Cyclic redundancy check in STM32H7 Series flash memory interface

Introduction

Whenever data is stored or transmitted, there is a risk of data corruption. One of the most reliable mechanisms to deal with the risk of data corruption is the cyclic redundancy check (CRC). The STM32H7 Series devices embed the CRC feature on their flash memory to ensure that the content of the flash memory is reliable.

Further details about CRC, its implementation on the STM32H7 flash memory, and how it is implemented on the HAL drivers are provided in this application note. In addition, a software simulation of the CRC hardware block calculation is presented. And at the end, details about the time needed to perform a CRC are presented.

Reference documents, available on www.st.com:

- STM32H742, STM32H743/753, and STM32H750 Value line advanced Arm®-based 32-bit MCUs (RM0433)
- STM32H745/755 and STM32H747/757 advanced Arm®-based 32-bit MCUs (RM0399)
- STM32H7A3/B3 and STM32H7B0 Value line advanced Arm®-based 32-bit MCUs (RM0455)
- STM32H723/733, STM32H725/735, and STM32H730 Value line advanced Arm®-based 32-bit MCUs (RM0468)

1 CRC overview

Cyclic redundancy check (CRC) is an error detection technique that checks the integrity of data during transmission or storage.

The CRC algorithm computes a checksum for each specific data that passes through it. For the calculation, it uses a polynomial, the initial CRC value, and the data to check.

The CRC-32 Ethernet polynomial is widely used and proved efficient for CRC calculation.

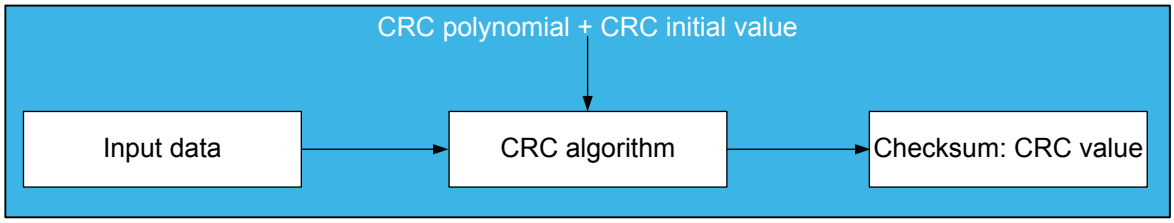
CRC-32 (Ethernet) polynomial:

x^32 + x^26 + x^23 + x^22 + x^16 + x^12 + x^11 + x^10 + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + x^0

which is 0x04C1 1DB7 in normal representation or 0xEDB8 8320 in reverse notation.

The figure below shows the block diagram of the CRC calculation algorithm:

Figure 1. CRC block diagram



A typical use case of CRC is the check of data integrity during a transfer of programming phase.

Example:

- The CRC value is computed on sender and receiver sides:
  - The sender calculates its CRC and appends it to the message to send.
  - The receiver, on its side, puts the data through the same process and compares it with the result appended by the sender.
- If the two calculated CRCs are equal, it means that no issue occurred and the data got sent/stored correctly.
- Otherwise, in case the CRCs do not match, that indicates data was corrupted during transmission. In such situations, the receiver can request to transmit again the block of data.

2 CRC hardware implementation on STM32H7 Series

The STM32 MCUs embed a CRC calculation unit that allows them to get the CRC value of a block of data. A use case of the CRC calculation unit is the verification of the flash memory integrity. The CRC helps to compute a signature of the software while booting or during runtime. This signature is compared with a reference signature which was generated at link-time and stored at a given memory location.

If the CRC value is the same after this comparison, the application may continue the execution of the flash memory content. If the CRC is not correct, this means that the flash memory content is corrupted. If the flash memory content is corrupted, an exceptional software could be launched (such as a jump to the bootloader or to an IAP).

This CRC calculation unit is very efficient. However, it is not only dedicated to check the integrity of the flash memory contents. Which means that, it could be used to verify the integrity of other memories. The CRC also needs other resources such as the CPU or the DMA to transfer data from the peripheral or memory. These data need to be checked on the CRC calculation unit data register (CRC_DR) where CRC is calculated.

As an improvement, the STM32H7 Series embed a dedicated CRC hardware module inside their flash memory interface, in addition to the CRC calculation unit. This module is exclusively reserved to verify the flash memory content. It does not need any further peripherals to accomplish this task.

The embedded CRC module in the flash memory interface facilitates the implementation of tests to detect permanent faults that might impact the flash memory including the memory cells and the address decoder by periodically computing the flash memory CRC signature and comparing it against its expected value.

3 Flash memory CRC module operating mode

The embedded flash memory interface of the STM32H7 Series manages the accesses of any master to their embedded nonvolatile memory (up to 2 Mbytes).

The size of one flash memory word is:

- 256 bits on STM32H723/733, STM32H725/735, STM32H730, STM32H742, STM32H743/753, STM32H750, STM32H745/755, and STM32H747/757.
- 128 bits on STM32H7A3, STM32H7B3, and STM32H7B0.

The flash memory interface on STM32H7 Series embeds a CRC hardware module that permits the check of the integrity of a specific content of the flash memory area. Depending on the programmer needs, this area can be defined as a whole bank, some consecutive sectors, or an area defined by start/end addresses.

The configuration steps recommended for a correct operation of the CRC hardware module are:

1. Unlock the flash memory to enable the flash memory control-register access.
2. Enable the CRC feature by setting the CRC_EN bit in the FLASH_CR1/2 register.
3. Program the desired data burst size in the CRC_BURST field of the FLASH_CRCCR1/2 register.

   The CRC hardware module processes flash memory data by chunks of 4, 16, 64 or 256 flash-words. Those chunks values are defined as burst size and are configurable by software.

```
CRC_InitStruct.BurstSize = FLASH_CRC_BURST_SIZE_4; //every burst has a size of 4 Flash
words
CRC_InitStruct.BurstSize = FLASH_CRC_BURST_SIZE_16; //every burst has a size of 16
Flash words
CRC_InitStruct.BurstSize = FLASH_CRC_BURST_SIZE_64; //every burst has a size of 64
Flash words
CRC_InitStruct.BurstSize = FLASH_CRC_BURST_SIZE_256; //every burst has a size of 256
Flash words
```

4. Define the flash memory area to compute the CRC.

   The area processed by the CRC module can be defined either between two addresses, on a list of sectors or on a whole bank. The configuration is done by software.

   – **Example of configuration between two addresses:**

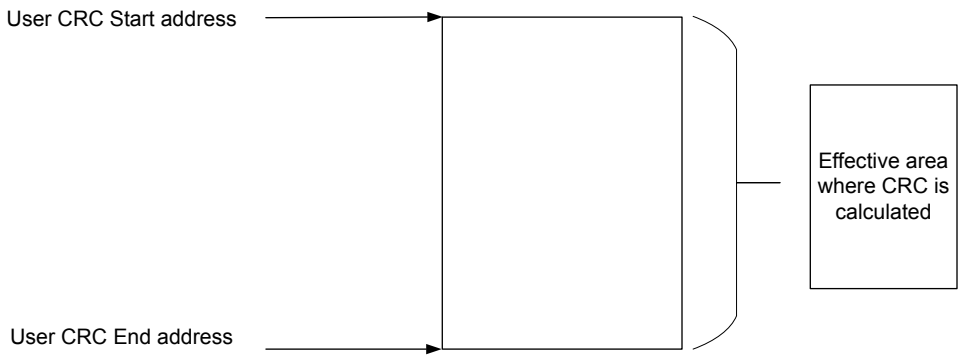
   Specify the start and end addresses to define the memory area where CRC must be calculated. This is done by programing FLASH_CRCSADD1/2R and FLASH_CRCEADD1/2R registers, respectively.

```
CRC_InitStruct.TypeCRC = FLASH_CRC_ADDR;
CRC_InitStruct.CRCStartAddr = UserCRCStartAddr;
CRC_InitStruct.CRCEndAddr = UserCRCEndAddr-1;
```

Note: When addresses are used to define the data to check, the CRC burst size has an impact on the result.

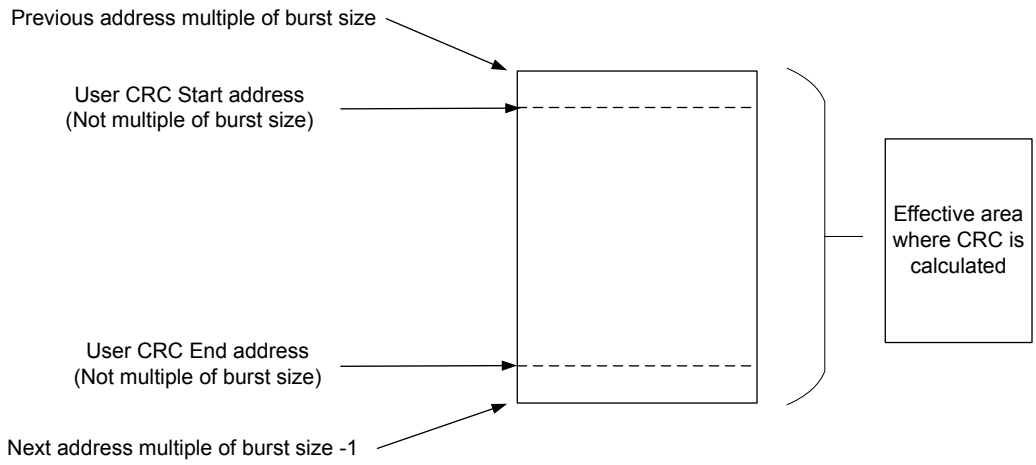
Case 1: The start address and the end address +1 are multiples of the chosen burst size (step 3). In this case, the CRC is calculated exactly from start to end address as shown below.

Figure 2. CRC computed in an address-defined area - case 1



Case 2: the start address and the end address are not multiples of the chosen burst size. In this case, the CRC is calculated in the area between the last address multiple of the burst size before the chosen start address and the next address multiple of the burst size -1 after the end address as shown below.

Figure 3. Case 2 CRC



– **Example of configuration between two sectors:**

When performed at sector level, CRC calculation is done on one or more selected user flash memory sectors. Those sectors are defined by indicating the first sector on which the CRC should be calculated, followed by the total number of sectors to check.

In order to select the targeted sectors, CRC_BY_SECT bit in the FLASH_CRCCR1/2 register should be set and the target sector numbers in the CRC_SECT field of the FLASH_CRCCR1/2 register need to be programmed. In addition, the ADD_SECT bit should be set after each CRC_SECT programming.

```
CRC_InitStruct.TypeCRC = FLASH_CRC_SECTORS;
CRC_InitStruct.Sector = FLASH_SECTOR_0; // Specify the initial sector
CRC_InitStruct.NbSectors = 1; // Number of sectors
```

– **Example of configuration on a whole bank:**

All bank 1 or 2 user sectors are added to the list of sectors on which the CRC should be calculated.

```
CRC_InitStruct.TypeCRC = FLASH_CRC_BANK;
CRC_InitStruct.Bank = FLASH_BANK_1; //Bank1 or 2 could be specified
```

5. Start the CRC operation by setting the START_CRC bit in FLASH_CRCCR1/2 register.
6. Wait until the CRC_BUSY flag in FLASH_SR1/2 register is reset.
7. Retrieve the CRC result in the FLASH_CRCDATAR register.

The steps listed above, excepting step 1, are already implemented on the function “HAL_FLASHEx_ComputeCRC” in the extended FLASH HAL module driver “stm32h7xx_hal_flash_ex.c”.

The software configuration above should be configured depending on the user's specific CRC calculation request.

Note:

- *The CRC operation is concurrent with option-byte change operations. If a CRC operation is requested while an option-byte change is ongoing, the option-byte change operation must be completed before serving the CRC operation, and vice versa.*
- *Running a CRC on PCROP-protected or secure-protected user flash memory area may alter the expected CRC value.*
- *Only one CRC check operation on bank 1 or 2 can be launched at the time. Thus, to avoid corruption, do not configure the CRC calculation on one bank, while calculating the CRC on the other bank.*

4 Software simulation of the hardware CRC block calculation

The “SW_crc32_Calcul” function below is developed to simulate the behavior of the CRC hardware module embedded in the flash memory interface of the STM32H7 Series.

CRC-32 polynomial should be defined: as follows:

```
/* CRC-32 polynomial */
#define POLY 0x4c11db7
/*******************************************************************************
* Title: SW_crc32_Calcul                                                       *
* Input parameters:                                                            *
*   - crc: Initial CRC value                                                   *
*   - buf: Pointer to first data byte                                          *
*   - len: Number of data bytes stored, for which CRC should be calculated    *
*   - BurstSize:  varies depending on the requested burst size "4, 16, 64 or
256 Flash words".                                                              *
*  Output: Calculated CRC result                                               *
*******************************************************************************/
uint32_t SW_crc32_Calcul(uint32_t crc, const uint32_t *buf, size_t len, uint32_t BurstSize)
{
    int k;
    /* Define the initial CRC value */
    uint32_t crc = 0;
    /* Define the length of data on which CRC has to be computed depending on CRC burst size
*/
    if ( len > BurstSize)
    {
      if ( (len % BurstSize) != 0)
      {
        len = BurstSize * ( ( len / BurstSize) + 1 );
      }
    }
    else
    {
      len = BurstSize;
    }
    /* Calculate CRC value */
    while (len--) {
        crc ^= *buf++;
        for (k = 0; k < 32; k++)
            crc = crc & 0x80000000 ? (crc << 1) ^ POLY : crc << 1;
        crc ^= 0x55555555;
    }
    /* Return CRC value */
    return crc;
}
Simple example:
#define N 10
int main(void)
{
    /* compute CRC for N words starting at address 0x08000000 */

            uint32_t u32CrcValue = SW_crc32_Calcul((const uint32_t *)0x08000000, N*4*8, 4);
    while(1);
}
```

5 System performance impact when using CRC

Using the CRC hardware module costs the application some CPU clock pulses. The number of pulses depends on the load of the flash memory interface. Any operation on the flash memory increases the load, and then increases the time for a CRC calculation.

In the case where only the CRC calculation is being executed in the flash memory, 32 bits are put in the CRC at every clock pulse. In this case, the time needed to complete the calculation can be estimated. The number of clock pulses needed to calculate one flash memory word depends on the flash memory word size:

- 8 clock pulses for STM32H723/733, STM32H725/735, STM32H730, STM32H742, STM32H743/753, STM32H750, STM32H745/755, and STM32H747/757 devices
- 4 clock pulses for STM32H7A3, STM32H7B3, and STM32H7B0 devices

In addition, a minimum of one-clock pulse is lost every time that the CRC hardware block loads a new burst of data.

The following tables describe the calculation time needed to calculate the CRC value for a flash memory area depending on the configured burst size on the STM32H7 Series.

Table 1. CRC calculation time on STM32H742, STM32H7X3, STM32H7x5, STM32H7x7, STM32H730, and STM32H750

Burst size	CPU clock pulses for 1 burst	CPU clock pulses for X burst
4 Flash word	4 x 8 = 32	(32 + 1) × X
16 Flash words	16 x 8 = 128	(128 + 1) × X
64 Flash words	64 x 8 = 512	(512 + 1) × X
256 Flash words	256 x 8 = 2048	(2048 + 1) × X

1. *X = number of bursts in the memory area in which the CRC is computed*

Table 2. CRC calculation time on STM32H7A3, STM32H7B3, and STM32H7B0

Burst size	CPU clock pulses for 1 burst	CPU clock pulses for X burst
4 Flash word	4 x 4 = 16	(16 + 1) × X
16 Flash words	16 x 4 = 64	(64 + 1) × X
64 Flash words	64 x 4 = 256	(256 + 1) × X
256 Flash words	256 x 4 = 1024	(1024 + 1) × X

1. *X = number of bursts in the memory area in which the CRC is computed*

As shown in the tables above, the CRC calculation time is minor compared to the application time. Thus, checking the Flash memory area integrity using the dedicated CRC hardware block is highly recommended.

6 Conclusion

CRC is a simple but an effective way to detect data corruption during transfer/programming or flash memory failure. This application note described the CRC feature and how the CRC hardware module embedded in STM32H7 Series devices flash memory interface works.

Revision history

Table 3. **Document revision history**

Date	Version	Changes
29-Mar-2022	1	Initial release.

Contents

List of tables

List of figures