regregex.bbcmicro.net

Catalogue of parametrised CRC algorithms

Conforming to the Rocksoft™ Model CRC Algorithm

For an explanation of the format of each record, please consult "A Painless Guide" (reference below.)

If your browser has problems printing this page, download the PDF version.

Catalogue of CRC algorithms with parameters according to the Rocksoft™ model.

		Evidence			
		I	**II**	**III**	**IV**
Rocksoft™ model record	**Notes**	**Primary documents**	**Implementations**	**Other documents**	**Codeword sets**
`Name : "CRC-3/ROHC"` `Width : 3` `Poly : 3` `Init : 7` `RefIn : True` `RefOut : True` `XorOut : 0` `Check : 6`	–	IETF RFC 3095 • Definition: Width, Poly, Init	–	Andreas Vernersson *et al.* rohc 1.0 module rohc-1.0/rohc /src/c_util.c • Code: C	–
`Name : "CRC-4/ITU"` `Width : 4` `Poly : 3` `Init : 0` `RefIn : True` `RefOut : True` `XorOut : 0` `Check : 7`	–	ITU-T Recommendation G.704 • Full mathematical description • Shift register diagram	–	–	–
`Name : "CRC-5/EPC"` `Width : 5` `Poly : 09` `Init : 09` `RefIn : False` `RefOut : False` `XorOut : 00` `Check : 00`	Residue = 0x00 Used in standardised RFID tags.	EPCglobal Inc™ UHF Class 1 Gen 2 Air Interface Protocol Standard v. 1.2.0 • Definition: Width, Poly, Init, Residue, RefIn • Shift register circuit diagram	–	–	H.B.Kang *et al.* High Security FeRAM-Based EPC C1G2 UHF (860 MHz-960 MHz) Passive RFID Tag Chip • 1 codeword www.lammertbies.nl Forum topic 1330 • 1 codeword
`Name : "CRC-5/ITU"` `Width : 5` `Poly : 15` `Init : 00` `RefIn : True` `RefOut : True` `XorOut : 00` `Check : 07`	Residue = 0x00	ITU-T Recommendation G.704 • Full mathematical description • Shift register diagram	–	–	–
`Name : "CRC-5/USB"` `Width : 5` `Poly : 05` `Init : 1F` `RefIn : True` `RefOut : True` `XorOut : 1F` `Check : 19`	Residue = 0x0C (internal form)	–	–	Anonymous "Cyclic Redundancy Checks in USB" (Draft) Courtesy of USB Implementers Forum, Inc. • Definition: Width, Poly, Init, XorOut • Code: Perl	See ← • 4 codewords
`Name : "CRC-6/DARC"` `Width : 6` `Poly : 19` `Init : 00` `RefIn : True` `RefOut : False` `XorOut : 00` `Check : 19`	See section 12 for details of the transmission order.	ETSI EN 300 751 V1.2.1 (2003-01) (registration required) • Definition: Width, Poly, RefIn, RefOut	–	–	See ← • 3 codewords
`Name : "CRC-6/ITU"` `Width : 6` `Poly : 03` `Init : 00` `RefIn : True` `RefOut : True` `XorOut : 00` `Check : 06`	–	ITU-T Recommendation G.704 • Full mathematical description • Shift register diagram	–	–	–
`Name : "CRC-7"` `Width : 7` `Poly : 09` `Init : 00`	Used in the MultiMediaCard interface.	JEDEC Standard No. JESD84-A441 (registration required) • Full definition	–	–	–

`RefIn  : False`
`RefOut : False`
`XorOut : 00`
`Check  : 75`
		• Shift register diagram			

`Name   : "CRC-7/ROHC"`
`Width  : 7`
`Poly   : 4F`
`Init   : 7F`
`RefIn  : True`
`RefOut : True`
`XorOut : 00`
`Check  : 53`
	–	IETF RFC 3095
		• Definition: Width, Poly, Init
			–	Andreas Vernersson *et al.*
				rohc 1.0 module rohc-1.0/rohc /src/c_util.c
				• Code: C
					–

`Name   : "CRC-8"`
`Width  : 8`
`Poly   : 07`
`Init   : 00`
`RefIn  : False`
`RefOut : False`
`XorOut : 00`
`Check  : F4`
	Residue = 0x00	–	• Checksum calculator, PVL Team	John Milios
				USAR Systems
				"CRC-8 firmware implementations for SMBus"
				SBS IF DevCon Japan 1999
				• Worked example
				• Code: Z80 assembler
					Libav 0.7.4 module libav-0.7.4/libavutil/crc.c
					• 1 codeword

`Name   : "CRC-8/DARC"`
`Width  : 8`
`Poly   : 39`
`Init   : 00`
`RefIn  : True`
`RefOut : True`
`XorOut : 00`
`Check  : 15`
	The single codeword is supported by the codewords confirming CRC-6/DARC, defined identically apart from Poly in the same standard. See section 12 for details of the transmission order.	ETSI EN 300 751 V1.2.1 (2003-01) (registration required)
		• Definition: Width, Poly, RefIn, RefOut
			–	–	See ←
					• 1 codeword

`Name   : "CRC-8/I-CODE"`
`Width  : 8`
`Poly   : 1D`
`Init   : FD`
`RefIn  : False`
`RefOut : False`
`XorOut : 00`
`Check  : 7E`
	Residue = 0x00	Philips Semiconductors SL2 ICS11 Product Specification (via NXP)
		• Definition: Width, Poly, Init
		• Code: C
		• Example (as code trace)
			–	–	–

`Name   : "CRC-8/ITU"`
`Width  : 8`
`Poly   : 07`
`Init   : 00`
`RefIn  : False`
`RefOut : False`
`XorOut : 55`
`Check  : A1`
	Used as the Asynchronous Transfer Mode Header Error Control sequence (ATM HEC).
	Single bit errors in the 4-byte ATM header can be automatically corrected.
		ITU-T Recommendation I.432.1
		Broadband Forum Technical Committee User-Network Interface Specification 3.0 (RTF)
		• Full mathematical description
			–	–	ITU-T Recommendation I.432.1
					• 2 codewords (trivial)

`Name   : "CRC-8/MAXIM"`
`Alias  : "DOW-CRC"`
`Width  : 8`
`Poly   : 31`
`Init   : 00`
`RefIn  : True`
`RefOut : True`
`XorOut : 00`
`Check  : A1`
	Residue = 0x00
	Used in Maxim 1-Wire® device registration numbers.
	AN27 contains a fast, table-less algorithm in 8051 assembler; compare Appendix M.
		Maxim Integrated Products DS1921G Datasheet
		• Definition: Width, Poly, Init, XorOut
		Application Note 27 (PDF, HTML)
		• Code: 8051 assembler
		• Shift register diagram
		• Worked example
			–	–	Maxim iButton® datasheets
					• 15 codewords
					• 2 **non-matching** codewords (DS1990A, DS1993)

`Name   : "CRC-8/ROHC"`
`Width  : 8`
`Poly   : 07`
`Init   : FF`
`RefIn  : True`
`RefOut : True`
`XorOut : 00`
`Check  : D0`
	–	IETF RFC 3095
		• Definition: Width, Poly, Init, XorOut
			–	Andreas Vernersson *et al.*
				rohc 1.0 module rohc-1.0/rohc /src/c_util.c
				• Code: C
					–

`Name   : "CRC-8/WCDMA"`
`Width  : 8`
`Poly   : 9B`
`Init   : 00`
`RefIn  : True`
`RefOut : True`
`XorOut : 00`
`Check  : 25`
	–	–	–	Andrew Richardson
				WCDMA Design Handbook
				Cambridge University Press
				ISBN 0-521-82815-5
				• Definition: Width, Poly, Residue
				• Shift register diagram
				Philip Koopman, Tridib
					www.lammertbies.nl Forum topic 1431
					• 46 codewords
					• 1 **non-matching** codeword (0 0 18 104)

				Chakravarty Cyclic Redundancy Code (CRC) Polynomial Selection for Embedded Networks • Assessment of polynomial performance (as 0xCD or WCDMA-8)	

`Name : "CRC-10"`
`Width : 10`
`Poly : 233`
`Init : 000`
`RefIn : False`
`RefOut : False`
`XorOut : 000`
`Check : 199`
	Used in Asynchronous Transfer Mode AAL 3/4 and OAM cells.
	Note there are 6 padding zero bits between each of the 46-byte information fields and their respective CRCs.
		ITU-T Recommendation I.610
		Broadband Forum Technical Committee Traffic Management Specification 2.1
		• Full mathematical description
			–	Charles M. Heard
				Generating and Checking CRC-10 in ATM AAL 3/4 or OAM Cells
				(via the Internet Archive)
				• Definition: Poly
				• Code: C
					ITU-T Recommendation I.610
					• 2 codewords
					Charles M. Heard
					Generating and Checking CRC-10 in ATM AAL 3/4 or OAM Cells
					(via the Internet Archive)
					• 6 additional codewords

`Name : "CRC-11"`
`Width : 11`
`Poly : 385`
`Init : 01A`
`RefIn : False`
`RefOut : False`
`XorOut : 000`
`Check : 5A3`
	–	FlexRay Consortium
		FlexRay Communications System Protocol Specification
		Version 3.0.1
		• Definition: Width, Poly, Init, RefOut
		• Pseudocode
			–	–	FlexRay Consortium
					FlexRay Protocol Conformance Test Specification
					Version 3.0.1
					• 1 codeword
					Robert Bosch GmbH
					E-Ray FlexRay IP Module
					Application Note 2
					• 2 codewords
					• 1 partial codeword
					• Researched by Vivek Rajan. See Appendix H

`Name : "CRC-12/3GPP"`
`Width : 12`
`Poly : 80F`
`Init : 000`
`RefIn : False`
`RefOut : True`
`XorOut : 000`
`Check : DAF`
	The reflection of the CRC against the payload is unusual but explicit.
	Thanks to markw_be at Lammert Bies' forum for the reference.
		3rd Generation Partnership Project (3GPP)
		TS 25.212 V10.1.0 (2010-12) (zipped MS Word document)
		— ETSI TS 125 212 V10.1.0 (2011-05) (registration required)
		• Mathematical description, defining Width, Poly, Init, Residue
		• Attachment relation, defining RefIn ^RefOut
			–	–	–

`Name : "CRC-12/DECT"`
`Alias : "X-CRC-12"`
`Width : 12`
`Poly : 80F`
`Init : 000`
`RefIn : False`
`RefOut : False`
`XorOut : 000`
`Check : F5B`
	–	ETSI EN 300 175-3 V2.4.0 (2011-12) (registration required)
		• Full mathematical description
			–	–	–

`Name : "CRC-14/DARC"`
`Width : 14`
`Poly : 0805`
`Init : 0000`
`RefIn : True`
`RefOut : True`
`XorOut : 0000`
`Check : 082D`
	The single codeword is supported by the codewords confirming CRC-6/DARC, defined identically apart from Poly in the same standard.
	The codeword, representing the "transmitted bits", is clearly reflected ASCII. The direct ASCII would be the input to this algorithm.
	See section 12 for details of the transmission order.
		ETSI EN 300 751 V1.2.1 (2003-01) (registration required)
		• Definition: Width, Poly, RefIn, RefOut
			–	–	See ←
					• 1 codeword

`Name : "CRC-15"`
`Width : 15`
`Poly : 4599`
`Init : 0000`
`RefIn : False`
`RefOut : False`
`XorOut : 0000`
`Check : 059E`
	–	Robert Bosch GmbH
		CAN 2.0 Specification
		• Full definition (except Check)
		• Pseudocode
			–	–	–

`Name   : "ARC"`
`Alias  : "CRC-16"`
`Alias  : "CRC-IBM"`
`Alias  : "CRC-16/ARC"`
`Alias  : "CRC-16/LHA"`
`Width  : 16`
`Poly   : 8005`
`Init   : 0000`
`RefIn  : True`
`RefOut : True`
`XorOut : 0000`
`Check  : BB3D`
	–	–	• ARC 5.20
			• LHA 2.55E
			• ZOO 2.1a
			• CRC calculator, Lammert Bies
			• Checksum calculator, PVL Team (as CRC16_arc)
				Ross N. Williams
				"A Painless Guide to CRC Error Detection Algorithms"
				Altera Corporation
				crc MegaCore Function Data Sheet
				(via the Internet Archive)
				• All parameters including Check
					–

`Name   : "CRC-16/AUG-CCITT"`
`Alias  : "CRC-16/SPI-FUJITSU"`
`Width  : 16`
`Poly   : 1021`
`Init   : 1D0F`
`RefIn  : False`
`RefOut : False`
`XorOut : 0000`
`Check  : E5CC`
	Init value is equivalent to an augment of 0xFFFF prepended to the message.	Fujitsu Semiconductor
		FlexRay ASSP MB88121B User's Manual
		• Definition: Width, Poly, Init
			• CRC calculator, Lammert Bies
			• Checksum calculator, PVL Team
				Berndt M. Gammel
				Matpack documentation "Crypto – Codes"
				• All parameters including Check
					–

`Name   : "CRC-16/BUYPASS"`
`Alias  : "CRC-16/VERIFONE"`
`Width  : 16`
`Poly   : 8005`
`Init   : 0000`
`RefIn  : False`
`RefOut : False`
`XorOut : 0000`
`Check  : FEE8`
	Reported for the multi-threaded portion of the Buypass transaction processing network.	Verifone, Inc.
		TCLOAD Reference Manual
		• Definition: Poly
		• CRC byte order, implying RefIn and RefOut
			• Checksum calculator, PVL Team	Emil Lenchak
				Texas Instruments, Inc.
				CRC Implementation With MSP430
				• All parameters including Check
					Libav 0.7.4 module libav-0.7.4/libavutil/crc.c
					• 1 codeword
					www.lammertbies.nl
					Forum topic 530
					• 2 codewords

`Name   : "CRC-16/CCITT-FALSE"`
`Width  : 16`
`Poly   : 1021`
`Init   : FFFF`
`RefIn  : False`
`RefOut : False`
`XorOut : 0000`
`Check  : 29B1`
	An algorithm commonly misidentified as CRC-CCITT. For the true CCITT algorithm see KERMIT.
	For the later ITU-T algorithm see X.25.
		Western Digital Corporation
		FD 179X-02 datasheet
		• Definition: Width, Poly, Init
			Floppy disc formats:
			• IBM 3740 (FM, e.g. Acorn DFS)
			• ISO/IEC 8860-2:1987 (DOS 720K)
			• ISO/IEC 9529-2:1989 (DOS 1.4M)
			• CRC calculator, Lammert Bies
			• Checksum calculator, PVL Team
				Ross N. Williams
				"A Painless Guide to CRC Error Detection Algorithms"
				• All parameters (except Check)
				Berndt M. Gammel
				Matpack documentation "Crypto – Codes"
				• All parameters including Check
				Altera Corporation
				crc MegaCore Function Data Sheet
				(via the Internet Archive)
				• All parameters including Check
					–

`Name   : "CRC-16/DDS-110"`
`Width  : 16`
`Poly   : 8005`
`Init   : 800D`
`RefIn  : False`
`RefOut : False`
`XorOut : 0000`
`Check  : 9ECF`
	Init value is equivalent to an augment of 0xFFFF prepended to the message.
	Used in the ELV DDS 110 function generator.
	In the ELV article, control characters are escaped according to the description, so the published codeword expands to
	02 00 10 82 00 73 10 82 FE F7.
		ELV Elektronik AG
		Software-Schnittstelle der Funktionsgeneratoren DDS 10/DDS 110
		• Definition: Width, Poly, CRC byte order
			–	www.lammertbies.nl
				Forum topic 1372
				• All parameters (except Check; solved by Gammatester)
					ELV Elektronik AG
					Software-Schnittstelle der Funktionsgeneratoren DDS 10/DDS 110
					• 1 codeword
					www.lammertbies.nl
					Forum topic 1372
					• 3 codewords

`Name   : "CRC-16/DECT-R"`
`Alias  : "R-CRC-16"`
`Width  : 16`
`Poly   : 0589`
`Init   : 0000`
`RefIn  : False`
`RefOut : False`
`XorOut : 0001`
`Check  : 007E`
	Used in DECT A-fields.	ETSI EN 300 175-3 V2.4.0 (2011-12)
		(registration required)
		• Full mathematical description
		• Performance of polynomial
			• pycrc 0.7.3	The Comprehensive GNU Radio Network
				GR_DECT - DECT receiver
				• Application of algorithm to DECT software
					–

`Name   : "CRC-16/DECT-X"`
`Alias  : "X-CRC-16"`
`Width  : 16`
`Poly   : 0589`
`Init   : 0000`
`RefIn  : False`
`RefOut : False`
`XorOut : 0000`
`Check  : 007F`
	The single codeword is supported by the implementation confirming CRC-16/DECT-R, defined identically apart from XorOut in the same standard.
	Used in DECT B-fields.
		ETSI EN 300 175-3 V2.4.0 (2011-12)
		(registration required)
		• Full mathematical description
		• Performance of polynomial
			–	–	StackOverflow
					Submitted question
					• 1 codeword

`Name   : "CRC-16/DNP"`
`Width  : 16`
`Poly   : 3D65`
`Init   : 0000`
`RefIn  : True`
`RefOut : True`
	–	–	• CRC calculator, Lammert Bies	–	–

XorOut : FFFF Check : EA82					
Name : "CRC-16/EN-13757" Width : 16 Poly : 3D65 Init : 0000 RefIn : False RefOut : False XorOut : FFFF Check : C2B7	Used in the Wireless M-Bus protocol for remote meter reading. In the Capt²web Web interface packet view, the bytes of the A and M fields are displayed in reverse, compared to transmission order.	–	• Capt²web sniffer, Steinbeis Transfer Center Embedded Design and Networking	Patrick Seem Texas Instruments, Inc. AN067: Wireless MBUS Implementation with CC1101 and MSP430 • Definition: Width, Poly, Init, XorOut • Describes synchronous transfer with MSBs sent first, implying RefIn and RefOut Dr.-Ing. Thomas Weinzierl Weinzierl Engineering GmbH Stack Implementation for KNX-RF • Radio link corresponds to Link A in AN067 • Definition: Poly • CRC byte order, implying RefIn and RefOut control.com Forum post • Width, Poly cited for ISO/IEC 60870-5-2	www.lammertbies.nl Forum topic 925 • 1 codeword and DNP poly cited for EN 13757 Forum topic 1315 • 1 codeword
Name : "CRC-16/GENIBUS" Alias : "CRC-16/EPC" Alias : "CRC-16/I-CODE" Alias : "CRC-16/DARC" Width : 16 Poly : 1021 Init : FFFF RefIn : False RefOut : False XorOut : FFFF Check : D64E	Used in standardised RFID tags. Presented high byte first. Residue = 0x1D0F	EPCglobal Inc™ UHF Class 1 Gen 2 Air Interface Protocol Standard v. 1.2.0 • Definition: Width, Poly, Init, Residue, RefIn • Shift register circuit diagram Philips Semiconductors SL2 ICS11 Product Specification (via NXP) • Definition: Width, Poly, Init • Code: C • Example (as code trace) ETSI EN 300 751 V1.2.1 (2003-01) (registration required) • Definition: Width, Poly	• Checksum calculator, PVL Team	www.lammertbies.nl Forum topic 216 • Quoted definition for GENIbus: Width, Poly, Init, XorOut Forum topic 907 • Reported definition for TI Tag-It: full (except Check)	EPCglobal Inc™ UHF Class 1 Gen 2 Air Interface Protocol Standard v. 1.2.0 • 7 codewords www.lammertbies.nl Forum topic 216 • 2 codewords cited for GENIbus Forum topic 907 • 4 codewords cited for TI Tag-It ETSI EN 300 751 V1.2.1 (2003-01) (registration required) • 1 codeword
Name : "CRC-16/MAXIM" Width : 16 Poly : 8005 Init : 0000 RefIn : True RefOut : True XorOut : FFFF Check : 44C2	Residue = 0xB001.	Maxim Integrated Products DS1921G Datasheet • Definition: Width, Poly, Init, XorOut Application Note 27 (PDF, HTML) • Definition: Init • Code: 8051 assembler • Shift register diagram • Worked example	–	–	–
Name : "CRC-16/MCRF4XX" Width : 16 Poly : 1021 Init : FFFF RefIn : True RefOut : True XorOut : 0000 Check : 6F91	Nibble oriented. For byte wide algorithms swap nibbles of each byte. CRC presented low nibble first.	Youbok Lee, PhD Microchip Technology Inc. "CRC Algorithm for MCRF45X Read/Write Device" • Definition: Width, Poly (reverse form), Init • Shift register diagram	–	Piers Desrochers "A quick guide to CRC" • Description • Worked example	W.H.Press *et al.* *Numerical Recipes in C* (embedded content) p.898 • 2 codewords www.lammertbies.nl Forum topic 578 • 2 codewords

		• Flowchart • Worked example • Code: C			

`Name   : "CRC-16/RIELLO"`
`Width  : 16`
`Poly   : 1021`
`Init   : B2AA`
`RefIn  : True`
`RefOut : True`
`XorOut : 0000`
`Check  : 63D0`
	Reported for a Riello Dialog UPS.	–	–	www.lammertbies.nl
				Forum topic 1305
				• Definition: Poly, Init
				• Algorithm reported to be CRC-CCITT, implying RefIn and RefOut.
					See ←
					• 1 codeword

`Name   : "CRC-16/T10-DIF"`
`Width  : 16`
`Poly   : 8BB7`
`Init   : 0000`
`RefIn  : False`
`RefOut : False`
`XorOut : 0000`
`Check  : D0DB`
	Used in the SCSI Data Integrity Field.
	Polynomial selected for its "proper" behaviour by Pat Thaler.
	XorOut = 0xBADB is proposed to mark known bad blocks.
		INCITS Working Group T10:
		Gerald Houlder
		End-to-End Data Protection Proposal
		• Definition: Poly, Init
		• Shift register diagram
		George O. Penokie
		Simplified End-to-End Data Protection
		• Full mathematical description
		Weber & Lohmeyer
		Minutes of Data Integrity Study Group - Aug 19-20, 2003
		Item 4.6
		• Definition: Init, XorOut
		• Acceptance of previous two documents
			–	Linux 2.6.31 module
				lib/crc-t10dif.c
				• Code: C
					INCITS Working Group T10:
					Gerald Houlder
					End-to-End Data Protection Proposal
					• 1 **non-matching** codeword with Init = 0xFFFF
					George O. Penokie
					Simplified End-to-End Data Protection
					• 5 codewords

`Name   : "CRC-16/TELEDISK"`
`Width  : 16`
`Poly   : A097`
`Init   : 0000`
`RefIn  : False`
`RefOut : False`
`XorOut : 0000`
`Check  : 0FB3`
	Used in the Teledisk disc archive format.
	DECnet and CRCK allegedly use a Sick-type algorithm but with this polynomial.
		–	• wteledsk v1.01, Will Kranz	Will Kranz
				wteledsk 1.0.1 module tdcrc.c
				• Code: C
					–

`Name   : "CRC-16/TMS37157"`
`Width  : 16`
`Poly   : 1021`
`Init   : 89EC`
`RefIn  : True`
`RefOut : True`
`XorOut : 0000`
`Check  : 26B1`
	–	Texas Instruments, Inc.
		TMS37157 datasheet
		• Full definition (except Check)
		• Shift register diagram
		• Flowchart
			–	StackOverflow
				Submitted answer
				• Code: C
					TI E2E™ Community
					Forum post
					• 1 codeword
					StackOverflow
					Submitted question
					• 2 codewords

`Name   : "CRC-16/USB"`
`Width  : 16`
`Poly   : 8005`
`Init   : FFFF`
`RefIn  : True`
`RefOut : True`
`XorOut : FFFF`
`Check  : B4C8`
	CRC appended low byte first.
	Residue = 0x800D
		–	–	Anonymous
				"Cyclic Redundancy Checks in USB" (Draft)
				Courtesy of USB Implementers Forum, Inc.
				• Definition: Width, Poly, Init, XorOut
				• Code: Perl
					See ←
					• 2 codewords

`Name   : "CRC-A"`
`Width  : 16`
`Poly   : 1021`
`Init   : C6C6`
`RefIn  : True`
`RefOut : True`
`XorOut : 0000`
`Check  : BF05`
	Used in contactless IC cards.	ISO/IEC FCD 14443-3
		• Definition: Init, XorOut
		• Citation for rest of algorithm: ISO/IEC 13239 (see X.25)
			–	–	See ←
					• 2 codewords

`Name   : "KERMIT"`
`Alias  : "CRC-16/CCITT"`
`Alias  : "CRC-16/CCITT-TRUE"`
`Alias  : "CRC-CCITT"`
`Width  : 16`
`Poly   : 1021`
`Init   : 0000`
`RefIn  : True`
`RefOut : True`
`XorOut : 0000`
`Check  : 2189`
	Kermit implements the true CCITT algorithm (according to *Numerical Recipes*, "Crypto - Codes" and others).
	V.41 is endianness-agnostic, referring only to bit sequences, but the CRC appears reflected when used with LSB-first modems.
	Ironically, the unreflected form is used in XMODEM.
	For the algorithm often misidentified as CCITT, see
		ITU-T Recommendation V.41
		• Full mathematical description
		• Shift register diagrams
		Frank da Cruz
		Kermit Protocol Manual, Sixth Edition (plain text)
		• Full definition (except
			• CRC calculator, Lammert Bies
			• Checksum calculator, PVL Team
				W.H.Press *et al.*
				Numerical Recipes in C (embedded content)
				p.898
				• All parameters (except Check)
				• Pseudocode
					See ←
					• 2 codewords

	CCITT-FALSE. For the later ITU-T algorithm see X.25.	Check) • Pseudocode			
Name : "MODBUS" Width : 16 Poly : 8005 Init : FFFF RefIn : True RefOut : True XorOut : 0000 Check : 4B37	CRC presented low byte first.	MODICON Inc. Modbus Protocol Reference Guide • Algorithm • Code: C	• CRC calculator, Lammert Bies • CRC calculator, Ondřej Karas	Control.com Forum post • Code: ObjectPascal	–
Name : "X-25" Alias : "CRC-16/IBM-SDLC" Alias : "CRC-16/ISO-HDLC" Alias : "CRC-B" Width : 16 Poly : 1021 Init : FFFF RefIn : True RefOut : True XorOut : FFFF Check : 906E	HDLC is defined in ISO/IEC 13239. CRC_B is defined in ISO/IEC 14443-3. Residue = 0xF0B8	ITU-T Recommendations T.30, V.42, X.25 • Full mathematical description IETF RFC 1171 Appendix B • Code: C	–	W.H.Press *et al.* *Numerical Recipes in C* (embedded content) p.898 • All parameters (except Check) • Pseudocode Berndt M. Gammel Matpack documentation "Crypto – Codes" • All parameters including Check	ITU-T Recommendation X.25 • 4 codewords ISO/IEC FCD 14443-3 • 3 codewords W.H.Press *et al.* *Numerical Recipes in C* (embedded content) p.898 • 2 codewords (before XorOut stage)
Name : "XMODEM" Alias : "ZMODEM" Alias : "CRC-16/ACORN" Width : 16 Poly : 1021 Init : 0000 RefIn : False RefOut : False XorOut : 0000 Check : 31C3	The MSB-first form of the V.41 algorithm. For the LSB-first form see KERMIT. CRC presented high byte first. Residue = 0x0000. Used in the MultiMediaCard interface. In XMODEM and Acorn MOS the message bits are processed out of transmission order, compromising the guarantees on burst error detection.	ITU-T Recommendation V.41 • Full mathematical description • Shift register diagrams JEDEC Standard No. JESD84-A441 (registration required) • Full definition • Shift register diagram Acorn Computers Ltd BBC Microcomputer User Guide • Pseudocode	• XMODEM 5.0 • Acorn MOS 1.20 (BBC Micro cassette format) • CRC calculator, Lammert Bies • Checksum calculator, PVL Team	Berndt M. Gammel Matpack documentation "Crypto – Codes" • All parameters including Check cited for XMODEM Altera Corporation crc MegaCore Function Data Sheet (via the Internet Archive) • All parameters including Check cited for ZMODEM	W.H.Press *et al.* *Numerical Recipes in C* (embedded content) p.898 • 2 codewords cited for XMODEM
Name : "CRC-24" Alias : "CRC-24/OPENPGP" Width : 24 Poly : 864CFB Init : B704CE RefIn : False RefOut : False XorOut : 000000 Check : 21CF02	–	IETF RFC 4880 • Definition: Width, Poly, Init • Code: C	• Checksum calculator, PVL Team	Berndt M. Gammel Matpack documentation "Crypto – Codes" • All parameters including Check	–
Name : "CRC-24/FLEXRAY-A" Width : 24 Poly : 5D6DCB Init : FEDCBA RefIn : False RefOut : False XorOut : 000000 Check : 7979BD	Channels A and B have different initial vectors to prevent frames crossing channels.	FlexRay Consortium FlexRay Communications System Protocol Specification Version 3.0.1 • Definition: Width, Poly, Init, RefOut • Pseudocode	–	–	FlexRay Consortium FlexRay Protocol Conformance Test Specification Version 3.0.1 • 5 codewords
Name : "CRC-24/FLEXRAY-B" Width : 24 Poly : 5D6DCB Init : ABCDEF RefIn : False RefOut : False XorOut : 000000 Check : 1F23B8	See ↑	See ↑	–	–	See ↑
Name : "CRC-32" Alias : "CRC-32/ADCCP" Alias : "PKZIP" Width : 32 Poly : 04C11DB7 Init : FFFFFFFF RefIn : True RefOut : True XorOut : FFFFFFFF Check : CBF43926	–	ITU-T Recommendation V.42 • Full mathematical description Lasse Collin, Igor Pavlov *et al.* The .xz file format Version 1.0.4 (2009-08-27) • Code: C	• PKZIP 2.04g • libpng 1.0.5 • XZ Utils 5.0.3 • CRC calculator, Lammert Bies • Checksum calculator, PVL Team	Ross N. Williams "A Painless Guide to CRC Error Detection Algorithms" Berndt M. Gammel Matpack documentation "Crypto – Codes" • All parameters including Check	–

`Name   : "CRC-32/BZIP2"`
`Alias  : "CRC-32/AAL5"`
`Alias  : "CRC-32/DECT-B"`
`Alias  : "B-CRC-32"`
`Width  : 32`
`Poly   : 04C11DB7`
`Init   : FFFFFFFF`
`RefIn  : False`
`RefOut : False`
`XorOut : FFFFFFFF`
`Check  : FC891918`
	Used in DECT B-fields. Black's example AAL5 cells, with bytes 00 00 00 28 inserted between the described data fields and their CRCs, equal the examples in I.363.5.	ITU-T Recommendation I.363.5
		• Full mathematical description
		• Definition: Residue
		
		ETSI EN 300 175-3 V2.4.0 (2011-12) (registration required)
		• Full mathematical description
			• bzip2 0.9.5d
			• Checksum calculator, PVL Team
				Emil Lenchak Texas Instruments, Inc. CRC Implementation With MSP430
				• All parameters including Check
				
				Richard Black Fast CRC32 in Software: Software Implementations
				• Code: C
					ITU-T Recommendation I.363.5
					Richard Black Fast CRC32 in Software: Some Examples
					• 3 codewords

`Name   : "CRC-32C"`
`Alias  : "CRC-32/ISCSI"`
`Alias  : "CRC-32/CASTAGNOLI"`
`Width  : 32`
`Poly   : 1EDC6F41`
`Init   : FFFFFFFF`
`RefIn  : True`
`RefOut : True`
`XorOut : FFFFFFFF`
`Check  : E3069283`
	–	IETF RFC 3720
		• Full definition (except Check)
			• Jacksum 1.7.0
			• Base91 level 1 version 2.12
				Mark Bakke, Julian Satran, Venkat Rangan IP Storage Mailing List thread
				• All parameters including Check (Bakke, Rangan)
				• Definition: Width, Poly, Init, XorOut (Satran)
				• Code: C (Rangan)
				
				Base91
				• Full mathematical description
				• All parameters including Check
				• Code: C
					Mark Bakke, Julian Satran, Venkat Rangan IP Storage Mailing List thread
					• 3 codewords (Bakke)

`Name   : "CRC-32D"`
`Width  : 32`
`Poly   : A833982B`
`Init   : FFFFFFFF`
`RefIn  : True`
`RefOut : True`
`XorOut : FFFFFFFF`
`Check  : 87315576`
	–	–	• Base91 level 1 version 2.12	Base91
				• Full mathematical description
				• All parameters including Check
				• Code: C
				
				Philip Koopman "32-Bit Cyclic Redundancy Codes for Internet Applications" Proceedings of The International Conference on Dependable Systems and Networks
				• Polynomial discovered by Castagnoli; properties confirmed by Koopman
					–

`Name   : "CRC-32/MPEG-2"`
`Width  : 32`
`Poly   : 04C11DB7`
`Init   : FFFFFFFF`
`RefIn  : False`
`RefOut : False`
`XorOut : 00000000`
`Check  : 0376E6E7`
	–	ISO/IEC 13818-1:2000 — ITU-T Recommendation H.222.0 Annex A
		• Definition: Width, Poly, Init, RefIn, RefOut, Residue
		• CRC checking algorithm
		• Partial shift register diagram
			• Jacksum 1.7.0	VLC 1.1.13 module vlc-1.1.13/modules /mux/mpeg/ps.c
				• Code: C
					–

`Name   : "CRC-32/POSIX"`
`Alias  : "CKSUM"`
`Width  : 32`
`Poly   : 04C11DB7`
`Init   : 00000000`
`RefIn  : False`
`RefOut : False`
`XorOut : FFFFFFFF`
`Check  : 765E7680`
	The cksum program processes a representation of the input stream length following the input. It returns 930766865 (0x377A6011) on the check string, processed internally as 31 32 33 34 35 36 37 38 39 09.
	See the definition for details.
		The Open Group Single Unix Specification, version 2 Commands & Utilities Issue 5 Reference Pages: cksum
		• Full definition (except Check)
			• GNU cksum 2.0a	–	Libav 0.7.4 module libav-0.7.4/libavutil/crc.c
					• 1 codeword (before XorOut stage)

`Name   : "CRC-32Q"`
`Width  : 32`
`Poly   : 814141AB`
`Init   : 00000000`
`RefIn  : False`
`RefOut : False`
	Recognised by the ICAO. Used for aeronautical data.	European Organisation for the Safety of Air Navigation (EUROCONTROL) AIXM Primer 4.5
		• Definition:
			–	–	See ←
					• 8 codewords

`XorOut : 00000000`
`Check  : 3010BF7F`
		Width, Poly, Init, XorOut, RefIn
		• Flowchart
		• Code: Java			

`Name   : "JAMCRC"`
`Width  : 32`
`Poly   : 04C11DB7`
`Init   : FFFFFFFF`
`RefIn  : True`
`RefOut : True`
`XorOut : 00000000`
`Check  : 340BC6D9`
	–	–	• Checksum calculator, PVL Team	Altera Corporation
				crc MegaCore Function Data Sheet
				(via the Internet Archive)
				• All parameters including Check
					–

`Name   : "XFER"`
`Width  : 32`
`Poly   : 000000AF`
`Init   : 00000000`
`RefIn  : False`
`RefOut : False`
`XorOut : 00000000`
`Check  : BD0BE338`
	–	–	• XFER in C, version 5.1a	W.H.Press *et al.*
				Numerical Recipes in C
				(embedded content)
				p.299
				• Presents polynomial and its properties
					–

`Name   : "CRC-40/GSM"`
`Width  : 40`
`Poly   : 0004820009`
`Init   : 0000000000`
`RefIn  : False`
`RefOut : False`
`XorOut : 0000000000`
`Check  : 2BE9B039B9`
	Reported for the Fire code for GSM control channels.	–	–	Patrick Geremia
				Texas Instruments, Inc.
				Cyclic Redundancy Check Computation: An Implementation Using the TMS320C54x
				• Code: TMS320C54x assembler
				Berndt M. Gammel
				Matpack documentation "Crypto – Codes"
				• Definition: Width, Poly
					–

`Name   : "CRC-64"`
`Width  : 64`
`Poly   : 42F0E1EBA9EA3693`
`Init   : 0000000000000000`
`RefIn  : False`
`RefOut : False`
`XorOut : 0000000000000000`
`Check  : 6C40DF5F0B497347`
	Used in DLT-1 tape cartridges.	ECMA standard ECMA-182
		• Full mathematical description
			–	–	–

`Name   : "CRC-64/WE"`
`Width  : 64`
`Poly   : 42F0E1EBA9EA3693`
`Init   : FFFFFFFFFFFFFFFF`
`RefIn  : False`
`RefOut : False`
`XorOut : FFFFFFFFFFFFFFFF`
`Check  : 62EC59E3F1A4F00A`
	–	–	• CRC/Hash plugin for FAR Manager, Wolfgang Ehrhardt	–	–

`Name   : "CRC-64/XZ"`
`Width  : 64`
`Poly   : 42F0E1EBA9EA3693`
`Init   : FFFFFFFFFFFFFFFF`
`RefIn  : True`
`RefOut : True`
`XorOut : FFFFFFFFFFFFFFFF`
`Check  : 995DC9BBDF1939FA`
	–	Lasse Collin, Igor Pavlov *et al.*
		The .xz file format
		Version 1.0.4
		(2009-08-27)
		• Code: C
			• XZ Utils 5.0.3	–	–

`Name   : "CRC-82/DARC"`
`Width  : 82`
`Poly   : 0308C0111011401440411`
`Init   : 000000000000000000000`
`RefIn  : True`
`RefOut : True`
`XorOut : 000000000000000000000`
`Check  : 09EA83F625023801FD612`
	The single codeword is supported by the codewords confirming CRC-6/DARC, defined identically apart from Poly in the same standard.
	The codeword, representing the "transmitted bits", is clearly reflected ASCII. The direct ASCII would be the input to this algorithm.
	The example input message is 190 bits long, considering that the inner CRC is 14 bits. See section 12 for details of the transmission order.
		ETSI EN 300 751
		V1.2.1 (2003-01)
		(registration required)
		• Definition: Width, Poly, RefIn, RefOut
			–	–	See ←
					• 1 codeword

Legend:

```
    Alias: Alternative name(s) for algorithm
    Check: CRC result for UTF-8 string "123456789"
           [31 32 33 34 35 36 37 38 39]

  Academic: It has not been confirmed that CRCs are actually
           calculated in the field according to this record.

Third-party: All parameters and codewords originate from
           unofficial sources.
```

Summary of the CRC catalogue

CRC width (bits)	Records				
	Attested	Confirmed	Academic	Third-party	Total
3	–	–	1	–	1
4	–	–	1	–	1
5	1	–	1	1	3
6	1	–	1	–	2
7	–	–	2	–	2
8	3	1	2	1	7
10	1	–	–	–	1
11	1	–	–	–	1
12	–	–	2	–	2
14	1	–	–	–	1
15	–	–	1	–	1
16	16	4	–	2	22
24	3	–	–	–	3
32	6	3	–	–	9
40	–	–	1	–	1
64	1	1	1	–	3
82	1	–	–	–	1
Total	35	9	13	4	61

Notes: For more CRC algorithms see the list of CRC models supported by pycrc.

To find the parameters of an unknown algorithm, try the author's CRC RevEng application, an arbitrary-precision CRC calculator and algorithm finder in C.

The scope of this catalogue is to record fully specified CRC algorithms, in particular those whose output is in evidence. A record is marked *academic* unless there is a worked example or at least two valid message-CRC pairs, or a widely available application that can calculate matching CRCs for any desired message. *Third-party* records are those supported neither by an official specification nor an accessible implementation.

While each specification document undoubtedly has its accompanying implementation, I have here listed the implementations I can vouch for as to matching the given record. To complete the column I would be interested in transcripts or recordings of actual sessions showing multiple valid CRCs, along with the make and model of the equipment, or name and version of the software generating the CRCs.

Being based on the Rocksoft™ model ("Model"), these algorithms implicitly augment processed messages. Williams deals with augmentation in Section 10 of the "Painless Guide", showing that an implicitly augmenting algorithm, with a revised initial value, is equivalent to one that requires an augmented message.

In particular, one explicit-augmentation variant of `"CRC-16/CCITT-FALSE"` generates a `Check` value of 0xE5CC. As implicit or explicit augmentation is not a parameter in the Model, the difference can only be expressed through the above equivalence. Fortunately this algorithm is easily catered for in the Model by specifying `"CRC-16/CCITT"` parameters but with an initial value of 0x1D0F. This value is derived from the shift register contents after the whole 'original' initial value, 0xFFFF, has been processed once. An example Model record is given here under `"CRC-16/AUG-CCITT"`.

Even so there is a class of implementations that the Rocksoft™ model cannot directly cover; those that use the explicit-augmentation form but do not augment the message. The last few bytes of the message are left 'in plain text', so to speak, XORed in the result. An example is the algorithm published in an unofficial guide to the *Acorn Atom*. A Rocksoft™-type implementation can emulate such routines by processing a 'diminished' message and then XORing the last bytes of the message into the result.

The POSIX utility `"cksum"` considers the length of the file as well as its content. If the file is empty then it returns `Init ^ XorOut` as the result. Otherwise after the content it processes each byte of the integer representing the length, LSB first, until all set bits have been processed.

The DNP and HDLC algorithms, and many other reflected algorithms, require the result of a Rocksoft™ Model implementation to have the high and low bytes swapped, for example the true HDLC CRC of "123456789" is 0x6E90. A Rocksoft™ algorithm would return 0x906E.

The true *CRC-CCITT* algorithm is the one as implemented by *KERMIT*; *Numerical Recipes* and "Crypto - Codes" claim that the `"KERMIT"` algorithm and *CRC-CCITT* are identical. However by a historical accident the majority of implementations claiming to use *CRC-CCITT* have actually adopted another algorithm, listed here under `"CRC-16/CCITT-FALSE"`.

Some devices made by Sick use a non-standard 16 bit algorithm that is not represented by the Rocksoft™ model.

References

Ross N. Williams, "A Painless Guide to CRC Error Detection Algorithms".

- http://www.ross.net/crc/download/crc_v3.txt

Thomas Pircher, *pycrc*. Python based parametrised CRC calculator and C code generator.

- http://www.tty1.net/pycrc/

Johann N. LöffImann, *Jacksum*. CRC and hash calculator in Java.

- http://www.jonelo.de/java/jacksum/index.html

Robert Bosch GmbH, CAN 2.0 Specification.

- http://www.semiconductors.bosch.de/media/pdf/canliteratur/can2spec.pdf

Robert Bosch GmbH, E-Ray FlexRay IP Module Documentation and Application Notes.

- http://www.semiconductors.bosch.de/en/ipmodules/flexray/flexray.asp
- http://www.semiconductors.bosch.de/media/en/pdf/ipmodules_1/flexray/eray_users_manual_1_2_7.pdf
- http://www.semiconductors.bosch.de/media/en/pdf/ipmodules_1/flexray/071203_an002_1r02.pdf

William H. Press, Brian P. Flannery, Saul A. Teukolsky, William T. Vetterling, *Numerical recipes in C: The art of scientific computing*. 2nd ed. Cambridge: Cambridge University Press; 1992. ISBN 0-521-43108-5

- http://apps.nrbook.com/c/index.html

Useful links

Lammert Bies, "On-line CRC calculation and free library".

- http://www.lammertbies.nl/comm/info/crc-calculation.html

Lammert Bies, "Error detection and correction" Web forum.

- http://www.lammertbies.nl/forum/viewforum.php?f=11

PicList MassMind, "Cyclic Redundancy Check error detection".

- http://www.piclist.com/techref/method/error/crc.htm

Jonathan Graham Harston, "Source Code for Calculating CRCs".

- http://mdfs.net/Info/Comp/Comms/CRCs.htm
- http://mdfs.net/Info/Comp/Comms/CRC16.htm
- http://mdfs.net/Info/Comp/Comms/CRC32.htm

"Sven", Parametrised online CRC calculator.

- http://www.zorc.breitbandkatze.de/crc.html

Wolfgang Ehrhardt, CRC / HASH utilities and plugin for FAR Manager.

- http://home.netsurf.de/wolfgang.ehrhardt/crchash_en.html

Tom Torfs, IOCCC winning entry, 1998, CRC generator.

- http://www.ioccc.org/years.html#1998_tomtorfs

Disclaimer

Every effort has been made to ensure accuracy, however there may be occasional errors or omissions. All trademarks and registered trademarks are the intellectual property of their respective owners. The code and documentation included in this document (including, but not limited to, Appendices A to M) are supplied without warranty, not even the implied warranties of merchantability or fitness for a particular purpose. In no event shall the author or his suppliers be liable for any loss, damage, injury or death, of any nature and howsoever caused, arising from the use of, or failure, inability or unwillingness to use, this software or documentation.

[Top of page]

Appendix A

Map of common 16-bit CRC algorithms.

Karnaugh map of the most common 16-bit CRCs, with *Check* values and algorithm citations.
All values are hexadecimal.

"123456789" (UTF-8)	**Polynomial**	**1021**		**8005**	
	Reflected?	**False**	**True**		**False**
Initial value	**Final XOR**				
0000	**0000**	31C3 (*XMODEM*)	2189 (*KERMIT*)	BB3D (*ARC*)	FEE8 (*BUYPASS*)
	FFFF	CE3C (–)	DE76 (–)	44C2 (*MAXIM*)	0117 (–)
FFFF		D64E (*GENIBUS*)	906E (*X.25*)	B4C8 (*USB*)	5118 (–)
	0000	29B1 (*FALSE CCITT*)	6F91 (*MCRF4XX*)	4B37 (*MODBUS*)	AEE7 (–)
	6502 code	Appendix B	Appendix B	Appendix B	–
	ARM code	Appendix F	Appendix F	Appendix F	Appendix L
	8080 / Z80 code	Appendix K	Appendix K	Appendix K	–
	8051 / 8052 code	Appendix M	Appendix M	Appendix M	–

[Top of page]

Appendix B

Merged with Appendix C

Sample 6502 assembly code to implement the nine major 16-bit algorithms in constant time, without the use of lookup tables.

```
   *= $0070
crclo:  .DB $FF
crchi:  .DB $FF

        *= $1900
        .START *
test:   CLD
        LDY #$FF        ; "CRC-16/CCITT-FALSE" init value = $FFFF
;       LDY #$00        ; "XMODEM" init value = $0000
        STY crclo       ; store init value
        STY crchi
        LDY #$00
byloop: LDA data,Y
        JSR crc16_ccitt_f
        INY
        CPY #lendata
        BMI byloop
;       LDA crclo       ; complement result
;       EOR #$FF        ; for I-CODE, X.25 or USB
;       STA crclo
;       LDA crchi
;       EOR #$FF
;       STA crchi
        BRK             ; result is in $0070 and $0071

; Add byte to false CCITT or XMODEM-style CRC
; On entry:
;       A = byte to add
;       crclo = low byte of CCITT CRC
;       crchi = high byte of CCITT CRC
;       (on the first call, crclo and crchi should be set
;       according to the algorithm in use)
; On exit:
;       crclo,crchi = New CCITT CRC with byte added
; On exit, when using alternative ending 1:
;       S,V,B,D,I = preserved (subject to interrupts)
;       A,X,Y,N,Z,C = undefined
; On exit, when using alternative ending 2:
;       Y,S,V,B,D,I = preserved (subject to interrupts)
;       A,X,N,Z,C = undefined
; Relocatable, non-re-entrant

crc16_ccitt_f:          ; add byte to false CCITT CRC
        EOR crchi       ; A contained the data
        STA crchi       ; XOR it into high byte
        LSR             ; right shift A 4 bits
        LSR             ; to make top of x^12 term
        LSR             ; ($1...)
        LSR
        TAX             ; save it
        ASL             ; then make top of x^5 term
        EOR crclo       ; and XOR that with low byte
        STA crclo       ; and save
        TXA             ; restore partial term
        EOR crchi       ; and update high byte
        STA crchi       ; and save
        ASL             ; left shift three
        ASL             ; the rest of the terms
        ASL             ; have feedback from x^12
        TAX             ; save bottom of x^12
        ASL             ; left shift two more
        ASL             ; watch the carry flag
        EOR crchi       ; bottom of x^5 ($..2.)

;                       ; alternative ending 1
;       TAY             ; save high byte
;       TXA             ; fetch temp value
;       ROL             ; bottom of x^12, middle of x^5!
;       EOR crclo       ; finally update low byte
;       STA crchi       ; then swap high and low bytes
;       STY crclo
;       RTS             ; 37 bytes, 68 cycles, AXYP undefined

                        ; alternative ending 2
        STA crchi       ; save high byte
        TXA             ; fetch temp value
        ROL             ; bottom of x^12, middle of x^5!
        EOR crclo       ; finally update low byte
        LDX crchi       ; then swap high and low bytes
        STA crchi
        STX crclo
        RTS             ; 40 bytes, 72 cycles, AXP undefined


; Add byte to KERMIT or X.25-style CRC
; This is a straightforward reflection of the false CCITT algorithm
; On entry:
;       A = byte to add
;       crclo = low byte of KERMIT CRC
;       crchi = high byte of KERMIT CRC
;       (on the first call, crclo and crchi should be set
;       according to the algorithm in use)
; On exit:
;       crclo,crchi = New KERMIT CRC with byte added
; On exit, when using alternative ending 1:
;       S,V,B,D,I = preserved (subject to interrupts)
;       A,X,Y,N,Z,C = undefined
; On exit, when using alternative ending 2:
;       Y,S,V,B,D,I = preserved (subject to interrupts)
;       A,X,N,Z,C = undefined
; Relocatable, non-re-entrant

kermit_f:               ; add byte to KERMIT CRC
        EOR crclo       ; A contained the data
        STA crclo       ; XOR into low byte
        ASL             ; create top of x^12 term
        ASL
        ASL
        ASL
        TAX             ; save it
        LSR             ; then make top of x^5 term
        EOR crchi       ; XOR into high byte
        STA crchi
        TXA             ; restore x^12
        EOR crclo       ; apply it to low byte
        EOR crclo
        LSR             ; create bottom of x^12
        LSR             ; (with feedback from top)
        LSR             ; watch the carry flag
        TAX             ; save it
        LSR             ; make bottom of x^5
        LSR
        EOR crclo       ; apply to low byte

;                       ; alternative ending 1
;       TAY             ; save in Y
;       TXA             ; restore bottom of x^12
;       ROR             ; rotate in middle of x^5
;       EOR crchi       ; apply to high byte
;       STA crclo       ; and swap bytes
;       STY crchi
;       RTS             ; 37 bytes, 68 cycles, AXYP undefined

                        ; alternative ending 2
        STA crclo       ; save low byte
        TXA             ; restore bottom of x^12
        ROR             ; rotate in middle of x^5
        EOR crchi       ; apply to high byte
        LDX crclo       ; pick up low byte
        STA crclo       ; and swap bytes
        STX crchi
        RTS             ; 40 bytes, 72 cycles, AXP undefined


; Add byte to ARC or MODBUS-style CRC
; On entry:
;       A = byte to add
;       crclo = low byte of ARC-style CRC
;       crchi = high byte of ARC-style CRC
;       (on the first call, crclo and crchi should = $00, $00)
;       temp = unused byte of memory
; On exit:
;       crclo,crchi = New ARC-style CRC with byte added
;       Y,S,V,B,D,I = preserved (subject to interrupts)
;       A,X,N,Z,C,temp = undefined
; Relocatable, non-re-entrant

crc16_arc_f:            ; add byte to ARC-style CRC
        EOR crclo
        STA crclo

                        ; parity_adc (thanks bogax at 6502.org)
        ASL
        EOR crclo
        AND #$AA
        ADC #$66        ; C has no effect
        AND #$88
        ADC #$78
        ASL             ; C contains even parity

        LDA #0

        ROR crclo       ; push parity in b7
        ROR
        STA temp        ; save crclo B0
        LDA crclo
        LSR
        ROR temp        ; save crclo B1
        EOR crclo       ; 'blur' crclo
        TAX             ; keep final crchi

        ASL             ; C contains parity again
        LDA temp        ; reload mask
        ROL             ; parity in b0, B0 in b7
        EOR temp        ; add B0 in b6, B1 in b7
        EOR crchi       ; apply completed mask
        STA crclo       ; and store, swapping bytes
        STX crchi
        RTS             ; 44 bytes, 73 cycles, AXP undefined
                        ; other parity algorithms can be used
                        ; their speed and size vary greatly

data:   .DB $31,$32,$33,$34,$35,$36,$37,$38
        .DB $39
lendata = *-data

        .END
```

[Top of page]

Appendix D

This appendix contained parity calculators for the ARC algorithm, and has been removed.

[Top of page]

Appendix E

Sample 6502 assembly code to implement the "CRC-8" algorithm in constant time, without the use of lookup tables.

```
   *=  $0070
crc:    .DB $00

        *=  $1900
        .START *
test:   CLD
        LDY #$00        ; init value
        STY crc
byloop: LDA data,Y
        JSR crc8_f
        INY
        CPY #lendata
        BMI byloop
        BRK             ; result is in $0070

; Add byte to CRC-8
; On entry:
;       A = byte to add
;       crc = CRC-8 value
;       (on the first call, crc should = $00)
; On exit:
;       crc = New CRC-8 value with byte added
;       X,Y,S,V,B,D,I = preserved (subject to interrupts)
;       A,N,Z,C,temp = undefined
; Relocatable, non-re-entrant

crc8_f:                 ; add byte to CRC-8
        EOR crc         ; A contained the data
        STA crc         ; XOR it with the byte
        ASL             ; current contents of A will become x^2 term
        BCC crc8_f_apply_1
                        ; if b7 = 1
        EOR #$07        ; then apply polynomial with feedback
crc8_f_apply_1:
        BCC *+2         ; ballast to ensure constant time
        EOR crc         ; apply x^1
        ASL             ; C contains b7 ^ b6
        BCC crc8_f_apply_2
        EOR #$07
crc8_f_apply_2:
        BCC *+2         ; ballast to ensure constant time
        EOR crc         ; apply unity term
        STA crc         ; save result
        RTS             ; 25 bytes, 37 cycles, AP undefined

data:   .DB $31,$32,$33,$34,$35,$36,$37,$38
        .DB $39
lendata = *-data

        .END
```

[Top of page]

Appendix F

Sample ARM assembly code to implement the nine major 16-bit algorithms in constant time, without the use of lookup tables. Thanks to Viktor Gottschald for correspondence and a 15-instruction routine for ARC/Modbus, leading to this updated version.

```
.crc16_arc_test
        STMDB   (sp)!,{R1-R4,R8,R9,lr}  \preserve registers
        ADR     R8,data                 \set pointer to string
        MOV     R9,#dataend - data      \set counter to string length
        MOV     R0,#0                   \set Init = 0 (XMODEM, KERMIT, ARC)
\       MOV     R0,#&FF00               \or set Init = 0xffff (others)
\       ORR     R0,R0,#&FF
        MOV     R2,#&FF000000           \R2 = 0xff0000ff
        ORR     R2,R2,#&FF
        MOV     R3,#&A000000A           \R3 = 0xa006000a
        ORR     R3,R3,#&60000
        MVN     R4,#&2D00               \R4 = 0x2d00d2ff
        EOR     R4,R4,R4,LSL #16
.crc16_arc_test_loop
        LDRB    R1,[R8],#1              \get character from string
        BL      crc16_ccitt_f           \update CRC using false CCITT
        SUBS    R9,R9,#1                \decrement counter
        BNE     crc16_arc_test_loop     \loop until end of string
        AND     R0,R0,R2,ROR #24        \clear high bytes of result
\       EOR     R0,R0,R2,ROR #24        \complement it for I-CODE/X.25/USB
        ADDS    R0,R0,#0                \BASIC V/RISC OS needs flags clear
        LDMIA   (sp)!,{R1-R4,R8,R9,pc}  \at end restore regs and return CRC


                                        \fast false CCITT / XMODEM engine
                                        \on entry R0=old CRC, R1=data byte,
                                        \R2 = 0xff0000ff
                                        \on exit R0=new CRC (bits 0..15),
                                        \R0 bits 16..31 undefined,
                                        \R1 undefined
.crc16_ccitt_f
        EOR     R0,R0,R1,LSL #8         \merge new byte into top byte
        AND     R0,R2,R0,ROR #8         \old CRC to top and bottom byte
        EOR     R1,R0,R0,LSR #4         \'blur' low byte in new register
        EOR     R0,R0,R1,LSL #21        \apply feedback to polynomial
        EOR     R0,R0,R1,LSL #12        \and again
        EOR     R0,R0,R0,LSR #16        \fold top half of word to bottom
        MOV     pc,lr                   \return; 6 (7) instructions


                                        \fast KERMIT / X.25 engine
                                        \on entry R0=old CRC, R1=data byte,
                                        \R2 = 0xff0000ff
                                        \on exit R0=new CRC (bits 0..15),
                                        \R0 bits 16..31 undefined,
                                        \R1 undefined
.kermit_f
        AND     R0,R2,R0,ROR #8         \merge new byte into bottom byte
        EOR     R0,R0,R1,LSL #24        \old CRC to top and bottom byte
        EOR     R1,R0,R0,LSL #4         \'blur' low byte in new register
        EOR     R0,R0,R1,LSR #21        \apply feedback to polynomial
        EOR     R0,R0,R1,LSR #12        \and again
        EOR     R0,R0,R0,LSR #16        \fold top half of word to bottom
        MOV     pc,lr                   \return; 6 (7) instructions


                                        \fast ARC / MODBUS engine
                                        \on entry R0=old CRC, R1=data byte,
                                        \R2 = 0xff0000ff, R3 = 0xa006000a,
                                        \R4 & 0x7f807f80 = 0x2d005280
                                        \on exit R0=new CRC (bits 0..15),
                                        \R0 bits 16..31 undefined,
                                        \R1 undefined
.crc16_arc_f
        AND     R0,R2,R0,ROR #8         \old CRC to top and bottom byte
        EOR     R0,R0,R1,LSL #24        \merge new byte into top byte
        EOR     R1,R0,R0,LSR #1         \'blur' top byte in new register
        EOR     R0,R0,R1,LSR #17        \merge blurred byte into new top
        ORR     R1,R3,R1,ROR #26        \and rotate it and mask with 1s
        ADDS    R1,R1,R4,ROR R1         \put parity into N using table
        EORMI   R0,R0,R3,LSR #3         \if odd parity invert three bits
        MOV     pc,lr                   \return; 7 (8) instructions


.data
        EQUS "123456789"
.dataend
        ALIGN
```

[Top of page]

Appendix G

A demonstration of selected 16-bit algorithms in Python. Reproduced by kind permission of James Luscher.

```
#!/usr/bin/env Python

def CharToBinary(char):
    n = ord(char)
    bits = ''
    for y in range(8):
        if ((n & (1 << y)) == 0):
            bits = '0' + bits
        else:
            bits = '1' + bits
    return bits

def StringToBinary(message, reflect):
    bytes = ''
    print 'message = "' + message + '"'
    if reflect:
        print '                binary    hex       reflected'
    else:
        print '                binary    hex'
    for x in range(len(message)):
        char = message[x:x+1]
        bits = CharToBinary(char)
        rbits = Reflect(bits)
        if reflect:
            print "char  = '"+char+"' -> "+bits+" (0x"+BinaryToHex(bits)+') <=> '+rbits+' (0x'+BinaryToHex(rbits)+')'
            bits = rbits
        else:
            print "char  = '"+char+"' -> "+bits+" (0x"+BinaryToHex(bits)+')'
        if bytes == '':
            bytes = bytes + '{' + bits
        else:
            bytes = bytes + ',' + bits
    bytes = bytes + '}'
    if len(bytes) > 57:
        print "bytes = "+bytes[0:57]+' ...'
    else:
        print "bytes = " + bytes
    return bytes


def XOR(s1,s2):
    if len(s1) != len(s2):
        print "XOR(): ERROR, unequal length strings: ["+s1+"]["+s2+"]"
        return
    r = ''
    for i in range(len(s1)):
        if  (s1[i:i+1] == '1' and s2[i:i+1] == '1'):
            r = r + '0'
        elif(s1[i:i+1] == '0' and s2[i:i+1] == '0'):
            r = r + '0'
        else:
            r = r + '1'
    return r

def Reflect(s):
    r = ''
    for i in range(len(s)):
        r = s[i:i+1] + r
    return r

def NOT(s):
    r = ''
    for i in range(len(s)):
        if  (s[i:i+1] == '1'):
            r = r + '0'
        else:
            r = r + '1'
    return r

def HexToBinary(n):
    h = ''
    for inx in range(16):
        if n & 0x1 == 0x1:
            h = '1' + h
        else:
            h = '0' + h
        n = n / 2
    return h

def BinaryToHex(s):
    h = ''
    for x in range((len(s)+3)/4):
        n = 0
        i = 1
        for y in range(4):
            if s[-1:] == '1':
                n = n + i
            s = s[:-1]
            i = (i * 2)
        if n <= 9:
            h = (chr(ord('0') +  n      )) + h
        else:
            h = (chr(ord('a') + (n - 10))) + h
    return h

def MessageStrip(M):
    while len(M) > 0 and M[0:1] != '0' and M[0:1] != '1':
        M = M[1:]
    return M

#============================================
# author: James Luscher  (owns all errors ;-)
#                         <jluscher-gmail-com>
# corrections: Greg Cook (thanks Greg!)
# copyright: 2007 James Luscher
# licensed under: GNU General Public License
# details at http://www.gnu.org/copyleft/
#
# This program was written for self-education.
# I hope others may find it informative also.
#===========================================

def crc(poly, register, reflect, message, invert):
#-----------------------------------
# CRC-16 polynomial is 0x8005
# CRC-16/CCITT polynomial is 0x1021
    P =  HexToBinary(poly)
#-----------------------------------
# register (initial value)
    R = HexToBinary(register)
#-----------------------------------
# reflect in/out ??
    if reflect != 0:
        print "reflect   = True"
    else:
        print "reflect   = False"
#-----------------------------------
# M -> message  (ASCII string)
    M = StringToBinary(message, reflect)
    length = len(M)
    print
    if len(M) > 53:
        print "message   = "+M[0:53]+' ...'
    else:
        print "message   = "+M
#-----------------------------------
    print "register  = "+R
    print "polynomial= "+P+"  (0x" + BinaryToHex(P) + ')'
    print
    M = MessageStrip(M)
    print            "       register               message..."
    print            "       ----------------       -------..."
    if len(M) > 40:
        print        "       "+R+'       '+M[0:40]+' ...'
    elif len(M) > 0:
        print        "       "+R+'       '+M
    else:
        print        "       "+R
    while len(M)>0:
        Rc = R[0:1]
        Mc = M[0:1]
        C = XOR(Rc,Mc)
        R = R[1:]+'0'
        M = M[1:]
        M = MessageStrip(M)
        if len(M) > 40:
            print        "  ("+Rc+") < "+R+'  ('+Mc+') < '+M[0:40]+' ...'
        else:
            print        "  ("+Rc+") < "+R+'  ('+Mc+') < '+M
        if C == '1':
            print        "["+Rc+'^'+Mc+"]=> "+P+"       (xor by 0x" + BinaryToHex(P) + ')'
            print        "       "+('-' * 16 )
            R = XOR(R, P)
            print        "       "+R+"       (0x" + BinaryToHex(R) + ')'
            print
    print
    if reflect:
        R = Reflect(R)
        print        "      reflected"
        print        "<=> => " + R
    if invert != 0:
        R = NOT(R)
        print        "NOT => " + R
    print            "      " + R + " = CRC" + " (0x" + BinaryToHex(R) + ")"


def d1():
    crcdemo(0x1021, 0x0000, 1, 0)
    print
    print "demo:  example:                 poly    reg   r  i      expect"
    print "-----  ------------------- -----------------------------     ------"
    print "d1()   KERMIT:             crcdemo(0x1021, 0x0000, 1, 0)  // 0x2189"
    print "================================================================"
    crcdemo_help()

def d2():
    crcdemo(0x8408, 0x0000, 1, 0)
    print
    print "demo:  example:                 poly    reg   r  i      expect"
    print "-----  ------------------- -----------------------------     ------"
    print "d2()   X-KERMIT:           crcdemo(0x8408, 0x0000, 1, 0)  // 0x0c73"
    print "================================================================"
    crcdemo_help()

def d3():
    crcdemo(0x1021, 0xffff, 1, 1)
    print
    print "demo:  example:                 poly    reg   r  i      expect"
    print "-----  ------------------- -----------------------------     ------"
    print "d3()   X-25:               crcdemo(0x1021, 0xffff, 1, 1)  // 0x906e"
    print "================================================================"
    crcdemo_help()

def d4():
    crcdemo(0x8005, 0x0000, 1, 0)
    print
    print "demo:  example:                 poly    reg   r  i      expect"
    print "-----  ------------------- -----------------------------     ------"
    print "d4()   CRC-16/ARC:         crcdemo(0x8005, 0x0000, 1, 0)  // 0xbb3d"
    print "=================================================================="
    crcdemo_help()

def d5():
    crcdemo(0x1021, 0xffff, 0, 0)
    print
    print "demo:  example:                 poly    reg   r  i      expect"
    print "-----  ------------------- -----------------------------     ------"
    print "d5()   CRC-16/CCITT-FALSE: crcdemo(0x1021, 0xffff, 0, 0)  // 0x29b1"
    print "=================================================================="
    crcdemo_help()

def d6():
    crcdemo(0x1021, 0x0000, 0, 0)
    print
    print "demo:  example:                 poly    reg   r  i      expect"
    print "-----  ------------------- -----------------------------     ------"
    print "d6()   CRC-16/XMODEM:      crcdemo(0x1021, 0x0000, 0, 0)  // 0x31c3"
    print "=================================================================="
    crcdemo_help()


def crcdemo_help():
    print """
        crcdemo()
        - show the detailed calculation for various kinds of 16 bit CRCs
        - ALL demo examples applied to the canonical message "123456789"
        - use function 'crc()' to apply to your own message string.

        crcdemo(poly, register, reflect, append, invert)
            poly     <- hex value for (truncated) generator polynomial
            register <- hex initial value for remainder register
            reflect  <- 0 == don't reflect message bytes & output CRC
            invert   <- 0 == don't invert remainder after calculation

 demo:  examples:                poly    reg   r  i      expect
 -----  ------------------- -----------------------------     ------
 d1()   KERMIT:             crcdemo(0x1021, 0x0000, 1, 0)  // 0x2189
 d2()   X-KERMIT:           crcdemo(0x8408, 0x0000, 1, 0)  // 0x0c73
 d3()   X-25:               crcdemo(0x1021, 0xffff, 1, 1)  // 0x906e
 d4()   CRC-16/ARC:         crcdemo(0x8005, 0x0000, 1, 0)  // 0xbb3d
 d5()   CRC-16/CCITT-FALSE: crcdemo(0x1021, 0xffff, 0, 0)  // 0x29b1
 d6()   CRC-16/XMODEM:      crcdemo(0x1021, 0x0000, 0, 0)  // 0x31c3
    """

def crcdemo(poly, register, reflect, invert):
    crc(poly, register, reflect, '123456789', invert)
    return

crcdemo_help()

// To run a demonstration of each algorithm, uncomment any of the next 6 lines. - GJC

//  d1()
//  d2()
//  d3()
//  d4()
//  d5()
//  d6()

// End of crc16demo.py
```

[Top of page]

Appendix H

Based on research by Vivek Rajan

Interpretation of the three CRC-11 samples in the Bosch E-Ray Application Note 002.

See Section 4 of the FlexRay Protocol Specification for a definition of the FlexRay frame format and the Header CRC Covered Area.

Example 1

In section 4.4.3.1 (pp. 26-7) of the Application Note is code for "Configuration of Coldstart Node A". It includes the lines

```
write32bit(WRHS1, 0x27000001); // transmit buffer, frame ID = 1
write32bit(WRHS2, 0x0008011B); // payload length = 8 two-byte words
```

As a coldstart node, according to section 4.4.3.1 (p.15) the Sync and Startup frame indicators are set. This corresponds to a Header CRC Covered Area of `11 00000000001 0001000 (0xc0088)` and a Header CRC of `001 0001 1011 (0x11b)`.

Example 2

In section 4.4.3.2 (pp. 28-9) of the Application Note is code for "Configuration of Coldstart Node B". It includes the lines

```
write32bit(WRHS1, 0x27000002); // transmit buffer, frame ID = 2
write32bit(WRHS2, 0x00080304); // payload length = 8 two-byte words
```

As a coldstart node, according to section 4.4.3.1 (p.15) the Sync and Startup frame indicators are set. This corresponds to a Header CRC Covered Area of `11 00000000010 0001000 (0xc0108)` and a Header CRC of `011 0000 0100 (0x304)`.

Example 3

In section 4.4.3.3 (pp. 30-1) of the Application Note is code for "Configuration of Integrating Node C". It includes the lines

```
write32bit(WRHS1, 0x27000003); // transmit buffer, frame ID = 3
write32bit(WRHS2, 0x000805D2); // payload length = 8 two-byte words
```

A cursory inspection does not reveal proof of whether the Sync or Startup frame indicators are set. This corresponds to a Header CRC Covered Area of `?? 00000000011 0001000 (0x?0188)` and a Header CRC of `101 1101 0010 (0x5d2)`.

By exhaustive search the bit string `00 00000000011 0001000 (0x00188)` is found to match the reported CRC. This gives us 511/512 confidence in Example 3, compared to

2047/2048 for the first two examples.

[Top of page]

Appendix J

Performing the Atom checksum algorithm with a Rocksoft™ Model implementation.

1. Make a copy of the file to be tested and do all work on this copy.
2. Truncate the last two bytes from the file. Save their values for later.

```
$ wc -c afloat.rom
   4096 afloat.rom
$ dd if=afloat.rom of=afloat.rom.diminished bs=1 count=4094
4094+0 records in
4094+0 records out
$ dd if=afloat.rom bs=1 skip=4094 | hexdump -x
2+0 records in
2+0 records out
00000000    605f
00000002
```

   Note: *hexdump* prints little-endian words. The second-to-last byte of this file is `0x5F` and the last byte is `0x60`.
3. Run a CRC check on the first part of the file.

```
$ reveng -A 16 -w 16 -p 002D -l -f -c afloat.rom.diminished
e50a
```

4. Treating the last two bytes of the original file as a little-endian word, XOR it with the CRC result.
   - `0xE50A ^ 0x605F == 0x8555`
5. Reverse the bits of this result, so that the LSB becomes the MSB and vice versa. This is the Atom CRC of the whole file.
   - `0x8555 == %1000010101010101 ==> %1010101010100001 == 0xAAA1`

This matches the 'signature' given for Atom Floating Basic in *Splitting the Atom*. For the Rocksoft™ check string:

1.
   ```
   $ reveng -A 16 -w 16 -p 002D -l -c 31323334353637
   f5f2
   ```
2. `0xF5F2 ^ 0x3938 == 0xCCCA`
3. `0xCCCA == %1100110011001010 ==> %0101001100110011 == 0x5333`

NB: The CRC RevEng application has direct support for non-augmenting algorithms. The above procedures can be replaced with:

-
  ```
  $ reveng -A 16 -w 16 -p 002D -l -B -M -f -c afloat.rom
  aaa1
  ```

-
  ```
  $ reveng -A 16 -w 16 -p 002D -l -B -M -c 313233343536373839
  5333
  ```

Incidentally, the unreflected form of this unusual algorithm reappears in the Meridian Lossless Packing format, installed in all DVD Audio players and developed by Meridian Audio Ltd., whose co-founder Allen Boothroyd designed the case for the Acorn Atom and BBC Micro.

[Top of page]

Appendix K

Sample 8080/Z80 assembly code to implement the nine major 16-bit algorithms in constant time, without the use of lookup tables. One of the routines is for faster calculation on a Z80 only.

```
        ; Main loop, for 8080/Z80

        ORG     100H
Start:  LD      SP,1000H
        LD      HL,0            ; for XMODEM, KERMIT and ARC
;       LD      HL,FFFFH        ; for false CCITT, X.25 and USB
        LD      DE,data         ; set pointer to test string
        LD      C,dataend-data  ; set counter to string length
tloop:  LD      A,(DE)          ; get character of string
        INC     DE              ; increment pointer
        PUSH    BC              ; save counter
        CALL    crc16_ccitt_f   ; do CRC on character
        POP     BC              ; restore counter
        DEC     C               ; decrement it
        JP      NZ,tloop        ; and loop until string done
;       LD      A,H             ; complement result
;       CPL                     ; for false CCITT, X.25 and USB
;       LD      H,A
;       LD      A,L
;       CPL
;       LD      L,A
        HALT

data:   DEFB    "123456789"     ; test string
dataend:NOP                     ; label for calculating length


        ; CRC-16/CCITT-FALSE for 8080/Z80
        ; On entry HL = old CRC, A = byte
        ; On exit HL = new CRC, A,B,C undefined

                                ; Ts  M/code    8080 assembly
crc16_ccitt_f:
        XOR     H               ;  4  AC        XRA     H
        LD      B,A             ;  4  47        MOV     B,A
        LD      C,L             ;  4  4D        MOV     C,L
        RRCA                    ;  4  0F        RRC
        RRCA                    ;  4  0F        RRC
        RRCA                    ;  4  0F        RRC
        RRCA                    ;  4  0F        RRC
        LD      L,A             ;  4  6F        MOV     L,A
        AND     0FH             ;  7  E6 0F     ANI     0FH
        LD      H,A             ;  4  67        MOV     H,A
        XOR     B               ;  4  A8        XRA     B
        LD      B,A             ;  4  47        MOV     B,A
        XOR     L               ;  4  AD        XRA     L
        AND     F0H             ;  7  E6 F0     ANI     F0H
        LD      L,A             ;  4  6F        MOV     L,A
        XOR     C               ;  4  A9        XRA     C
        ADD     HL,HL           ; 11  29        DAD     H
```

```
        XOR     H               ;  4  AC        XRA     H
        LD      H,A             ;  4  67        MOV     H,A
        LD      A,L             ;  4  7D        MOV     A,L
        XOR     B               ;  4  A8        XRA     B
        LD      L,A             ;  4  6F        MOV     L,A
        RET                     ; 10  C9        RET

        ; 115 T-states, 25 bytes


        ; CRC-16/CCITT-FALSE for 8080/Z80
        ; On entry HL = old CRC, A = byte
        ; On exit HL = new CRC, A,B undefined

                                ; Ts  M/code    8080 assembly
crc16_ccitt_c_f:
        XOR     H               ;  4  AC        XRA     H
        LD      H,A             ;  4  67        MOV     H,A
        AND     F0H             ;  7  E6 F0     ANI     F0H
        RRCA                    ;  4  0F        RRC
        RRCA                    ;  4  0F        RRC
        RRCA                    ;  4  0F        RRC
        RRCA                    ;  4  0F        RRC
        XOR     H               ;  4  AC        XRA     H
        LD      H,A             ;  4  67        MOV     H,A
        RRCA                    ;  4  0F        RRC
        RRCA                    ;  4  0F        RRC
        RRCA                    ;  4  0F        RRC
        LD      B,A             ;  4  47        MOV     B,A
        AND     E0H             ;  7  E6 E0     ANI     E0H
        XOR     H               ;  4  AC        XRA     H
        LD      H,A             ;  4  67        MOV     H,A
        LD      A,B             ;  4  78        MOV     A,B
        AND     1FH             ;  7  E6 1F     ANI     1FH
        XOR     L               ;  4  AD        XRA     L
        LD      L,A             ;  4  6F        MOV     L,A
        LD      A,B             ;  4  78        MOV     A,B
        RRCA                    ;  4  0F        RRC
        AND     F0H             ;  7  E6 F0     ANI     F0H
        XOR     L               ;  4  AD        XRA     L
        LD      L,H             ;  4  6C        MOV     L,H
        LD      H,A             ;  4  67        MOV     H,A
        RET                     ; 10  C9        RET

        ; 126 T-states, 31 bytes


        ; KERMIT for 8080/Z80
        ; On entry HL = old CRC, A = byte
        ; On exit HL = new CRC, A,B undefined

                                ; Ts  M/code    8080 assembly
kermit_f:
        XOR     L               ;  4  AD        XRA     L
        LD      L,A             ;  4  6F        MOV     L,A
        ADD     A,A             ;  4  87        ADD     A
        ADD     A,A             ;  4  87        ADD     A
        ADD     A,A             ;  4  87        ADD     A
        ADD     A,A             ;  4  87        ADD     A
        XOR     L               ;  4  AD        XRA     L
        LD      L,A             ;  4  6F        MOV     L,A
        RLCA                    ;  4  07        RLC
        RLCA                    ;  4  07        RLC
        RLCA                    ;  4  07        RLC
        LD      B,A             ;  4  47        MOV     B,A
        AND     07H             ;  7  E6 07     ANI     07H
        XOR     L               ;  4  AD        XRA     L
        LD      L,A             ;  4  6F        MOV     L,A
        LD      A,B             ;  4  78        MOV     A,B
        AND     F8H             ;  7  E6 F8     ANI     F8H
        XOR     H               ;  4  AC        XRA     H
        LD      H,A             ;  4  67        MOV     H,A
        LD      A,B             ;  4  78        MOV     A,B
        RLCA                    ;  4  07        RLC
        AND     0FH             ;  7  E6 0F     ANI     0FH
        XOR     H               ;  4  AC        XRA     H
        LD      H,L             ;  4  65        MOV     H,L
        LD      L,A             ;  4  6F        MOV     L,A
        RET                     ; 10  C9        RET

        ; 119 T-states, 29 bytes


        ; CRC-16/ARC for 8080/Z80
        ; On entry HL = old CRC, A = byte
        ; On exit HL = new CRC, A,B undefined

                                ; Ts  M/code    8080 assembly
crc16_arc_f:
        XOR     L               ;  4  AD        XRA     L
        LD      L,A             ;  4  6F        MOV     L,A
        RRCA                    ;  4  0F        RRC
        RRCA                    ;  4  0F        RRC
        ;AND    A               ; needed if running in ZINT / no ballast
        JP      PE,blur         ; 10  EA nn nn  JPE     blur
        SCF                     ;  0  37        STC
blur:   JP      PO,blur1        ; 10  E2 nn nn  JPO     blur1
        AND     A               ;  4  A7        ANA     A
blur1:  RRA                     ;  4  1F        RAR
        AND     E0H             ;  7  E6 E0     ANI     E0H
        RLA                     ;  4  17        RAL
        LD      B,A             ;  4  47        MOV     B,A
        RLA                     ;  4  17        RAL
        XOR     B               ;  4  A8        XRA     B
        XOR     H               ;  4  AC        XRA     H
        LD      B,A             ;  4  47        MOV     B,A
        XOR     H               ;  4  AC        XRA     H
        RRA                     ;  4  1F        RAR
        LD      A,L             ;  4  7D        MOV     A,L
        RRA                     ;  4  1F        RAR
        LD      L,A             ;  4  6F        MOV     L,A
        AND     A               ;  4  A7        ANA     A
        RRA                     ;  4  1F        RAR
        XOR     L               ;  4  AD        XRA     L
        LD      L,B             ;  4  68        MOV     L,B
        LD      H,A             ;  4  67        MOV     H,A
        RET                     ; 10  C9        RET

        ; 125 T-states, 31 bytes


        ; CRC-16/ARC for Z80 only
        ; On entry HL = old CRC, A = byte
        ; On exit HL = new CRC, A,B undefined

                                ; Ts  M/code
crc16_arc_z80_f:
        LD      B,0             ;  7  06 00
        XOR     L               ;  4  AD
        ;AND    A               ; needed if running in ZINT
        JP      PE,blur80       ; 10  EA nn nn
        SCF                     ;  0  37
blur80: JP      PO,blur81       ; 10  E2 nn nn
        NOP                     ;  4  00
blur81: RRA                     ;  4  1F
        RR      B               ;  8  CB 18
        LD      L,A             ;  4  6F
        SRL     A               ;  8  CB 3F
        RR      B               ;  8  CB 18
        XOR     L               ;  4  AD
        LD      L,A             ;  4  6F
        ADD     A,A             ;  4  87
        LD      A,B             ;  4  78
        RLA                     ;  4  17
        XOR     B               ;  4  A8
        XOR     H               ;  4  AC
        LD      H,L             ;  4  65
        LD      L,A             ;  4  6F
        RET                     ; 10  C9

        ; 113 T-states, 29 bytes

        END
```

[Top of page]

Appendix L

Parametrised, table-driven CRC algorithm in ARM assembler. The main loop takes just 8 instructions per byte, or 6 if inlined.

```
REM >Table3
REM Greg Cook 2008-09-11

REM Assemble algorithm-independent code

DIM code 1024+1024-1
sp=13:lr=14:pc=15

FOR pass=0 TO 3 STEP 3
P%=code
[OPT pass
.main
        \ do a CRC of the test string
        \ to demonstrate the algorithm
        stmdb   (sp)!,{r1-r4,lr}
        bl      doinit
        adr     r3,string
        ldr     r4,strlen
.mloop  ldrb    r1,[r3],#1
        bl      byte
        subs    r4,r4,#1
        bne     mloop
        bl      finish
        adds    r0,r0,#0 \for BASIC V/RISC OS
        ldmia   (sp)!,{r1-r4,pc}

.doinit
        stmdb   (sp)!,{r1,r3-r5,lr}
        \ set up registers
        adr     r2,table
        ldr     r1,poly
        bic     r1,r1,#&FF0000
        bic     r1,r1,#&FF000000
        \ set up table
        mov     r3,#&FF
.itbyte \ copy table offset to dividend
        mov     r0,r3,lsl #8
        mov     r5,r3
        mov     r4,#&01000000 \bit counter
        \ do pure polynomial division
.itbit  tst     r0,#&8000
        bic     r0,r0,#&8000
        mov     r0,r0,lsl #1
        eorne   r0,r0,r1
        \ shift bit counter and reverse offset
        movs    r5,r5,lsr #1
        adcs    r4,r4,r4
        bcc     itbit
        \ test RefIn
        ldr     r5,refin
        tst     r5,#1
        beq     split
        \ if RefIn = true reverse remainder
        \ (without swapping bytes)
        mov     r5,#&18000
.revrem movs    r0,r0,lsr #1
        adcs    r5,r5,r5
        bcc     revrem
        movs    r0,r5,ror #8 \clear Z
.split
        \split remainder to top and bottom byte
        orr     r5,r0,r0,lsl #16
        bic     r5,r5,#&FF00
        bic     r5,r5,#&FF0000
        \if RefIn = false store at direct offset
        orreq   r5,r5,r4,lsl #8
        streq   r5,[r2,r3,lsl #2]
        \if RefIn = true store at reflected offset
        orrne   r5,r5,r3,lsl #8
        strne   r5,[r2,r4,lsl #2]
        subs    r3,r3,#1
        bcs     itbyte
        \ set up shift register. test RefIn
        ldr     r5,refin
        tst     r5,#1
        ldr     r0,init
        \ move Init to top
        mov     r0,r0,lsl #16
        \ split Init and quit if RefIn = false
        orreq   r0,r0,r0,lsr #16
        beq     doneinit
        \ else split and reflect Init (bytes not swapped)
        mov     r5,r0,lsr #24
        and     r0,r0,#&FF0000
        ldr     r0,[r2,r0,lsr #14]
        ldr     r5,[r2,r5,lsl #2]
        mov     r5,r5,lsl #16
        orr     r0,r5,r0,lsr #8
.doneinit
        bic     r0,r0,#&FF00
        bic     r0,r0,#&FF0000
        ldmia   (sp)!,{r1,r3-r5,pc}

.byte
        \ merge byte in R1 into the CRC in R0
        eor     r0,r0,r1,lsl #24
        ldr     r1,[r2,r0,lsr #22]
        eor     r0,r1,r0,lsl #24
        mov     pc,lr

.finish
        \ test RefIn and RefOut
        ldr     r1,refout
        movs    r1,r1,lsr #1
        ldr     r1,refin
        adc     r1,r1,#0
        tst     r1,#1
        \ C = RefOut, Z = !(RefIn ^ RefOut)
        \ Bytes to top of r0 and r1
        \ Swap if RefOut = true
        movcc   r1,r0,lsl #24
        andcs   r1,r0,#&FF000000
        movcs   r0,r0,lsl #24
        \ Reflect bytes if RefIn != RefOut
        ldrne   r0,[r2,r0,lsr #22]
        ldrne   r1,[r2,r1,lsr #22]
        \ Join bytes in r0
        bic     r0,r0,#&FF
        orr     r0,r0,r1,lsr #8
        \ Move to top if RefIn != RefOut
        movne   r0,r0,lsl #16
        \ Apply XorOut to result
        ldr     r1,xorout
        eor     r0,r0,r1,lsl #16
        \ Shift result to bottom of r0
        mov     r0,r0,lsr #16
        mov     pc,lr

.poly   equd    0
.init   equd    0
.refin  equd    0
.refout equd    0
.xorout equd    0

.strlen equd    0
.string equs    STRING$(255," ")

        align
.table
]
P%+=1024
NEXT

REM Set parameters for this run
REM This is a RockSoft(TM) Model record
REM with adjusted syntax

Width   =  16   : REM not used
Poly    = &1021
Init    = &FFFF
RefIn   =  TRUE
RefOut  =  TRUE
XorOut  = &FFFF

REM Set the string to test
```

```
String$ = "123456789"

REM Store the parameters for the routine

!poly    = Poly
!init    = Init
!refin   = RefIn
!refout  = RefOut
!xorout  = XorOut
!strlen  = LEN(String$)
$string  = String$

REM Call code and print computed CRC

PRINT '"Result = ";~USR(main)

REM Dump table contents
PRINT "Contents of table:"
FOR T% = 0 TO 1023 STEP 4
IF T% MOD 32 = 0 THEN PRINT
PRINT ~table!T%;
NEXT
PRINT

REM Regression test: 16 results from Ross Williams' crcmodel.c
REM and values from
REM http://regregex.bbcmicro.net/crc-catalogue.htm#appendix.a
$string = "123456789"
!strlen=9
FOR T%=1 TO 30
READ !poly,!init,!refin,!refout,!xorout,expect%,name$
actual%=USR(main)
IF actual%<>expect% THEN
PRINT ~!poly,~!init,~!refin,~!refout,~!xorout,~expect%,~actual%:END
ENDIF
NEXT
PRINT "Regression test passed"
END

DATA &1021,&0000, 0, 0,&0000,&31C3,"XMODEM, ZMODEM, CRC16/ACORN"
DATA &1021,&0000, 0,-1,&0000,&C38C,""
DATA &1021,&0000,-1, 0,&0000,&9184,""
DATA &1021,&0000,-1,-1,&0000,&2189,"KERMIT, TRUE CRC-CCITT"
DATA &1021,&0000, 0, 0,&2357,&1294,""
DATA &1021,&0000, 0,-1,&2357,&E0DB,""
DATA &1021,&0000,-1, 0,&2357,&B2D3,""
DATA &1021,&0000,-1,-1,&2357,&02DE,""
DATA &1021,&1234, 0, 0,&0000,&EDEB,""
DATA &1021,&1234, 0,-1,&0000,&D7B7,""
DATA &1021,&1234,-1, 0,&0000,&4DAC,""
DATA &1021,&1234,-1,-1,&0000,&35B2,""
DATA &1021,&1234, 0, 0,&2357,&CEBC,""
DATA &1021,&1234, 0,-1,&2357,&F4E0,""
DATA &1021,&1234,-1, 0,&2357,&6EFB,""
DATA &1021,&1234,-1,-1,&2357,&16E5,""
DATA &8005,&0000,-1,-1,&0000,&BB3D,"CRC-16, ARC"
DATA &8005,&0000, 0, 0,&0000,&FEE8,"BUYPASS"
DATA &1021,&0000, 0, 0,&FFFF,&CE3C,""
DATA &1021,&0000,-1,-1,&FFFF,&DE76,""
DATA &8005,&0000,-1,-1,&FFFF,&44C2,""
DATA &8005,&0000, 0, 0,&FFFF,&0117,""
DATA &1021,&FFFF, 0, 0,&FFFF,&D64E,"I-CODE"
DATA &1021,&FFFF,-1,-1,&FFFF,&906E,"X.25, HDLC"
DATA &8005,&FFFF,-1,-1,&FFFF,&B4C8,"USB"
DATA &8005,&FFFF, 0, 0,&FFFF,&5118,""
DATA &1021,&FFFF, 0, 0,&0000,&29B1,"FALSE CRC-CCITT"
DATA &1021,&FFFF,-1,-1,&0000,&6F91,"MCRF4XX"
DATA &8005,&FFFF,-1,-1,&0000,&4B37,"MODBUS"
DATA &8005,&FFFF, 0, 0,&0000,&AEE7,""

REM End of Table3
```

NB: When `RefIn` and `RefOut` are constants, the `finish` subroutine can be condensed to one of the following:

```
        \RefIn = FALSE, RefOut = FALSE
.finish
        and     r1,r0,#&FF
        orr     r0,r1,r0,lsr #16
        ldr     r1,xorout
        eor     r0,r0,r1
        mov     pc,lr

        \RefIn = FALSE, RefOut = TRUE
.finish
        and     r1,r0,#&FF
        ldr     r0,[r2,r0,lsr #22]
        ldr     r1,[r2,r1,lsl #2]
        and     r0,r0,#&FF00
        and     r1,r1,#&FF00
        orr     r0,r1,r0,lsr #8
        ldr     r1,xorout
        eor     r0,r0,r1
        mov     pc,lr

        \RefIn = TRUE, RefOut = FALSE
.finish
        and     r1,r0,#&FF
        ldr     r0,[r2,r0,lsr #22]
        ldr     r1,[r2,r1,lsl #2]
        and     r0,r0,#&FF00
        and     r1,r1,#&FF00
        orr     r0,r0,r1,lsr #8
        ldr     r1,xorout
        eor     r0,r0,r1
        mov     pc,lr

        \RefIn = TRUE, RefOut = TRUE
.finish
        mov     r0,r0,ror #24
        bic     r0,r0,#&FF0000
        ldr     r1,xorout
        eor     r0,r0,r1
        mov     pc,lr
```

[Top of page]

Appendix M

Sample 8051/8052 assembly code to implement the nine major 16-bit algorithms in constant time, without the use of lookup tables.

```
   ; Main loop

        mov     r0,#00h         ; for XMODEM, KERMIT and ARC
        mov     r1,#00h
        ;mov    r0,#0ffh        ; for false CCITT, X.25 and USB
        ;mov    r1,#0ffh
        mov     dptr,#data
        mov     r3,#0
char:
        mov     a,r3
        movc    a,@a+dptr
        acall   ccitt
        inc     r3
        cjne    r3,#datalen,char
finish:
        ;xch    a,r0            ; complement result
        ;cpl    a               ; for false CCITT, X.25 and USB
        ;xch    a,r1
        ;cpl    a
        ;xch    a,r1
        ;xch    a,r0

        sjmp    $


        ; CRC-16/CCITT-FALSE for 8051/2
        ; On entry A  = byte
        ;          R0 = old CRC low byte
        ;          R1 = old CRC high byte
        ; On exit  R0 = new CRC low byte
        ;          R1 = new CRC high byte
        ;          A,R2 = undefined

                                ; Cs  M/code
ccitt:
        xrl     a,r1            ;  1  69
        mov     r1,a            ;  1  f9
        swap    a               ;  1  c4
        anl     a,#0fh          ;  1  54 0f
        xrl     a,r1            ;  1  69
        mov     r1,a            ;  1  f9
        swap    a               ;  1  c4
        mov     r2,a            ;  1  fa
        anl     a,#0f0h         ;  1  54 f0
        xrl     a,r0            ;  1  68
        xch     a,r2            ;  1  ca
        rl      a               ;  1  23
        mov     r0,a            ;  1  f8
        anl     a,#0e0h         ;  1  54 e0
        xrl     a,r1            ;  1  69
        xch     a,r0            ;  1  c8
        anl     a,#1fh          ;  1  54 1f
        xrl     a,r2            ;  1  6a
        mov     r1,a            ;  1  f9
        ret                     ;  2  22

        ;21 cycles, 24 bytes


        ; KERMIT for 8051/2
        ; On entry A  = byte
        ;          R0 = old CRC low byte
        ;          R1 = old CRC high byte
        ; On exit  R0 = new CRC low byte
        ;          R1 = new CRC high byte
        ;          A,R2 = undefined

                                ; Cs  M/code
kermit:
        xrl     a,r0            ;  1  68
        mov     r0,a            ;  1  f8
        swap    a               ;  1  c4
        anl     a,#0f0h         ;  1  54 f0
        xrl     a,r0            ;  1  68
        mov     r0,a            ;  1  f8
        swap    a               ;  1  c4
        mov     r2,a            ;  1  fa
        anl     a,#0fh          ;  1  54 0f
        xrl     a,r1            ;  1  69
        xch     a,r2            ;  1  ca
        rr      a               ;  1  03
        mov     r1,a            ;  1  f9
        anl     a,#07h          ;  1  54 07
        xrl     a,r0            ;  1  68
        xch     a,r1            ;  1  c9
        anl     a,#0f8h         ;  1  54 f8
        xrl     a,r2            ;  1  6a
        mov     r0,a            ;  1  f8
        ret                     ;  2  22
        ;21 cycles, 24 bytes


        ; ARC for 8051/2
        ; On entry A  = byte
        ;          R0 = old CRC low byte
        ;          R1 = old CRC high byte
        ; On exit  R0 = new CRC low byte
        ;          R1 = new CRC high byte
        ;          A,R2,C = undefined

                                ; Cs  M/code
arc:
        xrl     a,r0            ;  1  68
        mov     r0,a            ;  1  f8
        rr      a               ;  1  03
        rr      a               ;  1  03
        anl     a,#0c0h         ;  1  54 c0
        mov     r2,a            ;  1  fa
        mov     a,r0            ;  1  e8
        mov     c,p             ;  1  a2 d0
        rrc     a               ;  1  13
        mov     r0,a            ;  1  f8
        clr     c               ;  1  c3
        rrc     a               ;  1  13
        xrl     a,r0            ;  1  68
        mov     r0,a            ;  1  f8
        rlc     a               ;  1  33
        mov     a,r2            ;  1  ea
        rlc     a               ;  1  33
        xrl     a,r2            ;  1  6a
        xrl     a,r1            ;  1  69
        xch     a,r0            ;  1  c8
        mov     r1,a            ;  1  f9
        ret                     ;  2  22

        ;23 cycles, 24 bytes


datalen equ     9
data:
        db      "123456789"
        end
```

Greg Cook, debounce@yahoo.co.uk
http://regregex.bbcmicro.net/crc-catalogue.htm Last updated 2012-02-07
Links last verified 2012-01-18

Hosted by netNORTH

[Top of page]