APSR\_Exit, pass condition will continue with next test.

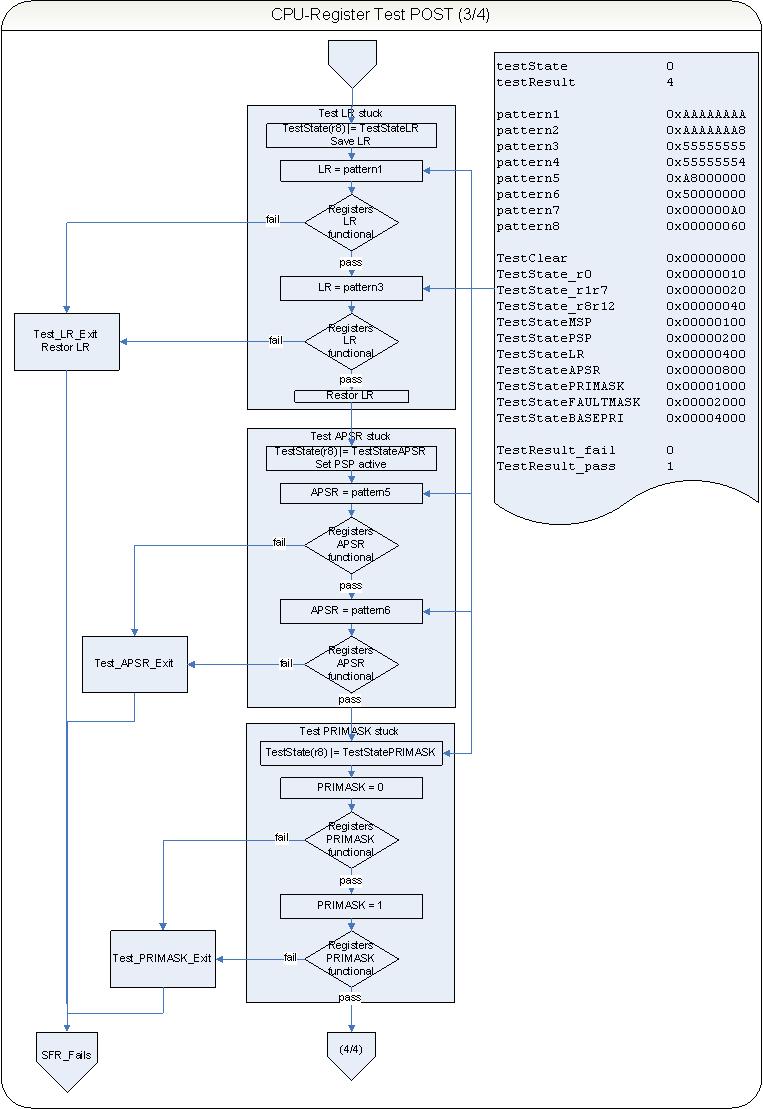
Register PRIMASK test:  
The PRIMASK contains one active bit which can be tested (Bit 0). This bit is set to zero and tested, set to 1 and verified again. Fail condition will branch to the label Test\_PRIMASK\_Exit, pass condition will continue with next test.

Image CPU Register Test LR, APSR, PRIMASK

Register FAULTMASK test:  
The FAULTMASK contains one active bit which can be tested (Bit 0). This bit is set to zero and tested, set to 1 and verified again. Fail condition will branch to the label Test\_FAULTMASK\_Exit, pass condition will continue with next test.

Register BASEPRI test:  
The Register BASEPRI is not fully implemented. Bit 4 to zero are always read as zero. BASEPRI is loaded indirect with msr command from r0 and read back with mrs to register r1. This is done with pattern7 and pattern8 from the constant table. Each read back is verified with the source in register r0. Fail condition will branch to the label Test\_BASEPRI\_Exit, pass condition will continue with next test.

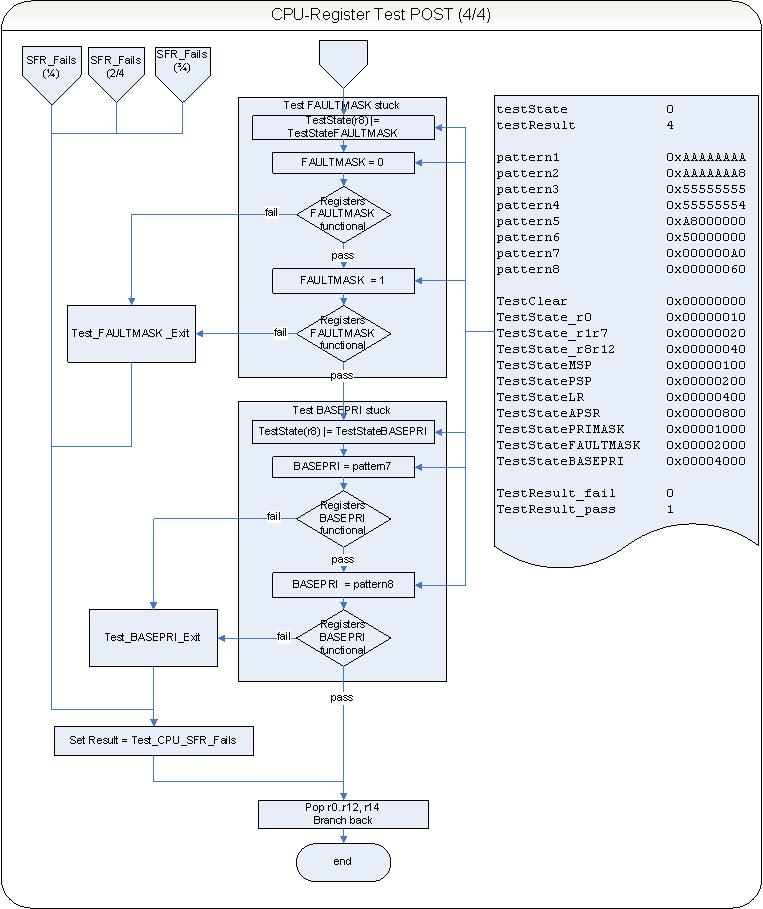
Exit and Error condition:  
If any error in the tests will exit the subroutines described above, the latest result status register bit can be identified to check the error condition. The result status register is addressed with an offset from register r8 relative to register r9. Register r9 points to the external struct IEC60335\_CPUregTestPOST. The calling function can evaluate the result accessing this variable. Fail condition will always set the test result to the constant TestResult\_fail from the constant table. The good condition is set in every individual subroutine test.

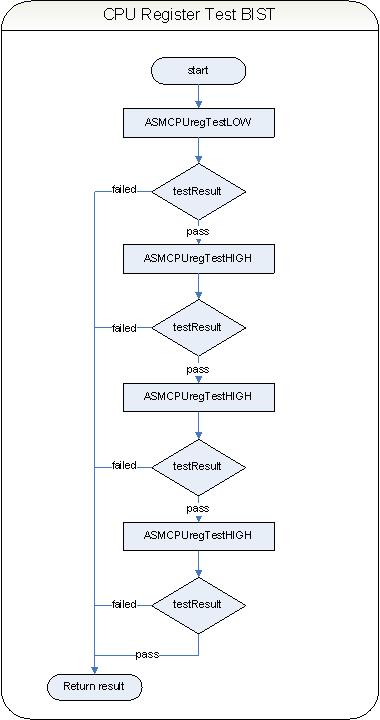
Image CPU Register Test FAULTMASK, BASEPRI

BIST test

|  |  |
| --- | --- |
| Functions specification |  |
| Function Name | IEC60335\_ClassB\_CPUregTest\_BIST |
| File (module) | iec60335\_class\_b\_cpureg\_test.c |
| Parameter (type) | None, - |
| Return value type | Result (testResult\_t) |
| Affected variables | IEC60335\_CPUregTestPOST (testState, testResult) |
| conditions | Compiler specific files with different mnemonics are called |

All CPU register BIST tests are to be executed in Privileged mode.

Image CPU register tests 1 to 4

The CPU register test for BIST is a consecutive call of all single subroutines testing the registers R0 to R7, R8 to R12, SP’s and the other special function registers of the CPU.

Fail condition in one subroutine will exit the function and always set the test result to the constant IEC60335\_testFailed from the constant definitions. The good condition is set in every individual subroutine test.

The global variable IEC60335\_CPUregTestBIST is set to adequate good result.

ASMCPUregTestLOW

|  |  |
| --- | --- |
| Functions specification |  |
| Function Name | ASMCPUregTestLOW |
| File (module) | iec60335\_class\_b\_cpureg\_test\_bist\_gcc.asm |
| Parameter (type) | None, - |
| Return value type | R0 |
| Affected variables | IEC60335\_CPUregTestBIST (testState, testResult) |
| conditions | Compiler specific files with different mnemonics are called |

Test description (BIST) from file iec60335\_class\_b\_cpureg\_test\_bist\_gcc.asm:

To trace the execution and fails each test adds a bit to the result status variable in register r8 to identify the last performed test or overall good condition. This is no mentioned in the following.

Starting the register test in the BIST test the registers are saved, especially the link register because the assembler coded tests are called from a C-file. Even is the stack area is uninitialized the push and pop calls are relevant for returning to the caller function. The result status register is cleared and set to the first (r0\_test) status.

Initially the register r0 is tested:

First a pattern (0xAA000000) is moved in the register. Second the register r0 is shifted left by 24 (3 bytes) which lead to the pattern residing in the lowest byte of the register. Third the pattern is compared with a literal 0xAA. If the content is correct the next step is performed.

A pattern (0x00AA0000) is moved in the register. Second the register r0 is shifted left by 16 (2 bytes) which lead to the pattern residing in the lowest byte of the register. Third the pattern is compared with a literal 0xAA. If the content is correct the next step is performed.

A pattern (0x0000AA00) is moved in the register. Second the register r0 is shifted left by 8 (1 byte) which lead to the pattern residing in the lowest byte of the register. Third the pattern is compared with a literal 0xAA. If the content is correct the next step is performed.

A pattern (0x000000AA) is moved in the register. Second the pattern is compared with a literal 0xAA. If the content is correct the next step is performed.

Fail condition will branch to the label Test\_r0\_Exit, pass condition will continue with next test r1 to r7.

Register r1 to r7 test:

All registers a loaded with the pattern1 from the constants list and each register is compared against a literal pattern1. This test is repeated with the pattern3 from the constants list. Fail condition will always set the test result to the constant TestResult\_fail from the constant table. The good condition is set to the constant TestResult\_pass.

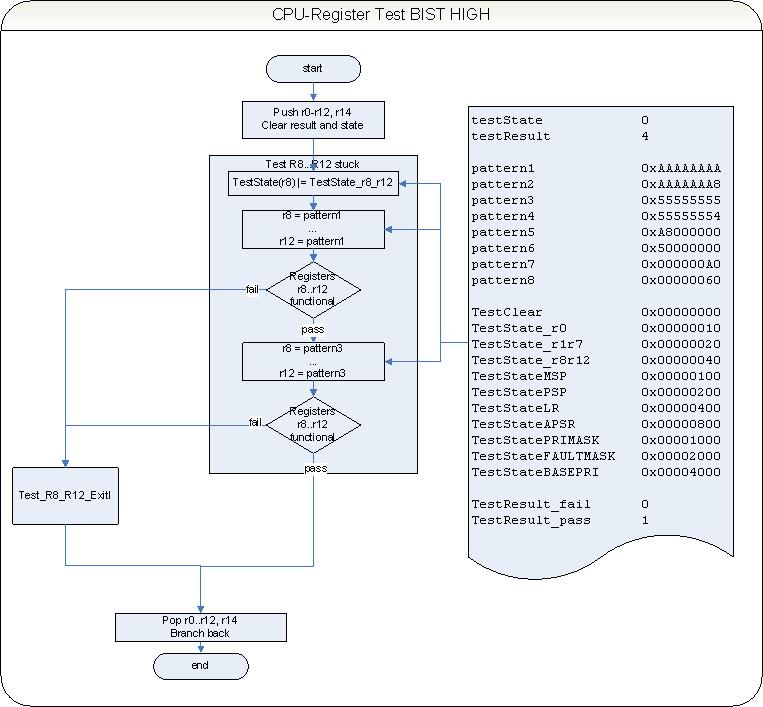


Image CPU Register Test HIGH

ASMCPUregTestHIGH

Register r8 to r12 test:  
All registers a loaded with the pattern1 from the constants list and each register is compared against a literal pattern1. This test is repeated with the pattern3 from the constants list. Fail condition will branch to the label Test\_r8\_r12\_Exit, pass condition will continue end this test.

Fail condition will always set the test result to the constant TestResult\_fail from the constant table. The good condition is set to the constant TestResult\_pass.

ASMCPUregTestSP

|  |  |
| --- | --- |
| Functions specification |  |
| Function Name | ASMCPUregTestSP |
| File (module) | iec60335\_class\_b\_cpureg\_test\_bist\_gcc.asm |
| Parameter (type) | None, - |
| Return value type | R0 |
| Affected variables | IEC60335\_CPUregTestBIST (testState, testResult) |
| conditions | Compiler specific files with different mnemonics are called |

Test description (BIST) from file iec60335\_class\_b\_cpureg\_test\_bist\_gcc.asm:

To trace the execution and fails each test adds a bit to the result status variable in register r5 to identify the last performed test or overall good condition. This is no mentioned in the following.

Register MSP test:  
The MSP register is accessible only in certain control states. So the MSP is set and verified to be the current and accessible stack pointer by clearing and reading the control register. In this case the register is also accessible via register r13 Because of the fact that MSP bit1 and 0 are always read as zero the MSP is filled with pattern2 and pattern4 from the constants list.   
Accessing the SMP register is possible with the commands mrs (move register from special register) and msr (move special register from register) or simple mov command. All combinations are used with the both pattern. After the register r0 with the constant pattern the MSP (r13) is loaded with a mov command. The content is checked by comparing the source r0 with r13. Writing the MSP with the msr command software is enabled to read the content back with the mrs command to register r1. An indirect comparison of the source register r0 and the MSP registers content in register r1 is made. Fail condition will branch to the label Test\_MSP\_Exit, pass condition will continue with next test.

Register PSP test:  
Accessing the PSP requires other state options set in the control register. Setting the control register to 0x02 enables the access. Because of the fact that PSP bit1 and 0 are always read as zero the PSP is filled with pattern2 and pattern4 from the constants list.   
Accessing the SMP register is possible with the commands mrs (move register from special register) and msr (move special register from register) or simple mov command. All combinations are used with the both pattern. After the register r0 with the constant pattern the MSP (r13) is loaded with a mov command. The content is checked by comparing the source r0 with r13. Writing the MSP with the msr command software is enabled to read the content back with the mrs command to register r1. An indirect comparison of the source register r0 and the MSP registers content in register r1 is made. Fail condition will branch to the label Test\_PSP\_Exit, pass condition will end this test.

Fail condition will always set the test result to the constant TestResult\_fail from the constant table. The good condition is set to the constant TestResult\_pass.

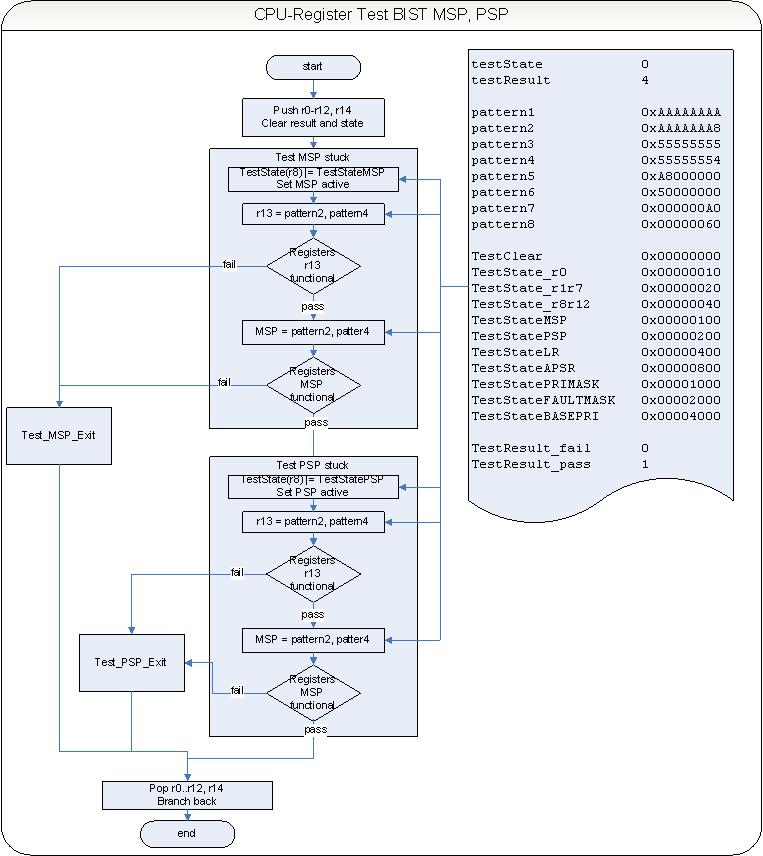


Image CPU Register Test SP

ASMCPUregTestSPEC

|  |  |
| --- | --- |
| Functions specification |  |
| Function Name | ASMCPUregTestSPEC |
| File (module) | iec60335\_class\_b\_cpureg\_test\_bist\_gcc.asm |
| Parameter (type) | None, - |
| Return value type | R0 |
| Affected variables | IEC60335\_CPUregTestBIST (testState, testResult) |
| conditions | Compiler specific files with different mnemonics are called |

Test description (BIST) from file iec60335\_class\_b\_cpureg\_test\_bist\_gcc.asm:

To trace the execution and fails each test adds a bit to the result status variable in register r8 to identify the last performed test or overall good condition. This is no mentioned in the following.

Register LR test:  
The link register must be saved before testing and restored after the test .The link register is filled with pattern1 and with pattern3 from the constant list only with mov commands. Each load action is verified by comparing the register content with the source register r0. Fail condition will branch to the label Test\_LR\_Exit, pass condition will continue with next test.

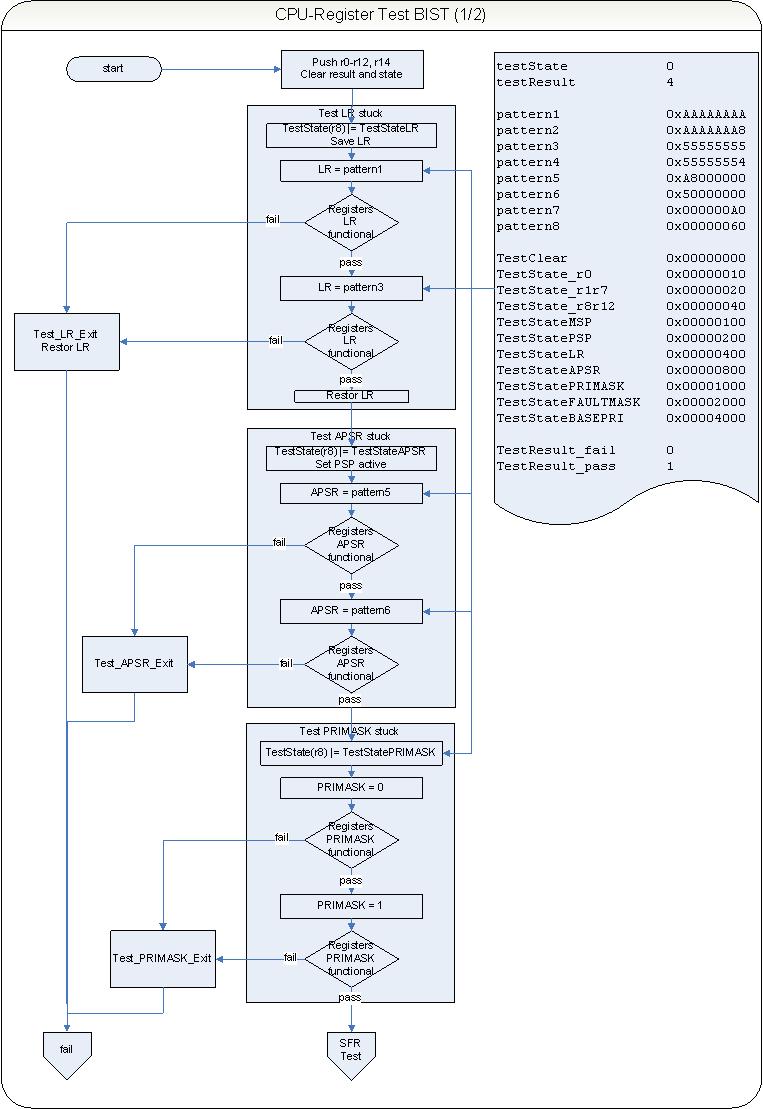
Register APSR test:  
The APSR can only be accessed with msr and mrs command indirect from other registers. APSR is filled with pattern5 and pattern6 from the constant list and read back to register r1 to be compared with. Fail condition will branch to the label Test\_APSR\_Exit, pass condition will continue with next test.

Register PRIMASK test:  
The PRIMASK contains one active bit which can be tested (Bit 0). This bit is set to zero and tested, set to 1 and verified again. Fail condition will branch to the label Test\_PRIMASK\_Exit, pass condition will continue with next test.

Register FAULTMASK test:  
The FAULTMASK contains one active bit which can be tested (Bit 0). This bit is set to zero and tested, set to 1 and verified again. Fail condition will branch to the label Test\_FAULTMASK\_Exit, pass condition will continue with next test.

Register BASEPRI test:  
The Register BASEPRI is not fully implemented. Bit 4 to zero are always read as zero. BASEPRI is loaded indirect with msr command from r0 and read back with mrs to register r1. This is done with pattern7 and pattern8 from the constant table. Each read back is verified with the source in register r0. Fail condition will branch to the label Test\_BASEPRI\_Exit, pass condition will end this test.

Exit and Error condition:  
If any error in the tests will exit the subroutines described above, the latest result status register bit can be identified to check the error condition. The result status register is addressed with an offset from register r8 relative to register r9. Register r9 points to the external struct IEC60335\_CPUregTestPOST. The calling function can evaluate the result accessing this variable. Fail condition will always set the test result to the constant TestResult\_fail from the constant table. The good condition is set in every individual subroutine test.



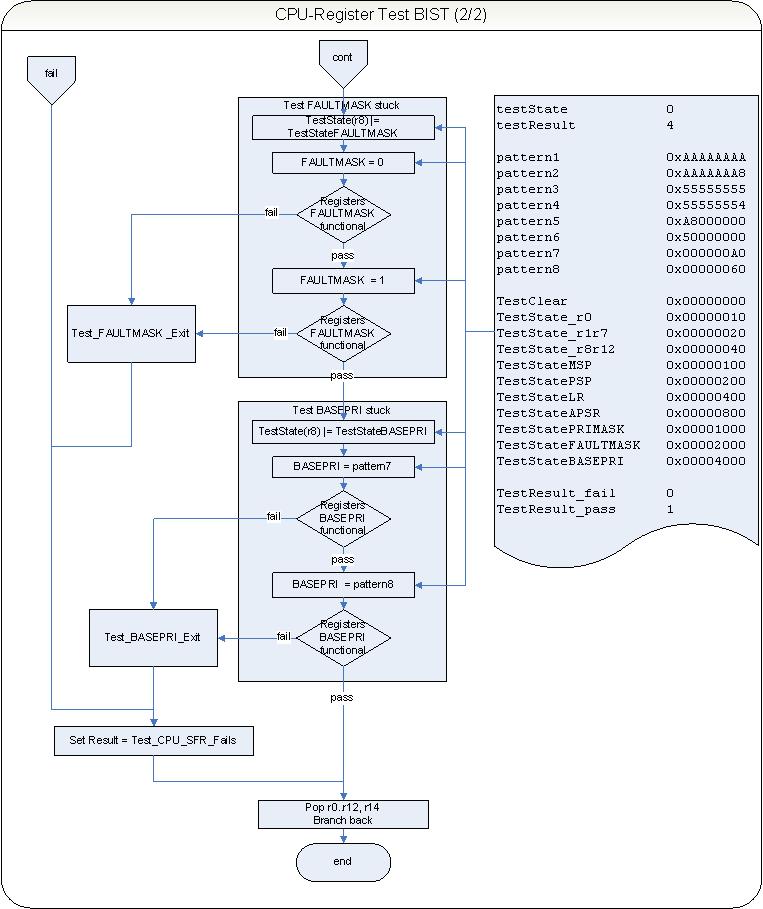


Image CPU Register Test SFRs

### Program Counter Test ()

This test checks the Program counter function.

To check it’s variability against register stuck the PC is initialized with several addresses from a Flash based array. Each address represents a small separate sub function which performs more or less useless code to prevent optimization. Each sub function is located to a separate Flash section to force the compiler to branch from the main caller function. After loading the PC with the address the sub routine is returning a return value from an also Flash based array. 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.

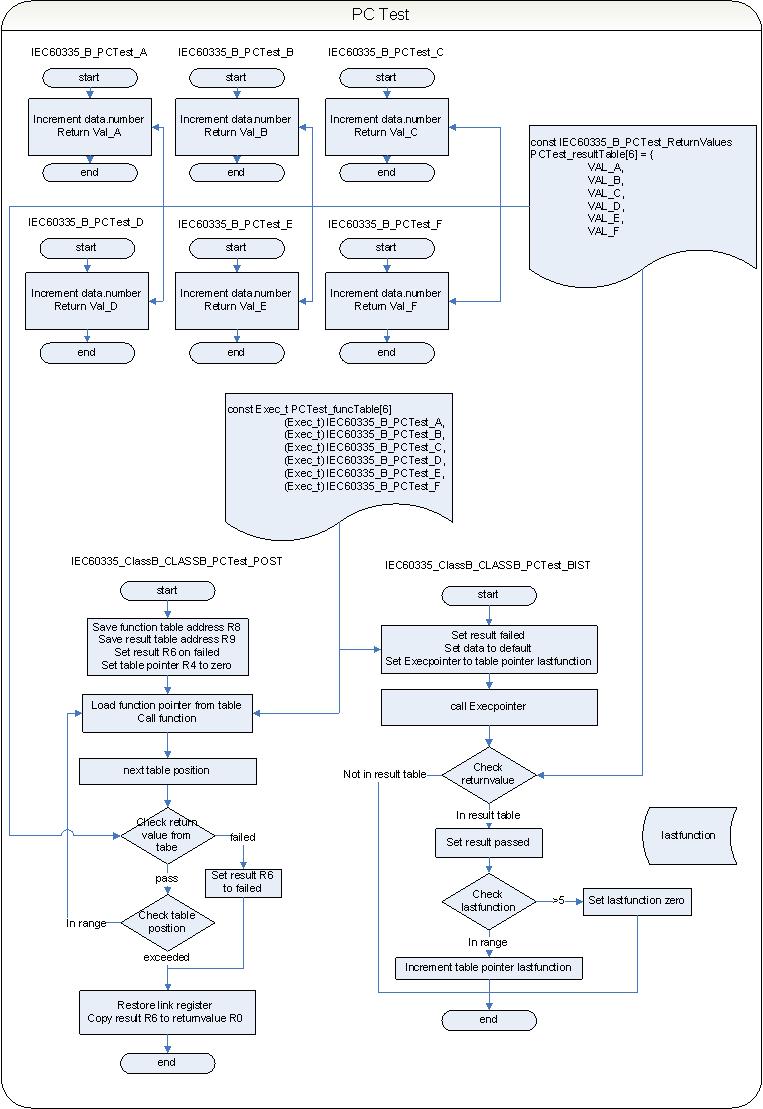


Image PC test POST and BIST

The POST function is inline assembler implemented because of the missing c-runtime environment. All sub routines are called in a loop. The BIST function calls only one routine from the list and changes an internal pointer to the next element for the following next call. So all sub functions are called after several times.

|  |  |
| --- | --- |
| Functions specification |  |
| Function Name | IEC60335\_ClassB\_CLASSB\_PCTest\_POST |
| File (module) | iec60335\_class\_b\_progamcounter\_test.c |
| Parameter, (type) | fTable (Exec\_t), rTable (IEC60335\_B\_PCTest\_ReturnValues) |
| Return value type | testResult\_t, - |
| Affected variables | none |
| conditions | Compiler specific files with different mnemonics |

Test description (POST) (from GCC implementation):

In a first step the input variables are checked to be not zero and saved to higher register not affected in the following program. R8 and R9 are holding the pointer to the constant tables of the sub-routines and the correct return values which are generated by the sub functions.

The loop counter R4 is set to zero and the register R5 holds the maximum address offset to the function table’s start.

The link register is saved to guarantee the return to the calling function without initialized C-environment

The first pointer from the subroutines table address is loaded to register R0 and the actual offset (initially it is zero) is added. The content of the table element is loaded to register R0 and a branch to this address is made.

Every function of the table is performing a little calculation to prevent from optimization. Also all subroutines are explicitly located in the Flash so the execution must be a branch. Each function returns a defined value which is in an ordered list referenced by the pointer rTable in register R9.

The value of the offset is incremented to access the next 32 bit addressed pointers

The return value is checked against the direct table access with the previously used offset (register R4) and the register R9. Fail condition will branch to the label “err” with the register R6 loaded with the value zero.

The value of the pointers offset (R4) is checked if reaching a size of 24. This means the maximum offset is already served.

For all good conditions the function is left with a “1” in the register R6 which is copied to R0 at the end of the function.

Finally the LinkRegister is restored.

|  |  |
| --- | --- |
| Functions specification |  |
| Function Name | IEC60335\_ClassB\_CLASSB\_PCTest\_BIST |
| File (module) | iec60335\_class\_b\_progamcounter\_test.c |
| Parameter, (type) | none |
| Return value type | testResult\_t, - |
| Affected variables | none |
| conditions | Compiler specific files with different mnemonics |

Test description (BIST):

The static local variable lastfunction is initialized with zero only with BSS section initialization. Afterwards it is simple variable which keeps the value even if the function is exited.

Data is prefilled with invalid return value and any number, test result is set to default with IEC60335\_testFailed.

The next function address is loaded from the function table.

The function is executed and the return value is compared with the according value of the result table.

If this test passes the result (return value) is set to IEC60335\_testPassed and the number of lastfunction is checked for exceeding the number of entries on the function table

Fail condition will always set the test result to IEC60335\_testFailed. The good condition is set in every individual subroutine test call.

### Interrupt Handling and Execution Test ()

The test for interrupt operation is strictly related to the application. In this test, the library delivers a mechanism to enable the testing of the functionality in an abstract way with or without activating the vector defined by the user application.

The test requires a initialization of the interrupt test structures and the tested interrupt number.

Defined struct for reference configuration:

typedef struct

{

UINT32 Active; /\*!< flag register indicating active on test \*/

UINT32 Mode; /\*!< flag register indicating original vector wanted \*/

} IEC60335\_IRQ\_Test\_t;

Defined struct for reference data:

typedef struct

{

UINT32 EffectiveCount; /\*!< counter Variable\*/

UINT32 MinThres; /\*!< minimum threshold target of interrupt occurrences\*/

UINT32 MaxThres; /\*!< maximum threshold target of interrupt occurrences\*/

} IRQTestData\_t;

|  |  |
| --- | --- |
| Functions specification |  |
| Function Name | IEC60335\_ClassB\_InitInterruptTest |
| File (module) | iec60335\_class\_b\_interrupt\_test.c |
| Parameter, type | IRQn\_Type IRQn,  INT8 Mode,  IRQTestData\_t \*CountSetup |
| Return value, type | None, - |
| Affected variables | Interrupt Vector Table |
| conditions | none |

The function builds a RAM allocated copy of the FLASH vector table if called the first time in an application. It also replaces the tested vector by a so called “replacement vector” which allows tracing the calls. In another step the user defined original vector can be called if the initialization is configuring this.

In a first step the function checks if the tested interrupt is in a valid range. If not the interrupt test function is deactivated and the flash-vector table is reactivated. If the range matches test vectors, the function checks the IEC60335\_IRQ\_Test vector to be valid. If not the function must first build a copy of the vector table in the RAM and set the IEC60335\_IRQ\_Test to a valid RAM location. In a next step the function checks the active state flag and toggles this. A first call will activate a second call will deactivate the test. Activation leads to a replacement of the user vector with the vector of the replacement vector handler. If active state also the mode flag is set according the input parameter.

Deactivation of the interrupt tests will reset the active vector setup of the NVIC and clear all flags in the configuration. Also the IEC60335\_IRQ\_Test vector is set to zero stating non-initialized state.

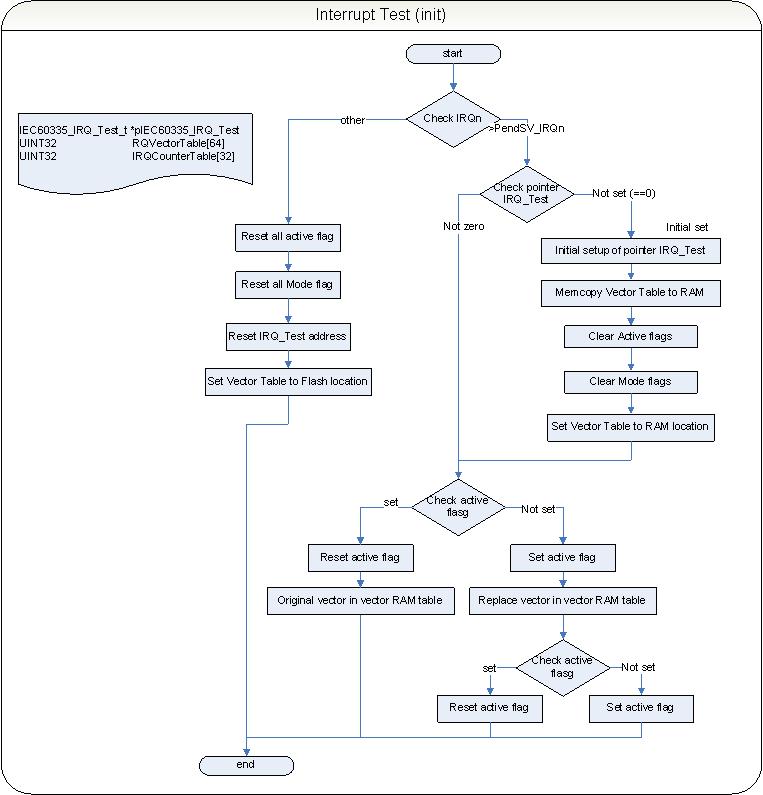


Image InterruptInit

|  |  |
| --- | --- |
| Functions specification |  |
| Function Name | \_\_call\_Vect |
| File (module) | iec60335\_class\_b\_interrupt\_test.c |
| Parameter, type | INT32 Vect |
| Return value, type | None, - |
| Affected variables | None, - |
| conditions | Compiler specific files with different inline mnemonics |

This function allows the test structure “IEC60335\_IRQReplacementHandler” to call the original vector defined by the FLASH vector table. If the routine IEC60335\_ClassB\_InitInterruptTest is called with a Mode parameter “CallIRQHandler” the function is called from the IEC60335\_IRQReplacementHandler function.

According to the compiler used the mnemonics can vary.

The function calculates the address of the flash based vector table and its original vector according to the input parameter “vect” and branches to this vector. For this call stack operations are necessary.

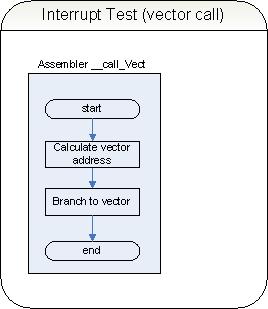


Image routine to call original vector

|  |  |
| --- | --- |
| Functions specification |  |
| Function Name | IEC60335\_IRQReplacementHandler |
| File (module) | iec60335\_class\_b\_interrupt\_test.c |
| Parameter, type | None, - |
| Return value, type | None, - |
| Affected variables | None, - |
| conditions | None, - |

The IRQReplacementVector routine offers the user a single global instance to call instead of the tested vector. To use this routine a previous initialization with the function IEC60335\_ClassB\_InitInterruptTest is necessary. The IRQReplacementVector has two tasks:

* Logging the occurrence of the tested vector with an individual counter
* If decided: call the original vector defined in the vector table

To reach these goals the routine is called by the NVIC on an occurrence of the interrupt which is tested. Because there is only one instance of this replacement the original vector is determined and an individual counter variable is modified by incrementing the variable.

Then the routine checks if the user configures the interrupt mode as active so the original vector routine is called with the \_\_call\_Vect routine.

This allows the user to simulate an interrupt occurrence without calling the vector routine. If the original vector routine is called the CPU modes and NVIC settings are not influenced and the routine is called in interrupt runtime environment.

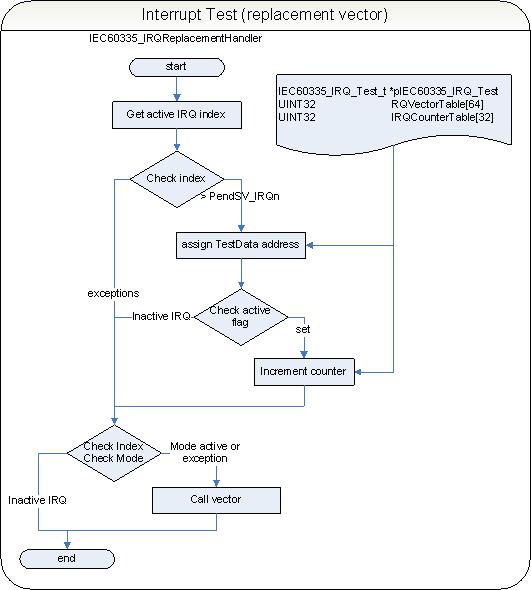
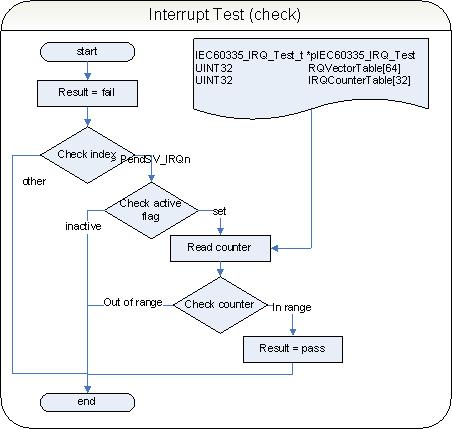


Image Replacement Vector

|  |  |
| --- | --- |
| Functions specification |  |
| Function Name | IEC60335\_ClassB\_InterruptCheck |
| File (module) | iec60335\_class\_b\_interrupt\_test.c |
| Parameter, type | IRQn\_Type IRQn, |
| Return value, type | testResult\_t |
| Affected variables | Interrupt Vector Table |
| conditions | none |

This routine offers a simple function to validate the number of occurrences of a tested interrupt.

Calling the function it will check if the specified interrupt vector is in an active test configuration. If so the assigned counter is read and tested for the initialized boundaries. If this test passes the function returns the result set to IEC60335\_testPassed. All other cases will return the result IEC60335\_testFailed. To use the function a call of the function IEC60335\_ClassB\_InitInterruptTest is necessary to activate the specified interrupt.



### Clock System Test ()

This test is intended to check the CPU and peripherals clock sources and frequency.

To achieve this, the test uses a second independent clock source to compare the tested timer occurrences in a tolerance aware way. The function is designed to be configured by compiler switches defined in the file iec60335\_class\_b.h. The options are USE\_SYSTICK, USE\_TIMER0 and USE\_TIMER1. The RTC is used as reference clock. All the peripheral timers are running with different clock sources than the RTC.

|  |  |
| --- | --- |
| Functions specification |  |
| Function Name | IEC60335\_ClassB\_initClockTest |
| File (module) | iec60335\_class\_b\_timer\_rtc\_test.c |
| Parameter, type | uint32\_t ratio, uint32\_t tolerance |
| Return value, type | None, - |
| Affected variables | None |
| conditions | Timer to test is specified in iec60335\_class\_b.h |

Clock system check enables the system to compare a specified timer with a defined period of the RTC. Because of the fact, that the RTC is on another clock domain than the timer the relative occurrences of timer interrupts during a specified real time is an indication for proper and correct working timer units.

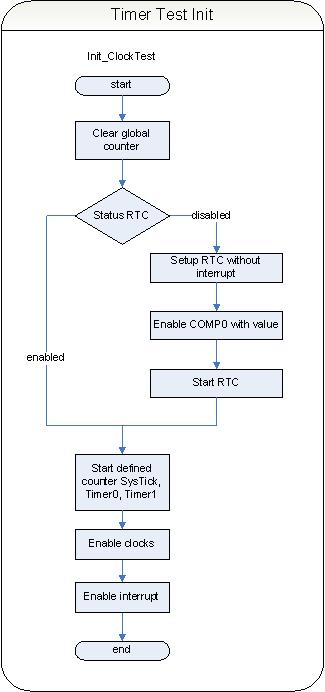


Image timer test initialization

The Timer test initialization starts the RTC module if not running already and the tested peripheral timer module. An individual variable is preset with ratio and tolerance values. The ration defines the number of clock interrupts expected during the measurement period. The tolerance defines the maximal allowed tolerance to state the timer running correct in addition to the ratio value.

|  |  |
| --- | --- |
| Functions specification |  |
| Function Name | IEC60335\_ClassB\_Clocktest\_TimerHandler |
| File (module) | iec60335\_class\_b\_timer\_rtc\_test.c |
| Parameter, type | None, - |
| Return value, type | None, - |
| Affected variables | ClockTest |
| conditions | Timer to test is specified in iec60335\_class\_b.h |

The ClockTest handler enables the tested timer to generate an interrupt counted in a global variable. Each time the interrupt occurs the counter is incremented.

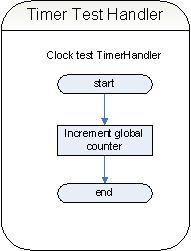


Image TimerTest Handler

To run the test the handler must be specified by the initialization routine as interrupt target for the timer in test. Which timer is in test is specified in the file iec60335\_class\_b.h by defines. The interrupt system is used within this test.

|  |  |
| --- | --- |
| Functions specification |  |
| Function Name | IEC60335\_ClassB\_Clocktest\_PollHandler |
| File (module) | iec60335\_class\_b\_timer\_rtc\_test.c |
| Parameter, type | None, - |
| Return value, type | testResult\_t, - |
| Affected variables | ClockTest |
| conditions | Timer to test is specified in iec60335\_class\_b.h |

The timer test poll handler is a frequently called control instance for the timer test. The function must be called periodically. The function checks if the timer interrupts are triggered, if the RTC interrupt is triggered and calculates the period in which the timer interrupts occur within the ratio defined during the initialization process. The result is stored in a global variable for later evaluation.

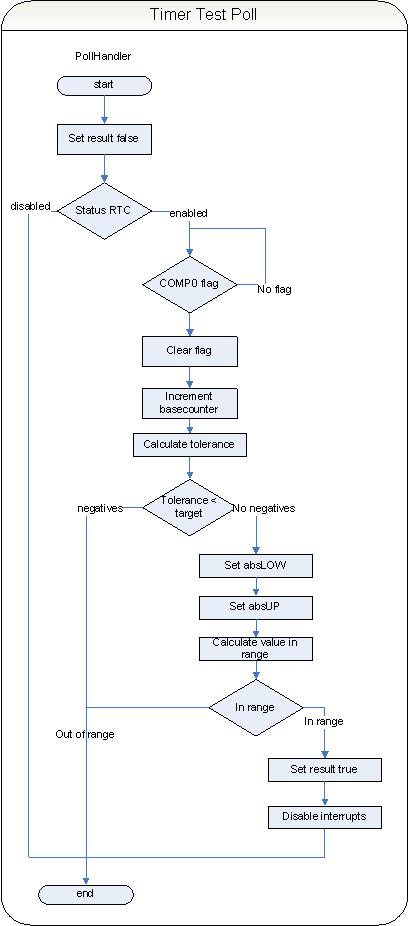


Image Timer Poll Handler

|  |  |
| --- | --- |
| Functions specification |  |
| Function Name | IEC60335\_ClassB\_Clocktest\_RTCHandler |
| File (module) | iec60335\_class\_b\_timer\_rtc\_test.c |
| Parameter, type | None, - |
| Return value, type | None, -- |
| Affected variables | ClockTest |
| conditions | Timer to test is specified in iec60335\_class\_b.h |

Generic RTC handler which is used to generate the basic counting system of the timer–RTC test. The function is set as interrupt handler for RTC by the initialization routine.

### Watchdog Timer Test (3.1)

This test is intended to check the Watchdog and its clock sources the interrupt generation and the reset function.

The watchdog test can only be used as a POST test in a very first order. It produces a hardware reset and evaluates the reset cause. For this behavior it requires an initial hardware or external reset as a starting condition

|  |  |
| --- | --- |
| Functions specification |  |
| Function Name | IEC60335\_ClassB\_initWDT |
| File (module) | iec60335\_class\_b\_wdt\_test.c |
| Parameter, type | None, - |
| Return value, type | testResult\_t |
| Affected variables | RMU->RSTCAUSE, WDOG->CTRL |
| conditions | POST test only! |

Calling the function from post() in the first entry it is assumed that a hardware or external reset has occurred as last reset cause. In a very first step the core-lock condition is checked. If lock is detected the function returns an error result because there is no way to have a sufficient test function in this case.

As second stage the function checks the reset cause for external or power on reset. If so the reset cause information is cleared and the WDOG module is enabled, started and locked. The result is set to IEC60335\_testInProgress.

Now it is expected that the watchdog timer exceeds its limits and produces a reset. In this case the next entry the function will run to the checking of the reset cause on the watchdog reset.

In this case the test has worked properly the result is set to IEC60335\_testPassed and the reset cause is cleared.

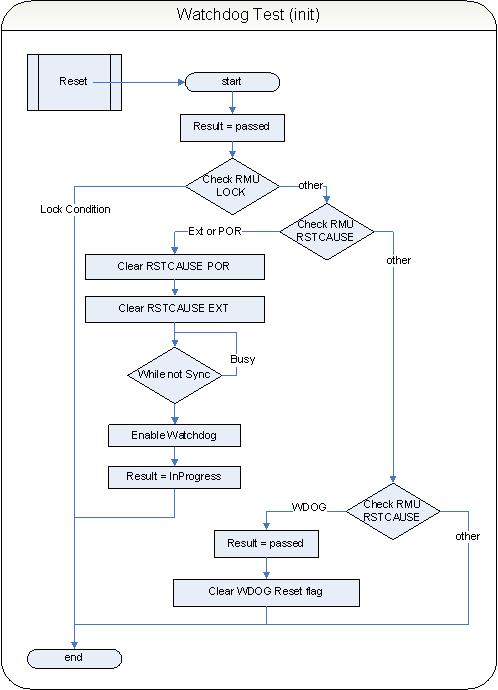


Image Watchdog Init

|  |  |
| --- | --- |
| Functions specification |  |
| Function Name | IEC60335\_ClassB\_Refresh\_WDT |
| File (module) | iec60335\_class\_b\_wdt\_test.c |
| Parameter, type | None, - |
| Return value, type | None, - |
| Affected variables | WDOG->CMD |
| conditions | None, - |

If using the watchdog in locked state the module can’t be deactivated again without hardware reset.

This function clears the watchdog counter and prevents the watchdog from overflow iIf the function is called periodically.

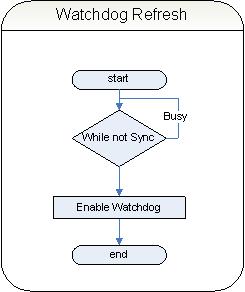


Image watchdog refresh

|  |  |
| --- | --- |
| Functions specification |  |
| Function Name | IEC60335\_ClassB\_Force\_WDT\_Reset |
| File (module) | iec60335\_class\_b\_wdt\_test.c |
| Parameter, type | None, - |
| Return value, type | None, - |
| Affected variables | ALL |
| conditions | Produces a hardware reset! |

This function is the recommend way to be called in the POST tests. Calling the function IEC60335\_ClassB\_initWDT until it returns a IEC60335\_testPassed will execute all stages of the IEC60335\_ClassB\_initWDT function without any additional code. Leaving of this function must be interpreted as IEC60335\_testPassed result.

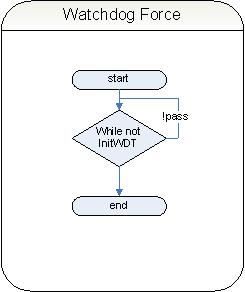


Image force reset

### Memory Tests and Safety Functions ()

Memory can be tested in POST condition and BIST condition. For POST testing of RAM the content of RAM must not be saved and restored. All of the RAM should be tested in one routine. Flash test requires defined independent information. This information is stored outside the code and rodata sections in the Flash memory. This partitioning requires some knowledge on linker processes. The corresponding data must be generated outside thee application.

Some derivatives may have MPU activated. In this case the memories are not transparent or accessible in ev3ry processor mode. The test requires unlimited access to all memory sections.

#### Invariable Memory (ROM / FLASH) ()

The Flash test module expects a defined CRC struct at a specified location outside the code section. The struct is defined as IEC60335\_Flash\_CRC\_REF from type FlashCRC\_t defined in the file “iec60335\_class\_b\_flash\_test.h”. the struct contains a pre-calculated 32 bit CRC32 value and other information on location, size and status of the CRC and the code

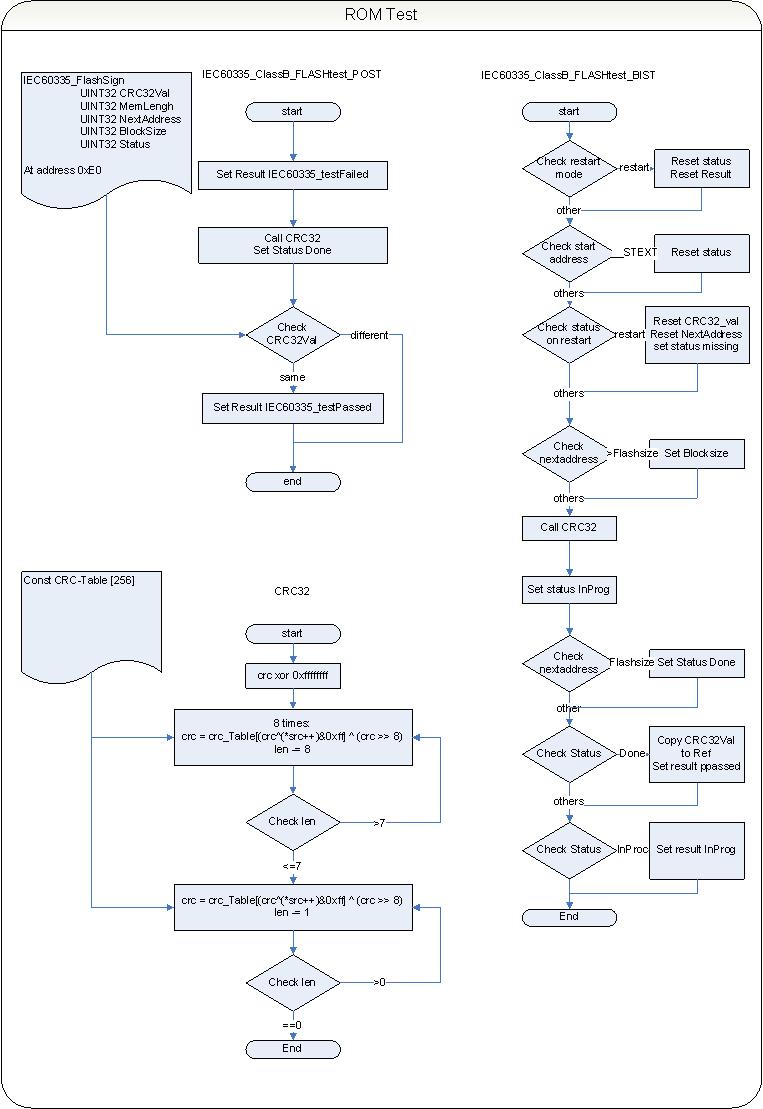


Image Flash Test and CRC

Defined struct for reference data:

typedef struct IEC60335\_FlashSign {

UINT32 CRC32Val; predefined CRC32 value

UINT32 MemLenght; length of Flash

UINT32 NextAddress; in case of variable use: a calculation reference

UINT32 BlockSize; block size in Flash

UINT32 Status; status of this struct

} FlashCRC\_t;

|  |  |
| --- | --- |
| Functions specification |  |
| Function Name | IEC60335\_ClassB\_FLASHtest\_POST |
| File (module) | iec60335\_class\_b\_flash\_test.c |
| Parameter, type | None, - |
| Return value, type | testResult\_t |
| Affected variables | None |
| Conditions | None |

Test description (POST):

The function calculates the CRC32 value from beginning of the code section until the end of the Flash device.

The result is compared with the predefined CRC32 value in the reference struct located at address 0xE0 in the Flash. This is an area outside the code section.

Mismatch will cause fail condition and will always set the test result to IEC60335\_testFailed.

If the value is initialized a POST test with complete check can be initialized and BIST check with smaller parts can be used at runtime.

|  |  |
| --- | --- |
| Functions specification |  |
| Function Name | IEC60335\_ClassB\_FLASHtest\_BIST |
| File (module) | iec60335\_class\_b\_flash\_test.c |
| Parameter, type | StartMode (UINT8) |
| Return value, type | testResult\_t |
| Affected variables | Static bistFlashCRC |
| Conditions | None |

Test description (BIST):

Initially the static variable bistFlashCRC is initialized with the bss section. It is not reinitialized in the following function calls.

The function checks a restart parameter which will reset the process parameter Status and Result.

The function checks nextaddress in boundary smaller than the valid start address of code than reser Status to restart.

The function checks the restart status condition and reinitializes the process parameter CRC32val (intermediate data) Nextaddress (pointer to the following sector) Blocksize and Status.

The function checks NextAddress in the Flash area then recalculate the block size

Calling the CRC32 function with parameter CRC32Val (so long calculated value), nextaddress and size.

Set the status inProgress

Checks end of flash then Status is Done

Compares the predefined CRC value with the calculated and set the result if matched

Mismatch will cause fail condition and will always set the test result to IEC60335\_testFailed. Pass condition will change the result to IEC60335\_testPassed still not ready will change the result to IEC60335\_testInProgress.

If the value is initialized a POST test with complete check can be initialized and BIST check with smaller parts can be used at runtime.

#### Flash Content Checksum Build

|  |  |
| --- | --- |
| Functions specification |  |
| Function Name | crc32 |
| File (module) | iec60335\_class\_b\_flash\_test.c |
| Parameter, type | crc (INT32), src (const UINT8 \*), len (UINT32) |
| Return value, type | UINT32 |
| Affected variables | None |
| conditions | None |

CRC32: Standard CRC32 checksum calculation from IEEE1684.

The checksum is expected to be part of the Flash content. The library provides a recalculation of the CRC.

#### Variable Memory (RAM) ()

|  |  |
| --- | --- |
| Functions specification |  |
| Function Name | IEC60335\_ClassB\_RAMtest\_POST |
| File (module) | iec60335\_class\_b\_ram\_test.c |
| Parameter, type | None, - |
| Return value, type | testResult\_t |
| Affected variables | None |
| conditions | All RAM is corrupted after this |

The RAM test is a marching algorithm with 9 elements:

For an Offset of 0, 1, 2 or 3 byte a loop is performed with 9 marching elements pointing to an address in the RAM area. Where the macro NOP\_R0 is writing a byte with content 0x00 is NOP\_R1 writing the inverted byte (0xFF) the macro NOP\_R0 is reading the byte back and compares it with 0x00 where NOP\_W1 compare with 0xFF. In fail condition the macros are jumping to the labels R0\_FAULT\_DETECTED or R1\_FAULT\_DETECTED.

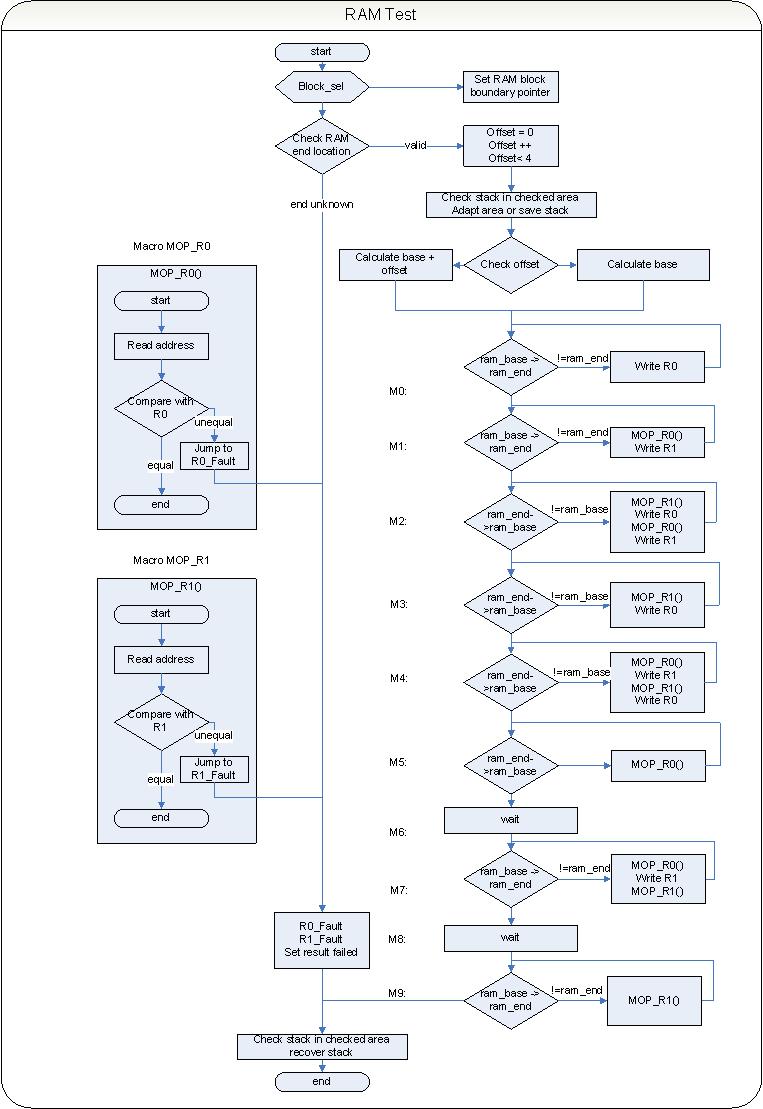


Image RAM test

March element M0:

Starting at RAM base it writes a 0x00 to every 32bit word with offset according to the loop status (offset).

March element M1:

Starting at RAM base it reads the expected 0x00 and swaps these bits on every 32 bit word with offset according to the loop status (offset).

March element M2:

Starting at RAM end it reads the expected 0xFF and swaps these bits on every 32 bit word with offset according to the loop status (offset). It is read immediately and changed back to 0xFF again.

March element M3:

Starting at RAM base it reads the expected 0xFF and swaps these bits on every 32 bit word with offset according to the loop status (offset).

March element M4:

Starting at RAM end it reads the expected 0x00 and swaps these bits on every 32 bit word with offset according to the loop status (offset). It is read immediately and changed back to 0x00 again.

March element M5:

Starting at RAM end it reads the expected 0x00.

March element M6:

No operation (waiting)

March element M7:

Starting at RAM end it reads the expected 0x00 and swaps these bits on every 32 bit word with offset according to the loop status (offset). It is read immediately and changed back to 0xFF again.

March element M8:

No operation (waiting)

March element M9:

Starting at RAM end it reads the expected 0xFF.

Failure in reading or writing will always leave the function setting test result to IEC60335\_testFailed. Good condition will pass IEC60335\_testPassed at the exit.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| March  element | March  direction | Offset = 0 | Offset = 1 | Offset = 2 | Offset = 3 |
| 0 write R0 | Up | 0xXXXXXX00 | 0xXXXX00XX | 0xXX00XXXX | 0x00XXXXXX |
| 1 read R0  1 write R1 | Up | 0xXXXXXX00  0xXXXXXX11 | 0xXXXX00XX  0xXXXX11XX | 0xXX00XXXX  0xXX11XXXX | 0x00XXXXXX  0x11XXXXXX |
| 2 read R1  2 write R0  2 read R0  2 write R1 | Down | 0xXXXXXX11  0xXXXXXX00  0xXXXXXX00  0xXXXXXX11 | 0xXXXX11XX  0xXXXX00XX  0xXXXX00XX  0xXXXX11XX | 0xXX11XXXX  0xXX00XXXX  0xXX00XXXX  0xXX11XXXX | 0x11XXXXXX  0x00XXXXXX  0x00XXXXXX  0x11XXXXXX |
| 3 read R1  3 write R0 | Down | 0xXXXXXX11  0xXXXXXX00 | 0xXXXX11XX  0xXXXX00XX | 0xXX11XXXX  0xXX00XXXX | 0x11XXXXXX  0x00XXXXXX |
| 4 read R0  4 write R1  4 read R1  4 write R0 | Down | 0xXXXXXX00  0xXXXXXX11  0xXXXXXX11  0xXXXXXX00 | 0xXXXX00XX  0xXXXX11XX  0xXXXX11XX  0xXXXX00XX | 0xXX00XXXX  0xXX11XXXX  0xXX11XXXX  0xXX00XXXX | 0x00XXXXXX  0x11XXXXXX  0x11XXXXXX  0x00XXXXXX |
| 5read R0 | Down | 0xXXXXXX00 | 0xXXXX00XX | 0xXX00XXXX | 0x00XXXXXX |
| 6 wait | - | - | - | - | - |
| 7 read R0  7 write R1  7 read R1 | Up | 0xXXXXXX00  0xXXXXXX11  0xXXXXXX11 | 0xXXXX00XX  0xXXXX11XX  0xXXXX11XX | 0xXX00XXXX  0xXX11XXXX  0xXX11XXXX | 0x00XXXXXX  0x11XXXXXX  0x11XXXXXX |
| 8 wait | - | - | - | - | - |
| 9 read R1 | down | 0xXXXXXX11 | 0xXXXX11XX | 0xXX11XXXX | 0x11XXXXXX |

Table marching algorithm

Stack is located according to the used compiler and linker. Corruption of this memory will lead to the fact that the function can’t be left and the previous calling function can’t be addressed again so the routine checks the location of the stack in the RAM and excludes this memory if the stack is located in the very end of the RAM or copies the content to a very low address with changing of the stack pointer to this new location. After finishing the test the stack is restored in this case and the stack pointer is set to its original value.

By this fact the RAM location where the Stack resides during the test is excluded from the test. To provide a full RAM check it is obvious to set the linker to a stack location somewhere in the RAM and checks the very low addresses o the RAM separately with the BIST function not accessing the stack location.

IEC60335\_ClassB\_RAMtest\_BIST!

|  |  |
| --- | --- |
| Functions specification |  |
| Function Name | IEC60335\_ClassB\_RAMtest\_BIST |
| File (module) | iec60335\_class\_b\_ram\_test.c |
| Parameter, type | UINT32 startAddr, UINT32 length- |
| Return value, type | testResult\_t |
| Affected variables | None |
| conditions | Affected RAM is restored after this test |

The RAM test is the same marching algorithm with 9 elements than POST test: As prologue the tested RAM area is saved to an array and restored after finishing the test. For this reason the size to check is limited to a buffer size defined in the header file.

The following checks are performed before testing the RAM

Check the tested RAM is in RAM area, if not exit with error

Check if the specified length is smaller than buffer size; if bigger set to buffer size

Check end address is in RAM area, if not exit with error

Check if the tested RAM is the same than the buffer with first and last element

If all conditions are valid the target memory content is saved to buffer and the RAM test is called with the parameter start address and length. After finishing the RAM content is restored.

Error conditions will always leave the function setting test result to IEC60335\_testFailed. Good condition will pass IEC60335\_testPassed at the exit.

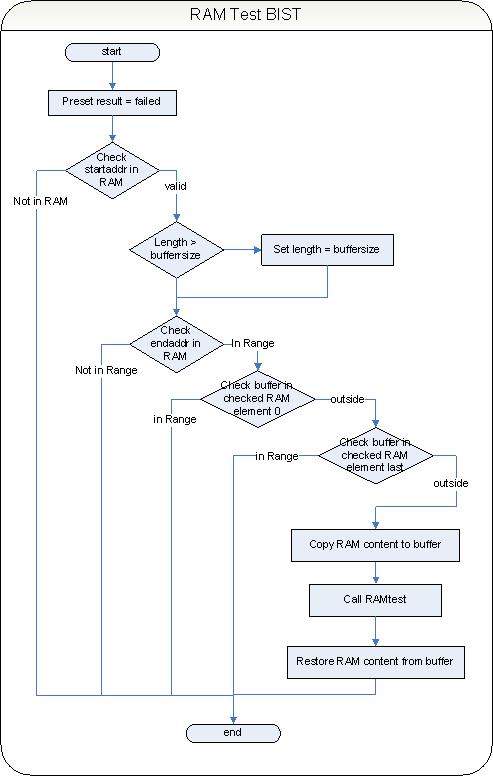


Image RMA Test BIST

#### Critical Data Storage ()

If there are stored a critical constant which are periodically used in critical calculations, then it is necessary to check this variable before every usage and ensure the validity of the data.

Theo enable the user in simple and reliable data handling the critical\_data header is realized.

The following types are available

|  |  |  |  |
| --- | --- | --- | --- |
| Type name | Usable type | Usable size [byte] | Memory usage [byte] |
| critical\_FLOAT64 | FLOAT64 | 8 | 16 |
| critical\_FLOAT32 | FLOAT32 | 4 | 8 |
| critical\_UINT64 | UINT64 | 8 | 16 |
| critical\_INT64 | INT64 | 4 | 8 |
| critical\_UINT32 | UINT32 | 4 | 8 |
| critical\_INT32 | INT32 | 4 | 8 |
| critical\_UINT16 | UINT16 | 2 | 4 |
| critical\_INT16 | INT16 | 2 | 4 |
| critical\_UINT8 | UINT8 | 1 | 1 |
| critical\_INT8 | INT8 | 1 | 1 |

|  |  |
| --- | --- |
| Functions specification |  |
| Macro Name | IEC60335\_ClassB\_CriticalDataInit |
| File (module) | iec60335\_class\_b\_critical\_data.h |
| Parameter, type | value |
| Return value, type | none |
| Affected variables | criticalVar |
| conditions | none |

Initialize a variable of a critical\_data\_type. The data is stored as bit inverted value and directly.

|  |  |
| --- | --- |
| Functions specification |  |
| Macro Name | IEC60335\_ClassB\_CriticalDataPush |
| File (module) | iec60335\_class\_b\_critical\_data.h |
| Parameter | criticalVar, value |
| Return value, type | None |
| Affected variables | criticalVar |
| conditions | Initialized Variable |

Rewrite the data of a critical\_data\_type. The macro must be used to keep data consistent.

|  |  |
| --- | --- |
| Functions specification |  |
| Macro Name | IEC60335\_ClassB\_CriticalDataPop |
| File (module) | iec60335\_class\_b\_critical\_data.h |
| Parameter | criticalVar |
| Return value, type | criticalVar |
| Affected variables | criticalVar |
| conditions | Initialized Variable |

Read a critical\_data\_type to use with simple data types

|  |  |
| --- | --- |
| Functions specification |  |
| Macro Name | IEC60335\_ClassB\_CriticalDataValidate |
| File (module) | iec60335\_class\_b\_critical\_data.h |
| Parameter, type | criticalVar |
| Return value, type | testResult\_t |
| Affected variables | criticalVar |
| conditions | Initialized Variable |

Check the validity of a critical data type.

Using the critical data types with the introduced macros will cause a double data holding. The data are stored first in bit-inverted form and then in direct form. If any critical type memorized variable cell is affected by other parts of an application, a critical data type variable is used without the macros in a direct way a change of the content can be easily detected by the validation macro. The application can react on the fact in a specified way.

### Test Execution Time

The described test routines have execution times which are limiting the number of executions in a runtime environment.

|  |  |  |  |
| --- | --- | --- | --- |
| Test Component | Routine (POST) | Routine (BIST) | Execution Time (cycles) |
| CPU Register Test | IEC60335\_ClassB\_CPUregTest\_POST | - | 618 |
|  | - | IEC60335\_ClassB\_CPUregTest\_BIST | 906 |
|  | - | ASMCPUregTestLOW | n/a |
|  | - | ASMCPUregTestHIGH | n/a |
|  | - | ASMCPUregTestSP | n/a |
|  | - | ASMCPUregTestSPEC | n/a |
| PC Test | IEC60335\_ClassB\_PC test\_POST | - | 444 |
|  | - | IEC60335\_ClassB\_PCTest\_BIST | 156 |
| RAM Test | IEC60335\_ClassB\_RAM test\_POST | - | 46809 |
|  | - | IEC60335\_ClassB\_RAMtest\_BIST(0x20000100, 0x10) | 3917 |
| Flash Test | IEC60335\_ClassB\_FLASH test\_POST | - | 168419 |
|  | - | IEC60335\_ClassB\_FLASHtest\_BIST | 32360 |
| Clock/RTC Test | - | IEC60335\_ClassB\_initClockTest | 332 |
|  | - | IEC60335\_ClassB\_Clocktest\_PollHandler | 272 |
|  | - | IEC60335\_ClassB\_Clocktest\_TimerHandler | 67 |
|  | - | IEC60335\_ClassB\_Clocktest\_RTCHandler | 92 |
| WDT Test | - | IEC60335\_ClassB\_initWDT | 253 + WDT timeout |
|  | - | IEC60335\_ClassB\_Refresh\_WDT | n/a |
|  | - | IEC60335\_ClassB\_ForceWDTreset | n/a |
| Interrupt Test | - | IEC60335\_ClassB\_InitInterruptTest | 8244 |
|  | - | IEC60335\_ClassB\_IRQReplacementHandler | 56 |
|  | - | IEC60335\_ClassB\_InterruptCheck | 72 |

Table Module Execution Time

Execution Time is defined in core clock cycles. Total time can be calculated by multiplying the cycles with the clock phase time (e.g. 1/10MHz = 100ns, IEC60335\_ClassB\_CPUregTest\_BIST = 906\*100ns = 90,6µs). The data are measured empirical.

* 1. Library Usage Description

This section defines rules for usage of the library. It is intended to show cases how the library is to use, and more important, how not to use. The reference for this chapter is the implemented test project available for IAR, GCC and ARM tool chain. The project details references to the GCC version.

### CPU Register Test Usage

Register tests are divided into POST and BIST tests. POST tests will destroy the memory content, so the POST test must be used before initializing the RAM (BSS section). The example application combines all the POST tests into one function called IEC60335\_CassB\_POST() calling subsequent all the POST tests before setup the runtime environment.

### Program Counter Test Usage

Program Counter tests are divided into POST and BIST tests. POST tests will run all possible Program Counter tests in one series where the BIST test calls separate subtest functions. The intention is to run the POST test as a monolithic test where the BIST test must be interruptable and short runtime for possible OS usage. The example project calls the POST test into one function called IEC60335\_CassB\_POST().

### Interrupt Operation Test Usage

Testing an Interrupt may vary in the test configuration during runtime. Each interrupt with an interrupt ID higher than -2 (PendSV\_IRQn) can be tested. Initialization of the test requires some information: which interrupt will be tested and which mode is intended. The mode parameter specifies if the original vector handler is executed (1) or the execution is skipped. This allows the user to run an interrupt with all the consequences without interference on the program flow. E.g. if protocol bytes on the USART are counted to serve any frame information, the test can be configured top skip the handler so that there is no effect on the frame counter. The interrupt tests are toggling the test feature. So disabling the test requires the same call of initialization for a second time. Unconfigure the complete mechanism a call with a unsupported vector Id is required. Only one interrupt can be tested at one time. This means for the fact that there is only one ClockTest structure the configuration and results are limited. To extend this the ClockTest have to be implemented as array for individual interrupt indices.

To check the occurrence of an interrupt there is the function IEC60335\_ClassB\_InterruptCheck available. The function checks the number of occurrences in relation to the desired number. The function is non-blocking so the call can be several times in the program loop.

### Clock System Test Usage

The Clock System test is intended to run selected clocks in relation to the RTC. Making sure that the RTC is driven by an external oscillator on the RTC-clock domain (LFACLKRTC) each other timer running on a different clock domain can be tested.

Running test can be implemented periodically or initial before using the tested timer.

Initializing the clock system test will enable the RTC and configure the timer specified in the file iec60335\_class\_b.h. Both timers are configured using interrupts. The initial call defines the factor and tolerance in percent how the timer produces interrupts relative to the RTC. For evaluation there are two functions available: one for polling usage e.g. in a main routine and one for interrupt usage called from an TRC interrupt. The main difference is, that the polling version is blocking until a RTC interrupt occurs, while the RTC interrupt routine is non-blocking. The interrupt versions result can be evaluated in the ClockTest struct evaluating the result member.

### Memory Test and Safety Functions Usage

#### Critical Data Storage

The library offers a set of storage macros which allows the user to store, read and verify data integrity in the volatile memory. Use the macro IEC60335\_ClassB\_CriticalDataInit to set an initial value to a data type e.g. critical\_uint32\_t. In this case the value is written into two memory locations with bit-inverted values. To update the data the macro IEC60335\_ClassB\_CriticalDataPush is necessary. The value is also written in the two locations, one inverted. Checking the data can be done with the macro IEC60335\_ClassB\_CriticalDataValidate. Writing on only one of the words of a critical data type will cause a validation error with this macro. IEC60335\_ClassB\_CriticalDataPop will read the data without checking for integrity.

#### Make an initialized or uninitialized instance

The difference of initialize a critical data type to simply storing the data to the critical data types data member is that the ones-complement is not initialized.

Initialize with the IEC60335\_ClassB\_CriticalDataInit macro will enable data verification from the beginning of the existence of the critical data. The uninitialized critical type will lead to a fail condition if the verification is made.

### FLASH Test Usage

The FLASH test checks the non volatile memory section of the application. This can include code and data section. By linker script definition a section CRCs is defined at the address 0x0000.00ec. This location is behind the interrupt vector table of the Cortex M3 and before the text section which holds the program code. In this CRCs area a struct called ENTRY\_FLASH\_CRC is located. It hold a CRCvalue, information on the FlashSize and where the CRC checked area begins. Also a flag is available if the CRC is valid or if there is any exceptional condition like during development cycles. This is defined in the file iec60335\_class\_b\_flash\_test.h and must be adjusted before running the test.

The POST function is already calling the IEC60335\_ClassB\_FLASHtest\_POST function which runs a CRC recalculation on the complete Flash except the non text sections. The calculated CRC is compared to the former mentioned fix CRC value. If the numbers are not matching the result should lead to a shut down because the code was inconsistent. Because of the long term (more than 128 ms) to run this test on the complete Flash content the IEC60335\_ClassB\_FLASHtest\_BIST function checks the Flash consecutive in blocks of 1024 byte defined in the file iec60335\_class\_b\_flash\_test.h. This takes much less time and the result will be available automatically after the necessary number of calls. This method needs a startup condition which is given by a single call with parameter 5 (FLASH\_CRC\_Restart) before entering the main loop. According to the Flash size the IEC60335\_ClassB\_FLASHtest\_BIST function must be called several times until the result changes from FLASH\_CRC\_InProg to the calculated result.

#### Functions

IEC60335\_ClassB\_FLASHtest\_POST called by the POST test is checking the complete Flash which takes some times.

IEC60335\_ClassB\_FLASHtest\_BIST is called from the main routine with an initial value FLASH\_CRC\_Restart. Calling the function from the main loop with parameter 0 will cause the function to recalculate consequent the CRC in blocks of 1024 byte until the end of the CRC defined are is reached and compares this to the predefined CRC value from the sections CRCs.

#### Type definitions

To give the routine fixed data for evaluation the section CRCs contains the CRC data predefined as CRC32 value, memory length, next address, block size and status.

#define ENTRY\_FLASH\_CRC.

Hint: to calculate the CRC32 value external several tools are available from the internet. Another method is the inbound calculation. With a debugger system run the code until the CRC value is recalculated and checkout the temporary value. This value can be representing the correct predefinition.

### RAM Test Usage

Like the Flash test the RAM test is divided into two parts: IEC60335\_ClassB\_RAMtest\_POST to run a complete test before system start and IEC60335\_ClassB\_RAMtest\_BIST which is intended to check smaller portions of RAM with the ability to save the content before and restore the content after the test. In this case there is no data corruption. The RAM memory is checked for stuck and crosstalk effects. There is no way to check the content against some fixed values. To provide data integrity the critical data storage macros can be used.

The POST test of the RAM is destructive. This means no valid data is left in the RAM after the tests is finished. So the test must be used before the data section is copied to the RAM and before the zero filling is done (bss).

The BIST test of the RAM delivers the result immediately after the test is finished. Parameters are the start address and the length of the portion which will be tested. The function is not able to check the RAM part holding the stack. So the application will automatically test the RAM parts outside, even is split into two part before and after, with a maximum size defined by the IEC60335\_RAM\_buffersize in the file iec60335\_class\_b\_ram\_test.h.

### Watchdog Timer Test Usage

The watchdog timer test cannot be simulated. By this fact the test is forced to run the watchdog timer and force a timeout which will reset the controller. This fact leads to the inherent precondition that there is no code before this test. User must be sure that there is no data which are missing after this test. In the example application the watch dog timer (WDT) test is the first test which is called by the post function before the chip is initialized and RAM is preset.

The function IEC60335\_ClassB\_Force\_WDT\_Reset calls the function IEC60335\_ClassB\_initWDT which is checking the last reset condition, initializing the WDTT hardware if a system reset was detected and check for a watchdog reset to state the test was passed or failed. To enter the function the second time the watch dog must have made a reset on the core. Since there is no possibility to simulate this behavior a reset is the only way to provide this functional test.

During development cycles this function should be disabled (file iec60335\_class\_b.h) because a reset will cause connection loss to any debugging devices.

* 1. The Development Environment

As development hardware, the EFM32 Gecko Development Kit was used. The CPU module hosts a EFM32G890F128 device and contains a JTAG interface.

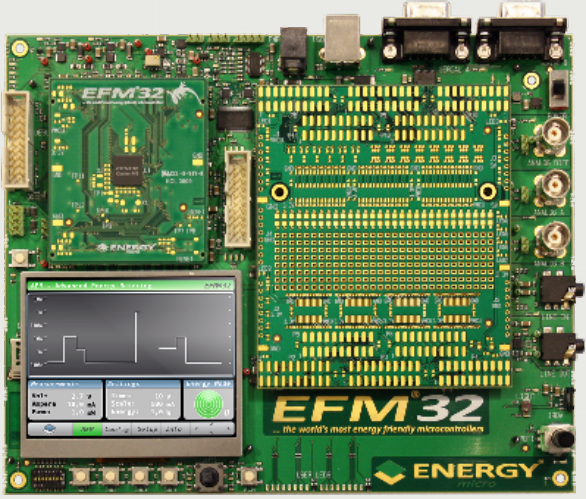


Image EFM32 Hardware

The example project was build with Hitex HiTOP 5.4 CTX Environment IDE and Debugger with the GNU ARM compiler for Cortex. Debug wiggler is the Hitex Cortino. The project for IAR was build with IAR Kickstart edition and The Keil project with µVision 4.

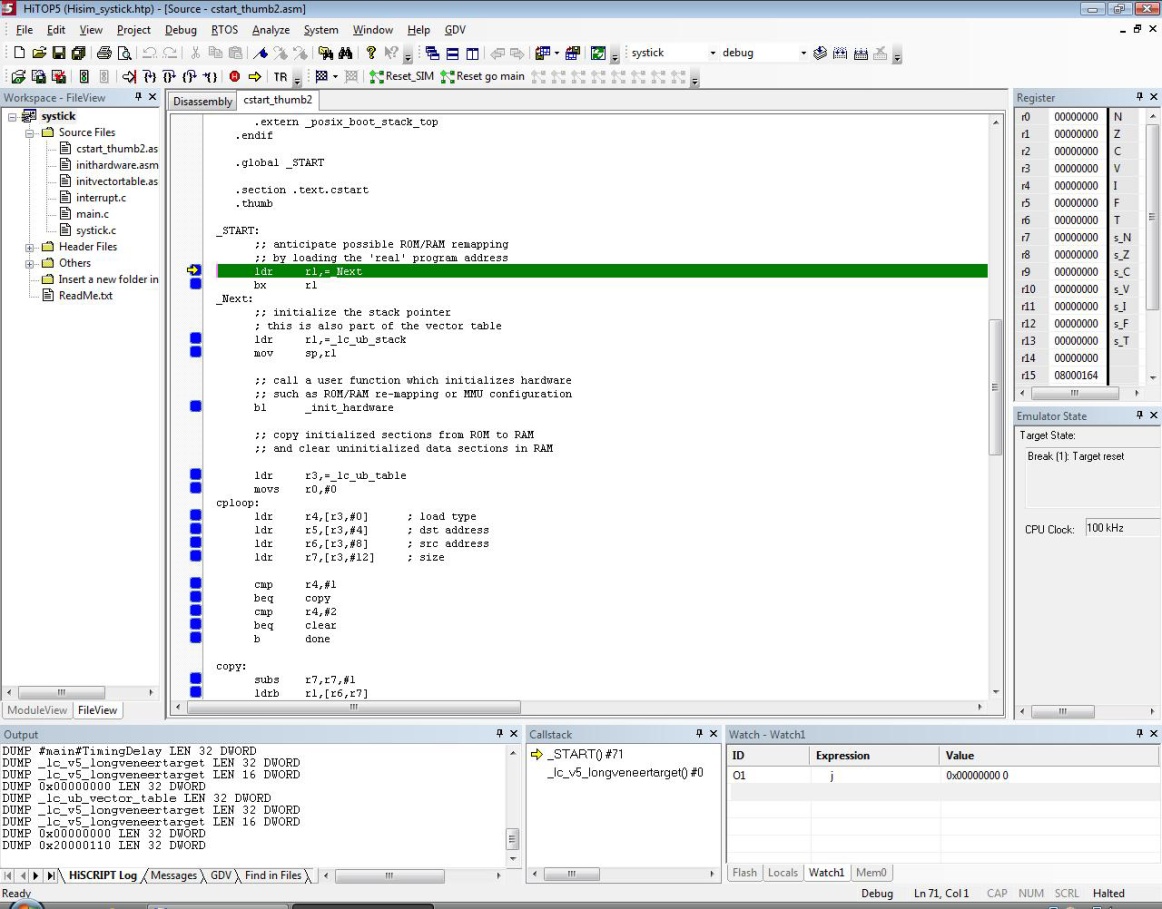


Image HiTOP

* 1. List of tools

|  |  |  |
| --- | --- | --- |
| **ToolName** | **Description** | **Version** |
| EFM32 Gecko Development Kit | Hardware platform including JTAG | PCB3300 Rev A02 |
| HiTOP54-CTX | IDE and Debugger Software | 5.4.0074 |
| Tessy | Unit test environment | 2.6.29 |
| GNU C compiler for ARM Cortex | C compiler | 4.5.0 |
| Keil µVision IDE and compiler | IDE, compiler and Debugger Software | 4.14.10 |
| IAR Kickstart Edition | IDE, compiler and Debugger Software | 6.10 |
| DAC | Static Code Analysis | 4.2.088 |

Compiler specific setting:

The compilers add their intrinsic definitions to the command lines:

GCC \_\_GNUC\_\_

Keil µvision and Realview compiler: \_\_CC\_ARM

IAR compiler: \_\_ICCARM\_\_

Some code is very compiler specific and the tool chains will disable the competitive code sections.

Some code is nearly fixed to some hardware related issues like RAM and Flash size. According to the used PCB and Controller the example is configured to the described hardware.

# Sources

IEC60355 library content:

.\iec60335\_classb\inc\

iec60335\_class\_b.h main include file, configuration

iec60335\_class\_b\_cpureg\_test.h proto and defines

iec60335\_class\_b\_critical\_data.h critical data macros

iec60335\_class\_b\_def.h proto and defines

iec60335\_class\_b\_flash\_test.h proto and defines

iec60335\_class\_b\_interrupt\_test.h proto and defines

iec60335\_class\_b\_post.h proto and defines

iec60335\_class\_b\_programcounter\_test.h proto and defines

iec60335\_class\_b\_ram\_test.h proto and defines

iec60335\_class\_b\_timer\_rtc\_test.h proto and defines

iec60335\_class\_b\_typedef.h proto and defines

iec60335\_class\_b\_wdt\_test.h proto and defines

.\iec60335\_classb\src\

iec60335\_class\_b\_cpureg\_test.c entry for CPU register tests

iec60335\_class\_b\_critical\_data.c critical data protos

iec60335\_class\_b\_flash\_test.c Flash test

iec60335\_class\_b\_interrupt\_test.c Interrupt test

iec60335\_class\_b\_post.c POST test collection

iec60335\_class\_b\_programcounter\_test.c PC test

iec60335\_class\_b\_ram\_test.c RAM test

iec60335\_class\_b\_timer\_rtc\_test.c Timer test

iec60335\_class\_b\_wdt\_test.c WDOG test

iec60335\_class\_b\_cpureg\_test\_bist\_gcc.asm GCC Register BIST test

iec60335\_class\_b\_cpureg\_test\_post\_gcc.asm GCC Register POST test

iec60335\_class\_b\_cpureg\_test\_bist\_arm.s Keil Register BIST test

iec60335\_class\_b\_cpureg\_test\_bist\_iar.s IAR Register BIST test

iec60335\_class\_b\_cpureg\_test\_post\_arm.s Keil Register POST test

iec60335\_class\_b\_cpureg\_test\_post\_iar.s IAR Register POST test

documentation:

.\doc\

Doxyfile\_ie60335\_lib doxygen file

EM\_ClassB\_Library.pdf this document

Go\_index.html link to html docu

.graph\ graphics

.html\ doxygen output

test example:

.\boards\examples\iec60335\_classb\_example\

arm\ project files for Keil µVision

doc\ documentation (doxygen and html)

hitop-gcc\ project files for HiTOP5.4 and scripts

iar\ project files for IAR Kickstart

source\ source files

visualC\ project files for MS Visual C++

# Bibliography

1. **ARM ltd.** *DDI 0337G.* Cortex-M3 r2p0 Technical Reference Manual (2008) : s.n.

2. **EnergyMicro.** *EFM32G Reference Manual.*

3. **CEI/IEC 60335-1.** *Household and similar electrical appliances Safety.* :2001+A1:2004+A2:2006 : s.n.

4. **IEC 60730-1.** *Automatic electrical controls for household and similar use.* :1999+A1:2003+A2:2007(E) : s.n.

5. **EnergyMicro.** *EFM32G Cortex-M3 Reference Manual.*

6. **ARM ltd.** ARM website. [Online] http://www.arm.com.

7. **EnergyMicro.** *EFM32G Datasheet.*

8. —. *EFM32G errata.*

9. **Sadasivan, Shyam.** An Introduction to the ARM Cortex-M3 Processor. [book auth.] White paper. October 2006.

10. **Yiu, Joseph.** The definitive guide to the ARM Cortex-M3.

# List of Images

[Image 1 Cortex™-M3 core architecture 6](#_Toc299609303)

[Image 2 Structural overview 8](#_Toc299609304)

[Image 3 EFM32G Cortex™-M3 Implementation 10](#_Toc299609305)

[Image 4 Processor Registers 11](#_Toc299609306)

[Image 5 Interrupt Operation 12](#_Toc299609307)

[Image 6 High Frequency Domain 13](#_Toc299609308)

[Image 7 Low Frequency Domain 14](#_Toc299609309)

[Image 8 Cortex-M3 Memory Model 15](#_Toc299609310)

[Image 9 Detailed Memory Map 16](#_Toc299609311)

[Image 10 Bus Matrix 18](#_Toc299609312)

[Image 11 External Bus Interface 19](#_Toc299609313)

[Image 12 logical Library Representation 22](#_Toc299609314)

[Image 13 V-model 23](#_Toc299609315)

[Image 14 Memory Mapping 26](#_Toc299609316)

[Image 15 CPU-Register Test 29](file:///C:\Work\Projekte\5-EM-Classb\Lieferung\Distribution%20package\documentation\EM_ClassB_Library_User_Documentation.docx#_Toc299609317)

[Image 16 CPU Register Test LOW 31](file:///C:\Work\Projekte\5-EM-Classb\Lieferung\Distribution%20package\documentation\EM_ClassB_Library_User_Documentation.docx#_Toc299609318)

[Image 17 CPU Register Test High, MSP, PSP 32](file:///C:\Work\Projekte\5-EM-Classb\Lieferung\Distribution%20package\documentation\EM_ClassB_Library_User_Documentation.docx#_Toc299609319)

[Image 18 CPU Register Test LR, APSR, PRIMASK 33](file:///C:\Work\Projekte\5-EM-Classb\Lieferung\Distribution%20package\documentation\EM_ClassB_Library_User_Documentation.docx#_Toc299609320)

[Image 19 CPU Register Test FAULTMASK, BASEPRI 34](file:///C:\Work\Projekte\5-EM-Classb\Lieferung\Distribution%20package\documentation\EM_ClassB_Library_User_Documentation.docx#_Toc299609321)

[Image 20 CPU register tests 1 to 4 35](file:///C:\Work\Projekte\5-EM-Classb\Lieferung\Distribution%20package\documentation\EM_ClassB_Library_User_Documentation.docx#_Toc299609322)

[Image 21 CPU Register Test HIGH 37](#_Toc299609323)

[Image 22 CPU Register Test SP 39](#_Toc299609324)

[Image 23 CPU Register Test SFRs 42](#_Toc299609325)

[Image 24 PC test POST and BIST 43](#_Toc299609326)

[Image 25 InterruptInit 47](#_Toc299609327)

[Image 26 routine to call original vector 48](#_Toc299609328)

[Image 27 Replacement Vector 49](#_Toc299609329)

[Image 28 timer test initialization 51](#_Toc299609330)

[Image 29 TimerTest Handler 52](#_Toc299609331)

[Image 30 Timer Poll Handler 53](#_Toc299609332)

[Image 31 Watchdog Init 54](#_Toc299609333)

[Image 32 watchdog refresh 55](#_Toc299609334)

[Image 33 force reset 56](#_Toc299609335)

[Image 34 Flash Test and CRC 57](#_Toc299609336)

[Image 35 RAM test 60](#_Toc299609337)

[Image 36 RMA Test BIST 63](#_Toc299609338)

[Image 38 EFM32 Hardware 71](#_Toc299609339)

[Image 39 HiTOP 72](#_Toc299609340)