

DediWare Checksum Calculation Methods

© DediProg Technology Co., Ltd. 2023 All rights reserved.



V1.0

2

Table of Contents

I.	The Scope of Checksum Calculation	3
	File Checksum	3
	Chip Checksum	3
	Option Checksum	6
II.	Checksum Algorithm Types	6
	ByteACC (Byte sum (x8))	7
	ByteACC_2s complement	7
	CRC8_CCITT	7
	CRC16	7
	CRC32	8
	MD5	8
	MD5_UseSourceFile	8
	WoldAccLE (WORD)	9
	WoldAccBE (WORD)	10
	DWoldAccLE (DWORD)	11
	DWoldAccBE (DWORD)	12
	SHA1	13
	SHA256	13
III.	Additional Information	13
	What are Big-Endian and Little-Endian	13
IV.	Revision History	15

Important notice:

This document is provided as a guideline and must not be disclosed without consent of DediProg. However, no responsibility is assumed for errors that might appear.

DediProg reserves the right to make any changes to the product and/or the specification at any time without notice. No part of this document may be copied or reproduced in any form or by any means without prior written consent of DediProg.



I. The Scope of Checksum Calculation

There are two methods to calculate the checksum scope for the programmer, File Checksum and Chip Checksum. When the files are loaded into the buffer of the programmer software, the File Checksum will be displayed in the red box, while the Chip Checksum will be displayed in the green box.

Chip	File CheckSum	File Size	IC Partition	File Name	
0xEE023EE5	0x01003EE5	0x22000	Flash/	1Mbit.bin	
Option					
0x0	<				

The differences between File Checksum and Chip Checksum

File Checksum

It calculates the Checksum values for the programming file, and the calculation scope depends on the file size.

Chip Checksum

It calculates the checksum value of all partitions available in the programmer software. In other WORDs, it adds up the checksum values of the data stored in the memory partitions that can be programmed. Users can select which partitions to include in the Chip Checksum in the programmer software, as demonstrated in Example 2 below.

Example 1:

The programmed IC type is EEPROM, and its model name is ACE024C02. It only has one memory partition that can be programmed, called EEPROM, with a memory size of 256 bytes (which converts to hexadecimal as 0x100).

The programming file size is 128Byte (which converts to hexadecimal as 0x80).



岱鐠科技股份有限公司 DediProg Technology Co., Ltd.

4

Programming File (image1)] [Programmer Buffer	
Address offest (h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F	Address offest (h)	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F	
0x0	0x0		
0x10	0x10		
0x20	0x20		
0x30 image 1 data (0x0~0x7F)	0x30	image 1 data (0x0~0x7F)	
0x40 File Checksum	0x40		
0x50	0x50		
0x60	0x60		_ Chip Checksum
0x70	0x70		
	0x80		
	0x90		
	0xA0		
	0xB0	The remaining portion of the buffer that did not load with the file data will be the default value for the blank IC. For EEPROM, the	
	0xC0	default value is 0xFF.	
	0xD0		
	0xE0		
	0×E0]	

File Checksum Calculation Scope = File's own data (0x0~0x7F)

Chip Checksum Calculation Scope =

Data imported from the file to the buffer address (0x0~0x7F)

+ Blank Data (0xFF) of the Buffer Address (0x80~0xFF)

To put it another way, it calculates the checksum for the range of the buffer addresses (0x0 to 0xFF) that have been loaded.

Example 2:

The programmed IC type is SPI NOR Flash, and its model name is S25FL032P. It has two memory partitions that can be programmed, that are called

- 1. Flash; memory size is 4194304 Bytes (which converts to hexadecimal as 0x400000)
- 2. 512Bytes OTP; memory size is 512 Bytes (which converts to hexadecimal as 0x200)

The programming file size is 2097152 Bytes (which converts to hexadecimal as 0x200000)

			Pr	ogra	amm	ning	File	(ima	ge1)						
Address offest (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
0x0																
0x100000					in	nage					LFFFI	FF)				
							File	Ch	ecks	um						
0x1FFFFF																



岱鐠科技股份有限公司 DediProg Technology Co., Ltd.

5

File Checksum Calculation Scope = File's own data (0x0~ 0x1FFFF)

			P	rogr	amn	ner E	Buffe	r (Fla	ash)							
Address offest (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
0x0																
0x100000					in	ane	1 da	ata (l		.0v1	FFFF	E)				
						aye	i ua	atel (1	0.00	UX II		1)				
0x1FFFFF																
0x200000																
	_						- 6 41-				ي اماله				<i>c</i>	
0X300000															he fi R, th	
	"	ala	WIII I	eur	e ue		efaul								ix, u	
0X3FFFFF																

		P	rogr	amm	ner B	uttei	r (51	2ВУ	es C	NP)						
Address offest (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
0x0																
	8	Since	e no	file is									this	parti	tion	is
					bla	ink, t	the c	lefau	ult va	ilue i	is Ox	FF.				
0x1FF																

Chip Checksum = Flash(0x0~0x3FFFFF) + 512Bytes OTP(0x0~0x1FF)

Chip Checksum Calculation Scope =

Data imported from the file to the Buffer (Flash) Address (0x0~0x1FFFFF)

- + Blank Data (0xFF) of the Buffer (Flash) Address (0x200000~0x3FFFF)
- + Blank Data (0xFF) of the Buffer (512Bytes OTP) Address (0x0~0x1FF)

<pre>✓ Flash 2 ✓ 512Bytes OTP 3 Calculate Cancel</pre>	1				Save Log Clear Log
Indier Chip 0x2FB8 Optio 0xFFI	n 0x0FD640CE	File Size 0x200000	IC Partition Flash/	File Name 16Mbit.bin	>

Programmer software allows the user to select which Buffer Partition can be included in the Chip Checksum calculation. To do so, please follow the steps below:

- 1. Click the Chip button
- 2. Choose the partition that will be calculated in the chip checksum
- 3. Click the Calculate button
- Example 2-1

If the user selects only the Buffer (Flash) Partition, then only that partition will be included in the calculation of the Chip Checksum.



Option Checksum

Check Sum				
Chip	File CheckSum	File Size	IC Partition	File Name
0x778959AD	0x77875BAD	0x800000	Flash/	all.bin
Option				
0x2156				
	L			

The Option Checksum is calculated using the CRC-16 algorithm based on the option bytes in the Config setting. The default value for the Option Checksum is 0xFFFF.

II. Checksum Algorithm Types

The programmer software supports the below Checksum algorithm types.

When loading a file, you can choose the type of algorithm you want to use to calculate the File Checksum and Chip Checksum.

Load File			
File1 +			
FileFormat: Binary(*.bin)	~		
FileChecksum: ByteAcc	~	ChipCheckSum:	ByteAcc 🗸
FileOffset: ByteAcc ByteAcc_2sCor	nplement	StartProgAddr:	ByteAcc ByteAcc_2sComplement Crc8 CCITT
FilePath: Crc8_CCITT	DCTEST	ProgramLen:	Crc16
Disable auto Disable auto MD5_UseSoury WoldAccLE WoldAccBE	æFile	FillUnusedByte:	Crc32 MD5 MD5_UseSourceFile WoldAccLE WoldAccBE
ShowFileList DWoldAccLE DWoldAccBE SHA1 SHA256		Log Window	DWoldAccLE DWoldAccBE SHA1 SHA256

Note: For SPI NAND and Parallel NAND chips, the Chip Checksum only supports ByteACC.



Take this file as an example for calculation.

B DOCTEST Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 解碼的文字 00000000 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF ..."3DD撮w^{™™™}»ÌÝiÿ

ByteACC (Byte sum (x8))

Algorithm Description:

Sum the data bytes byte by byte and disregard any carry beyond 32 bits.

Result:

00 + 11 + 12 + 33 + 44 + 55 + 66 + 77 + 88 + 99 + AA + BB + CC + DD + EE + FF = 0x7F8

ByteACC_2s complement

Algorithm Description:

Sum the data bytes byte-by-byte, and take the two's complement out of the sum. Any carry beyond 32 bits will be ignored.

Result: 0xFFFFF808

CRC8_CCITT

Algorithm Description:

Data are summed by bytes to sum by bytes to WORD using standard CRC-8 algorithm with polynomial x^8+x^2+x+1 , (0x7), init value 0, and XOR out 0

Result: 0x000004D

CRC16

Algorithm Description:

Data are summed by bytes to sum by bytes to WORD using standard CRC-16 algorithm with polynomial $x^{16}+x^{12}+x^{5}+1$ (0x8408), init value 0, and XOR out 0

Result: 0x00007842



CRC32

Algorithm Description:

Buffer data are summed by bytes to DWORD using standard CRC-32 algorithm with 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 (0x04C11DB7), init value 0xFFFFFFF, and XOR out 0xFFFFFFF

Result: 0x8407759B

MD5

Algorithm Description:

The MD5 Message-Digest Algorithm is a widely used cryptographic hash function for passwords that can generate 128-bit (16-byte) hash values.

Result: 0x6E8311168EE16D6AA1AA48C64145003C

MD5_UseSourceFile

Algorithm Description:

Also use the MD5 algorithm, but the Chip Checksum will be calculated based on the programming file rather than the loaded Buffer data of the programmer.

Example:

The IC that is going to be programmed only has one available programmable memory partition. The memory size is 128 Bytes (equivalent to 0x80 in hexadecimal), and the programming file size is 256 Bytes (equivalent to 0x100 in hexadecimal).



MD5 UseS

岱鐠科技股份有限公司 DediProg Technology Co., Ltd.

9

				Р	rogr	amn	ning	File	(ima	ge1)							
	Address offest (h)	00 ()1	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
	0x0																
	0x10																
	0x20																
	0x30						ima	1	dat	a (0)	0~0	x7F)	、 、				
	0x40						IIIId	iye i	uat	a (0)	(0.40	,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	,				
5_	0x50																
SourceFile	0x60																
	0x70																
	0x80																
	0x90																
	0xA0																
	0xB0	The	data	a lo	cate	d at	addr	esse	es Ox	(80 t	o 0x	FF ir	n the	file	exce	eds	the
	0xC0		si	ze o	of the	e bu	fer a	and \	vill n	ot be	e loa	ded	into	the b	ouffe	r.	
	0xD0																
	0×E0																

				P	rogra	amm	er B	uffei								
Address offest (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
0x0																
0x10	Ι															
0x20	Ī															
0x30	1					ima	ao 1	dat	- (Ov	0~0	v7E)					
0x40	1	image 1 data (0x0~0x7F)														
0x50	1															
0x60	I															
0x70	1															

Result: 0x6E8311168EE16D6AA1AA48C64145003C

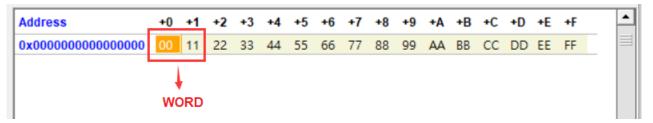
WoldAccLE (WORD)

Algorithm Description:

0xF0

The actual name is WORD sum Little-Endian (x16). Sum the data WORD-by-WORD; any carry beyond 32 bits will be ignored. The technical term Little-Endian means that the checksum is calculated based on the WORD read from the programmer Buffer or the programming file in Little-Endian mode (The highest byte of the programming file or the programmer buffer data will be read and placed at the highest address).

Assume the programmer software Buffer has the data loaded as shown below:



Then, there will be eight sets of WORDs:

0x1100 in Little-Endian mode, the WORD will be read as 0x1100 0x3322 in Little-Endian mode, the WORD will be read as 0x3322 0x5544 in Little-Endian mode, the WORD will be read as 0x5544 0x7766 in Little-Endian mode, the WORD will be read as 0x7766 0x9988 in Little-Endian mode, the WORD will be read as 0x9988 0xBBAA in Little-Endian mode, the WORD will be read as 0xBBAA 0xDDCC in Little-Endian mode, the WORD will be read as 0xDDCC



岱鐠科技股份有限公司 DediProg Technology Co., Ltd.

V1.0

10

0xFFEE in Little-Endian mode, the WORD will be read as 0xFFEE The total Checksum of the above eight sets of WORDs = 0x000443B8

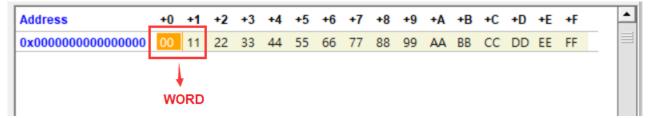
Result: 0x000443B8

WoldAccBE (WORD)

Algorithm Description:

The actual name is WORD sum Big-Endian (x16). The data is summed up WORD-by-WORD byte by byte, and any carry beyond 32 bits will be ignored. Technical term Big-Endian means that the Checksum is calculated based on the WORD read from the programmer buffer or the programming file in the Big-Endian mode (The highest byte of the programming file or the programmer Buffer data will be read and placed at the lowest address).

Assume the programmer software Buffer has the data loaded as shown below:



Then, there will be eight sets of WORDs:

0x1100 in Big-Endian mode, the WORD will be read as 0x0011 0x3322 in Big-Endian mode, the WORD will be read as 0x2233 0x5544 in Big-Endian mode, the WORD will be read as 0x4455 0x7766 in Big-Endian mode, the WORD will be read as 0x6677 0x9988 in Big-Endian mode, the WORD will be read as 0x8899 0xBBAA in Big-Endian mode, the WORD will be read as 0xAABB 0xDDCC in Big-Endian mode, the WORD will be read as 0xCCDD 0xFFEE in Big-Endian mode, the WORD will be read as 0xEEFF The total Checksum of the above eight sets of WORDs = 0x0003BC40

Result: 0x0003BC40



.0

11

DWoldAccLE (DWORD)

Algorithm Description:

The actual name is DWORD sum Little-Endian (x16). Sum the data DWORD-by-DWORD; any carry beyond 32 bits will be ignored. The technical term Little-Endian means that the checksum is calculated based on the DWORD read from the programmer Buffer or the programming file in Little-Endian mode (The highest byte of the programming file or the programmer buffer data will be read and placed at the highest address).

Assume the programmer software Buffer has the data loaded as shown below:

/pe																	×
EEPROM																	
Address	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F	-
0x000000000000000000	00	11	22	33	44	55	66	77	88	99	AA	BB	CC	DD	EE	FF	
			♦		•												
		DV	VORI	D													

Then, there will be four sets of DWORDs:

0x33221100 in Little-Endian mode, the DWORD will be read as 0x33221100 0x77665544 in Little-Endian mode, the DWORD will be read as 0x77665544 0xBBAA9988 in Little-Endian mode, the DWORD will be read as 0xBBAA9988 0xFFEEDDCC in Little-Endian mode, the DWORD will be read as 0xFFEEDDCC The total Checksum of the above four sets of DWORDs = 0x26621DD98 Since any carry beyond 32 bits will be ignored, the 33rd bit and beyond will also be ignored. Therefore, the final Checksum = 0x6621DD98.

Result: 0x6621DD98



DWoldAccBE (DWORD)

Algorithm Description:

The actual name is DWORD sum Big-Endian (x16). The data is summed up DWORD-by-DWORD, and any carry beyond 32 bits will be ignored. Technical term Big-Endian means that the Checksum is calculated based on the DWORD read from the programmer buffer or the programming file in the Big-Endian mode (The highest byte of the programming file or the programmer Buffer data will be read and placed at the lowest address).

Assume the programmer software Buffer has the data loaded as shown below:

/ре																	>
● EEPROM																	
Address	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F	
0x0000000000000000	00	11		33										DD			-
			1		•												
		DV	VORI	D													

Then, there will be four sets of DWORDs:

0x33221100 in Big-Endian mode, the DWORD will be read as 0x00112233 0x77665544 in Big-Endian mode, the DWORD will be read as 0x44556677 0xBBAA9988 in Big-Endian mode, the DWORD will be read as 0x8899AABB 0xFFEEDDCC in Big-Endian mode, the DWORD will be read as 0xCCDDEEFF The total Checksum of the above four sets of DWORDs = 0x199DE2264 Since any carry beyond 32 bits will be ignored, the 33rd bit and beyond will also be ignored. Therefore, the final Checksum = 0x99DE2264

Result: 0x99DE2264



SHA1

Algorithm Description:

SHA-1 (Secure Hash Algorithm 1) is a type of cryptographic hash function designed by the US National Security Agency and published as a Federal Information Processing Standard (FIPS) by the US National Institute of Standards and Technology (NIST). SHA-1 can generate a 160-bit (20-byte) hash value known as a message digest, which is typically represented as 40 hexadecimal digits.

Result: 0x739E0E8490EACBCB2EA11D4A5DBEFBAE888B092E

SHA256

Algorithm Description:

SHA-256" is the English abbreviation for "Secure Hash Algorithm 256-bit". It is a cryptographic hash function that enhances security through encryption. SHA-256 can generate a 256-bit (32-byte) hash value known as a message digest, which is typically represented as 64 hexadecimal digits

Result:

0xA8FAED6ABBF35C12A4B26E40F6FEB19D736D90045C83B9F9A31F638D323E6811

III. Additional Information

What are Big-Endian and Little-Endian

Big-Endian: When data is imported into the buffer of the programmer software, the highest byte of the data will be placed at the lowest address of the buffer.

Little-Endian: When data is imported into the buffer of the programmer software, the highest byte of the data will be placed at the highest address of the buffer.

For example, if there is a 32-bit data such as 0x11223344, when it is input by a Big-Endian system into the buffer of the programmer software, it will be placed in the buffer according to the rule shown in the image below:



Big-Endian The highest bytes of the data will be placed at the lowest address of the Buffer.

0x11223344							
		1					
		· · · · · · · · · · · · · · · · · · ·					
	Programmer Buffer						
Address offest (h)	00 (Buffer lowest address)	01	02	03 (Buffer highest address)			
0x0	0x11	0x22	0x33	0x44			

However, if it is input by a Little-Endian system, it will be placed in the buffer in reverse order, as shown below:

Little-Endian The highest bytes of the data will be placed at the highest address of the Buffer.

0x11223344

		•					
Programmer Buffer							
Address offest (h)	00 (Buffer lowest address)	01	02	03 (Buffer highest address)			
0x0	0x44	0x33	0x22	0x11			



IV. Revision History

Date	Version	Description
2023/3/16	1.0	Initial Release

DediProg Technology Co., Ltd. (Headquarter)

No. 142, Ankang Rd., Neihu Dist., Taipei City, Taiwan, R.O.C 114044 TEL: 886-2-2790-7932 FAX: 886-2-2790-7916

Technical Support: support@dediprog.com Sales Support: sales@dediprog.com

Information furnished is believed to be accurate and reliable. However, DediProg assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties which may result from its use. Specifications mentioned in this publication are subject to change without notice.

This publication supersedes and replaces all information previously supplied.

All rights reserved Printed in Taiwan.