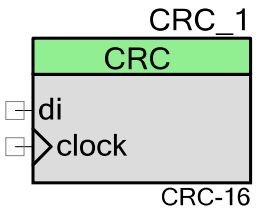
Cyclic Redundancy Check (CRC)

1.20

Features



- 1 - 64 Bits
- Requires clock and data for serial bit stream input
- Serial data in, parallel result out
- Standard (CRC-1, CRC-4-ITU, CRC-5-USB etc) or custom polynomial
- Standard or custom seed value

General Description

The default use of the Cyclic Redundancy Check (CRC) component is to compute CRC from a serial bit stream of any length. The input data is sampled on the rising edge of the data clock. The CRC value is reset to 0 before starting or can optionally be seeded with an initial value. On completion of the bitstream the computed CRC value may be read out.

When to use a CRC

The default the CRC component can be used as a checksum to detect alteration of data during transmission or storage. CRCs are popular because they are simple to implement in binary hardware, are easy to analyze mathematically, and are particularly good at detecting common errors caused by noise in transmission channels.

Input/Output Connections

This section describes the various input and output connections for the CRC. An asterisk (*) in the list of I/O's states that the I/O may be hidden on the symbol under the conditions listed in the description of that I/O.

clock – Input

The CRC requires a data input that provides the serial bitstream used to calculate the CRC. A data clock input is also required in order to correctly sample the serial data input. The input data is sampled on the rising edge of the data clock.

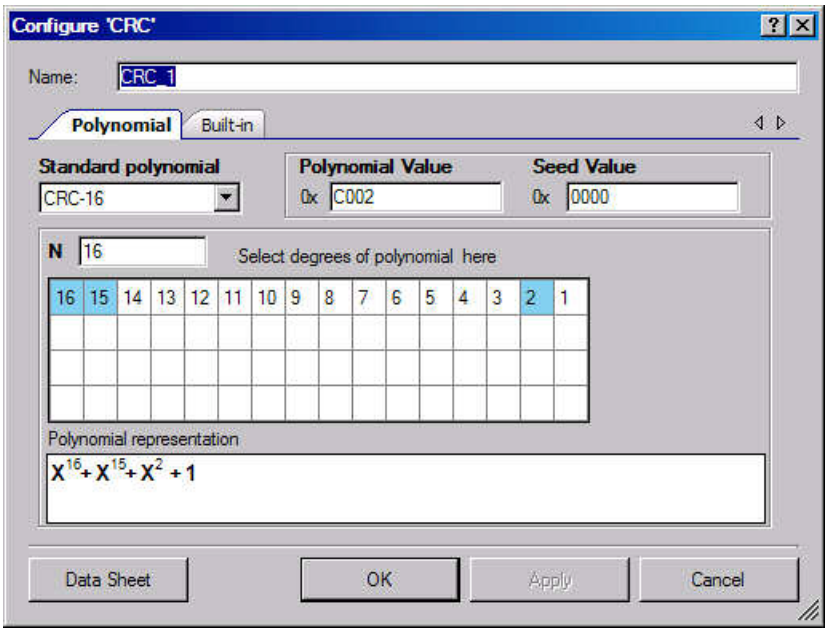
Note Generation of the proper CRC sequence for Resolution, which is greater than 8, requires two clock transitions.

di – Input

Data input that provides the serial bitstream used to calculate the CRC.

Parameters and Settings

Drag a CRC component onto your design and double-click it to open the Configure dialog.



Standard Polynomial

Allows you to choose one of the standard CRC polynomials provided in the Standard polynomial combo box or generate a custom polynomial. The additional information about each standard polynomial is given in the tool tip.

Polynomial Name	Polynomial	Use
Custom	User defined	General
CRC-1	$x + 1$	Parity
CRC-4-ITU	$x^4 + x + 1$	ITU G.704
CRC-5-ITU	$x^5 + x^4 + x^2 + 1$	ITU G.704
CRC-5-USB	$x^5 + x^2 + 1$	USB
CRC-6-ITU	$x^6 + x + 1$	ITU G.704
CRC-7	$x^7 + x^3 + 1$	telecom systems, MMC
CRC-8-ATM	$x^8 + x^2 + x + 1$	ATM HEC
CRC-8-CCITT	$x^8 + x^7 + x^3 + x^2 + 1$	1-Wire bus
CRC-8-Maxim	$x^8 + x^5 + x^4 + 1$	1-Wire bus
CRC-8	$x^8 + x^7 + x^6 + x^4 + x^2 + 1$	General
CRC-8-SAE	$x^8 + x^4 + x^3 + x^2 + 1$	SAE J1850

Polynomial Name	Polynomial	Use
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x + 1$	General
CRC-12	$x^{12} + x^{11} + x^3 + x^2 + x + 1$	telecom systems
CRC-15-CAN	$x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$	CAN
CRC-16-CCITT	$x^{16} + x^{12} + x^5 + 1$	XMODEM,X.25, V.41, Bluetooth, PPP, IrDA, CRC-CCITT
CRC-16	$x^{16} + x^{15} + x^2 + 1$	USB
CRC-24-Radix64	$x^{24} + x^{23} + x^{18} + x^{17} + x^{14} + x^{11} + x^{10} + x^7 + x^6 + x^5 + x^4 + x^3 + x + 1$	General
CRC-32-IEEE802.3	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$	Ethernet, MPEG2
CRC-32C	$x^{32} + x^{28} + x^{27} + x^{26} + x^{25} + x^{23} + x^{22} + x^{20} + x^{19} + x^{18} + x^{14} + x^{13} + x^{11} + x^{10} + x^9 + x^8 + x^6 + 1$	General
CRC-32K	$x^{32} + x^{30} + x^{29} + x^{28} + x^{26} + x^{20} + x^{19} + x^{17} + x^{16} + x^{15} + x^{11} + x^{10} + x^7 + x^6 + x^4 + x^2 + x + 1$	General
CRC-64-ISO	$x^{64} + x^4 + x^3 + x + 1$	ISO 3309
CRC-64-ECMA	$x^{64} + x^{62} + x^{57} + x^{55} + x^{54} + x^{53} + x^{52} + x^{47} + x^{46} + x^{45} + x^{40} + x^{39} + x^{38} + x^{37} + x^{35} + x^{33} + x^{32} + x^{31} + x^{29} + x^{27} + x^{24} + x^{23} + x^{22} + x^{21} + x^{19} + x^{17} + x^{13} + x^{12} + x^{10} + x^9 + x^7 + x^4 + x + 1$	ECMA-182

Polynomial Value

Represented in the hexadecimal form. It is calculated automatically when one of the standard polynomials is selected. You may also enter it manually (see Custom Polynomials).

Seed Value

Represented in the hexadecimal form. The maximum possible value is 2N-1.

N

Defines the degree of polynomial. Possible values include 1- 64 bits. The table with numbers indicates which degrees will be included in the polynomial. Cells with selected numbers are blue; others are white. The number of active cells is equal to N. Numbers are arranged in the reverse order. You may click on the cell to select or deselect a number.

Polynomial Representation

Displays the resulting polynomial with the mathematical notation.

Custom Polynomials

You may enter a custom polynomial in three different ways:

Small Changes to Standard Polynomial

- Choose one of the standard polynomials.
- Select the necessary degrees in the table by clicking on the appropriate cells; the text in **Standard Polynomial** will change to “Custom.”
- The polynomial value will be recalculated automatically based on the polynomial representation.

Use Polynomial Degrees

- Enter a custom polynomial in the **N** textbox; the text in **Standard Polynomial** will change to “Custom.”
- Select the necessary degrees in the table with numbers.
- Check the view of the polynomial with the **Polynomial Representation**.
- The polynomial value will be recalculated automatically based on the polynomial representation.

Use Hexadecimal Format

- Enter a polynomial value in the hexadecimal form in the **Polynomial Value** text box.
- Press [Enter] or switch to another control; the text in **Standard Polynomial** will change to “Custom."
- The N value and degrees of polynomial will be recalculated based on the entered polynomial value.

Clock Selection

TBD

Placement

TBD

Resources

TBD

Resolution	Digital Blocks					API Memory (Bytes)		Pins (per External I/O)
	Datapaths	Macro cells	Status Registers	Control Registers	Counter7	Flash	RAM	
1..8-Bits	1	2	0	1	0	3782	578	1
9..16-Bits	1	9	0	1	0	4094	585	1
17..24-Bits	2	10	0	1	0	4486	599	1
25..32-Bits	2	10	0	1	0	4589	599	1
33..64-Bits	4	10	0	1	0	5446	615	1

Application Programming Interface

Application Programming Interface (API) routines allow you to configure the component using software. The following table lists and describes the interface to each function. The subsequent sections cover each function in more detail.

By default, PSoC Creator assigns the instance name "CRC_1" to the first instance of a component in a given design. You can rename it to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is "CRC”.

Function	Description
void CRC_Start(void)	Initializes seed and polynomial registers. Computation of CRC starts on rising edge of input clock.
void CRC_Stop(void)	Stops CRC computation.
void CRC_ResetSeed(void)	Set default seed value.
void CRC_WriteSeed(uint8/16/32 seed)	Writes seed value.
void CRC_WriteSeedUpper(uint32 seed)	Writes upper half of seed value. Only generated for 33-64-bit CRC.
void CRC_WriteSeedLower(uint32 seed)	Writes lower half of seed value. Only generated for 33-64-bit CRC.
uint8/16/32 CRC_ReadCRC(void)	Reads CRC value.
uint32 CRC_ReadCRCUpper(void)	Reads upper half of CRC value. Only generated for 33-64-bit CRC.
uint32 CRC_ReadCRCLower(void)	Reads lower half of CRC value. Only generated for 33-64-bit CRC.

Function	Description
void CRC_WritePolynomial(uint8/16/32 polynomial)	Writes CRC polynomial value.
void CRC_WritePolynomialUpper(uint32 polynomial)	Writes upper half of CRC polynomial value. Only generated for 33-64-bit CRC.
void CRC_WritePolynomialLower(uint32 polynomial)	Writes lower half of CRC polynomial value. Only generated for 33-64-bit CRC.
uint8/16/32 CRC_ReadPolynomial(void)	Reads CRC polynomial value.
uint32 CRC_ReadPolynomialUpper(void)	Reads upper half of CRC polynomial value. Only generated for 33-64-bit CRC.
uint32 CRC_ReadPolynomialLower(void)	Reads lower half of CRC polynomial value. Only generated for 33-64-bit CRC.

void CRC_Start(void)

Description: Initializes seed and polynomial registers. Computation of CRC starts on rising edge of input clock.

Parameters: None

Return Value: None

Side Effects: None

void CRC_Stop(void)

Description: Stops CRC computation.

Parameters: None

Return Value: None

Side Effects: None

void CRC_ResetSeed(void)

Description: Sets default seed value.

Parameters: None

Return Value: None

Side Effects: None

void CRC_WriteSeed(uint8/16/32 seed)

Description: Writes seed value.

Parameters: (uint8/16/32) seed: Seed value.

Return Value: None

Side Effects: None

void CRC_WriteSeedUpper(uint32 seed)

Description: Writes upper half of seed value.Only generated for 33-64-bit CRC.

Parameters: (uint32) seed: Upper half of seed value.

Return Value: None

Side Effects: None

void CRC_WriteSeedLower(uint32 seed)

Description: Writes lower half of seed value. Only generated for 33-64-bit CRC.

Parameters: (uint32) seed: Lower half of seed value.

Return Value: None

Side Effects: None

uint8/16/32 CRC_ReadCRC(void)

Description: Reads CRC value.

Parameters: None

Return Value: (uint8/16/32) Returns CRC value.

Side Effects: None

uint32 CRC_ReadCRCUpper(void)

Description: Reads upper half of CRC value. Only generated for 33-64-bit CRC.

Parameters: None

Return Value: (uint32) Returns upper half of CRC value.

Side Effects: None

uint32 CRC_ReadCRCLower(void)

Description: Reads lower half of CRC value. Only generated for 33-64-bit CRC.

Parameters: None

Return Value: (uint32) Returns lower half of CRC value..

Side Effects: None

void CRC_WritePolynomial(uint8/16/32 polynomial)

Description: Writes CRC polynomial value.

Parameters: (uint8/16/32) polynomial: CRC polynomial.

Return Value: None

Side Effects: None

void CRC_WritePolynomialUpper(uint32 polynomial)

Description: Writes upper half of CRC polynomial value. Only generated for 33-64-bit CRC.

Parameters: (uint32) polynomial: Upper half CRC polynomial value.

Return Value: None

Side Effects: None

void CRC_WritePolynomialLower(uint32 polynomial)

Description: Writes lower half of CRC polynomial value.Only generated for 33-64-bit CRC.

Parameters: (uint32) polynomial: Lower half of CRC polynomial value.

Return Value: None

Side Effects: None

uint8/16/32 CRC_ReadPolynomial(void)

Description: Reads CRC polynomial value.

Parameters: None

Return Value: (uint8/16/32) Returns CRC polynomial value.

Side Effects: None

uint32 CRC_ReadPolynomialUpper(void)

Description: Reads upper half of CRC polynomial value. Only generated for 33-64-bit CRC.

Parameters: None

Return Value: (uint32) Returns upper half of CRC polynomial value.

Side Effects: None

uint32 CRC_ReadPolynomialLower(void)

Description: Reads lower half of CRC polynomial value.Only generated for 33-64-bit CRC.

Parameters: None

Return Value: (uint32) Returns lower half of CRC polynomial value.

Side Effects: None

Sample Firmware Source Code

The following is a C language example demonstrating the basic functionality of the CRC. This example assumes the component has been placed in the schematic and renamed to "CRC." The CRC component resolution is set to 32bits. Also, a Character LCD component has been placed to show CRC calculation results; it was renamed to "LCD."

```
#include <device.h>

void main()
{
    uint32 crc_val = 0;
    uint16 crc_part1 = 0;
    uint16 crc_part2 = 0;
    uint8 i = 0;
    uint8 j = 0;

    clock_Enable();
    di_Enable();
    LCD_Start();
    CRC_Start();

    for(i=0;i<4;i++)
    {
        for(j=0;j<=13;j+=5)
        {
            crc_val = CRC_ReadCRC();
            crc_part2 = HI16(crc_val);
            crc_part1 = LO16(crc_val);

            LCD_Position(i, j);
            LCD_PrintInt16(crc_part2);
```

```
            j+=4;
            LCD_Position(i, j);
            LCD_PrintInt16(crc_part1);

            CyDelay(500);
        }
    }

      for(;;){}
}
```

Functional Description

The CRC is implemented as a linear feedback shift register (LFSR). The Shift register computes the LFSR function; the Polynomial register holds the polynomial that defines the LFSR polynomial; and the Seed register enables initialization of the starting data.

This component requires that the Seed and Polynomial registers are initialized prior to start.

Computation of an N-bit LFSR result is specified by a polynomial with N+1 terms, the last of which is the X0 term where X^0=1. For example, the widely used CRC-CCITT 16-bit polynomial is $X^{16}+X^{12}+X^5+1$. The CRC algorithm assumes the presence of the X^0 term, so that the polynomial for an N-bit result can be expressed by an N bit rather than N+1-bit specification.

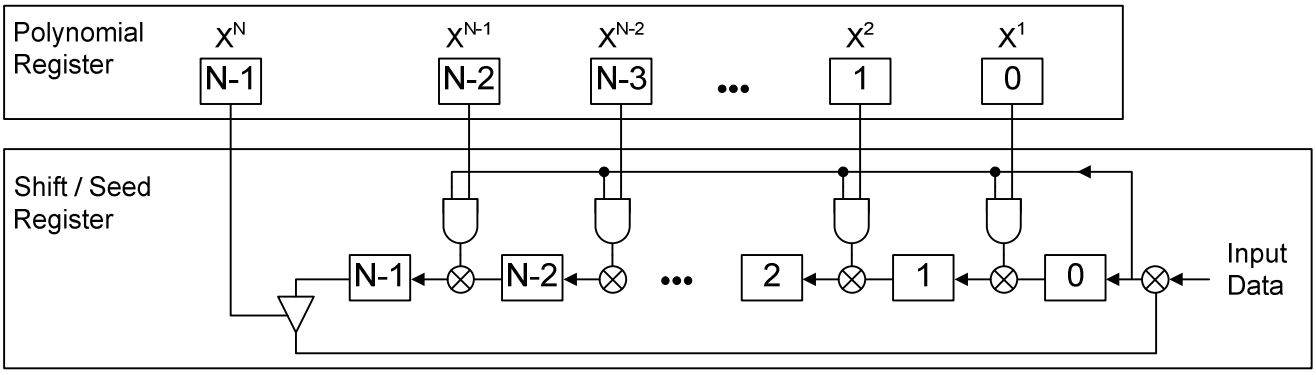
To specify the polynomial specification, write an N+1 bit binary number corresponding to the full polynomial, with 1's for each term present. The CRC-CCITT polynomial would be 10001000000100001b. Then, drop the right-most bit (the X^0 term) to obtain the CRC polynomial value. To implement the CRC-CCITT example, the Polynomial register is loaded with the value of 8810h.

A rising edge of the input clock shifts each bit, MSB first, of the input data stream through the Shift register, computing the specified CRC algorithm. Eight clocks are required to compute the CRC for each byte of input data.

Note that the initial seed value is lost. This is usually of no consequence since the seed value is only used to initialize the Shift register once, per data set.

Block Diagram and Configuration

Add information here about the data paths used and how the registers are used inside of those data paths. Also include if writing a register causes something to happen etc.



DC and AC Electrical Characteristics

The following values are indicative of expected performance and based on initial characterization data.

5.0V/3.3V DC and AC Electrical Characteristics

Parameter	Typical	Min	Max	Units	Conditions and Notes
Input					
Input Voltage Range	---		Vss to Vdd	V	
Input Capacitance	---		---	pF	
Input Impedance	---		---	Ω	
Maximum Clock Rate	---		67	MHz	

Component Changes

This section lists the major changes in the component from the previous version.

Version	Description of Changes	Reason for Changes / Impact
1.20	Changed method of API generation. In version 1.10, APIs were generated by settings from the customizer. For 1.20, APIs are provided by the .c and .h files like most other components.	This change allows users to view and make changes to the generated API files, and they will not be overwritten on subsequent builds.
	Seed and Polynomial parameters were changed to have hexadecimal representation.	Change was made to comply with corporate standard.