crc MegaCore Function

Parameterized CRC Generator/Checker

August 1997, ver. 1 — Data Sheet

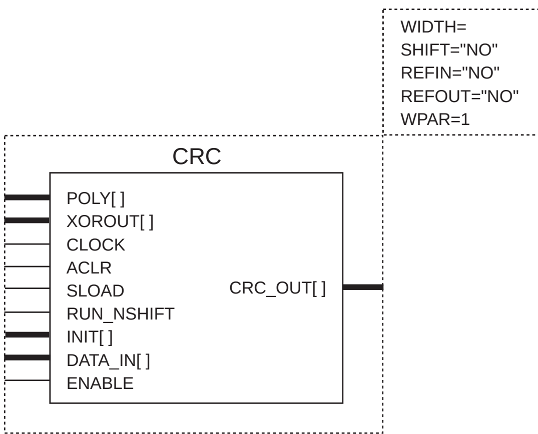
Features

- crc MegaCore™ function, general-purpose cyclic redundancy code (CRC) generator and checker
- Optimized for the FLEX® device architecture
- Supported by the MAX+PLUS® II development system
- High-speed operation, over 100 MHz for many configurations
- Fully parameterized, including:
  - Any length generator polynomial
  - Input data width, from 1 bit to the width of the polynomial
  - Any initial value
- Built-in support for:
  - Inverting output data
  - Reflecting (reversing bit order) input and output data

General Description

The crc MegaCore function is a general-purpose CRC generator and checker that validates data frames and ensures that data corruption during transmission is detected. The crc function is fully parameterized, and therefore can be used in virtually any design that requires a CRC checker. See Figure 1.

Figure 1. crc Symbol



A-DS-CRC-01

AHDL Function Prototype

The Altera® Hardware Description Language (AHDL™) Function Prototype of the crc function is shown below:

```
FUNCTION crc (poly[WIDTH-1..0], xorout[WIDTH-1..0],
  clock, aclr, sload, run_nshift, init[WIDTH-1..0],
  data_in[WPAR-1..0], enable)
  WITH (WIDTH, SHIFT, REFOUT, REFIN, WPAR)
  RETURNS (crc_out[WIDTH-1..0]);
```

VHDL Component Declaration

The VHDL Component Declaration of the crc function is shown below:

```
COMPONENT crc
  GENERIC (
    WIDTH      : POSITIVE;
    SHIFT      : STRING          := "NO";
    REFOUT     : STRING          := "NO";
    REFIN      : STRING          := "NO";
    WPAR       : POSITIVE        := 1);
  PORT (
    poly       : IN STD_LOGIC_VECTOR
                 (WIDTH-1 DOWNTO 0);
    xorout     : IN STD_LOGIC_VECTOR(WIDTH-1 DOWNTO 0)
                 := (OTHERS => '0');
    clock      : IN STD_LOGIC;
    aclr       : IN STD_LOGIC   := '0';
    sload      : IN STD_LOGIC;
    run_nshift : IN STD_LOGIC;
    init       : IN STD_LOGIC_VECTOR(WIDTH-1 DOWNTO 0);
    data_in    : IN STD_LOGIC_VECTOR(WPAR-1 DOWNTO 0);
    enable     : IN STD_LOGIC   := '1';
    crc_out    : OUT
    STD_LOGIC_VECTOR(WIDTH-1  DOWNTO 0) );
END COMPONENT;
```

Parameters

Table 1 describes the parameters of the crc function.

Table 1. crc Parameters

Name	Required	Default	Description
WIDTH	Yes	–	Width of the generator polynomial.
SHIFT	No	"NO"	If "YES" is specified, the run_nshift input is used. If "NO" is specified, the run_nshift input is not used.
REFIN	No	"NO"	If "YES" is specified, the crc function will reflect (bit reverse) the input data. The REFIN parameter allows a different bit order, e.g., some algorithms require the most significant bit (MSB) first, while others require the least significant bit (LSB) first.
REFOUT	No	"NO"	Specifies whether or not the output data bits are reflected.
WPAR	No	1	The WPAR parameter indicates the width of the input word. Some systems require data to be processed one bit at a time. In this case, WPAR would be set to 1. Other systems require that data be processed in bytes, words, or double words. In this case, WPAR would be set to 8,16, or 32 respectively. WPAR may be any factor of WIDTH.

Ports

Table 2 describes the input and output ports of the crc function.

Table 2. crc Ports (Part 1 of 2)

Name	Required	Type	Description
poly[WIDTH-1..0]	Yes	Input	The poly[] input is used to define the generator polynomial. However, the polynomial must first be converted to a binary value. For example, the CRC-16 generator polynomial is defined as: $X^{16} + X^{15} + X^{2} + X^{0}$, and can be transformed into a binary number by placing a logic 1 in every position where there is a non-zero power in the generator polynomial. Thus, the CRC-16 generator polynomial equals the following 17-bit binary number: 11000000000000101. Because every generator polynomial has a logic 1 as its MSB, the MSB is left off when specifying the polynomial as a binary number. Thus, the 17-bit binary number, which represents the CRC-16 generator polynomial, becomes the following 16-bit binary number: B"1000000000000101" or H"8005."
xorout[WIDTH-1..0]	No (Default = GND)	Input	Some CRC algorithms specify that the CRC register value be inverted before being output. When using one of these algorithms, the xorout[] input specifies which bits should be inverted, e.g., any bit with a logic 1 value in the xorout[] word will be inverted between the CRC register and the crc_out[] output.
clock	Yes	Input	Clock input.
aclr	No	Input	Asynchronous clear.
sload	Yes	Input	Synchronous load. Loads the value on the init[] bus into the CRC register.
run_nshift	No	Input	Run/shift. When high, the crc function is operating. When low, the crc register is serially shifted to the right. This input has an affect only when the shift port is set to "Yes."

Table 2. crc Ports (Part 2 of 2)

Name	Required	Type	Description
`init[WIDTH-1..0]`	Yes	Input	The `init[]` input specifies the initial value of the CRC register when the algorithm starts. This input allows the designer to dynamically place any value into the CRC register whenever the `sload` input is high, which also allows the designer to initialize the CRC register synchronously. The `init[]` input is used for both setting the initial value of the CRC and for starting mid-stream. In many systems, such as networking applications, data frames from different data streams are received in an interleaved order. For multiple data streams where the CRC must be calculated over many data frames, intermediate CRC values can be stored and loaded from a RAM buffer.
`data_in[WPAR-1..0]`	Yes	Input	Input data stream.
`enable`	No	Input	Clock enable
`crc_out[WIDTH-1..0]`	Yes	Output	Computed CRC output.

Table 3 summarizes the parameters for several standard CRC algorithms, including the algorithm width, polynomial value, initial value (hexadecimal radix), whether the bit order of the input and output data is reversed (reflected), the `XOR`ed output, and `check` values.

Table 3. Parameters for Various Standard CRC Algorithms Note (1)

NAME	WIDTH	POLY	INIT	REFIN	REFOUT	XOROUT	CHECK
		(Hexadecimal)				(Hexadecimal)	
CRC-16/ARC	16	8005	0000	`"YES"`	`"YES"`	0000	BB3D
CRC-16/CITT	16	1021	FFFF	`"NO"`	`"NO"`	0000	29B1
Kermit	16	8408	0000	`"YES"`	`"YES"`	0000	0C73
CRC-32/ ADCCP	32	04C11DB7	FFFFFFFF	`"YES"`	`"YES"`	FFFFFFFF	CBF43926
JamCRC	32	04C11DB7	FFFFFFFF	`"YES"`	`"YES"`	00000000	340BC6D9
ZMODEM	16	1021	0000	`"NO"`	`"NO"`	0000	31C3

Note:

(1) `check` is not a parameter, but a simple way to verify that the algorithm is working properly. The `check` word is the CRC output (`crc_out[]`) value when the ASCII string `"123456789"` is input to the CRC algorithm.

Functional Description

The crc function validates data streams via redundant encoding. CRCs are a preferred type of redundant encoding, where redundant bits are spread over more bits than the original data stream. Similar to parity checking, CRC encoding is a method of generating a code to verify the integrity of the data stream. However, while parity checking uses one bit to indicate even or odd parity, CRC encoding uses multiple bits, and therefore catches more errors in the data stream.

CRCs are particularly effective for two reasons:

- CRCs provide excellent protection against common errors such as burst errors, in which consecutive bits in a data stream are corrupted during transmission.
- The original data is the first part of the transmission, which makes systems that use CRCs easy to understand and implement.

The crc function is fully parameterized. Thus, virtually any CRC algorithm can be defined using the parameters described in this data sheet (see "Parameters" on page 3). To maximize flexibility, the crc function also allows designers to set port values, e.g., initial register values can be set via the init[ ] input. See Table 4.

Table 4. Sample crc MegaCore Function Performance & Logic Cell Usage

CRC Configuration with FLEX Devices *Note (1)*		**Size (LEs)** *Note (2)*	**Performance (MHz),** *Note (2)*	**Performance (Mbits/s)**
CRC-32 generator polynomial	32-bit wide input	318	28	896
	8-bit wide input	87	70	560
	1-bit wide input	32	>125	>125
CRC-16/CCITT generator polynomial	16-bit wide input	39	75	1200
	8-bit wide input	24	100	800
	1-bit wide input	16	>125	>125

Notes:
(1) The fastest speed grade from the FLEX 10K, FLEX 8000, and FLEX 6000 device families was used.
(2) The size and performance of the crc function will vary depending on the logic synthesis settings, device fitting, and the chosen polynomial.

The size of the crc function's generator polynomial can be defined to meet designer specifications. The larger the CRC polynomial length, the greater the chance of transmission error detection.

Refer to Rocksoft Corporation for more information on a generic parameterized model for CRC algorithms (see "References" on page 8 for details).

Pattern Generation Program

A vector generation program, available with the `crc` function, has the same parameters as the `crc` function and generates vector files to verify the operation of the `crc` function.

Figure 2 shows a sample implementation of the `crc` function, a CRC-32 algorithm with a byte-wide input.

Figure 2. crc Function Implementing a Byte-Wide CRC-32 Algorithm

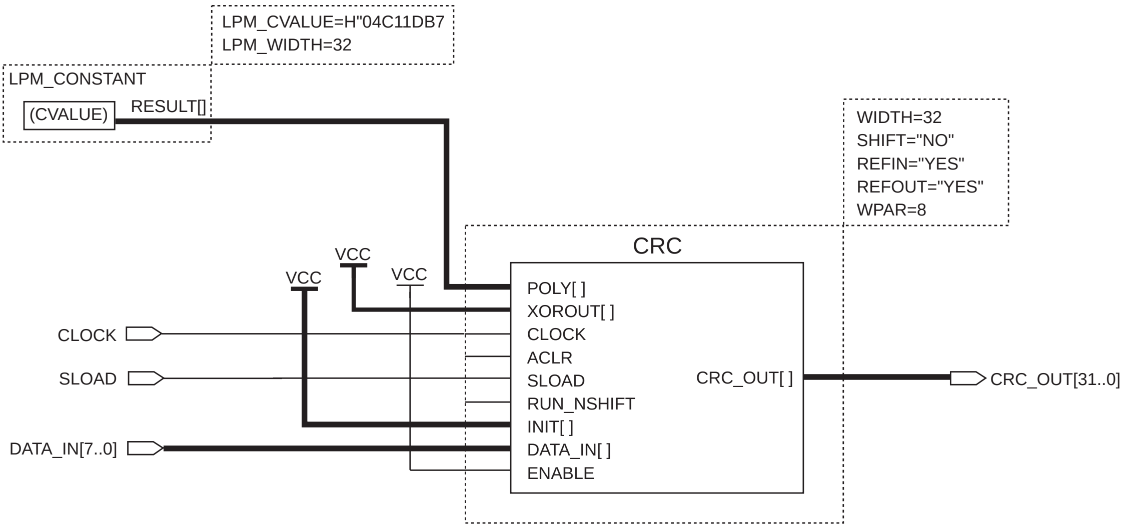
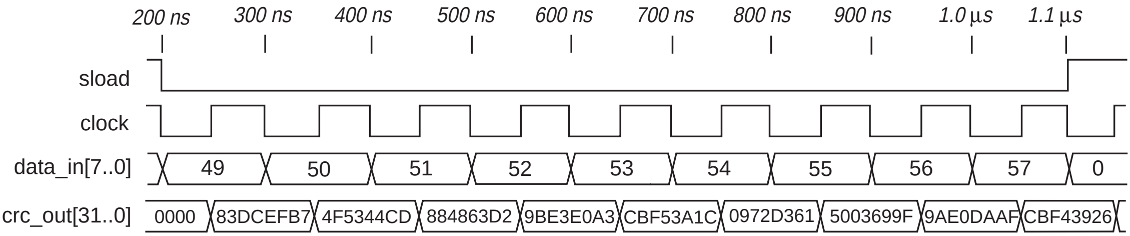


Figure 3 shows a simulation timing waveform of the `crc` function designed to implement a CRC-32 algorithm using the byte-wide (ASCII) input string "`123456789`." In ASCII format, "`1`" equals 49, so the string of characters is 49 to 57. The hexadecimal output of Figure 3 is H"CBF43926." Operating at 60 MHz in an EPF10K10-3 device, the design uses 101 FLEX LEs.

Figure 3. Simulation Timing Waveform of Byte-Wide CRC-32 Algorithm



References

Williams, Ross N. ***A Painless Guide to CRC Error Detection Algorithms.*** Version 3. Hazelwood Park, Australia: Rocksoft PTY Ltd, 1996.

This document explains CRCs and their table-driven implementations, and also provides a generic parameterized model CRC algorithm. For more information on this document, go to the Rocksoft world-wide web site at: **http://www.rocksoft.com.**

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
http://www.altera.com
Applications Hotline:
(800) 800-EPLD
Customer Marketing:
(408) 544-7104
Literature Services:
(888) 3-ALTERA
lit_req@altera.com