Cyclic Redundancy Check

April 2011 **Reference Design RD1105**

Introduction

The Cyclic Redundancy Check (CRC) is an efficient technique for detecting errors during digital data transmissions between a source and a destination. The destination device calculates the CRC of the received data. If the CRC calculated by the destination device does not match the one calculated by the source device, then the received data contains an error. This technique is used in a wide variety of applications from Ethernet transmission to daily file transfers. It provides quick and easy insurance of data integrity within digital communication systems The CRC is based on polynomial manipulations which treat each received message as a binary number. The received message is then divided by a fixed value, also known as the generator polynomial, using modulo-2 arithmetic. The characteristic of the CRC implementation is determined by the generator polynomial selection. The generator polynomials are selected to maximize the error detection capability without using too many resources. Generator polynomials that have been incorporated into standards such as CRC-8, CRC-16 and CRC-CCIT are commonly known and are well tested.

This reference design describes the use of Lattice programmable devices to implement the CRC generator and checker. The design allows users to implement the CRC using different generator polynomials.

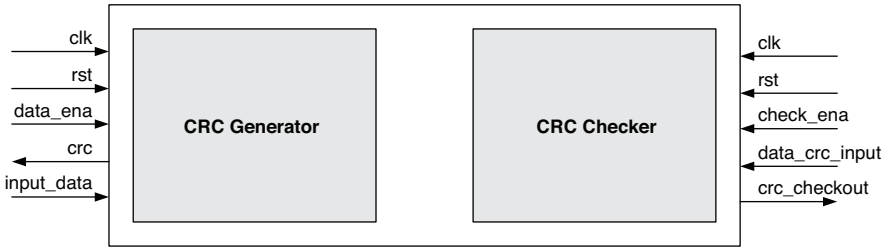
Features

- Parameterized data width
- Supports polynomial orders from CRC-1 to CRC-64
- Supports both CRC generators and CRC checkers
- Allows transposing of incoming data bytes, transposing CRC output bytes
- Allows complement input data bytes and complement output CRC bytes

Functional Description

The functional block diagram of this reference design is shown in Figure 1.

Figure 1. Functional Block Diagram



Signal Descriptions

Table 1. Signal Descriptions

Signal Name	Direction	Active State	Description
CRC Generator			
clk	Input	N/A	Clock signal
rst	Input	High	Asynchronous reset signal
data_ena	Input	High	Input data enable signal
input_data[INPUT_DATA_WIDTH-1 : 0]	Input	N/A	Input data being transmitted
crc[CRC_WIDTH-1 : 0]	Output	N/A	CRC code
CRC Checker			
clk	Input	N/A	Clock signal
rst	Input	High	Asynchronous reset signal
check_ena	Input	High	Input enable signal
data_crc_input [INPUT_DATA_WIDTH+ CRC_WIDTH-1 : 0]	Input	N/A	Received message appending CRC code
crc_checkout[CRC_WIDTH-1 : 0]	Output	N/A	CRC checker out code

Table 2. Parameter Descriptions

Parameter	Description	Active Value	Default Value
INPUT_DATA_WIDTH	Specifies the width of the message to be transmitted.	2 to 256	8
CRC_WIDTH	Specifies the CRC code width.	1 to 64	16
INPUT_DATA TRANSPOSE	Specifies whether the message to be transmitted is transposed before generating the CRC code.	0 = Not transposed 1 = Transposed	0
CRC_TRANSPOSE	Specifies whether the CRC code is transposed after generating the CRC code.	0 = Not transposed 1 = Transposed	0
INPUT_DATA_COMPLEMETN	Specifies whether the message to be transmitted is complemented before generating the CRC code.	0= Not complemented 1= Complemented	0
CRC_COMPLEMENT	Specifies whether the CRC code is complemented after generating the CRC code.	0= Not complemented 1= Complemented	0
POLYNOMINAL	The CRC polynomial, which can be specified between CRC-1 and CRC-64. The LSB and MSB positions of the polynomial value must be 1. The default value is 0x8005 (CRC-16).	CRC-1 to CRC-64	0x8005

CRC Generator

A polynomial called generator polynomial must be chosen before the user computes the CRC of a transmitted message. The generator polynomial must have a degree greater than zero and a non-zero coefficient in the MSB and LSB positions. An attribute of the generator polynomial is that its length is equal to the degree +1. For example, in CRC-8, the degree is 8 and the length is 9. The degree of the generator polynomial determines the length of the CRC code. For example, if the degree of the generator polynomial is 16, then the length of the CRC code is 16.

In this reference design, a *n* number of zero ('0') bits (*n* is the degree of the generator polynomial) is appended to the transmitted message before the *n*-bit CRC code is computed. Modulo-2 arithmetic (XOR operation) is implemented when computing *n*-bit CRC code, as shown in the example below. In this example, the generator polynomial is chosen as CRC-16-IBM (11000000000000101) and the transmitted message is chosen as 0xAA (10101010). Before the CRC code is computed, 16 bits of zeros are appended to the 0xAA and line the bits in a row, as follows:

10101010000000000000000

To compute the CRC code, position the generator polynomial underneath the left-hand end of the row, then XOR the generator polynomial into the input, as follows:

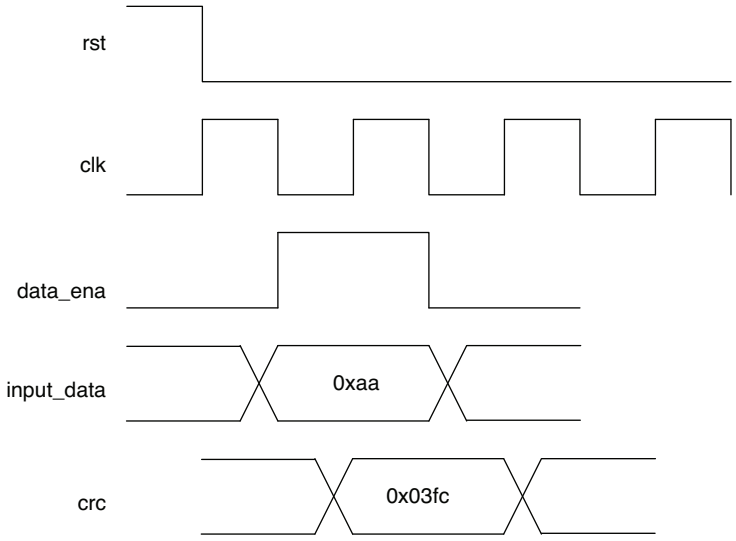
```
10101010000000000000000  <-- Transmitted message appending n-bit zero
11000000000000101        <-- Generator polynomial
-----------------------  <-- XOR
01101010000001010000000  <-- Result
```

If the resulting MSB bit is a '0', the result is shifted to the right until the left-most bit (LSB) is a '1'. Position the generator polynomial under the non-zero (left-hand end) of the result and do the XOR calculation again. This process is repeated until the generator polynomial reaches the right-hand end of the result. The following is the entire calculation of this example:

```
10101010000000000000000  <-- Transmitted message appending n-bit zero
11000000000000101        <-- Generator polynomial
-----------------------  <-- XOR
01101010000001010000000  <-- Result
 11000000000000101       <-- Generator polynomial
 ----------------------  <-- XOR
 00010100000001111000000 <-- Result
  11000000000000101      <-- Generator polynomial
  ---------------------  <-- XOR
  01100000001111101000   <-- Result
   11000000000000101     <-- Generator polynomial
   --------------------  <-- XOR
   00000000011111111100  <-- CRC (0x03fc )
```

Figure 2 shows the timing diagram of the CRC generator.

Figure 2. CRC Generator Timing (Polynomial = 11000000000000101)



Before calculating the CRC code, the user can transpose the input data. For example, if the input data is 0x55 and the parameter INPUT_DATA TRANSPOSE is set to '1', then this design uses data 0xAA to generate the CRC code.

Before calculating the CRC code, the user can complement the input data. For example, if the input data is 0xF0 and the parameter INPUT_DATA_COMPLEMETN is set to '1', then this design uses data 0x0F to generate the CRC code.

After calculating the CRC code, the user can transpose it. For example, if the CRC code is 0x55 and the parameter CRC_TRANSPOSE is set to '1', then the CRC code becomes 0xAA.

After calculating the CRC code, the user can complement the it. For example, if the CRC code is 0xF0 and the parameter CRC_COMPLEMENT is set to '1', then the CRC code becomes 0x0F.

CRC Checker

To check the CRC after the data transmission, the user passes the transmitted message appending the CRC code through the CRC checker module. The calculation of the CRC checker is the same as the CRC generator.

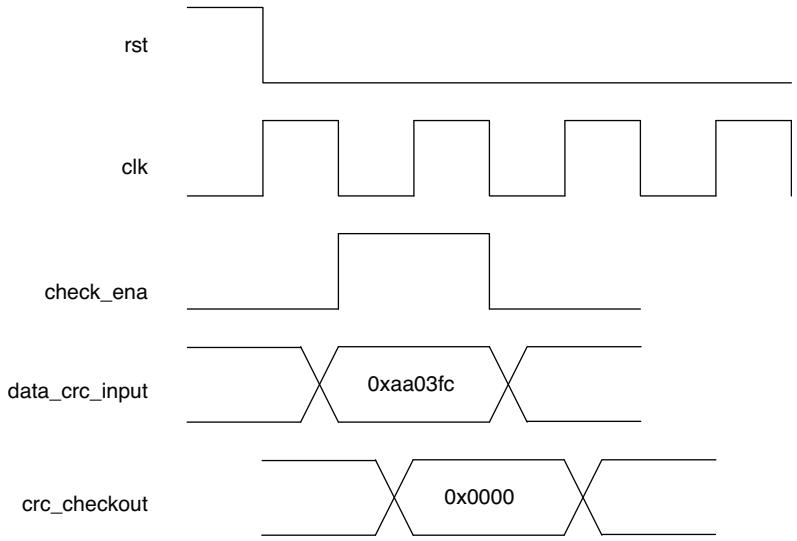
The above example is used to explain the calculation of the CRC checker, as shown below:

```
10101010000000111111100   <-- Transmitted message appending crc code
11000000000000101         <-- Generator polynomial
--------------------------<-- XOR
01101010000000101111100   <-- Result
 11000000000000101        <-- Generator polynomial
 -------------------------<-- XOR
 00010100000000000111100  <-- Result
    11000000000000101     <-- Generator polynomial
    ------------------    <-- XOR
    01100000000000010100  <-- Result
     11000000000000101    <-- Generator polynomial
     -----------------    <-- XOR
     00000000000000000    <-- CRC_checkout (0x0000 )
```

If the checkout value is '0' (all bits '0'), then there was no error during the transmitting of the message.

Figure 3 shows the timing of CRC checker.

Figure 3. CRC Checker Timing Diagram (Polynomial = 11000000000000101)



Test Bench Description

The test bench simulates the CRC generator and CRC checker in this design with default settings.

Figure 4. CRC Generator

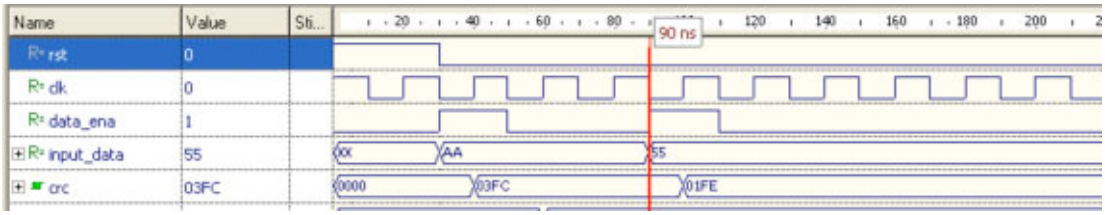
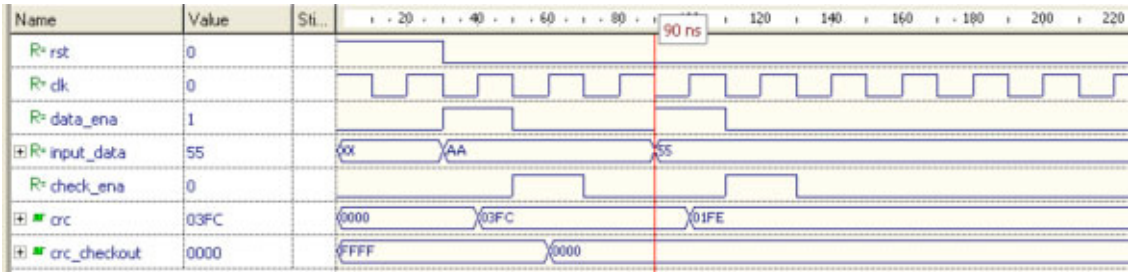


Figure 5. CRC Checker



Implementation

This design is implemented in Verilog and VHDL. When using this design in a different device, density, speed, or grade, performance and utilization may vary. Default settings are used during the fitting of the design.

Table 3. Performance and Resource Utilization[1]

Device Family	Language	Speed Grade	Utilization	f_{MAX} (MHz)	I/Os	Architecture Resources
MachXO2™ [2]	Verilog	-6	26 LUTs	>50	44	N/A
	VHDL	-6	26 LUTs	>50	44	N/A
MachXO™ [3]	Verilog	-3	25 LUTs	>50	44	N/A
	VHDL	-3	25 LUTs	>50	44	N/A

1. Use default settings: INPUT_DATA_WIDTH=8; CRC_WIDTH=16; INPUT_DATA_TRANSPOSE=0; CRC_TRANSPOSE=0; INPUT_DATA_COMPLEMENT=0; CRC_COMPLEMENT=0; POLYNOMIAL=0x8005.
2. Performance and utilization characteristics are generated using LCMXO2-1200HC-6TG144C with Lattice Diamond™ 1.2 design software.
3. Performance and utilization characteristics are generated using LCMXO640C-3T100C with Lattice Diamond 1.2 design software.

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)
+1-503-268-8001 (Outside North America)
e-mail: techsupport@latticesemi.com
Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
April 2011	01.0	Initial release.