University of Pisa

MSc in Computer Engineering

Electronics and Communication Systems

Design and implementation of a CRC algorithm on a Xilinx Zynq Board

Supervisors
Prof. Luca Fanucci
Prof. Massimiliano Donati
Ing. Pietro Nannipieri

Student
Marco Pinna

April 10, 2022

Contents

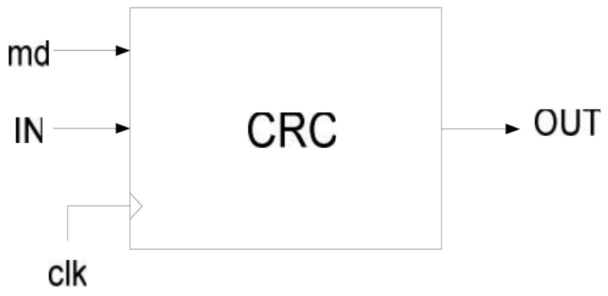
Specifications

In what follows, the design and implementation of a CRC algorithm on a Xilinx Zynq Board is carried out.
The specifications are the following (translated from Italian):

Implementation of a CRC algorithm

Design a digital circuit that implements the Cyclic Redundancy Check (CRC) algorithm, both for the *sender* and for the *receiver*, according to the specifications below. Such algorithm is a powerful control method that exploits the idea of redundancy; a sequence of F redundant bits (Frame Control Sequence FCS)is added (by the sender) to a data sequence M, so that the transmitted message, on M+F bit, is divisible by a predefined divisor called CRC polynomial. The receiver, through a division by the same polynomial used by the sender, can detect the correctness of the received data.

Message M=56 bits
Frame Control Sequence F=8 bits
CRC polynomial of degree 9: $x^8 + x^4 + x^3 + x^2 + 1$
(The binary representation of the polynomial will therefore be 100011101)
Transmitted message M+F=64 bit



Through the `md` signal one can set whether the circuit is to be used as sender or as receiver.
When operating as receiver the last F values of OUT indicate the correctness of the received message.

The final project report has to include:

- Introduction (description of the algorithm, possible applications, possible architectures, etc.)
- Description of the architecture selected for the realization (block diagram, inputs/outputs, etc.)
- VHDL code (with detailed comments)
- Test-plan and relative testbench for verification
- Results of the automated logic synthesis on Xilinx FPGA Zync platform: resources used (slice, LUT, etc.), maximum operating frequency, critical path, etc. commenting potential warning messages
- Conclusions

Chapter 1

Introduction

The work is organized as follows:

- In
- In
- In

1.1 Algorithm description

A Cyclic Redundancy Check (CRC) algorithm is a technique used in digital networks that uses redundant bits to detect accidental changes in digital data.
Let us supposed to have a *sender* (S) and a *receiver* (R) at the two end of a communication channel.
To each message M of `m` bits being sent by S, an additional section (the Frame Control Sequence, or FCS) of `f` redundant bits is added, whose value is computed performing a polynomial division between the message M (the dividend) and a polynomial G (the divisor) called *generator* and calculating the remainder of such division.
Upon reception, R performs the same division and, depending on the value of the remainder, it is able to check whether the message M has been corrupted during transmission.

More in detail:
- let M be an `m` bits long binary string.
An `m-1` degree polynomial $M(x)$ is associated to it, such that the i-th coefficient of the polynomial is equal to the i-th bit of the string (e.g. 100101111 $\Rightarrow x^8+x^5+x^3+x^2+x+1$).

- Let $G(x)$ be the generator polynomial whose binary representation is `f + 1` bits long (the degree of G(x) will therefore be `f`).
M is shifted to the left by `f` positions, padding to the right with `f` zeros. This corresponds to multiplying $M(x)$ by x^f.

- The FCS is built as follows:
  a polynomial *long division* between $x^f \cdot M(x)$ and $G(x)$ is performed, using finite field arithmetic on the Galois Field GF(2). Let $Q(x)$ and $R(X)$ be the quotient and the remainder of such division, respectively.
  It follows that

$$x^f \cdot M(x) = Q(x) \cdot G(x) + R(x) \tag{1.1}$$

  If we subtract $R(x)$ from both sides of the equation, we get

$$x^f \cdot M(x) - R(x) = Q(x) \cdot G(x) \tag{1.2}$$

- The polynomial on the left-hand side of the equation is divisible by $G(x)$ and contains the original message M in the `m` highest bits and the FCS in the `f` lower bits (by definition, the degree of $R(X)$ is strictly less than `f`, so it can be represented on `f` bits).

- This `m+f` bits long string M|FCS is what will be sent to the receiver.

- The receiver can check the integrity of the message by dividing M|FCS by $G(x)$ and checking whether the remainder is equal to 0: if it is, then the message M has likely not been corrupted during transmission, otherwise it has and has to be discarded.

In this particular implementation, M will be 56 bits long, the generator will be

$$x^8 + x^4 + x^3 + x^2 + 1 \quad \leftrightarrow \quad 100011101 \tag{1.3}$$

therefore the FCS will be 8 bits long, for a total of 64 bits to be sent for each message.

Example
Let us use as an example the transmission of the word "hi" encoded in ASCII.
The binary string corresponding to "hi" is:

$$01101000\ 01101001 \tag{1.4}$$

We first pad it with zeros

$$01101000\ 01101001\ 00000000 \tag{1.5}$$

then we compute the FCS by performing the long division between 1.5 and 1.3.

```
0110100001101001000000000
 100011101
 0101111001
  100011101
  00110010001
    100011101
    0100011000
     100011101
     000000101010000
           100011101
           00100110100
             100011101
             00010100100
```

The remainder R is 10100100. R is subtracted from (1.5) to create the FCS in the 8 lower bits, yielding the following string

$$01101000\ 01101001\ 01011011 \tag{1.6}$$

1.2 Possible applications

CRC algorithms are used extensively in various data transmission technologies and protocols: SATA, USB, Ethernet, CDMA, Bluetooth, etc.
They are also found in several standards, programs and file formats such as MPEG-2, PNG, Gzip, to name a few.
There are different implementations of CRC algorithms, each with its own polynomial. The choice of the polynomial (both its degree and its coefficients) depend on several factors such as the maximum desired overhead and the sensitivity to different errors.

An important remark to be made is that such algorithms protect against *accidental* corruption of data, but they are not suited against *intentional* alteration of data, since a valid CRC to attach to the tampered message can easily be computed once the algorithm is known.

1.3 Possible architectures

Two different architectures were identified to implement the algorithm:

- the first one simply implements the long division shown in 1.1;

- the second one exploits the fact that, during the division, the divisor (i.e. the generator) is fixed. It is therefore possible to precompute the division for each possible byte and store the results in a LUT. This solution is likely to be more efficient in terms of speed, at the expense of a greater use of resources since a 258x1B LUT has to be created.

Chapter 2

Overview

Chapter 3

Appendices