Application Report
SPRACR3–April 2020

CRC Engines in C2000™ Devices

Veena Kamath, Prasanth Viswanathan Pillai, Sira Rao, and Baskaran Chidambaram

ABSTRACT

This application report is an introduction to the various CRC modules that are available on the C2000 device families. References are provided to software examples and the differences between the CRC modules are highlighted. Also, guidance about selecting a CRC module for a specific use-case is discussed in Section 9.

Contents

List of Tables

Trademarks

1 Introduction

Cyclic Redundancy Check (CRC) is an error detection mechanism used in communication networks and data storage. The C2000 family of devices discussed in Section 11 contains many CRC modules (with different properties discussed in Section 10) that provide the end application builder the options to pick the right engine for the application. Additionally there are CRC engines present within C2000 devices in CLB, FSI, USB, CAN, EMAC, EtherCAT, and so forth that are not discussed in this document. If the data and corresponding CRC are modified together that will not be detectable by this mechanism.

1.1 Acronyms

Table 1. Acronyms

Acronym	Expansion
BGCRC	Background Cyclic Redundancy Check
CAN	Controller Area Network
CLA	Control Law Accelerator (Co-processor for F28x family of devices)
CLB	Configurable Logic Block
CM	Connectivity Manager
CRC	Cyclic Redundancy Check
ECC	Error Correction Code
EMAC	Ethernet Media Access Controller
ERAD	Embedded Real-Time Analysis and Diagnostic Unit
FSI	Fast Serial Interface
PAB	Program Address Bus
GCRC	General Cyclic Redundancy Check
PROMCRC	Program ROM CRC
PSA	Parallel Signature Analysis
VCU	Viterbi, Complex Math and CRC Unit

2 BGCRC

Back-Ground Cyclic Redundancy Check (BGCRC) engine can be used to calculate CRC on a memory region without needing to use the CPU or the CLA. The CRC computation can be then used for detecting memory corruption. BGCRC needs just one-time configuration for its functionality. BGCRC after computing the CRC, compares against the configured golden value and can then trigger a NMI or CLA task if there is any error.

The BGCRC module has the ability to continuously read data from the memory as a background process and work in the background right through the application run-time. The reads happen during the idle time (when none of the other masters such as CPU, CLA, DMA are accessing the memory block) and hence the functional accesses are not impacted. The module also does ECC/parity check while reading the memories. Any ECC or parity errors that occur during the read will be indicated by setting the respective NMI flag and generating an interrupt, if configured.

The module also provides an option to lock and commit the register values once configured. The CRC computation time can also be monitored using an internal watchdog. The engine takes one cycle for computing the CRC per 32-bit word. But, the total time taken for a memory block may vary based on CPU/DMA/CLA memory accesses to the configured memory block as BGCRC works only during idle time.

The device has separate BGCRC instances for CPU and CLA in both CPU1 and CPU2 subsystems.

For more details on computation time and other features present in the module, see the *TMS320F2838x Microcontrollers Technical Reference Manual*.

For software support, see C2000ware:

- driverlib can be referred from <C2000Ware installation folder>\driverlib\<device>\driverlib
- examples can be referred from <C2000Ware installation folder>\driverlib\<device>\examples\c28x\bgcrc

3 GCRC

General Cyclic Redundancy Check (GCRC) is available only with the Connectivity Manager (Cortex-M) core of F2838x. This CRC engine provides option for custom polynomial for CRC calculation. It has the ability to calculate CRC on byte, half-word or word data. After configuring the module, CPU/DMA is expected to provide the data to the engine.

Other features include:

- Ability to define the endianness of data and datatype of the source data
- Ability to reverse the bit order
- Ability to select data bits which participate in the CRC computation

The module also has support for fixed polynomial path where the CRC configuration is fixed with following values. This fixed data path will compute the CRC of a given data set in a single cycle.

- Polynomial : 0x04C11DB7
- Endianness : Little
- Bit Reverse : No
- Datatype : 32 bit
- DataMask : None

GCRC takes 2n+2 cycles for n bytes of data (for polynomials other than 0x04c11db7).

For more details on the module, the *TMS320F2838x Microcontrollers Technical Reference Manual*.

For software support, see C2000ware:

- driverlib can be referred from <C2000Ware installation folder>\driverlib\f2838x\driverlib_cm
- examples can be referred from <C2000Ware installation folder>\driverlib\f2838x\examples\cm\gcrc

4 VCU CRC

The VCU module on C2000 devices performs CRC computation in addition to Viterbi and Complex Math operations. It provides special instructions to speed up CRC computation which otherwise take several cycles on the C28x CPU. 3 different types of VCU modules are present on C2000 devices. To determine the specific VCU module (if any) available on a specific device, see the *C2000 Real-Time Control Peripherals Reference Guide*.

- VCU Type0 or Type1 (variously represented as VCU0, VCU-I) – they are identical.
- VCU Type2 (variously represented as VCU2, VCU-II). This is a newer version of VCU0.
- VCRC - This is the latest version of the module which contains only CRC functionality. Viterbi and Complex Math functionality have been removed.

The module supports computation of 8-bit, 16-bit, 24-bit (except in VCU0), or 32-bit CRCs. The VCRC module supports user configurable polynomials, flexible both in value and size (1 to 32 bits). It also supports user configurable data sizes (1 to 8 bits).

Table 2. Features of Different VCU Modules

	8-Bit CRC Polynomial	16-Bit CRC Polynomial	24-Bit CRC Polynomial	32-Bit CRC Polynomial	Configurable Data and Polynomial
VCU0	0x07	0x8005 0x1021	N/A	0x4C11DB7	N/A
VCU2	0x07	0x8005 0x1021	0x5D6DCB	0x4C11DB7 0x1EDC6F41	N/A
VCRC	0x07	0x8005 0x1021	0x5D6DCB	0x4C11DB7 0x1EDC6F41	1-8 bit data 1-32 bit polynomial

Fixed polynomials are supported by VCU0/VCU2 (listed in Table 2), and their error detection capabilities are as follows:

- Can detect all single-bit and double-bit errors in the message (of arbitrary length)
- Can detect any error pattern that has an odd number of errors
- Can detect all burst errors of length up to the length of the CRC, and a fraction of all longer bursts ($1 - 2^{-n}$)

For more information, see the *TMS320C28x Extended Instruction Sets Technical Reference Manual*.

The above mentioned capabilities provide the user with many options to meet varied application requirements. CRC computation can be performed on data from ROM, RAM, or FLASH. VCU also features bit-order support - CRC can be computed on data taken from memory "as is" or flipped. The latter is called a "Reflected CRC".

C2000Ware contains assembly-optimized software libraries for the VCU, and examples that demonstrate their use through C-callable assembly functions. In addition, for comparison, CRCs are computed using a Look-up Table approach (written in C), and the Linker generated CRC (at link-time; this is a feature provided by the C2000 Code Generation Tools).

The libraries support even or odd Parity support (Endianness). With even parity, CRC input computation begins on the low-byte in memory, while with odd parity, it begins with the high-byte in memory.

For software support, see C2000ware:

- VCU0 - CRC C-callable assembly implementations can be found at C2000Ware_X_XX_XX_XX\libraries\dsp\VCU\c28\source\vcu0\crc
- VCU2 - CRC C-callable assembly implementations can be found at C2000Ware_X_XX_XX_XX\libraries\dsp\VCU\c28\source\vcu2\crc.
- Examples can be found at: C2000Ware_X_XX_XX_XX\libraries\dsp\VCU\c28\examples\crc

5 ERAD CRC

The CRC units in the ERAD module monitor CPU buses and compute CRC when the self-test code is executed. The main purpose of the CRC units is to ensure that the CPU functionally remains intact when it is executing the same software test library over multiple iterations. These CRC units monitor different CPU interfaces and cannot be used for CRC calculation on program or data memory.

For more details, see the *TMS320F28004x Microcontrollers Technical Reference Manual* and the *TMS320F2838x Microcontrollers Technical Reference Manual*.

6 CLA CRC (PSA)

6.1 CLA PSA

The Parallel Signature Analysis (PSA) ensures the integrity of code execution on Control Law Accelerator (CLA). CLA-PSA logic allows computing the signature of Program Address Bus (PAB) and Data Write Data Bus (DWDB) of the CLA. The golden signature can be either computed offline or during the initialization phase of the application. This can subsequently be compared with the signature computed during execution to ensure the correctness of the executed code.

6.1.1 PSA for PAB

Cross-checking PSA for PAB will ensure the correct sequence of the code execution. Unintended code branches as a result of the fault can be detected by using the PSA for PAB. This is computed using the polynomial $1 + x + x^2 + x^{22} + x^{32}$.

6.1.2 PSA for DWDB

PSA for DWDB can be used to ensure the integrity of written data. If it is required to compute the PSA for a set of memory locations (which is required to periodically check the integrity of configuration registers, static memory content, and so forth), it needs to be first read to the CLA and then written to a dummy location. PSA for DWDB allows the polynomial to be configured using MPSACTL.MPSA2CFG configuration.

- 00 – PSA $(1 + x + x^2 + x^{22} + x^{32})$
- 01 – CRC32 $(1 + x + x^2 + x^4 + x^5 + x^7 + x^8 + x^{10} + x^{11} + x^{12} + x^{16} + x^{22} + x^{23} + x^{26} + x^{32})$

6.1.3 Considerations While Computing the PSA

- It is recommended to compute the PSA in event mode (MPSACTL.MPABCYC = 1 and MPSACTL.MDWDBCYC = 1). Computing PSA in cycle mode has strong dependency on memory access which might be difficult to ensure during a real application.
- Polynomial configuration should be performed when PSA2 is stopped.

7 CLA-PROMCRC – CLA Program Integrity check

The CLAPROMCRC module calculates CRC-32 value of a configurable block of data in CLA program ROM. The CLAPROMCRC calculates the CRC-32 in a non-intrusive manner when the CLA is not accessing the ROM on the CLA program bus. Neither the CPU nor the CLA has the capability to compute a CRC on the CLA program ROM and this limitation is handled by the CLAPROMCRC unit. However, CLAPROMCRC cannot be used for any other memory block. The BGCRC engine has the capability to perform CRC on CLA Program ROM. Hence, in devices with BRCRC it can be used. CLAPROMCRC module can be used on devices where BGCRC support is not available.

For software support, see C2000ware: <C2000Ware>\driverlib\<device>\examples\clapromcrc.

For more details, *TMS320F28004x Microcontrollers Technical Reference Manual*.

8 CRC Calculation Using Software

There are some sample C functions (for C28x) available in <C2000ware>\libraries\dsp\VCU\c28\source\common\c\crc which compute CRC8, CRC16 and CRC32 using lookup tables. In these examples, a lookup table of size 512 bytes is generated prior to the CRC computation. This method significantly reduces CRC computation time, but needs additional memory. At compiler optimization level -o2, these functions take around 50 cycles per 32-bit word.

For software support, see C2000ware: <C2000ware>\device_support\<device>\examples\cpu1\cla_crc8.

9 CRC Recommendation for Use-Cases

The different use cases and which CRC type can help in addressing the requirements are given in Table 3. The functional safety standards ISO 26262 and IEC 61508 require the integrity check of various hardware units. The different CRC blocks implemented in the device helps to address some of these requirements.

Table 3. Use Cases and Preferred CRC Modules

No	Use-case	Preferred (1) CRC Type to be Used
1	In order to ensure the integrity of code execution (for example, to detect unintended branches in the code, wrong program data due to a fault), use CRC on the program bus. CRC computed on the fetched program data will provide coverage to the fetch unit, address generation unit and memory interconnect.	1. ERAD CRC for C28x 2. CLA-PSA for CLA
2	Configuration registers hold important content that determine the device behavior. It is required to check the integrity of the configuration registers periodically. CRC modules can be used to complete this test faster.	1. ERAD CRC for C28x 2. VCUCRC for C28x (for devices without support for ERAD CRC) 3. CLA PSA for CLA 4. GCRC for CM sub-system with fixed polynomial (0x04C11DB7)
3	Similar to configuration registers, it is required to check the integrity of the static SRAM/ROM contents to make sure that they are free from faults.	1. BGCRC for C28x and CLA 2. VCUCRC for C28x (for devices without support for ERAD CRC) 3. GCRC for CM sub-system (polynomial - 0x04C11DB7) 4. CLA PROMCRC can be used in devices where BGCRC is not available.
4	Similar to configuration registers and SRAM/ROM memory, it is required to check the integrity of static flash contents	1. ERAD CRC for C28x 2. GCRC for CM sub-system (polynomial: 0x04C11DB7) 3. VCUCRC for C28x (for devices without support for ERAD CRC)
5	Many times, there might be use cases where the user may have to compute CRC using a polynomial which is not supported in hardware. The configurable CRC support available with VCUCRC and GCRC module help to implement this in an efficient manner.	1. VCUCRC for C28x (configure appropriate polynomial) 2. If the polynomial is available in CLA-PSA, it can be used. Else, software needs to be used 3. GCRC with the appropriate polynomial configured
6	Communication interfaces implement end to end safing as a back channel safing approach to provide integrity to the transmitted data. Hardware accelerators in the device can be used to implement this and reduce the impact on CPU MIPS.	1. VCUCRC for C28x 2.Software-based computation for CLA 3. GCRC for CM sub-system

(1) Other CRC modules can be used as appropriate, the options mentioned here are just the preferred options.

10 CRC Modules Comparison

Table 4. Properties of CRC Engines

Properties of CRC engines	BGCRC	GCRC	VCU CRC	ERAD CRC	CLA PSA
Accessible cores	C28x, CLA	CM	C28x	C28x	CLA
CRC polynomial(s) used	Fixed value = 0x04C11DB7	User-programmable (up to 32 -bit polynomial)	• 8-bit - 0x07 • 16-bit - CRC16 802.15.4, 0x8005, CRC-CCITT, 0x1021 • 24-bit - 0x5d6dcb • 32-bit - CCITT-32, 0x04C11DB7, 0x1EDC6F41 VCRC supports user configurable size of data and configurable polynomials (value and size (up to 32 bits))	Fixed polynomial	PAB : 1 + x + x^2 + x^{22} + x^{32} (0x00400007) DWDB: polynomial is configurable PSA mode : 1 + x + x2 + x22 + x32 (0x00400007) CRC32 mode : 1 + x + x2 + x4 + x5 + x7 + x8 + x10 + x11 + x12 + x16 + x22 + x23 + x26 + x^{32} (0x04C11DB7) CRC-16-IBM : 1 + x2 + x15 + x16 (0x8005) CRC16-CCITT : 1 + x5 + x12 + x16 (0x1021)
Seed value	User-programmable	User-programmable	User-programmable	User-programmable	User-programmable
Input size	Multiples of 256 Bytes Min : 256 Bytes Max : 256 KBytes	Any size	Any size	-	Any size
Reverse bit order (reflected input)	No	Yes	Yes	No	No
Number of cycles for CRC computation	1 cycle per 32-bit word	Fixed polynomial : 1 cycle per 32-bit word Others : 2n+2 cycles for n bytes of data	Fixed polynomials - 1 cycle per byte Configurable polynomials - 3 cycles for 1-8 bits	1	1
Mode of data access	Direct access to memory	CPU/DMA needs to write the data one by one to a specific register in GCRC	Direct access to memory	Monitoring bus access	Monitoring bus access

11 CRC Engines vs Devices Mapping Table

Table 5. CRC Engines vs Devices Mapping Table

Device/ CRC Engine	BGCRC	GCRC	VCU CRC	CLA PSA	CLAPROMCRC	ERAD CRC
F2837xD/F2837xS/ F2807x	NA	NA	C28x (VCU2)	NA	NA	NA
F28004x	NA	NA	C28x (VCU0)	CLA	CLA	NA
F2838x	C28x, CLA	CM	C28x (VCRC)	CLA	NA	C28x
F28002x	C28x	NA	C28x (VCRC)	NA	NA	C28x

12 References

- *TMS320F2838x Microcontrollers Technical Reference Manual*
- *TMS320C28x Extended Instruction Sets Technical Reference Manual*
- *TMS320F28004x Microcontrollers Technical Reference Manual*
- *C2000 Real-Time Control Peripherals Reference Guide*

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265