Introduction

The cyclic redundancy check (CRC) is a technique used for detecting errors in digital data, but without making corrections when errors are detected. It is used in data transmission or data storage integrity check. The CRC is a powerful and easily implemented technique to obtain data reliability. Diagnostic coverage of this technique satisfies requirements of basic safety standards. This is why the CRC implementation feature is used at Flash content integrity self-test check at ST firmware certified for compliance with IEC 60335-1 and IEC 607030-1 standards (known as "Class B" requirements). For more information, refer to application note AN3307 and the associated firmware packages dedicated for different family products. It is advised to check all the necessary CRC settings at compilers’ manuals when CRC checksum information has to be placed directly into user code by linker (mostly in format of a CRC descriptor data table).

The CRC is based on polynomial arithmetic. It computes the remainder of the division of a polynomial in GF(2) by another. The remainder is called checksum, while the dividend is the data and the divisor is the generator polynomial.

Note: A polynomial in GF(2) (Galois field with two elements) is a polynomial with a single variable x whose coefficients are 0 or 1.

This application note describes the features of the cyclic redundancy check peripheral embedded in all STM32 series (F0, F1, F2, F3, F4, L1) and the steps required to configure it.

This application note is structured as follows:
- **Section 1** describes the STM32 CRC implementation algorithm and its hardware implementation 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 provided as well:
- CRC_usage example: how to configure the CRC using CPU as data transfer controller.
- CRC_DMA example: how to use the DMA as CRC data transfer controller.

The measuring of execution time in both examples is supported.

<table>
<thead>
<tr>
<th>Table 1. Applicable products</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
</tr>
<tr>
<td>Microcontrollers</td>
</tr>
</tbody>
</table>
Contents

1 CRC peripheral overview ............................................ 5
   1.1 CRC computing algorithm .................................... 5
   1.2 CRC peripheral configuration ................................. 7
   1.3 CRC hardware implementation benefits ...................... 9

2 Using CRC through DMA ........................................... 10
   2.1 DMA back-to-back transfer mode ............................ 10
   2.2 DMA configuration ............................................ 11
   2.3 DMA usage benefits ......................................... 12

3 CRC migration through STM32 series ......................... 13

4 Reference documents .............................................. 14

5 Revision history .................................................. 15
List of tables

Table 1. Applicable products ................................................................. 1
Table 2. Comparison of CRC algorithm and CRC peripheral execution time ................. 9
Table 3. DMA configuration summary ...................................................... 11
Table 4. Comparison results of CPU versus DMA execution time usage ..................... 12
Table 5. STM32 CRC peripheral features ............................................... 13
Table 6. Document revision history ................................................................ 15
# List of figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CRC block diagram</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>CRC algorithm flowchart</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>Step-by-step CRC computing example</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>CRC calculation unit block diagram</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>CRC computing through DMA transfer</td>
<td>10</td>
</tr>
</tbody>
</table>
1 CRC peripheral overview

The CRC peripheral embedded in all STM32 microcontroller devices is used to provide a CRC checksum code of any supported data type.

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](image)

One of the known CRC algorithms is the polynomial division with the bitwise message XORing technique. This is the most suitable technique for hardware or low-level implementation, as used in microcontrollers.

The input parameters of this algorithm 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 below shows the CRC algorithm flowchart.
At start up, the algorithm sets CRC to the Initial_Crc XOR with the Input_Data.

Once CRC MSB is equal to one, 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

<table>
<thead>
<tr>
<th>Index</th>
<th>Execution step</th>
<th>Binary format</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Crc &lt;&lt; 1</td>
<td>01111110</td>
<td>0x3E</td>
</tr>
<tr>
<td>1</td>
<td>Crc &lt;&lt; 1</td>
<td>11111000</td>
<td>0x7C</td>
</tr>
<tr>
<td>2</td>
<td>Crc &lt;&lt; 1</td>
<td>11110000</td>
<td>0xF0</td>
</tr>
<tr>
<td>3</td>
<td>Crc &lt;&lt; 1</td>
<td>11001111</td>
<td>0xCB</td>
</tr>
<tr>
<td>4</td>
<td>Crc &lt;&lt; 1</td>
<td>01110110</td>
<td>0x76</td>
</tr>
<tr>
<td>5</td>
<td>Crc &lt;&lt; 1</td>
<td>11101100</td>
<td>0xEC</td>
</tr>
<tr>
<td>6</td>
<td>Crc &lt;&lt; 1</td>
<td>11001011</td>
<td>0x13</td>
</tr>
<tr>
<td>7</td>
<td>Crc &lt;&lt; 1</td>
<td>01001100</td>
<td>0x26</td>
</tr>
</tbody>
</table>

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

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). It is 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) and the new one.

Figure 4. CRC calculation unit block diagram
To compute a CRC of any supported data, you must follow these 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).\(^{(a)}\)
3. Set the I/O reverse bit order through the REV_IN\[1:0\] and REV_OUT bits respectively in CRC Control register (CRC_CR).\(^{(a)}\)
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.\(^{(b)}\)
5. Reset the CRC peripheral through the Reset bit in 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 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, please refer to the file Readme.txt in the CRC_usage folder.

\(^{(a)}\) Applicable only for STM32F0xx and STM32F3xx devices

\(^{(b)}\) Applicable only for STM32F3xx devices
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 registers
  - 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 2 shows the results of the comparison of the execution time of the CRC algorithm versus the STM32 CRC peripheral.

<table>
<thead>
<tr>
<th>Optimization level</th>
<th>CRC algorithm (system clock cycle)</th>
<th>CRC peripheral (system clock cycle)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 3 + Optimize for time</td>
<td>78094</td>
<td>1287</td>
</tr>
</tbody>
</table>

Table 2. Comparison of CRC algorithm and CRC peripheral execution time

The hardware implementation of the CRC is about 60 times faster than the software algorithm. The CPU controls the data transfer and no instruction process is allowed during this phase. Application developers can choose another peripheral as controller in order to free the CPU for other tasks. Since the DMA manages data transfer, it becomes the alternative choice 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 on the architecture chosen. There are two categories: DMA with channels and DMA with stream. For both DMA architectures, the configuration is almost identical.

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_IRQHandler handler routine callback.

In this case of use, the DMA controls the data transfer counter and waits for the transfer complete flag to execute the NVIC DMA_IRQHandler handler routine. The NVIC DMA_IRQHandler routine has to stop the system timer (systick) counter and return the CRC computed value. The CPU usage is only limited 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 3 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 of architecture, the NVIC DMA_IRQHandler routine must disable the channel for further use.

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

<table>
<thead>
<tr>
<th>DMA configuration</th>
<th>DMA with channel</th>
<th>DMA with stream</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transfer direction</td>
<td>Memory to Memory</td>
<td>Memory to Memory</td>
</tr>
<tr>
<td>Peripheral address</td>
<td>Flash Base pointer</td>
<td>Flash Base pointer</td>
</tr>
<tr>
<td>Memory address</td>
<td>CRC data register</td>
<td>CRC data register</td>
</tr>
<tr>
<td>Peripheral address increment</td>
<td>Enable</td>
<td>Enable</td>
</tr>
<tr>
<td>Memory address increment</td>
<td>Disable</td>
<td>Disable</td>
</tr>
<tr>
<td>Buffer size</td>
<td>Data Buffer size</td>
<td>Data Buffer size</td>
</tr>
<tr>
<td>Peripheral data size</td>
<td>Supported data type (byte, half-word or word)</td>
<td>Supported data type (byte, half-word or word)</td>
</tr>
<tr>
<td>Memory data size</td>
<td>Supported data type (byte, half-word or word)</td>
<td>Supported data type (byte, half-word or word)</td>
</tr>
<tr>
<td>Transfer Mode</td>
<td>Normal</td>
<td>Normal</td>
</tr>
<tr>
<td>Peripheral Burst</td>
<td>NA</td>
<td>Single</td>
</tr>
<tr>
<td>Memory Burst</td>
<td>NA</td>
<td>Single</td>
</tr>
<tr>
<td>FIFO mode(1)</td>
<td>NA</td>
<td>Disable</td>
</tr>
<tr>
<td>FIFO threshold(1)</td>
<td>NA</td>
<td>--</td>
</tr>
</tbody>
</table>

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 intervenes only 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 4. Comparison results of CPU versus DMA execution time usage

<table>
<thead>
<tr>
<th>Transfer handler</th>
<th>Peripheral configuration(^{(1)})</th>
<th>CRC computing(^{(1)})</th>
<th>CPU load</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>66(^{(2)})</td>
<td>40962</td>
<td>100%</td>
</tr>
<tr>
<td>DMA</td>
<td>295(^{(3)})</td>
<td>40968</td>
<td>0.72%</td>
</tr>
</tbody>
</table>

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, it is important to note that the CPU load is equal to 100% when the CPU is used as the data transfer handler, while it is reduced to 0.72% when using DMA.
3 CRC migration through STM32 series

The CRC peripheral features can vary from one STM32 series to another. Table 5 lists the CRC features and offers a software compatibility analysis for each STM32 series.

Table 5. STM32 CRC peripheral features

<table>
<thead>
<tr>
<th>Feature</th>
<th>F1 series</th>
<th>L1 series</th>
<th>F2 series</th>
<th>F4 series</th>
<th>F0 series</th>
<th>F3 series</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single input/output 32-bit data register</td>
<td>YES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>General-purpose 8-bit register</td>
<td></td>
<td>YES</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input buffer to avoid bus stall during calculation</td>
<td>NO</td>
<td></td>
<td></td>
<td>YES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reversibility option on I/O data</td>
<td>NO</td>
<td></td>
<td></td>
<td>YES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CRC initial value</td>
<td>Fixed to 0xFFFFFFFF</td>
<td></td>
<td>Programmable on 32 bits</td>
<td>Programmable on 8, 16, 32 bits</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Handled data size in bits</td>
<td>32</td>
<td></td>
<td>8, 16, 32</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Polynomial size in bits</td>
<td>32</td>
<td></td>
<td>7, 8, 16, 32</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Polynomial coefficients</td>
<td>Fixed to 0x4C11DB7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Programmable</td>
</tr>
</tbody>
</table>
Reference documents

- STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced ARM-based 32-bit MCUs reference manual (RM0008)
- STM32L151xx, STM32L152xx and STM32L162xx advanced ARM-based 32-bit MCUs reference manual (RM0038)
- STM32F100xx advanced ARM-based 32-bit MCUs reference manual (RM0041)
- STM32F405xx, STM32F407xx, STM32F415xx and STM32F417xx advanced ARM-based 32-bit MCUs reference manual (RM0090)
- STM32F05xxx advanced ARM-based 32-bit MCUs reference manual (RM0091)
- STM32F37xx and STM32F38xx advanced ARM-based 32-bit MCUs reference manual (RM00313)
- STM32F302xx, STM32F303xx and STM32F313xx advanced ARM-based 32-bit MCUs reference manual (RM00316)
- Guidelines for obtaining IEC 60335 Class B certification in STM32 applications (AN3307)

Note: The above documents are available at the following URL: http://www.st.com/stm32
5 Revision history

Table 6. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>06-Jun-2013</td>
<td>1</td>
<td>Initial release.</td>
</tr>
</tbody>
</table>
Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

ST PRODUCTS ARE NOT AUTHORIZED FOR USE IN WEAPONS. NOR ARE ST PRODUCTS DESIGNED OR AUTHORIZED FOR USE IN: (A) SAFETY CRITICAL APPLICATIONS SUCH AS LIFE SUPPORTING, ACTIVE IMPLANTED DEVICES OR SYSTEMS WITH PRODUCT FUNCTIONAL SAFETY REQUIREMENTS; (B) AERONAUTIC APPLICATIONS; (C) AUTOMOTIVE APPLICATIONS OR ENVIRONMENTS, AND/OR (D) AEROSPACE APPLICATIONS OR ENVIRONMENTS. WHERE ST PRODUCTS ARE NOT DESIGNED FOR SUCH USE, THE PURCHASER SHALL USE PRODUCTS AT PURCHASER’S SOLE RISK, EVEN IF ST HAS BEEN INFORMED IN WRITING OF SUCH USAGE, UNLESS A PRODUCT IS EXPRESSLY DESIGNATED BY ST AS BEING INTENDED FOR “AUTOMOTIVE, AUTOMOTIVE SAFETY OR MEDICAL” INDUSTRY DOMAINS ACCORDING TO ST PRODUCT DESIGN SPECIFICATIONS.

PRODUCTS FORMALLY ESCC, QML OR JAN QUALIFIED ARE DEEMED SUITABLE FOR USE IN AEROSPACE BY THE CORRESPONDING GOVERNMENTAL AGENCY.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2013 STMicroelectronics - All rights reserved

STMicroelectronics group of companies
Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com