CRC Generator for Verilog or VHDL

By Evgeni Stavinov, OutputLogic.com

Description

CRC Generator is a command-line application that generates Verilog or VHDL code for CRC of any data width between 1 and 1024 and polynomial width between 1 and 1024. The code is written in C and is cross-platform compatible

Parameters

language: verilog or vhdl

data_width : data bus width {1..1024}
poly_width : polynomial width {1..1024}

poly_string : a string that describes CRC polynomial.

Examples: 05 = x^5+x^2+1
8005 = $x^{16}+x^{15}+x^2+1$

Note: string representation (0x05, 0x8005) doesn't include highest degree coefficient in polynomial representation (x^5 and x^{16} in the above examples)

Output Examples

[1] C:\OutputLogic> crc-gen

usage:
crc-gen language data_width poly_width poly_string

parameters:
language : verilog or vhdl
data_width : data bus width {1..1024}
poly_width : polynomial width {1..1024}
poly_string : polynomial string in hex

example: usb crc5 = x^5+x^2+1
crc-gen verilog 8 5 5

[2] C:\OutputLogic> crc-gen verilog 8 5 05

```
//------------------------------------------------------------------------------
// Copyright (C) 2009 OutputLogic.com
// This source file may be used and distributed without restriction
// provided that this copyright statement is not removed from the file
// and that any derivative work contains the original copyright notice
// and the associated disclaimer.
// THIS SOURCE FILE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS
// OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
// WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.
//------------------------------------------------------------------------------
// CRC module for
//      data[7:0]
//      crc[4:0]=1+x^2+x^5;
//
module crc(
      input [7:0] data_in,
      input       crc_en,
      output [4:0] crc_out,
      input       rst,
      input       clk);

      reg [4:0] lfsr_q,
                lfsr_c;
      assign crc_out = lfsr_q;
      always @(*) begin
            lfsr_c[0] = lfsr_q[0] ^ lfsr_q[2] ^ lfsr_q[3] ^ data_in[0] ^ data_in[3] ^ data_in[5] ^ data_in[6];
            lfsr_c[1] = lfsr_q[1] ^ lfsr_q[3] ^ lfsr_q[4] ^ data_in[1] ^ data_in[4] ^ data_in[6] ^ data_in[7];
            lfsr_c[2] = lfsr_q[0] ^ lfsr_q[3] ^ lfsr_q[4] ^ data_in[0] ^ data_in[2] ^ data_in[3] ^ data_in[6] ^ data_in[7];
            lfsr_c[3] = lfsr_q[0] ^ lfsr_q[1] ^ lfsr_q[4] ^ data_in[1] ^ data_in[3] ^ data_in[4] ^ data_in[7];
            lfsr_c[4] = lfsr_q[1] ^ lfsr_q[2] ^ data_in[2] ^ data_in[4] ^ data_in[5];


      end // always

      always @(posedge clk, posedge rst) begin
            if(rst) begin
                  lfsr_q  <= {5{1'b1}};
            end
            else begin
                  lfsr_q  <= crc_en ? lfsr_c : lfsr_q;
            end
      end // always
endmodule // crc
```

[3] C:\OutputLogic> crc-gen verilog 8 16 8005

```
//------------------------------------------------------------------------------
// Copyright (C) 2009 OutputLogic.com
// This source file may be used and distributed without restriction
// provided that this copyright statement is not removed from the file
// and that any derivative work contains the original copyright notice
// and the associated disclaimer.
// THIS SOURCE FILE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS
// OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
// WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.
```

```
//------------------------------------------------------------------------------
// CRC module for
//       data[7:0]
//       crc[15:0]=1+x^2+x^15+x^16;
//
module crc(
      input [7:0] data_in,
      input       crc_en,
      output [15:0] crc_out,
      input       rst,
      input       clk);

      reg [15:0] lfsr_q,
                 lfsr_c;
      assign crc_out = lfsr_q;

      always @(*) begin
            lfsr_c[0] = lfsr_q[8] ^ lfsr_q[9] ^ lfsr_q[10] ^ lfsr_q[11] ^ lfsr_q[12] ^ lfsr_q[13] ^ lfsr_q[14] ^ lfsr_q[15] ^ data_in[0]
^ data_in[1] ^ data_in[2] ^ data_in[3] ^ data_in[4] ^ data_in[5] ^ data_in[6] ^ data_in[7];
            lfsr_c[1] = lfsr_q[9] ^ lfsr_q[10] ^ lfsr_q[11] ^ lfsr_q[12] ^ lfsr_q[13] ^ lfsr_q[14] ^ lfsr_q[15] ^ data_in[1] ^ data_in[2]
 ^ data_in[3] ^ data_in[4] ^ data_in[5] ^ data_in[6] ^ data_in[7];
            lfsr_c[2] = lfsr_q[8] ^ lfsr_q[9] ^ data_in[0] ^ data_in[1];
            lfsr_c[3] = lfsr_q[9] ^ lfsr_q[10] ^ data_in[1] ^ data_in[2];
            lfsr_c[4] = lfsr_q[10] ^ lfsr_q[11] ^ data_in[2] ^ data_in[3];
            lfsr_c[5] = lfsr_q[11] ^ lfsr_q[12] ^ data_in[3] ^ data_in[4];
            lfsr_c[6] = lfsr_q[12] ^ lfsr_q[13] ^ data_in[4] ^ data_in[5];
            lfsr_c[7] = lfsr_q[13] ^ lfsr_q[14] ^ data_in[5] ^ data_in[6];
            lfsr_c[8] = lfsr_q[0] ^ lfsr_q[14] ^ lfsr_q[15] ^ data_in[6] ^ data_in[7];
            lfsr_c[9] = lfsr_q[1] ^ lfsr_q[15] ^ data_in[7];
            lfsr_c[10] = lfsr_q[2];
            lfsr_c[11] = lfsr_q[3];
            lfsr_c[12] = lfsr_q[4];
            lfsr_c[13] = lfsr_q[5];
            lfsr_c[14] = lfsr_q[6];
            lfsr_c[15] = lfsr_q[7] ^ lfsr_q[8] ^ lfsr_q[9] ^ lfsr_q[10] ^ lfsr_q[11] ^ lfsr_q[12] ^ lfsr_q[13] ^ lfsr_q[14] ^ lfsr_q[15]
^ data_in[0] ^ data_in[1] ^ data_in[2] ^ data_in[3] ^ data_in[4] ^ data_in[5] ^ data_in[6] ^ data_in[7];


      end // always

      always @(posedge clk, posedge rst) begin
            if(rst) begin
                  lfsr_q  <= {16{1'b1}};
            end
            else begin
                  lfsr_q  <= crc_en ? lfsr_c : lfsr_q;
            end
      end // always
endmodule // crc
```

About the Author

Evgeni Stavinov is the creator and main developer of OutputLogic.com. Evgeni has more than 10 years of diverse design experience in the areas of FPGA logic design, embedded software and communication protocols. He holds MSEE from University of Southern California and BSEE from Technion – Israel Institute of Technology. For more information contact evgeni@outputlogic.com

About OutputLogic.com

OutputLogic.com is a web portal that offers online tools for FPGA and ASIC designers.

The MIT License