Cyclic Redundancy Code Generator Macro

Features

The highlights of the Cyclic Redundancy Codes (CRC) Generator are as follow:

- CRC Calculation
- Multiple Algorithm Support
  - CRC8 ATM (HEC)
  - CRC10 ATM (OAM)
  - CANbus
  - X25 (SDLC, HDLC, CRC-CCITT)
  - CRC16
  - CRC16 inverted
  - CRC32
  - User defined polynomial
- A Win32 console application that generates VHDL code
- Performance (CRC32, 54SXA –2)
  - Serial → 200 MHz
  - 32-bit Parallel, 8-bit Data—130 MHz

The CRC Generator application generates VHDL code specifically for use in the SX-A, ProASIC, and ProASIC^{PLUS} families. The RTL VHDL code generated is simply designed to take advantage of the SX-A silicon features. This is necessary to meet the timing requirement of greater than 125 MHz—oriented toward telecommunications applications.

Functional Description

Many designers use the CRC as an alternative to parity and checksum calculation for checking (and sometimes correcting) data transmission errors. A 16-bit CRC detects all single and double-burst errors and ensures detection of 99.998% of all possible errors. This level of error protection is considered sufficient for data transmission blocks of 4 kilobytes or less. For larger transmissions, a 32-bit CRC is often employed.

Gigabit Ethernet, ATM, and other higher-speed protocols require the use of CRC, although the exact CRC polynomial changes depend upon the transmission protocol (Figure 1). In ATM, an 8-bit CRC is used for the header error check, and in addition (optional) CRC error checking is often done for data. For larger transmissions, a 32-bit CRC is often used.

The CRC method treats the data frame as a large binary number. This number is then divided (at the generator end) by a fixed binary number (the generator polynomial) and the resulting CRC value is appended to the end of the data frame. The receiver then repeats the calculation and compares its CRC value with the generated CRC value. The classic serial implementation method uses a shift register with XOR gates and feedback taps as shown in Figure 1 for the X25 polynomial ($G(x)=x^{16}+x^{12}+x^5=1$).

A serial CRC implementation is suitable up to about 100 Mb/s using FPGAs. The blocks described here can be used for higher rates including Gigabit Ethernet (1 Gb/sec serial rate stepped down to 125 MHz and ATM (620 Mb/sec serial rate stepped down to 77.5 MHz).

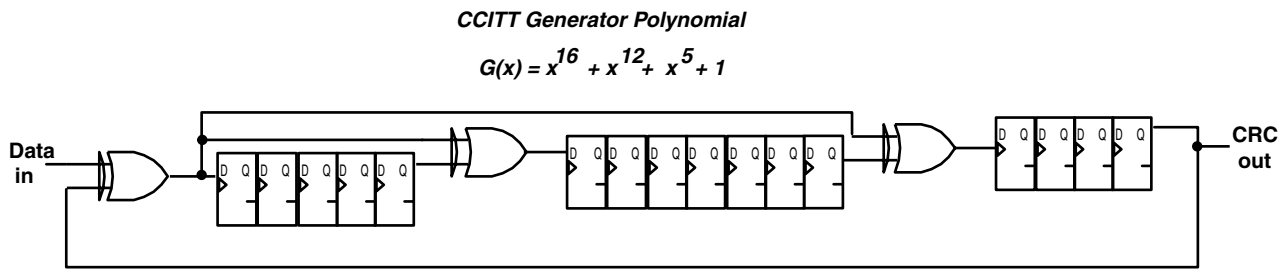

Figure 1 • Serial CRC Calculation

The CRC Shift Sequence

The initial contents of the feedback shift register are shown in the top row; namely, C15 through C0. After the first data bit D0 is shifted into the shift register, the new contents of the shift register are functions of D0 and the previous contents. We continue to shift until eight shifts have been made, as shown in the diagram. The new shift register contents are now functions of both the original contents and the 8 data bits after 8 clocks, as shown in Figure 2.

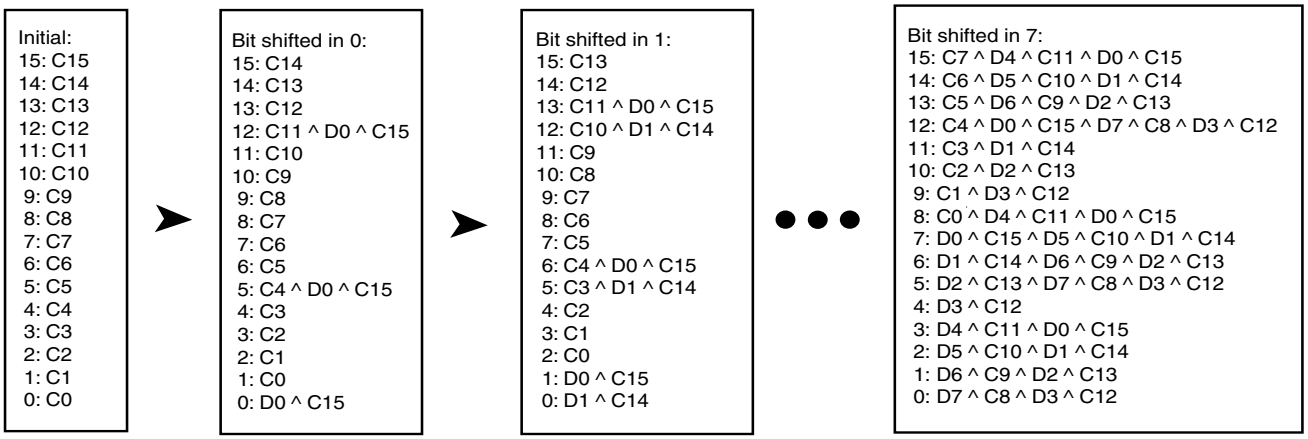

Figure 2 • *Serial to Parallel CRC Sequence*

If input data bits are available in parallel, the number of steps to generate the CRC can be reduced. In our example, the 16-bit CRC generation would be a two-step process- one for each set of eight input data bits. The same technique extends to 32-bit CRCs. Alternatively, we can expand the equations to do a full 16- or 32-bit CRC in one step.

There is no pipelining other than the CRC register itself. In an N-bit serial CRC, the data becomes available on the Nth clock edge after the first one (which clocks the first bit into the shift register). This is actually cycle N+1. For an N-bit parallel CRC, with M data bits available, N/M should be an integer. Again, if cycle 1 is the cycle on which data is clocked into the N/M bit shift register, the first M bits are available in parallel on cycle N/M+1, the next M bits on cycle N/M+2, etc.

This widely used method is slow for Gigabit Ethernet and high-speed ATM, where bit rates can top 100 Mbps. An alternative method is parallel computation of the CRC. Serial-to-parallel conversion of the data effectively divides the input clock frequency by 8, 16, or 32. We can now calculate an N-bit CRC M-bits at a time. A typical implementation is shown in Figure 3 for an 8-bit parallel 16-bit CRC.

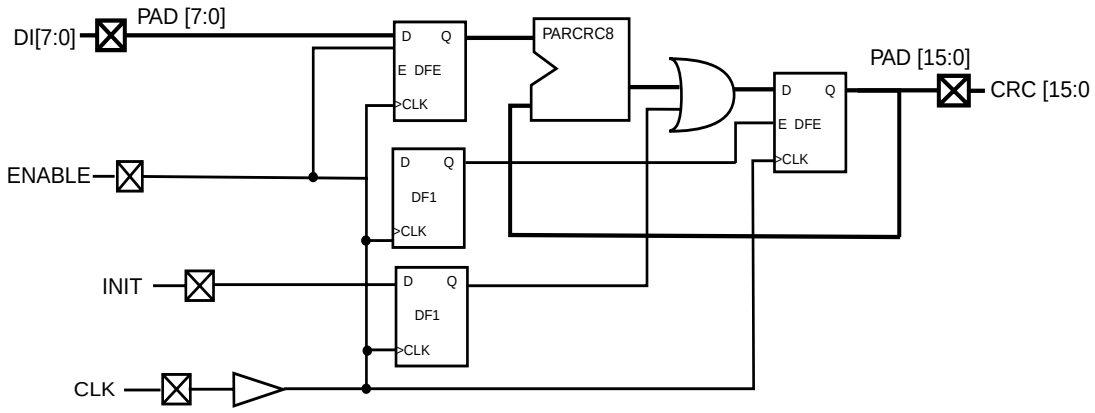

Figure 3 • *8-Bit Parallel CCITT-16 CRC*

Top-Level Interface

Top-Level Block Diagrams

Figure 4 illustrates the serial and parallel cases for the CRC block. In addition, Table 1 describes the signals for the serial and parallel CRC.

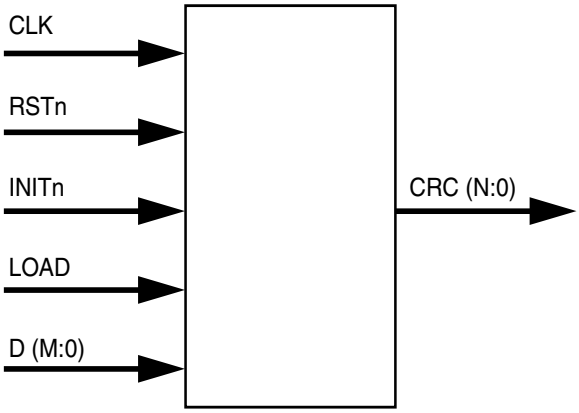

Figure 4 • CRC Block

Table 1 • Top-Level Signal Description

CLK	Input	This is the master clock signal.
RSTn	Input	Asynchronous system reset (initialization) signal.
INITn	Input	Synchronous reset (initialization) signal.
LOAD	Input	In parallel mode, when this signal is high the macro is enabled allowing the CRC to be calculated on incoming data. When this signal is low the previous CRC value is retained on the CRC output. In serial mode, when this signal is high the macro is enabled allowing the CRC to be calculated on incoming data. When this signal is low the calculated CRC is shifted out unaltered by incoming data.
D(M:0)	Input	CRC input data – 8, 16, or 32 bits. For serial CRC computation, this is a single bit.
CRC(N:)	Output	CRC output data – 8, 16, or 32 bits. For serial CRC computation, this is a single bit.

Deliverables

A Win32 console application (NT/Win95/Win98/Win2000 compatible) allows choosing the desired polynomial from the menu. There are several common generator polynomials supported with the Actel CRC Macro as follow:

sCRC8, ATM (HEC)

x^8+x^2+x+1

CRC10, ATM (OAM Cell)

$x^{10}+x^9+x^5+x^4+x+1$

CANbus

$x^{15}+x^{14}+x^{10}+x^8+x^7+x^4+x^3+1$

CRC16

$x^{16}+x^{12}+x^2+1$

CRC16 inverted

$x^{16}+x^{14}+x+1$

X25 (SDLC, HDLC, CRC-CCITT)

$x^{16}+x^{12}+x^5+1$

CRC32 (Ethernet, FDDI)

$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$

If the user wants a different polynomial, an "Enter your polynomial" option is available. For example, the X25 polynomial would be entered as 10001000000100001. User-defined polynomials up to 32nd order (comparable to CRC32, for example) are supported.

In addition, the number of input data bits to be calculated in parallel (up to 32) can be selected. The program will generate a fully synthesizable, completely behavioral VHDL description.

A simple test bench has been implemented illustrating CRC generation using the X25 polynomial and is supplied with the macro. The test bench reads in random data and calculates the CRC both serially and with 8-bit parallel data. At the end of the simulation, the two CRCs are compared to verify that the parallel CRC was implemented correctly (serial CRCs are correct by inspection). The test bench can be easily modified to accommodate different polynomials.

Estimated Performance and Device Utilization

The expected performance and utilization statistics are given in Table 2 and Table 3 for serial and parallel CRC implementations using the SX-A –3 and ProASIC^{PLUS} devices. The clock frequency is the rate at which the CRC is clocked. The serial data rate is the effective system clock rate assuming serialization and deserialization of the data for the parallel CRC cases.

Table 2 • *54SX-A Utilization and Performance Statistics*

CRC Configuration with an SXA08 Device		Sequential/Total Used Modules	Percent (%) 54SXA08	Maximum Clock Frequency (MHz)[1]	Maximum System Frequency
CRC-32	32-bit wide input	32/392	50	114	3.6 GHz
Generator	8-bit wide input	32/36	19	149	1.2 GHz
Polynomial	1-bit wide input	32/115	9	255	255MHz
CRC-16/CCITT	16-bit wide input	16/106	11	166	2.6 GHz
Generator	8-bit wide input	16/57	7	240	1.9 GHz
Polynomial	1-bit wide input	16/35	6.5	297	297 MHz

1. TDPR was used to obtain the timing results. Pipelining the "d" inputs to the macro were also used to obtain parallel timing results.

Table 3 • *ProASIC^{PLUS} Utilization and Performance Statistics*

CRC Configuration with an APA750 Device		Sequential/Total Used Tiles	Percent (%) APA750	Maximum Clock Frequency (MHz)[1]	Maximum System Frequency
CRC-32	32-bit wide input	177/32768	0.5	82	2.4 GHz
Generator	8-bit wide input	177/32768	0.5	82	656 MHz
Polynomial	1-bit wide input	105/32768	0.3	130	130 MHz
CRC-16/CCITT	16-bit wide input	128/32768	0.4	102	1.6 GHz
Generator	8-bit wide input	80/32768	0.2	106	848 MHz
Polynomial	1-bit wide input	38/32768	0.1	169	160 MHz

1. TDPR was used to obtain the timing results. Pipelining the "d" inputs to the macro were also used to obtain parallel timing results.

http://www.actel.com

Actel Europe Ltd.
Maxfli Court, Riverside Way
Camberley, Surrey GU15 3YL
United Kingdom
Tel: +44 (0)1276 401450
Fax: +44 (0)1276 401590

Actel Corporation
955 East Arques Avenue
Sunnyvale, California 94086
USA
Tel: 408-739-1010
Fax: 408-739-1540

Actel Asia-Pacific
EXOS Ebisu Bldg. 4F
1-24-14 Ebisu Shibuya-ku
Tokyo 150 Japan
Tel: +81-(0)3-3445-7671
Fax: +81-0)3-3445-7668

5172158-1/1.02