AN4187
Application note

Using the CRC peripheral on STM32 microcontrollers

Introduction

The cyclic redundancy check (CRC) is a powerful and easily implemented technique to obtain data reliability. It is used to detect errors in data transmission or storage integrity check, without making corrections when errors are detected.

The diagnostic coverage satisfies the requirements of basic safety standards, hence is used for flash memory content integrity self-test check in ST firmware certified for compliance with IEC 60335-1 and IEC 607030-1 standards (“Class B” requirements). For more information, refer to AN3307 (available on *www.st.com),* and the associated firmware packages dedicated to different family products.

Check the necessary CRC settings in compiler manuals when CRC checksum information must be placed directly into user code by linker (mostly in the format of a CRC descriptor data table).

The CRC is based on polynomial arithmetic, computing the remainder of the division of a polynomial in the Galois field with two elements by another. The remainder is the checksum, the dividend is the data, and the divisor is the generator polynomial.

This document describes the features of the CRC peripheral embedded in STM32 32-bit Arm® Cortex® MCUs, and the steps required to configure it:

- *Section 1* describes the STM32 CRC implementation algorithm and its benefits
- *Section 2* describes the use of the DMA as CRC data transfer controller
- *Section 3* describes the migration of the CRC through STM32 devices

Two examples are described, with the measurement of execution time:

- CRC_usage example: how to configure the CRC using the CPU as data transfer controller.
- CRC_DMA example: how to use the DMA as CRC data transfer controller.

Contents

List of tables

List of figures

1 CRC peripheral overview

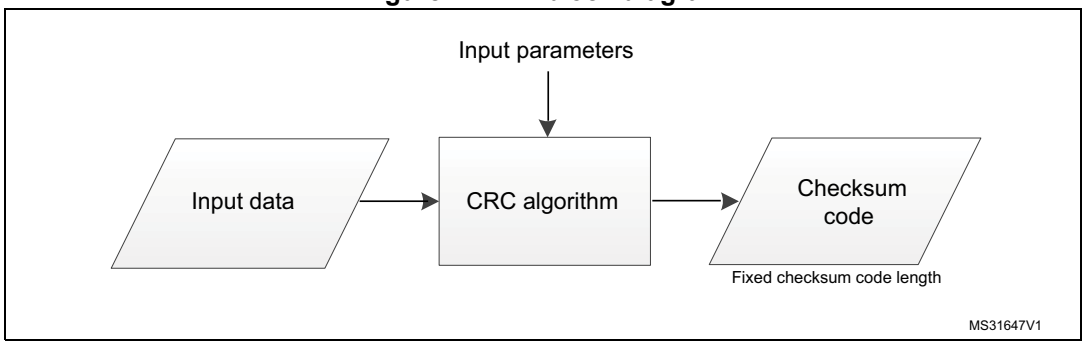
The CRC peripheral embedded in all STM32 microcontrollers (based on Arm®(a) Cortex® cores) is used to provide a CRC checksum code of supported data types.

This section first introduces the software CRC algorithm, then the STM32 CRC hardware accelerator, highlighting its benefits.

1.1 CRC computing algorithm

As shown in *Figure 1*, the CRC algorithm has a data input and generates a fixed checksum code length, depending on the input parameters.

Figure 1. CRC block diagram



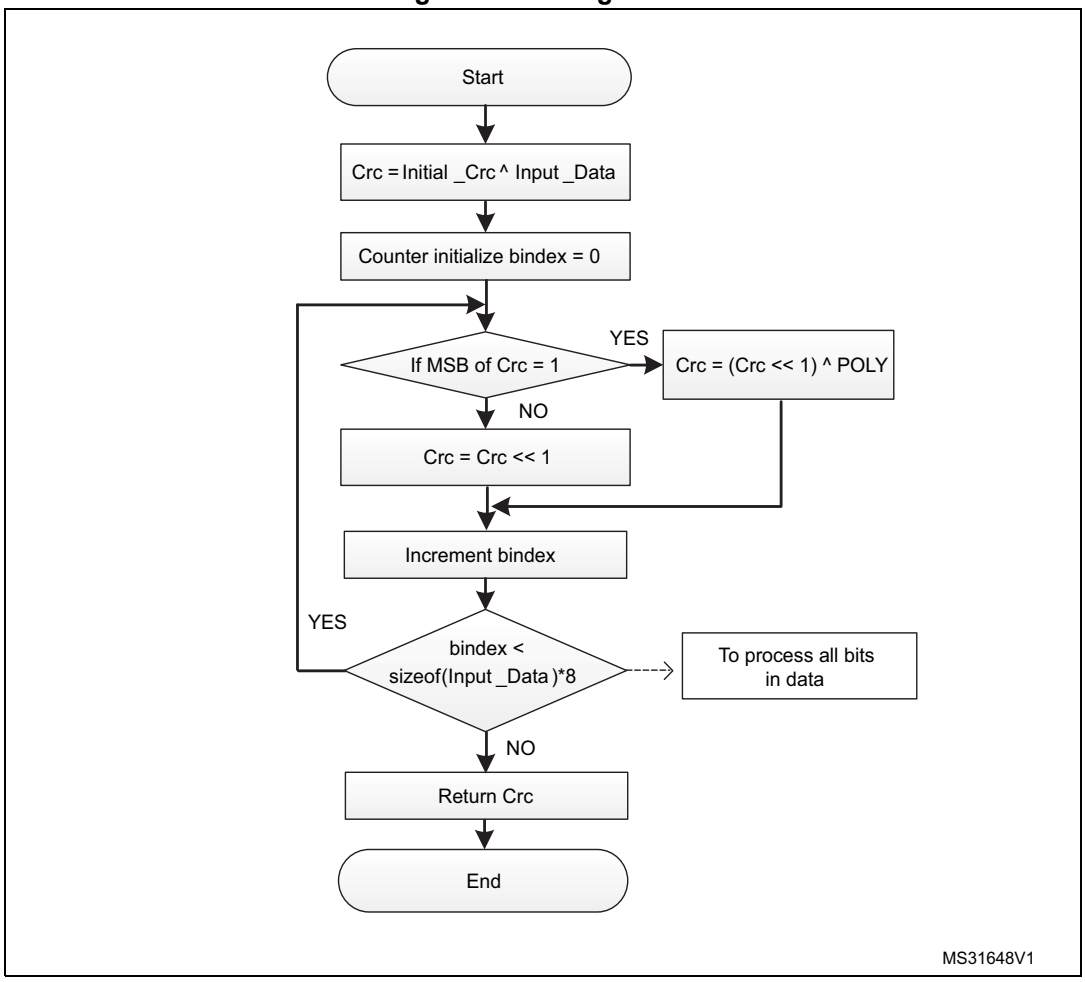
One CRC algorithm is the polynomial division with bitwise message XOR-ing technique, the most suitable for hardware or low-level implementation. The input parameters are:

- the dividend: also called input data, abbreviated to *"Input_Data"*
- the divisor: also called generator polynomial, abbreviated to "*POLY*"
- an initial CRC value: abbreviated to "*Initial_Crc*"

Figure 2 shows the CRC algorithm.

a. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

Figure 2. CRC algorithm



At startup, the algorithm sets the CRC to the *Initial_Crc* XOR with the *Input_Data.*

Once CRC MSB is equal to 1, the algorithm shifts CRC one bit to the left, and XORs it with the POLY. Otherwise, it only shifts CRC one bit to the left.

Figure 3 shows the step-by-step algorithm execution for the following conditions:

– Input_Data = 0xC1
– POLY = 0xCB
– Initial_Crc = 0xFF

Figure 3. Step-by-step CRC computing example

bindex	Execution step	Binary format	Hex
	Crc = *Initial_Crc* ^ *Input_Data*	1 1 1 1 1 1 1 1 (*Initial_Crc*) ^ 1 1 0 0 0 0 0 1 (*Input_Data*)	0xFF 0xC1
0	Crc << 1	[0] 0 1 1 1 1 1 0	0x3E
1	Crc << 1	[0] 1 1 1 1 1 0 0	0x7C
2	Crc << 1	[1] 1 1 1 1 0 0 0	0xF8
	Crc = Crc ^ POLY	[1] 1 1 1 0 0 0 0 ^ 1 1 0 0 1 0 1 1 (*POLY*)	0xF0 0xCB
3	Crc << 1	[0] 0 1 1 1 0 1 1	0x3B
4	Crc << 1	0 1 1 1 0 1 1 0	0x76
5	Crc << 1	[1] 1 1 0 1 1 0 0	0xEC
	Crc = Crc ^ POLY	[1] 1 0 1 1 0 0 0 ^ 1 1 0 0 1 0 1 1 (*POLY*)	0xD8 0xCB
6	Crc << 1	[0] 0 0 1 0 0 1 1	0x13
7	Crc << 1	[0] 0 1 0 0 1 1 0	0x26
	Crc (Returned value)	0 1 0 0 1 1 0 0	0x4C

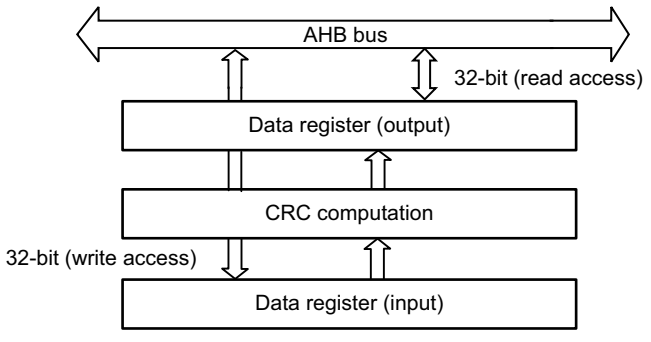
1. The returned CRC value (0x4C) has been verified by the CRC peripheral on STM32F37x products, with the above mentioned configurations.
2. This routine has been implemented under CrcSoftwareFunc function in the CRC_usage example.

1.2 CRC peripheral configuration

All STM32 devices implement a CRC peripheral, as described in *Section 1.1*. The CRC calculation unit has a single 32-bit read/write data register (CRC_DR), used to input new data (write access), and hold the result of the previous CRC calculation (read access).

Each write operation to the data register creates a combination of the previous CRC value (stored in CRC_DR) with the new one.

Figure 4. Block diagram of the CRC calculation unit



To compute a CRC of the supported data, go through the following steps:

1. Enable the CRC peripheral clock via the RCC peripheral.
2. Set the CRC data register to the initial CRC value by configuring the initial CRC value register (CRC_INIT). In the more recent STM32 Series, it is possible to chain a CRC calculation based on the previous CRC calculation as initial value. In this case, the CRC_IDR register (not affected by the reset bit in CRC_CR) can be used. In HAL, this is implemented by HAL_CRC_Calculate.
3. Set the I/O reverse bit order through the REV_IN[1:0] and REV_OUT bits, respectively, in the CRC control register (CRC_CR).
4. Set the polynomial size and coefficients through the POLYSIZE[1:0] bits in CRC control register (CRC_CR) and CRC polynomial register (CRC_POL), respectively.
5. Reset the CRC peripheral through the Reset bit in the CRC control register (CRC_CR).
6. Set the data to the CRC data register.
7. Read the content of the CRC data register.
8. Disable the CRC peripheral clock.

In the firmware package, the CRC_usage example runs the CRC checksum code computing an array data (*DataBuffer*) of 256 supported data type. For a full description, refer to the file *Readme.txt* in the CRC_usage folder.

1.3 CRC hardware implementation benefits

The CRC_usage example has been developed to check the compatibility between the software CRC algorithm implementation and the CRC peripheral, as well as to measure their execution times.

The example has been executed under the following conditions:

- Hardware: STM32373C-EVAL board (STM32F37x device)
- System clock: HSE (8 MHz crystal oscillator)
- Toolchain: Keil® V4.60.0.0
- CRC configurations: default values of the CRC register
  CRC_CR: 0x0000 0000; POLYSIZE is 32, no REV_IN and no REV_OUT
  CRC_INIT: 0XFFFF FFFF
  CRC_POLY: 0X04D11 CDB7
- Input data: 256 words

Table 1 compares the execution time of the CRC algorithm versus that achieved using the STM32 CRC peripheral.

Table 1. Comparison of CRC algorithm and CRC peripheral execution time

Optimization level	CRC algorithm (system clock cycle)	CRC peripheral (system clock cycle)
Level 3 + Optimize for time	78094	1287

The hardware implementation is about 60 times faster than the software algorithm. The CPU controls the data transfer and no instruction processing is allowed during this phase. Application developers can choose another peripheral as controller to free the CPU for other tasks. As the DMA manages data transfer, it becomes the alternative for computing the CRC checksum code of the data buffer.

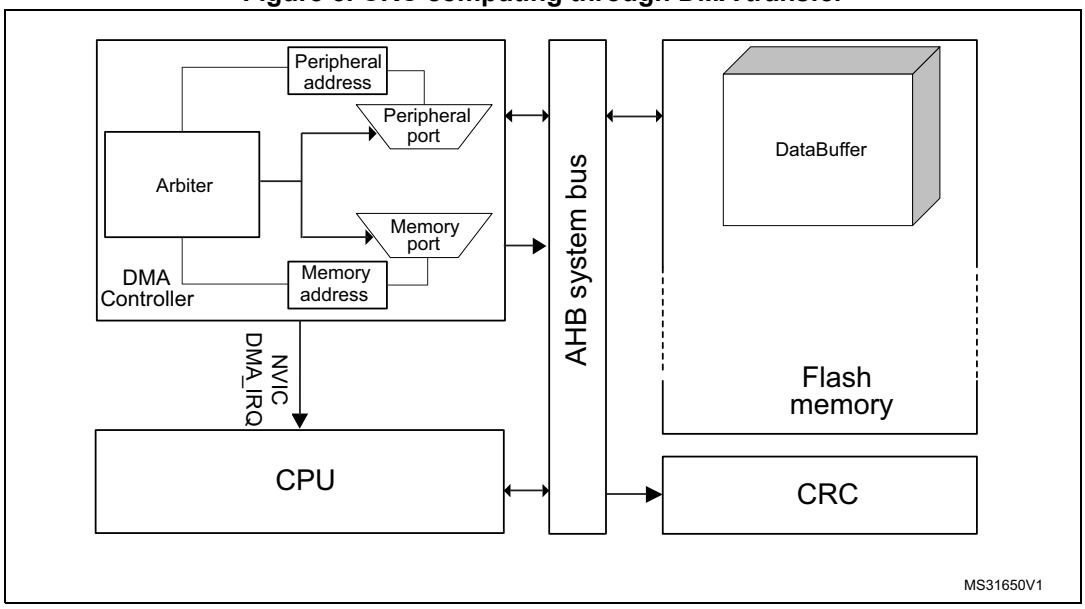
2 Using CRC through DMA

The STM32 embedded DMA can be used as the data transfer handler to avoid the use of the CPU as controller. The DMA configuration depends upon the architecture chosen. There are two categories, namely DMA with channels and DMA with stream, with almost identical configurations.

2.1 DMA back-to-back transfer mode

The CRC_DMA example available in the firmware package implements the technique illustrated in *Figure 5*. This technique is the DMA back-to-back data transfer mode with the DMA_IRQ handler routine callback.

Figure 5. CRC computing through DMA transfer



In this case, the DMA controls the data transfer counter and waits for the transfer complete flag to execute the NVIC DMA_IRQ handler routine. The NVIC DMA_IRQ routine must stop the system timer (systick) counter and return the CRC computed value. The CPU usage is limited only to the execution of the DMA interrupt request routine.

2.2 DMA configuration

As mentioned above, STM32 devices integrate a multiple DMA architecture. The configuration steps below are common for both DMA architectures:

1. Enable the DMA peripheral clock via the RCC peripheral.
2. Configure the DMA channel/stream (see *Table 2* for the two configurations).
3. Configure the CRC, as described in the first five steps of *Section 1.2*.
4. Enable the DMA transfer complete interrupt.
5. Configure the DMA NVIC IRQ.
6. Enable the DMA channel/stream.
7. Wait for the DMA transfer complete to occur.
8. Disable the DMA channel. In the case of DMA with stream architecture, the controller disables automatically the channel when the transfer ends, while in the other case, the NVIC DMA_IRQ routine must disable the channel for further use.
9. Clear DMA IT pending bit.

Note: For any additional information, refer to the file Readme.txt under CRC_DMA example folder.

Table 2. DMA configuration summary

DMA configuration	DMA with channel	DMA with stream
Transfer direction	Memory to memory	
Peripheral address	Flash base pointer	
Memory address	CRC data register	
Peripheral address increment	Enable	
Memory address increment	Disable	
Buffer size	Data buffer size	
Peripheral data size	Supported data type (byte, half-word or word)	
Memory data size	Supported data type (byte, half-word or word)	
Transfer mode	Normal	
Peripheral burst	NA	Single
Memory burst	NA	Single
FIFO mode(1)	NA	Disable
FIFO threshold(1)	NA	-

1. Enabling the FIFO mode does not affect the execution time. Therefore, the FIFO mode and the FIFO threshold configuration have no impact.

The CRC_DMA example has been developed to check the compatibility results between the use of CPU and DMA as transfer handlers, and measure the CPU load during the use of DMA as transfer handler.

2.3 DMA usage benefits

During the DMA back-to-back data transfer mode, the CPU acts during the CRC and DMA configurations, and during the DMA interrupt handler execution.

The execution conditions of the example are the same as those listed in *Section 1.3*, except for the input data size that became 8192 of the supported data type.

Table 3. Comparison results of CPU versus DMA execution time usage

Transfer handler	Peripheral configuration(1)	CRC computing(1)	CPU load
CPU	66(2)	40962	100%
DMA	295(3)	40968	0.72%

1. The systick timer tick is the measurement unit, while the systick clock source is the CPU clock.
2. CRC configuration time.
3. CRC configuration, DMA configuration and DMA IRQ handler execution times.

Overall, the CPU load is equal to 100% when the CPU is used as data transfer handler, it is reduced to 0.72 % when using DMA.

3 CRC migration

Table 4 lists the CRC features, and offers a software compatibility analysis for each Series.

Table 4. CRC peripheral features by Series

Feature	F1, F2, F4, L1	F0, F3, F7, G0, G4, H7, L0, L4, L4+, U5, WB, WL
Single input/output 32-bit data register	Yes	
General-purpose 8-bit register	Yes	
Input buffer to avoid bus stall during calculation	No	Yes
Reversibility option on I/O data	No	Yes
CRC initial value	Fixed to 0xFFFFFFFF	Programmable on 32 bits
Handled data size in bits	32	8, 16, 32
Polynomial size in bits	32	7, 8, 16, 32
Polynomial coefficients	Fixed to 0x4C11DB7	Programmable size: 7, 8, 16, or 32 bits(1)

1. Available only on STM32F07x and STM32F09x devices, fixed to 0x4C11DB7 for other products of the STM32F0 Series.

4 Reference documents

- STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced Arm®-based 32-bit MCUsl (RM0008)
- STM32F205xx, STM32F207xx, STM32F215xx and STM32F217xx advanced ARM-based 32-bit MCUs (RM0033)
- STM32L151xx, STM32L152xx and STM32L162xx advanced Arm®-based 32-bit MCUs (RM0038)
- STM32F100xx advanced ARM®-based 32-bit MCUs (RM0041)
- STM32F405xx, STM32F407xx, STM32F415xx and STM32F417xx advanced Arm®-based 32-bit MCUs (RM0090)
- STM32F05xxx advanced Arm®-based 32-bit MCUs (RM0091)
- STM32F37xx and STM32F38xx advanced Arm®-based 32-bit MCUs (RM0313)
- STM32F302xx, STM32F303xx and STM32F313xx advanced ARM-based 32-bit MCUs (RM0316)
- Guidelines for obtaining IEC 60335 Class B certification in STM32 applications (AN3307)

Note: The above documents are available at www.st.com.

5 Revision history

Table 5. Document revision history

Date	Revision	Changes
06-Jun-2013	1	Initial release.
07-Nov-2022	2	Updated document title and *Introduction*. Removed former *Table 1: Applicable products*. Updated *Table 4: CRC peripheral features by Series*. Minor text edits across the whole document.