Gbps CRC generator in 0.35um CMOS technology implemented with standard cells

José María Nadal Serrano
Lunds Tekniska Högskola - Lund Institute of Technology

Version 0.6
June 11, 2002

A mi familia y amigos,
que están cerca a pesar de
que son muchos los kilómetros
que nos separan.

Merci beaucoup à R.J. Glaise,
qui m'a aidé beaucoup avec les
matrices et m'a appris l'esprit
des ingenieurs qui aiment ce
qu'ils faisent. Merci beaucoup
pour votre temps!!

To my advisors for the project,
Prof. Peter Nilsson and Thomas Olsson
for their support and help during
the time I spent doing this project.

I would also like to acknowledge
the great help received by
Prof. Mats Cederwall, who gave me
some insight into CRCs.
Tack allihop!!

Table of contents

1 Introduction

Nowadays the world of digital electronics is changing rapidly. Moreover, the latest tendencies point in a direction where the more and more complex circuits are no longer being designed at the transistor level, but instead some CAD tools and more high-level tools help the designer in order to facilitate the response to customer's necesities and also reduce the so-called "time to market".

Besides this, a general tendency is to integrate several modules in a single piece of silicon, thus making huge chips that can be even made of pieces in which different technologies were used.

These new SoC, or "Systems on Chip" need on the other hand to make the designer have to think in a higher level of abstraction, and use now blocks with a more complex functionality. This is the only way of being able to deal with the ever-growing complexity.

In this context is where the so-called IPs appear. They have nothing to do with the well known internet protocol, but with patents. Those IPs (IP stands for "Intellectual Properties") are one step forward in the abstraction process. We were already used to the propietary cells that could keep our hands clean and far away from the full custom design. Those cells were mainly provided by the manufacturers in such a way that we could keep our design almost manufacturer-independent, and at the same time the manufacturer could hide the details of his technology. Now the concept is more or less the same one, but at a higher level of abstraction. Somebody could sell us a more o less complex block that performed a specific algorithm optimized for speed, area, comsumption... We will keep on building difficult and bigger and bigger circuits, but we will be using complete "rooms" to build them, instead of just using small "bricks".

It is in this context where the CRC-generator makes sense. Probably none would like to have a chip with such a circuit, since it is just worth if it is going to be used as a part of a bigger circuit, as an ATM transmiter or as a OC-XXX speed link.

2 General description: the problem

2.1 A word on Cyclic Redundancy Codes

As it is easily imaginable, we all would like to have no errors when transmiting data over the Internet or any other media, but deffects in materials, interferences and other sources can corrupt the data, so we have to provide the communication with ways of detecting and if possible correcting those errors.

It could be highly desirable to be able to check this in an accurate and fast way, and therefore many different strategies have been developed in the early years, but almost all were based in performing some kind of mathematical operation over the bits we are transmitting.

In this way, the so-called CRC was rapidly widespread as a good way of checking errors. A mathematical operation is performed over the data that is going to be sent, and the result is appended at the end of the data, thus resulting the packet that will actually be sent. At the far end of the channel, the same operation is made. If the result is the one expected (all 1's in our case) then there have not been problems. Else, the data is corrupted and something will have to be made (this "something" is out of the scope of the CRC generator; it is usually an issue corresponding to the upper layers of the communication protocol).

2.2 Special constraints

Our task consists in doing a CRC generator that could handle really high speeds and this is the reason why we will have to implement it in hardware instead of in software, as most of the CRC generators are done.

The final goal is to be able to achieve a 10Gbps data throughput. The standard polinomial for Ethernet must be used to generate the 32 bit CRC, and the throughput is supposed to be achieved in a $0.18\mu m$. However, we will have to deal with many problems derived from the fact that we will have to use a $0.35\mu m$ technology. Other issues as the way we will measure the performance of the chip will be covered in later sections.

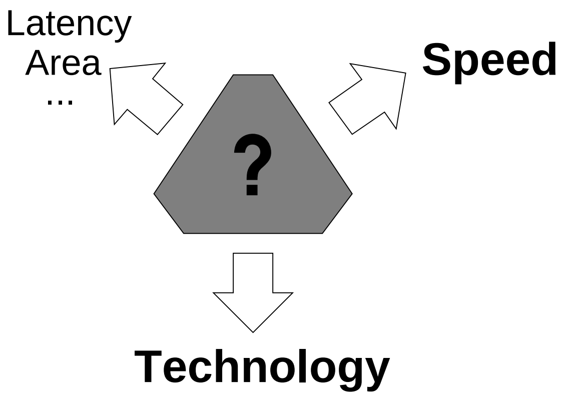

Figure 1: Our problem

3 The paper and pencil design

3.1 The algorithm

There are several ways of calculating a CRC[1] out of a block of data. We will name the most important ones just before explaining in detail the one we will use.

A straightforward implementation of a CRC generator could be as simple as a shift register in which the incoming bits are shifted in and XOR-ed with the polynomial that generates the code (in our case, the Ethernet polynomial). However, this implementation is rapidly discarded. Why? It is easy to think about the problems of making a GHz shift register in $0.35\mu m$ CMOS process. No way to implement this in a reasonable way. First conclusion then, we should try to look for something that could take bytes or words instead of bits. This way we will paralelize the architecture and therefore we will be able to reduce significantly the clock speed.

Other approach is to use a table-driven implementation. It consists in indexing a table with the result (or the so to speak WIP -"work in progress"- in the input register) and XOR-ing it with the result of the search. There are several versions of this approach, but they all surpass by far what is reasonable in on-chip memories. Even if the area is not our main problem, one of the constraints was not to increase the area beyond a "reasonable amount". Option discarded again.

Although this could have been quite discouraging, there is another approach, which is much closer to what we need and can be found in several papers[2]. This description is based in what they call the "Galois Fields" and reduce the whole problem to some logic. This is specially useful for us, since we will just have to try to modify the architecture in order to speed it up. And besides that, isn't that marvellous to be able to reduce the calculation of a reminder to some cascaded XORs? It seems great to me!. I finally decided that so far this was the best solution posible.

3.2 In-depth look at the algorithm

3.2.1 Galois and friends

It is strongly recommended for those who really want to understand how the circuit works to have a look at the paper by Jacquart and Glaise, and for further details, to have a look at the book by Peterson and Weldon[3].

The polynomial used in Ethernet and ATM is the same as the one we will use in our generator. It is the next one:

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

This polynomial has several advantages, such as the fact that it is irreducible, that is, it is like a "prime" polynomial. Besides that, if we recall the main property of a Galois Field:

$$\alpha^i \otimes \alpha^j = \alpha^{i+j mod(2^{32}-1)}$$

we will have that, provided our polynomial is an element of the Galois Field, the shift[4] we will have to do will then just be a multiplication by α^n, where n is the length of the word we are using (it has to be smaller than 32 bits, or α^{31}). For convenience, we chose a $n = 16$, which allows us to lowen the speed of the clock to about $10GHz/16 = 625MHz$ in a first approach (we will see that this is not that true -we will not be able to process one word per clock cycle-,

[1]A good introduction on CRCs can be found at http://www.repairfaq.org/filipg/LINK/F_crc_v31.html

[2]It is specially recomended to have a look at this paper: Fast CRC calculation, *R. J. Glaise and X. Jacquart*, IBM CER La Gaude (France) dep. TNN Hardware development

[3]"Error correcting codes", *Peterson and Weldon*, the MIT Press, 2nd edition 1972

[4]What we are doing here is to change the straight forward implementation into something more useful in our "hardware world". The description for this can be found in many places. If the reader needs to recall this, a good reference to start with is the one available in the Internet.

but I didn’t realize of this “bug” in these early stages of the design). Anyway this is a good register lenght, since it provides us with a good clock speed reduction and it does not increase the area too much either.

Taking $n = 16$ then, we have that:

$$CRC(T+1) = CRC(T) \otimes \alpha^8 \oplus InputWord(T+1)$$

In practice, the operation above corresponds to this next scheme:

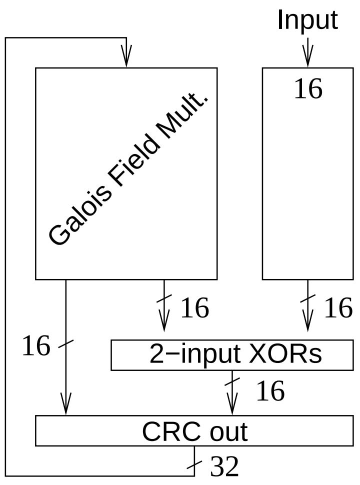

Figure 2: Straight forward schematic

Once we have the general idea of what we are going to do, the next step is to find the so-called *“H-matrix”*, which is the core of the Galois Field multiplier itself. That matrix is the one that makes the multiplication by α^{16}. This matrix is not very difficult to generate (if one knows how to do it) but it is very tedious. The matrix to compute 16 bits at a time is included here:

```
   31                             0
1...1.11..1.....1...............  31
 .1...1.11..1.....1..............  |
 ..1...1.11..1.....1.............  |
 ...1...1.11..1.....1............  |
 ....1...1.11..1.....1...........  |
 .....1...1.11..1.....1..........  |
 1...1..1....11........1.........  |
 .1...1..1....11........1........  |
 1.1...1..1....11........1.......  |
 .1.11.1........1.........1......  |
 ..1..11...1...............1.....  |
 ...1..11...1...............1....  |
 1...1..11...1...............1...  |
 11...1..11...1...............1..  |
 .11...1..11...1...............1.  |
 ..11...1..11...1...............1 16
 1..1..111.111...................  |
 11..1..111.111..................  |
 .11..1..111.111.................  |
 1.11..1..111.111................  |
11.1..1....11.11................  |
.11...1...1.11.1................  |
..111.1...11.11.................  |
...111.1...11.11................  |
1....1.11.1.11.1................  |
.1..1..11111.11.................  |
..1..1..11111.11................  |
1..11..1.1.111.1................  |
11...1111...111.................  |
.11...1111...111................  |
..111.1.11....11................  |
...1.11..1.....1................  0
```

It is now straightforward to convert this matrix into the logic that implements it. As the additions are XORs in Galois Field arithmetic, what we have to do is to convert the $'1'$ into XORs. This way, the $CRC_{(T+1)}[31] = CRC_{(T)}[31] \oplus CRC_{(T)}[27] \oplus CRC_{(T)}[25] \oplus CRC_{(T)}[24] \oplus CRC_{(T)}[21] \oplus CRC_{(T)}[15]$. We can easily expand the whole matrix into this new form and then implement the hardware directly. If we then have a look at what we have, we will see that we have a 10-XOR delay (maximum) in the Galois Field multiplier. This is quite a lot, but it is much smaller that in the case of computing 32 bits at a time. The price we will have to pay for the simplicity will probably be the latency as we will see.

The last consideration that must be made is that in order to be compliant with the Ethernet standard, we have to set the initial CRC to a well defined value ($H'46AF6449$). This element is the one that once multiplied by α^{31} yields the all one pattern.

3.2.2 Hardware constraints

Once we have gone through the first "hardware extraction" of the circuit we have to think about what we really have and how demanding our problem is, in order to be able to adapt this first implementation so that it meets all the requirements. And it is therefore here where the technology dependent decisions come, since the delays in the gates and the wires condition the whole design. We have to keep in mind that we have to make our design run very, very fast.

As we said before, if we have a look at the matrix we will see that the longest delay is 10 XORs. It is not possible to have such a big critical path if we want the circuit to run at several hundred megahertz. Therefore partitions must be made. And this means introducing pipeling and two phase logic in our circuit as well as some other problem-generators.

A good advantage of this architecture is the fact that the multiplier is *completely* "pipelineable", reducing in this way the critical path to one gate plus wires. This is the preferable architecture for our $0.35\mu m$ process. Why? Because if we take the data provided by the manufacturer and we take into account the load seen by the gate (just the input of one flip-flop, in this highly-pipelined process) we have the possibility to have a rather high clock frequency (in theory, up to 1GHz). If we used two gates as the maximum critical path, we wouldn't be able to handle the incoming data.

However, in a more modern process this last case (several gates per pipelining stage) could me more desirable, because the delay due to the gates and the wires is potentially smaller. It is possible that the reader thinks that this all is a little bit bizarre...

Why are we taking such low-level considerations into account? Well, let me explain that. We will describe the chip with VHDL, which in theory is used to get the designer rid of all those considerations. However, as this application is speed-critical, I thought that the best thing was to describe the hardware thinking in the final layout, so to speak, a full-custom, VHDL described *(=with standard cells)* chip. Even if this does not sound good, this is exactly what we will do.

We also have to ensure that the input arrives at the "big XOR" (the one we have just before the final register) at the same time as the processed -multiplied- data. This basically means that we will also have to include some pipelining at the input. The registers in this part will just delay the moment in which the signal arrives at the big XOR, and it will also sinchronize it with the data coming from the multiplier. The general sketch of the circuit can be seen in figure 3.

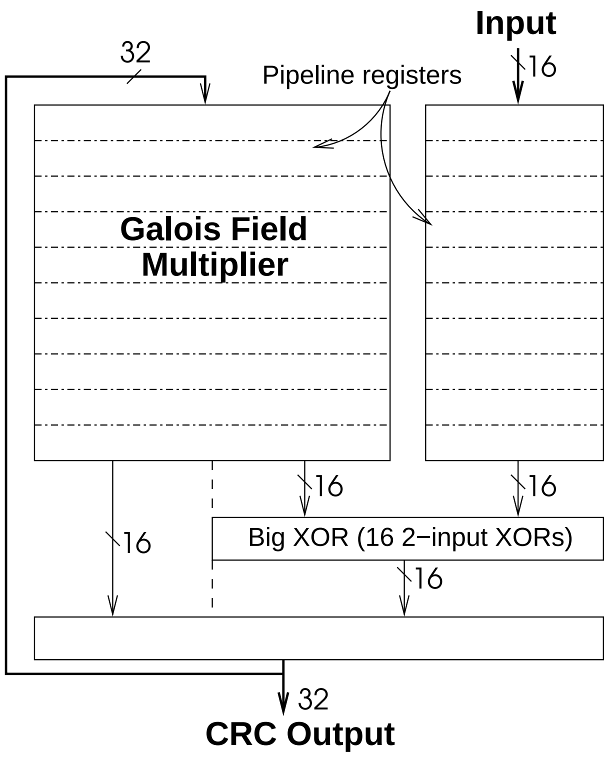

Figure 3: General scheme of the CRC generator

And when it comes to the multiplier, we will not be able to just include the registers to pipeline the outputs of the XORs (this is what I call the “WIP registers”, where WIP stands for “Work In Progress”), but also some others that will have to carry the original data (we need it in order to feed the inputs of the gates properly). We will have increased the size of some of the registers to 64 bits, but the increase in area is still affordable. The structure can be seen in figure 4.

The rest of the generator (the big XOR and the final register) does not have big problems when it comes to its design, but the parts also pipelined in order to keep the one-gate critical path (see figure 5). Special care has to be taken in order to respect the timing, for the output and for the feedback. And it is here where our problems start again.

The feedback and the pipelining. This is the most difficult part of the whole design, and since -I will have to admit that- I did not notice that there could be some problems because of this, the “errors” (looses in performance, actually) due to the feedback were the last ones to be discovered.

These errors (better said “features”) make the circuit have one fifth the throughput it should have, this is, $2Gbps$ measured at the input or $4Gbps$ at the output (we generate 32bits for each word of 16 bits). Sad but true.

Why does this happen?[5] Well, if we think a little bit, we will see that we just achieve a correct multiplication every five samples (it takes four clock cycles to compute a multiplication, and we get the result in the fifth one). The rest of the clock cycles are necessary, but provide us with no valid data. What to do, then? Well, the question arises much faster than the answer. It could be posible to increase the two phase logic to a multiphase logic, for instance, four phase logic, with the same clock frequency. This solution, besides the difficulties to generate the clock signal and the problems derived of the skew in the circuit (we depend on the automatic

[5]The interested reader is encouraged to have a look at the timing in the “Implementation” section in order to understand the details of the explanation.

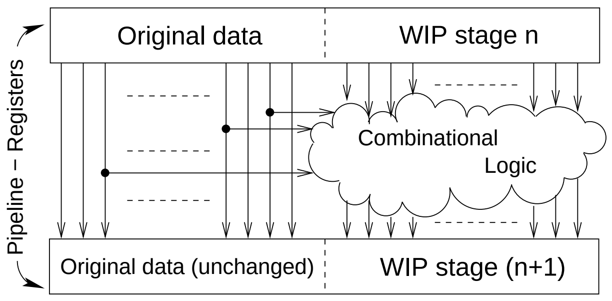

Figure 4: Structural scheme of the registers in the Galois Field multiplier

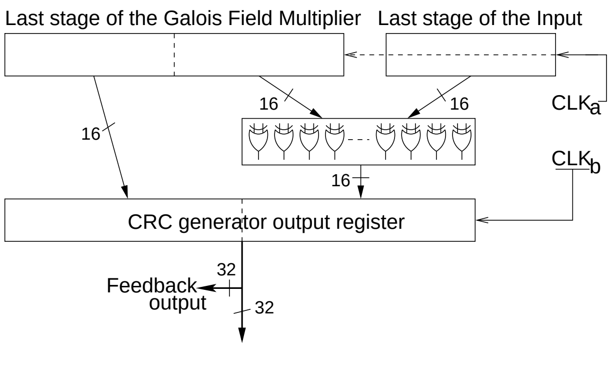
Last stage of the Galois Field Multiplier
Last stage of the Input
16
16
16
16
CLKa
CLKb
CRC generator output register
32
Feedback output
32

Figure 5: Output stage

place and route, so we cannot guarantee an H-shaped clock lines plus the fact of having really high clock frequencies, which turns out in smaller skew margins) does not solve our problem. More solutions... maybe try to increase the clock signal so that we have 625Mhz per word (data) and therefore a clock of 3.125 GHz. This is not possible nowadays. If so, just in the $0.18\mu m$ version, but not in this technology (we cannot do fine-grain pipelining and break the gates, but even in that case we would not be improving anything).

The last chance is to try to use the algebraic properties of the multiplier and all the operations we are using. In that case, the aim should be to paralelize the whole architecture, including minimal changes in the rest of the modules, and thus achieving the high throughput required. This means increasing the area of the chip at least five times in order to have one valid output at a time, and then making some operation to get the valid output *at any time*.

Due to lack of time (and why not to say it, deep knowledge on the topic) I could not come up with an idea that could solve these problems, because all the attempts where useless.

Then, the CRC generator design will be able to run (at most) at $4Gbps$ (output), faster if we used a better technology. The input will therefore consist of one word of data each five clock cycles, in the way that is shown in figure 6:

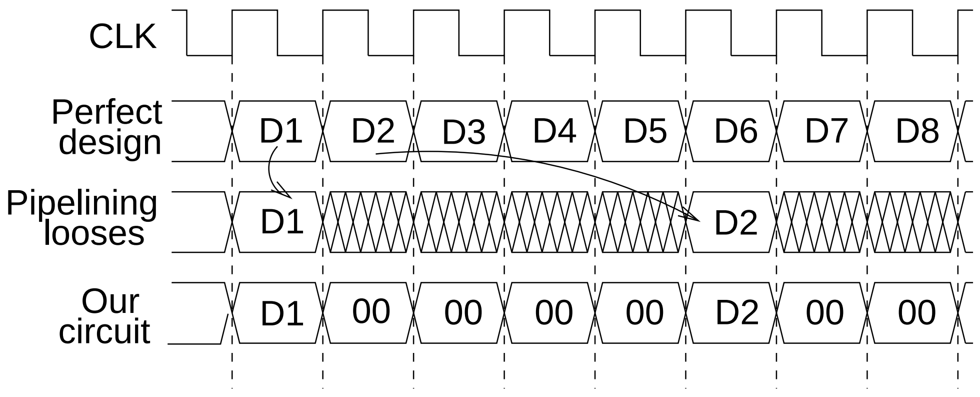

Figure 6: Input data flow

The idle clock cycles will contain zeros, but could contain anything else. The output should be also synchronized to get the parallel output each five clock cycles.

4 Implementation: down to the problem

4.1 The design and its structure

We will describe the CRC generator now from an architecture perspective, from the high level, almost-functional architecture, to the low level architecture (mainly timing and connections).

If we have a look at figure 7 we will see the architecture that should be familiar by now. Four main blocks are part of the CRC generator, the Galois Field multiplier, which is the biggest one and has the heaviest functional load, the input block, with the pipelining stages we mentioned, and then the two small blocks, one consisting of 16 XORs and the other one, the output stage, which consists of a register. These two are put together in the “big_xor” block. The bus widths are also shown.

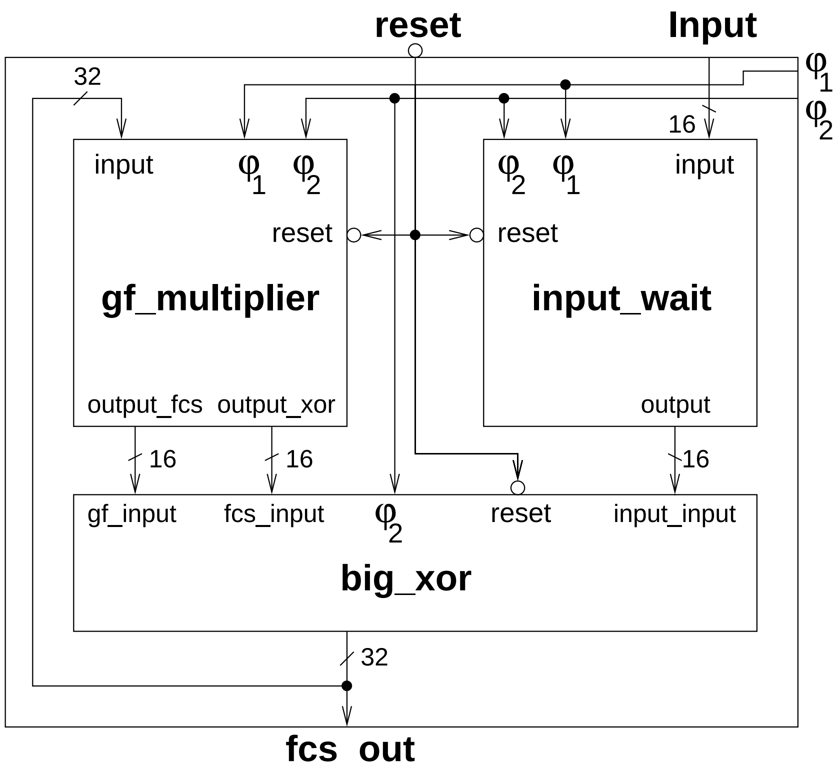

Figure 7: Block structure of the generator

4.1.1 Two phase logic

We will use the so-called strict two-phase design criterion. We will therefore use a square wave clock with two non-ovelapping phases (see figure 8).

Thus, the general scheme for both φ_1 and φ_2 is shown in figurẽreffigure:impl-phi12.

Then, to have a balanced tree (so that the signals get to the big XOR at the same time) we have to design the timing for the whole chip as shown in the figure 10. This design allows us to keep the power consumption controlled, because the number of the switchings remains somewhat controlled (we have separated the switching in two different times).

It is also interesting to have a look at the pipelining registers, in the input side and in the multiplier side. They are all represented in the figure where the general scheme for the timing is shown.

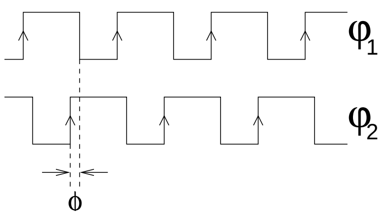

Figure 8: Two non-overlapping clock phases

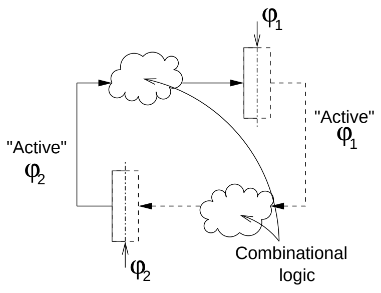

Figure 9: General two phase design scheme

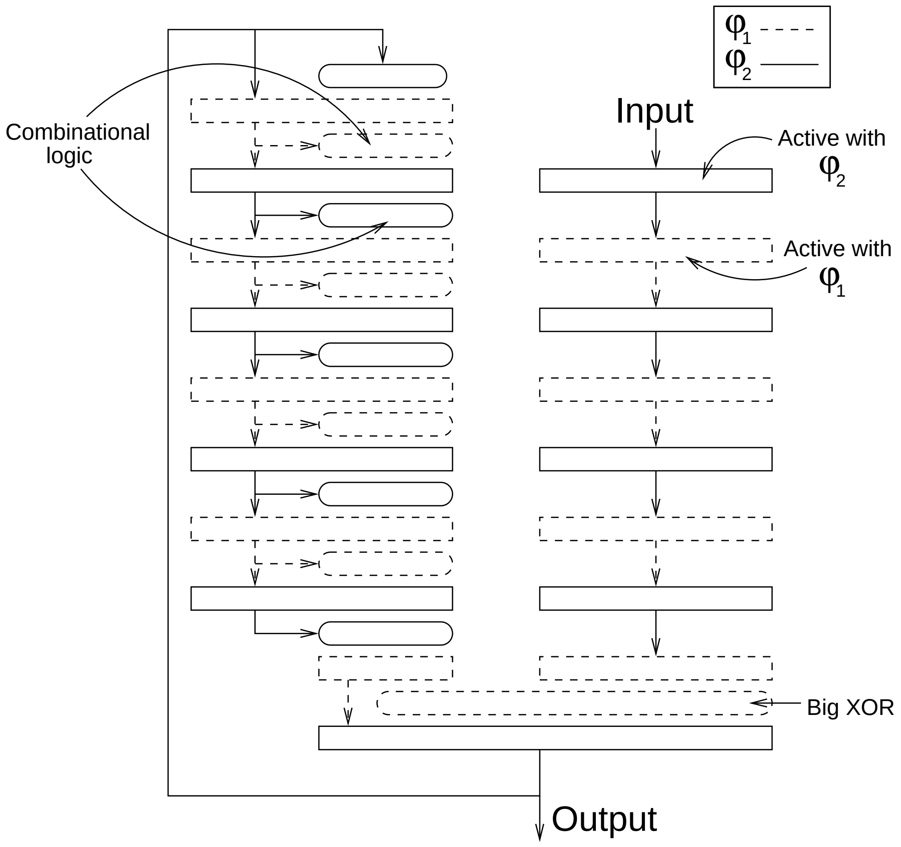

Figure 10: Detailed timing scheme of the whole design

Once the algorithm in the previous sections has been understood, there is not much to explain besides the timing.

4.2 Coding "policy"

When reading the coding, it is very important to say that it has not been writing as an example of the best coding possible (particulary the use of parameters could make it smaller and smarter), but just bearing in mind that the final aim is to describe hardware with a high level tool but thinking at the same time in the final layout, much more than one usually does.

And this all causes a fairly weird coding style, with very small blocks (in functionality and also in area), as well as an important amount of signals. Big care has been put so that the names of the signals are self-explaining. However, the figures showing how the connections are made should help to clarify any doubt.

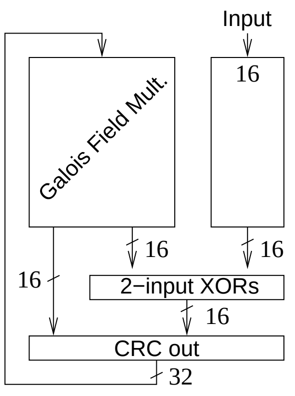

Figure 11: Structural block model showing connections and interfaces

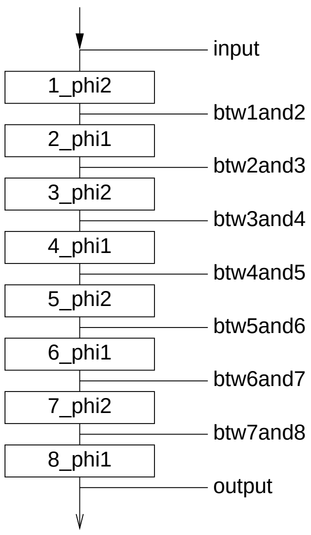

Figure 12: Detailed block diagram of the input block

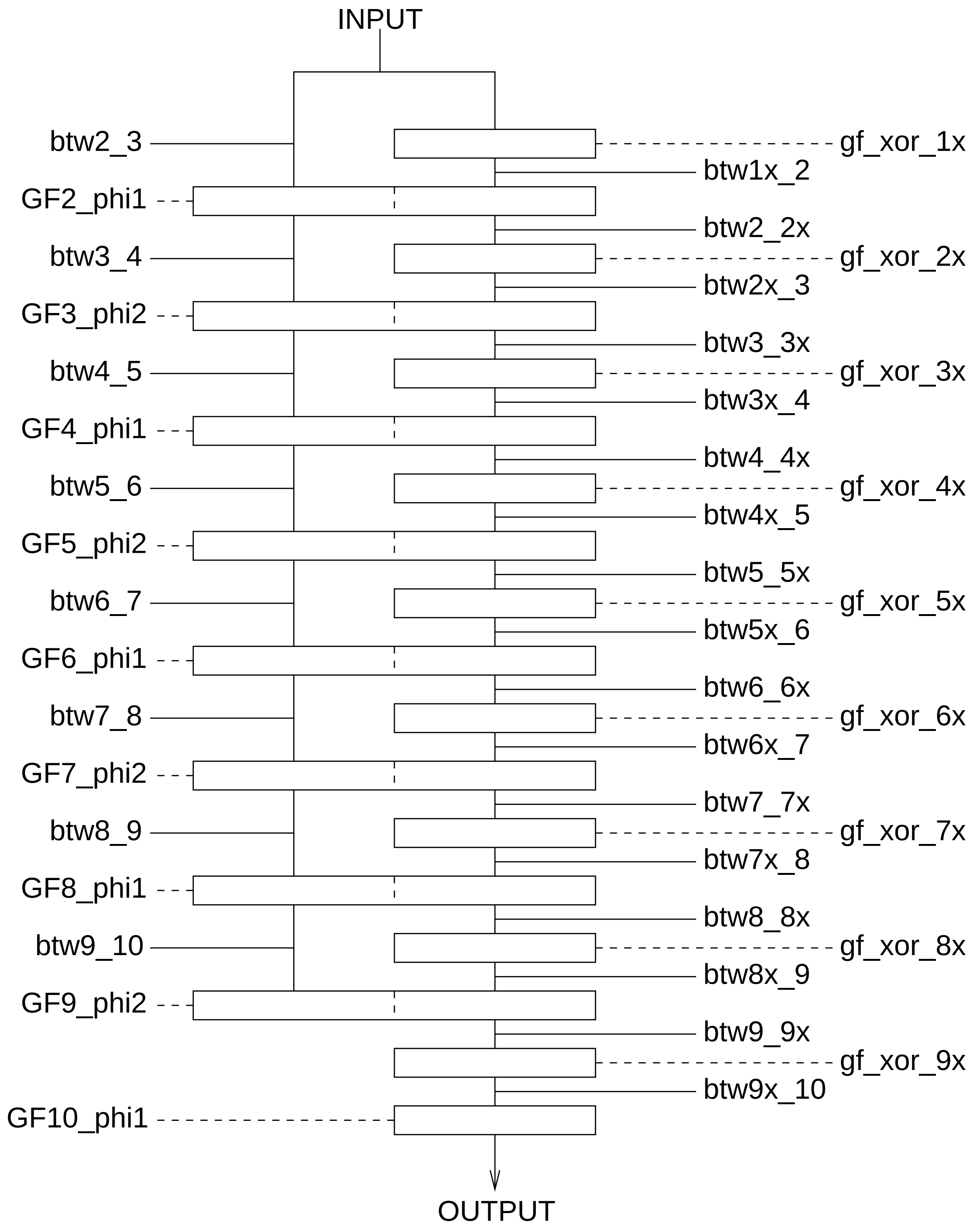

Figure 13: Detailed block diagram of the Galois Field multiplier

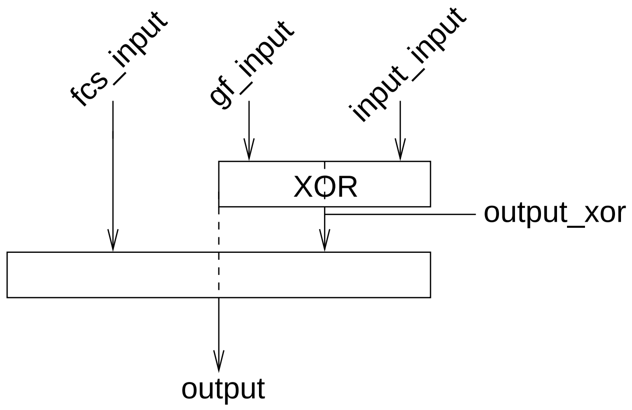

Figure 14: Detailed block diagram of the output block

5 Measurements and testing

Every design shall provide a test vector to be able to test if the functionality is the expected one and to be able to see if the fabrication process was optimum.

In our case, there is a test vector that should give an all-1's output. This is due to the fact that this vector was generated with a CRC generator compliant with the Ethernet standard. Then, what we will actually be doing is to make our CRC generator act as a receiver, instead of as a transmiter when there has not been any corruption in the channel. The output that we will get will therefore be all-1's.

The input vector is:

```
 96-BIT TEST VECTOR:
MSB
0 0 0 0 1 1 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 0 1 1 1 1 0 0 1 1 0 1
1 0 1 0 1 1 0 0 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 0 1 0 0 0 1 1 0 1
                                                             LSB
```

This vector has proven to be a good test vector, since it tests the functionality of the chip, and at the same time, it gets a good node stuck-at error coverage.

It is also important to remark that even if the testing is esential in this design it could not be taken to the extreme. This means that there are no special devices to observe/control all the nodes, we just have the input and the output. This is dangerous, but in these kind of projects it is not affordable to do two circuits, one thinking in the testability and the next one thinking in speed, because it is difficult to include testing hardware without seriously affecting the capabilities of the circuit.

Some of the designs sent for fabrication were requested to have the fastest packaging of the available ones (QFP), and not to pack the rest.

Therefore, the use of special probes (low-inductance, low-capacitance) will be mandatory in the measuring phase. The use of analog PADs for the inputs increases the risk of failure due to static discharge, so additional protections will have to be used in order to be able to handle the die without breaking it.

As it can be seen, all precautions must be extremed during the test phase, since we are near the limits of this technology (if we are not trying to surpase them).

A VHDL code of the CRC generator

A.1 Files dependency tree

Different files and different levels in the hierarchy have been used in the description of the CRC generator for the sake of clearness.

The structure is graphically represented in figure 15

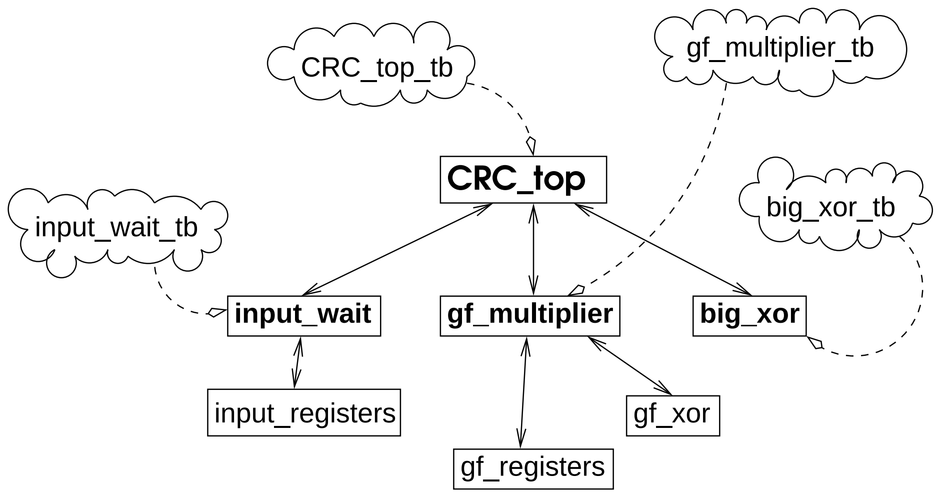

Figure 15: Hierarchy of the design and VHDL files

A.2 VHDL code

A.2.1 input_registers.vhd

```
--=============================================================================
-- File        : input_registers.vhd
-- Author      : José María Nadal
-- Date        : Mar, 2001
-- Description : The two types of registers used in the input block for the
--               pipelining.

-- Copyright (C) 2001  José María Nadal <chema@scouts-es.org>

-- This program is free software; you can redistribute it and/or
-- modify it under the terms of the GNU General Public License
-- as published by the Free Software Foundation; either version 2
-- of the License, or (at your option) any later version.

-- This program is distributed in the hope that it will be useful,
-- but WITHOUT ANY WARRANTY; without even the implied warranty of
-- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
-- GNU General Public License for more details.

-- You should have received a copy of the GNU General Public License
-- along with this program; if not, write to the Free Software
-- Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
--=============================================================================
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

-------------------------------------------------------------------------------
-- We first describe the entities of the two types of registers used in the
-- pipelining.

entity input_phi1_register is

  port (
    reset  : in  std_logic;
    phi1   : in  std_logic;
    input  : in  std_logic_vector(15 downto 0);
    output : out std_logic_vector(15 downto 0));

end input_phi1_register;

architecture behavior of input_phi1_register is

begin  -- behavior

  -- purpose: Pipelining register activated by phi1
  -- type   : sequential
  -- inputs : phi1, reset, input (16 bits)
  -- outputs: output (16 bits)
  p_input_phi1_register : process (phi1, reset)
  begin  -- process
    if reset = '0' then                 -- asynchronous reset (active low)
      output <= (others => '0');
    elsif phi1'event and phi1 = '1' then  -- rising clock edge
      output <= input;
    end if;
  end process;

end behavior;

-------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;


entity input_phi2_register is

  port (
    reset  : in  std_logic;
    phi2   : in  std_logic;
    input  : in  std_logic_vector(15 downto 0);
    output : out std_logic_vector(15 downto 0));

end input_phi2_register;

architecture behavior of input_phi2_register is
```

```
begin  -- behavior

  -- purpose: Pipelining register activated by phi2
  -- type   : sequential
  -- inputs : phi2, reset, input (16 bits)
  -- outputs: output (16 bits)
  p_input_phi2_register: process (phi2, reset)
  begin  -- process p_input_phi2_register
    if reset = '0' then                 -- asynchronous reset (active low)
      output <= (others => '0');
    elsif phi2'event and phi2 = '1' then  -- rising clock edge
      output <= input;
    end if;
  end process p_input_phi2_register;

end behavior;
```

A.2.2 input_wait.vhd

```
--=============================================================================
-- File        : input_wait.vhd
-- Author      : José María Nadal
-- Date        : Mar, 2001
-- Description : This file contains the code for the "wait states" of the in
--               put. It is also one of the three main blocks of the generator.

-- Copyright (C) 2001  José María Nadal <chema@scouts-es.org>

-- This program is free software; you can redistribute it and/or
-- modify it under the terms of the GNU General Public License
-- as published by the Free Software Foundation; either version 2
-- of the License, or (at your option) any later version.

-- This program is distributed in the hope that it will be useful,
-- but WITHOUT ANY WARRANTY; without even the implied warranty of
-- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
-- GNU General Public License for more details.

-- You should have received a copy of the GNU General Public License
-- along with this program; if not, write to the Free Software
-- Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
--=============================================================================

library IEEE;
use IEEE.std_logic_1164.all;

entity input_wait is port (
    phi1      : in  std_logic;           -- Two phase discipline
    phi2      : in  std_logic;
    reset     : in  std_logic;           -- #RESET
    input     : in  std_logic_vector(15 downto 0);
                                       -- The serial/parallel conversion has
                                       -- been made somewhere else
    output    : out std_logic_vector(15 downto 0));

end input_wait;
```

```
architecture structural of input_wait is


  component input_phi1_register
    port (
      reset  : in  std_logic;           -- #RESET
      phi1   : in  std_logic;           -- Clock
      input  : in  std_logic_vector(15 downto 0);
      output : out std_logic_vector(15 downto 0));
  end component;

  component input_phi2_register
    port (
      reset  : in  std_logic;           -- #RESET
      phi2   : in  std_logic;           -- Clock
      input  : in  std_logic_vector(15 downto 0);
      output : out std_logic_vector(15 downto 0));
  end component;

  signal btw1and2   : std_logic_vector(15 downto 0);
  signal btw2and3   : std_logic_vector(15 downto 0);
  signal btw3and4   : std_logic_vector(15 downto 0);
  signal btw4and5   : std_logic_vector(15 downto 0);
  signal btw5and6   : std_logic_vector(15 downto 0);
  signal btw6and7   : std_logic_vector(15 downto 0);
  signal btw7and8   : std_logic_vector(15 downto 0);

begin  -- structural

Input1:  input_phi2_register port map (reset  => reset, phi2   => phi2,
                                       input  => input, output => btw1and2);

Input2:  input_phi1_register port map (reset  => reset,    phi1   => phi1,
                                       input  => btw1and2, output => btw2and3);

Input3:  input_phi2_register port map (reset  => reset,    phi2   => phi2,
                                       input  => btw2and3, output => btw3and4);

Input4:  input_phi1_register port map (reset  => reset,    phi1   => phi1,
                                       input  => btw3and4, output => btw4and5);

Input5:  input_phi2_register port map (reset  => reset,    phi2   => phi2,
                                       input  => btw4and5, output => btw5and6);

Input6:  input_phi1_register port map (reset  => reset,    phi1   => phi1,
                                       input  => btw5and6, output => btw6and7);

Input7:  input_phi2_register port map (reset  => reset,    phi2   => phi2,
                                       input  => btw6and7, output => btw7and8);

Input8:  input_phi1_register port map (reset  => reset,    phi1   => phi1,
                                       input  => btw7and8, output => output);
end structural;


configuration cfg_input_wait_structural of input_wait is
```

```
  for structural
    for Input1 : input_phi2_register
      use entity work.input_phi2_register(behavior);
    end for;

    for Input2 : input_phi1_register
      use entity work.input_phi1_register(behavior);
    end for;

    for Input3 : input_phi2_register
      use entity work.input_phi2_register(behavior);
    end for;

    for Input4 : input_phi1_register
      use entity work.input_phi1_register(behavior);
    end for;

    for Input5 : input_phi2_register
      use entity work.input_phi2_register(behavior);
    end for;

    for Input6 : input_phi1_register
      use entity work.input_phi1_register(behavior);
    end for;

    for Input7 : input_phi2_register
      use entity work.input_phi2_register(behavior);
    end for;

    for Input8 : input_phi1_register
      use entity work.input_phi1_register(behavior);
    end for;
  end for;

end cfg_input_wait_structural;
```

A.2.3 gf_registers.vhd

```
--=========================================================================
-- File        : gf_registers.vhd
-- Author      : José María Nadal
-- Date        : Mar, 2001
-- Description : The different types of registers needed for the pipelining
--               are described here.

-- Copyright (C) 2001  José María Nadal <chema@scouts-es.org>

-- This program is free software; you can redistribute it and/or
-- modify it under the terms of the GNU General Public License
-- as published by the Free Software Foundation; either version 2
-- of the License, or (at your option) any later version.

-- This program is distributed in the hope that it will be useful,
-- but WITHOUT ANY WARRANTY; without even the implied warranty of
-- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
-- GNU General Public License for more details.
-- You should have received a copy of the GNU General Public License
-- along with this program; if not, write to the Free Software
-- Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
--=========================================================================

library IEEE;
use IEEE.std_logic_1164.all;
---------------------------------------------------------------------------
entity gf_phi1_register_out is

  port (
      reset        : in  std_logic;           -- #RESET
      phi1         : in  std_logic;           -- Clock
      input_wip    : in  std_logic_vector(31 downto 0);
      output_final : out std_logic_vector(31 downto 0));

end gf_phi1_register_out;


architecture behavior of gf_phi1_register_out is

begin  -- gf_phi1_register_out

  -- purpose: This is the final register in the GF multiplier.
  -- It is coded as a different entity since it is much smaller than the
  -- "standard" ones

  p_gf_phi1_register_out: process (phi1, reset)
  begin  -- process p_gf_phi1_register_out
    if reset = '0' then                 -- asynchronous reset (active low)
      output_final <= (others => '0');
    elsif phi1'event and phi1 = '1' then  -- rising clock edge
      output_final <= input_wip;
    end if;
  end process p_gf_phi1_register_out;

end behavior;

---------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;

entity gf_phi1_register is

  port (
      reset      : in  std_logic;                      -- #RESET
      phi1       : in  std_logic;                      -- Clock
      input_wip  : in  std_logic_vector(31 downto 0);  -- The incoming WIP
      input_fcs  : in  std_logic_vector(31 downto 0);
                   -- The original data for that step. As we are using
                   -- pipelining we have to grant that we will have the original
                   -- FCS data available.
      output_wip : out std_logic_vector(31 downto 0);
                   -- The modified data -our "WIP"-
      output_fcs : out std_logic_vector(31 downto 0));
                   -- The original data is kept untouched
```

```
end gf_phi1_register;


architecture behavior of gf_phi1_register is

begin  -- behavior

  -- purpose: 63 bit pipeline register
  -- type   : sequential
  -- inputs : phi1, reset, input_fcs, input_wip
  -- outputs: output_fcs, output_wip
  p_gf_phi1_register: process (phi1, reset)
  begin  -- process p_gf_phi1_register
    if reset = '0' then                -- asynchronous reset (active low)
      output_fcs <= (others => '0');
      output_wip <= (others => '0');
    elsif phi1'event and phi1 = '1' then  -- rising clock edge
      output_fcs <= input_fcs;
      output_wip <= input_wip;
    end if;
  end process p_gf_phi1_register;

end behavior;

-------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;

entity gf_phi2_register is

    port (
      reset      : in  std_logic;                      -- #RESET
      phi2       : in  std_logic;                      -- Clock
      input_wip  : in  std_logic_vector(31 downto 0);  -- The incoming WIP
      input_fcs  : in  std_logic_vector(31 downto 0);
                   -- The original data for that step. As we are using
                   -- pipelining we have to grant that we will have the original
                   -- FCS data available.
      output_wip : out std_logic_vector(31 downto 0);
                   -- The modified data -our "WIP"-
      output_fcs : out std_logic_vector(31 downto 0));
                   -- The original data is kept untouched

end gf_phi2_register;


architecture behavior of gf_phi2_register is

begin  -- behavior

  -- purpose: 63 bit pipeline register
  -- type   : sequential
  -- inputs : phi2, reset, input_fcs, input_wip
  -- outputs: output_fcs, output_wip
  p_gf_phi2_register: process (phi2, reset)
  begin  -- process p_gf_phi2_register
```

```
    if reset = '0' then                -- asynchronous reset (active low)
      output_fcs <= (others => '0');
      output_wip <= (others => '0');
    elsif phi2'event and phi2 = '1' then  -- rising clock edge
      output_fcs <= input_fcs;
      output_wip <= input_wip;
    end if;
  end process p_gf_phi2_register;

end behavior;
```

A.2.4 gf_xor.vhd

```
--===============================================================================
-- File        : gf_xor.vhd
-- Author      : José María Nadal
-- Date        : Mar, 2001
-- Description : Codification of the combination logic in each pipelining
--               stage

-- Copyright (C) 2001  José María Nadal <chema@scouts-es.org>

-- This program is free software; you can redistribute it and/or
-- modify it under the terms of the GNU General Public License
-- as published by the Free Software Foundation; either version 2
-- of the License, or (at your option) any later version.

-- This program is distributed in the hope that it will be useful,
-- but WITHOUT ANY WARRANTY; without even the implied warranty of
-- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
-- GNU General Public License for more details.

-- You should have received a copy of the GNU General Public License
-- along with this program; if not, write to the Free Software
-- Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
--===============================================================================


library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.all;

-------------------------------------------------------------------------------

entity gf_xor_input is

  port (
    input_fcs  : in  std_logic_vector(31 downto 0);
    output_wip : out std_logic_vector(31 downto 0));

end gf_xor_input;

architecture behavior of gf_xor_input is

begin  -- behavior
```

```
  -- purpose: GF1x
  -- type   : combinational
  -- inputs : input_fcs
  -- outputs: output_wip
  p_gf_xor_input: process (input_fcs)
  begin  -- process p_gf_xor_input
               output_wip(31) <= (input_fcs(31) xor input_fcs(27));
               output_wip(30) <= (input_fcs(30) xor input_fcs(26));
               output_wip(29) <= (input_fcs(29) xor input_fcs(25));
               output_wip(28) <= (input_fcs(28) xor input_fcs(24));
               output_wip(27) <= (input_fcs(27) xor input_fcs(23));
               output_wip(26) <= (input_fcs(26) xor input_fcs(22));
               output_wip(25) <= (input_fcs(31) xor input_fcs(27));
               output_wip(24) <= (input_fcs(30) xor input_fcs(26));
               output_wip(23) <= (input_fcs(31) xor input_fcs(29));
               output_wip(22) <= (input_fcs(30) xor input_fcs(28));
               output_wip(21) <= (input_fcs(29) xor input_fcs(26));
               output_wip(20) <= (input_fcs(28) xor input_fcs(25));
               output_wip(19) <= (input_fcs(31) xor input_fcs(27));
               output_wip(18) <= (input_fcs(31) xor input_fcs(30));
               output_wip(17) <= (input_fcs(30) xor input_fcs(29));
               output_wip(16) <= (input_fcs(29) xor input_fcs(28));
               output_wip(15) <= (input_fcs(31) xor input_fcs(28));
               output_wip(14) <= (input_fcs(31) xor input_fcs(30));
               output_wip(13) <= (input_fcs(30) xor input_fcs(29));
               output_wip(12) <= (input_fcs(31) xor input_fcs(29));
               output_wip(11) <= (input_fcs(31) xor input_fcs(30));
               output_wip(10) <= (input_fcs(30) xor input_fcs(29));
               output_wip(9) <= (input_fcs(29) xor input_fcs(28));
               output_wip(8) <= (input_fcs(28) xor input_fcs(27));
               output_wip(7) <= (input_fcs(31) xor input_fcs(26));
               output_wip(6) <= (input_fcs(30) xor input_fcs(27));
               output_wip(5) <= (input_fcs(29) xor input_fcs(26));
               output_wip(4) <= (input_fcs(31) xor input_fcs(28));
               output_wip(3) <= (input_fcs(31) xor input_fcs(30));
               output_wip(2) <= (input_fcs(30) xor input_fcs(29));
               output_wip(1) <= (input_fcs(29) xor input_fcs(28));
               output_wip(0) <= (input_fcs(28) xor input_fcs(26));
  end process p_gf_xor_input;

end behavior;

-------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.all;

entity gf_xor_2x is

   port (
      input_wip  : in  std_logic_vector(31 downto 0);
      input_fcs  : in  std_logic_vector(31 downto 0);
      output_wip : out  std_logic_vector(31 downto 0));
```

```
end gf_xor_2x;

architecture behavior of gf_xor_2x is

begin  -- behavior

  p_gf_xor_2x: process (input_fcs, input_wip)
  begin  -- process p_gf_xor_2x
               output_wip(31) <= (input_wip(31) xor input_fcs(25));
               output_wip(30) <= (input_wip(30) xor input_fcs(24));
               output_wip(29) <= (input_wip(29) xor input_fcs(23));
               output_wip(28) <= (input_wip(28) xor input_fcs(22));
               output_wip(27) <= (input_wip(27) xor input_fcs(21));
               output_wip(26) <= (input_wip(26) xor input_fcs(20));
               output_wip(25) <= (input_wip(25) xor input_fcs(24));
               output_wip(24) <= (input_wip(24) xor input_fcs(23));
               output_wip(23) <= (input_wip(23) xor input_fcs(25));
               output_wip(22) <= (input_wip(22) xor input_fcs(27));
               output_wip(21) <= (input_wip(21) xor input_fcs(25));
               output_wip(20) <= (input_wip(20) xor input_fcs(24));
               output_wip(19) <= (input_wip(19) xor input_fcs(24));
               output_wip(18) <= (input_wip(18) xor input_fcs(26));
               output_wip(17) <= (input_wip(17) xor input_fcs(25));
               output_wip(16) <= (input_wip(16) xor input_fcs(24));
               output_wip(15) <= (input_wip(15) xor input_fcs(25));
               output_wip(14) <= (input_wip(14) xor input_fcs(27));
               output_wip(13) <= (input_wip(13) xor input_fcs(26));
               output_wip(12) <= (input_wip(12) xor input_fcs(28));
               output_wip(11) <= (input_wip(11) xor input_fcs(28));
               output_wip(10) <= (input_wip(10) xor input_fcs(25));
               output_wip(9) <= (input_wip(9) xor input_fcs(27));
               output_wip(8) <= (input_wip(8) xor input_fcs(26));
               output_wip(7) <= (input_wip(7) xor input_fcs(24));
               output_wip(6) <= (input_wip(6) xor input_fcs(24));
               output_wip(5) <= (input_wip(5) xor input_fcs(23));
               output_wip(4) <= (input_wip(4) xor input_fcs(27));
               output_wip(3) <= (input_wip(3) xor input_fcs(26));
               output_wip(2) <= (input_wip(2) xor input_fcs(25));
               output_wip(1) <= (input_wip(1) xor input_fcs(27));
               output_wip(0) <= (input_wip(0) xor input_fcs(25));
  end process p_gf_xor_2x;

end behavior;

-------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.all;

entity gf_xor_3x is

   port (
      input_wip  : in  std_logic_vector(31 downto 0);
      input_fcs  : in  std_logic_vector(31 downto 0);
      output_wip : out  std_logic_vector(31 downto 0));

end gf_xor_3x;


architecture behavior of gf_xor_3x is

begin  -- behavior

p_gf_xor_3x: process (input_fcs, input_wip)
 begin  -- process p_gf_xor_3x
                 output_wip(31) <= (input_wip(31) xor input_fcs(24));
                 output_wip(30) <= (input_wip(30) xor input_fcs(23));
                 output_wip(29) <= (input_wip(29) xor input_fcs(22));
                 output_wip(28) <= (input_wip(28) xor input_fcs(21));
                 output_wip(27) <= (input_wip(27) xor input_fcs(20));
                 output_wip(26) <= (input_wip(26) xor input_fcs(19));
                 output_wip(25) <= (input_wip(25) xor input_fcs(19));
                 output_wip(24) <= (input_wip(24) xor input_fcs(18));
                 output_wip(23) <= (input_wip(23) xor input_fcs(22));
                 output_wip(22) <= (input_wip(22) xor input_fcs(25));
                 output_wip(21) <= (input_wip(21) xor input_fcs(21));
                 output_wip(20) <= (input_wip(20) xor input_fcs(20));
                 output_wip(19) <= (input_wip(19) xor input_fcs(23));
                 output_wip(18) <= (input_wip(18) xor input_fcs(23));
                 output_wip(17) <= (input_wip(17) xor input_fcs(22));
                 output_wip(16) <= (input_wip(16) xor input_fcs(21));
                 output_wip(15) <= (input_wip(15) xor input_fcs(24));
                 output_wip(14) <= (input_wip(14) xor input_fcs(24));
                 output_wip(13) <= (input_wip(13) xor input_fcs(23));
                 output_wip(12) <= (input_wip(12) xor input_fcs(25));
                 output_wip(11) <= (input_wip(11) xor input_fcs(25));
                 output_wip(10) <= (input_wip(10) xor input_fcs(21));
                 output_wip(9) <= (input_wip(9) xor input_fcs(25));
                 output_wip(8) <= (input_wip(8) xor input_fcs(24));
                 output_wip(7) <= (input_wip(7) xor input_fcs(23));
                 output_wip(6) <= (input_wip(6) xor input_fcs(23));
                 output_wip(5) <= (input_wip(5) xor input_fcs(22));
                 output_wip(4) <= (input_wip(4) xor input_fcs(24));
                 output_wip(3) <= (input_wip(3) xor input_fcs(25));
                 output_wip(2) <= (input_wip(2) xor input_fcs(24));
                 output_wip(1) <= (input_wip(1) xor input_fcs(25));
                 output_wip(0) <= (input_wip(0) xor input_fcs(22));
    end process p_gf_xor_3x;

 end behavior;

-------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.all;

entity gf_xor_4x is

   port (
      input_wip  : in  std_logic_vector(31 downto 0);
      input_fcs  : in  std_logic_vector(31 downto 0);
      output_wip : out  std_logic_vector(31 downto 0));

end gf_xor_4x;


architecture behavior of gf_xor_4x is

begin  -- behavior

  p_gf_xor_4x: process (input_fcs, input_wip)
  begin  -- process p_gf_xor_4x
                 output_wip (31) <= (input_wip(31) xor input_fcs(21));
                 output_wip (30) <= (input_wip(30) xor input_fcs(20));
                 output_wip (29) <= (input_wip(29) xor input_fcs(19));
                 output_wip (28) <= (input_wip(28) xor input_fcs(18));
                 output_wip (27) <= (input_wip(27) xor input_fcs(17));
                 output_wip (26) <= (input_wip(26) xor input_fcs(16));
                 output_wip (25) <= (input_wip(25) xor input_fcs(18));
                 output_wip (24) <= (input_wip(24) xor input_fcs(17));
                 output_wip (23) <= (input_wip(23) xor input_fcs(17));
                 output_wip (22) <= (input_wip(22) xor input_fcs(16));
                 output_wip (21) <= (input_wip(21) xor input_fcs(5));
                 output_wip (20) <= (input_wip(20) xor input_fcs(4));
                 output_wip (19) <= (input_wip(19) xor input_fcs(19));
                 output_wip (18) <= (input_wip(18) xor input_fcs(22));
                 output_wip (17) <= (input_wip(17) xor input_fcs(21));
                 output_wip (16) <= (input_wip(16) xor input_fcs(20));
                 output_wip (15) <= (input_wip(15) xor input_fcs(23));
                 output_wip (14) <= (input_wip(14) xor input_fcs(23));
                 output_wip (13) <= (input_wip(13) xor input_fcs(22));
                 output_wip (12) <= (input_wip(12) xor input_fcs(22));
                 output_wip (11) <= (input_wip(11) xor input_fcs(20));
                 output_wip (10) <= (input_wip(10) xor input_fcs(19));
                 output_wip (9) <= (input_wip(9) xor input_fcs(21));
                 output_wip (8) <= (input_wip(8) xor input_fcs(20));
                 output_wip (7) <= (input_wip(7) xor input_fcs(21));
                 output_wip (6) <= (input_wip(6) xor input_fcs(22));
                 output_wip (5) <= (input_wip(5) xor input_fcs(21));
                 output_wip (4) <= (input_wip(4) xor input_fcs(22));
                 output_wip (3) <= (input_wip(3) xor input_fcs(24));
                 output_wip (2) <= (input_wip(2) xor input_fcs(23));
                 output_wip (1) <= (input_wip(1) xor input_fcs(23));
                 output_wip (0) <= (input_wip(0) xor input_fcs(16));
  end process p_gf_xor_4x;

end behavior;

-------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.all;

entity gf_xor_5x is
```

```
   port (
      input_wip  : in  std_logic_vector(31 downto 0);
      input_fcs  : in  std_logic_vector(31 downto 0);
      output_wip : out  std_logic_vector(31 downto 0));

end gf_xor_5x;

architecture behavior of gf_xor_5x is

begin  -- behavior

  p_gf_xor_5x: process (input_fcs, input_wip)
  begin  -- process p_gf_xor_5x
                 output_wip (31) <= (input_wip(31) xor input_fcs(15));
                 output_wip (30) <= (input_wip(30) xor input_fcs(14));
                 output_wip (29) <= (input_wip(29) xor input_fcs(13));
                 output_wip (28) <= (input_wip(28) xor input_fcs(12));
                 output_wip (27) <= (input_wip(27) xor input_fcs(11));
                 output_wip (26) <= (input_wip(26) xor input_fcs(10));
                 output_wip (25) <= (input_wip(25) xor input_fcs(9));
                 output_wip (24) <= (input_wip(24) xor input_fcs(8));
                 output_wip (23) <= (input_wip(23) xor input_fcs(16));
                 output_wip (22) <= (input_wip(22) xor input_fcs(6));
                 output_wip (21) <= (input_wip(21));
                 output_wip (20) <= (input_wip(20));
                 output_wip (19) <= (input_wip(19) xor input_fcs(3));
                 output_wip (18) <= (input_wip(18) xor input_fcs(18));
                 output_wip (17) <= (input_wip(17) xor input_fcs(17));
                 output_wip (16) <= (input_wip(16) xor input_fcs(16));
                 output_wip (15) <= (input_wip(15) xor input_fcs(21));
                 output_wip (14) <= (input_wip(14) xor input_fcs(22));
                 output_wip (13) <= (input_wip(13) xor input_fcs(21));
                 output_wip (12) <= (input_wip(12) xor input_fcs(21));
                 output_wip (11) <= (input_wip(11) xor input_fcs(19));
                 output_wip (10) <= (input_wip(10) xor input_fcs(18));
                 output_wip (9) <= (input_wip(9) xor input_fcs(20));
                 output_wip (8) <= (input_wip(8) xor input_fcs(19));
                 output_wip (7) <= (input_wip(7) xor input_fcs(19));
                 output_wip (6) <= (input_wip(6) xor input_fcs(21));
                 output_wip (5) <= (input_wip(5) xor input_fcs(20));
                 output_wip (4) <= (input_wip(4) xor input_fcs(20));
                 output_wip (3) <= (input_wip(3) xor input_fcs(23));
                 output_wip (2) <= (input_wip(2) xor input_fcs(22));
                 output_wip (1) <= (input_wip(1) xor input_fcs(22));
                 output_wip (0) <= (input_wip(0));
  end process p_gf_xor_5x;

end behavior;

-------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.all;
```

```
entity gf_xor_6x is

   port (
      input_wip  : in  std_logic_vector(31 downto 0);
      input_fcs  : in  std_logic_vector(31 downto 0);
      output_wip : out  std_logic_vector(31 downto 0));

end gf_xor_6x;


architecture behavior of gf_xor_6x is

begin  -- behavior

  p_gf_xor_6x: process (input_fcs, input_wip)
  begin  -- process p_gf_xor_6x
                 output_wip (31) <= input_wip(31);
                 output_wip (30) <= input_wip(30);
                 output_wip (29) <= input_wip(29);
                 output_wip (28) <= input_wip(28);
                 output_wip (27) <= input_wip(27);
                 output_wip (26) <= input_wip(26);
                 output_wip (25) <= input_wip(25);
                 output_wip (24) <= input_wip(24);
                 output_wip (23) <= (input_wip(23) xor input_fcs(7));
                 output_wip (22) <= input_wip(22);
                 output_wip (21) <= input_wip(21);
                 output_wip (20) <= input_wip(20);
                 output_wip (19) <= input_wip(19);
                 output_wip (18) <= (input_wip(18) xor input_fcs(2));
                 output_wip (17) <= (input_wip(17) xor input_fcs(1));
                 output_wip (16) <= (input_wip(16) xor input_fcs(0));
                 output_wip (15) <= (input_wip(15) xor input_fcs(20));
                 output_wip (14) <= (input_wip(14) xor input_fcs(20));
                 output_wip (13) <= (input_wip(13) xor input_fcs(19));
                 output_wip (12) <= (input_wip(12) xor input_fcs(20));
                 output_wip (11) <= (input_wip(11) xor input_fcs(17));
                 output_wip (10) <= (input_wip(10) xor input_fcs(16));
                 output_wip (9) <= (input_wip(9) xor input_fcs(18));
                 output_wip (8) <= (input_wip(8) xor input_fcs(17));
                 output_wip (7) <= (input_wip(7) xor input_fcs(18));
                 output_wip (6) <= (input_wip(6) xor input_fcs(20));
                 output_wip (5) <= (input_wip(5) xor input_fcs(19));
                 output_wip (4) <= (input_wip(4) xor input_fcs(19));
                 output_wip (3) <= (input_wip(3) xor input_fcs(19));
                 output_wip (2) <= (input_wip(2) xor input_fcs(18));
                 output_wip (1) <= (input_wip(1) xor input_fcs(17));
                 output_wip (0) <= input_wip(0);
  end process p_gf_xor_6x;

end behavior;

-------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.all;

entity gf_xor_7x is

   port (
      input_wip  : in  std_logic_vector(31 downto 0);
      input_fcs  : in  std_logic_vector(31 downto 0);
      output_wip : out  std_logic_vector(31 downto 0));

end gf_xor_7x;

architecture behavior of gf_xor_7x is

begin  -- behavior

  p_gf_xor_7x: process (input_fcs, input_wip)
  begin  -- process p_gf_xor_7x
                 output_wip (31) <= input_wip(31);
                 output_wip (30) <= input_wip(30);
                 output_wip (29) <= input_wip(29);
                 output_wip (28) <= input_wip(28);
                 output_wip (27) <= input_wip(27);
                 output_wip (26) <= input_wip(26);
                 output_wip (25) <= input_wip(25);
                 output_wip (24) <= input_wip(24);
                 output_wip (23) <= input_wip(23);
                 output_wip (22) <= input_wip(22);
                 output_wip (21) <= input_wip(21);
                 output_wip (20) <= input_wip(20);
                 output_wip (19) <= input_wip(19);
                 output_wip (18) <= input_wip(18);
                 output_wip (17) <= input_wip(17);
                 output_wip (16) <= input_wip(16);
                 output_wip (15) <= (input_wip(15) xor input_fcs(19));
                 output_wip (14) <= (input_wip(14) xor input_fcs(19));
                 output_wip (13) <= (input_wip(13) xor input_fcs(18));
                 output_wip (12) <= (input_wip(12) xor input_fcs(18));
                 output_wip (11) <= (input_wip(11) xor input_fcs(16));
                 output_wip (10) <= input_wip(10);
                 output_wip (9) <= (input_wip(9) xor input_fcs(17));
                 output_wip (8) <= (input_wip(8) xor input_fcs(16));
                 output_wip (7) <= (input_wip(7) xor input_fcs(16));
                 output_wip (6) <= (input_wip(6) xor input_fcs(18));
                 output_wip (5) <= (input_wip(5) xor input_fcs(17));
                 output_wip (4) <= (input_wip(4) xor input_fcs(18));
                 output_wip (3) <= (input_wip(3) xor input_fcs(18));
                 output_wip (2) <= (input_wip(2) xor input_fcs(17));
                 output_wip (1) <= (input_wip(1) xor input_fcs(16));
                 output_wip (0) <= input_wip(0);
  end process p_gf_xor_7x;

end behavior;

-------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.all;

entity gf_xor_8x is

   port (
      input_wip  : in  std_logic_vector(31 downto 0);
      input_fcs  : in  std_logic_vector(31 downto 0);
      output_wip : out  std_logic_vector(31 downto 0));

end gf_xor_8x;

architecture behavior of gf_xor_8x is

begin  -- behavior

  p_gf_xor_8x: process (input_fcs, input_wip)
  begin  -- process p_gf_xor_8x
                 output_wip (31) <= input_wip(31);
                 output_wip (30) <= input_wip(30);
                 output_wip (29) <= input_wip(29);
                 output_wip (28) <= input_wip(28);
                 output_wip (27) <= input_wip(27);
                 output_wip (26) <= input_wip(26);
                 output_wip (25) <= input_wip(25);
                 output_wip (24) <= input_wip(24);
                 output_wip (23) <= input_wip(23);
                 output_wip (22) <= input_wip(22);
                 output_wip (21) <= input_wip(21);
                 output_wip (20) <= input_wip(20);
                 output_wip (19) <= input_wip(19);
                 output_wip (18) <= input_wip(18);
                 output_wip (17) <= input_wip(17);
                 output_wip (16) <= input_wip(16);
                 output_wip (15) <= input_wip(15);
                 output_wip (14) <= (input_wip(14) xor input_fcs(18));
                 output_wip (13) <= (input_wip(13) xor input_fcs(17));
                 output_wip (12) <= (input_wip(12) xor input_fcs(17));
                 output_wip (11) <= input_wip(11);
                 output_wip (10) <= input_wip(10);
                 output_wip (9) <= input_wip(9);
                 output_wip (8) <= input_wip(8);
                 output_wip (7) <= input_wip(7);
                 output_wip (6) <= (input_wip(6) xor input_fcs(17));
                 output_wip (5) <= (input_wip(5) xor input_fcs(16));
                 output_wip (4) <= (input_wip(4) xor input_fcs(16));
                 output_wip (3) <= (input_wip(3) xor input_fcs(17));
                 output_wip (2) <= (input_wip(2) xor input_fcs(16));
                 output_wip (1) <= input_wip(1);
                 output_wip (0) <= input_wip(0);
  end process p_gf_xor_8x;

end behavior;

-------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.all;

entity gf_xor_9x is

   port (
      input_wip  : in  std_logic_vector(31 downto 0);
      input_fcs  : in  std_logic_vector(31 downto 0);
      output_wip : out  std_logic_vector(31 downto 0));

end gf_xor_9x;

architecture behavior of gf_xor_9x is

begin  -- behavior

  p_gf_xor_9x: process (input_fcs, input_wip)
  begin  -- process p_gf_xor_9x
                 output_wip (31) <= input_wip(31);
                 output_wip (30) <= input_wip(30);
                 output_wip (29) <= input_wip(29);
                 output_wip (28) <= input_wip(28);
                 output_wip (27) <= input_wip(27);
                 output_wip (26) <= input_wip(26);
                 output_wip (25) <= input_wip(25);
                 output_wip (24) <= input_wip(24);
                 output_wip (23) <= input_wip(23);
                 output_wip (22) <= input_wip(22);
                 output_wip (21) <= input_wip(21);
                 output_wip (20) <= input_wip(20);
                 output_wip (19) <= input_wip(19);
                 output_wip (18) <= input_wip(18);
                 output_wip (17) <= input_wip(17);
                 output_wip (16) <= input_wip(16);
                 output_wip (15) <= input_wip(15);
                 output_wip (14) <= input_wip(14);
                 output_wip (13) <= input_wip(13);
                 output_wip (12) <= (input_wip(12) xor input_fcs(16));
                 output_wip (11) <= input_wip(11);
                 output_wip (10) <= input_wip(10);
                 output_wip (9) <= input_wip(9);
                 output_wip (8) <= input_wip(8);
                 output_wip (7) <= input_wip(7);
                 output_wip (6) <= input_wip(6);
                 output_wip (5) <= input_wip(5);
                 output_wip (4) <= input_wip(4);
                 output_wip (3) <= input_wip(3);
                 output_wip (2) <= input_wip(2);
                 output_wip (1) <= input_wip(1);
                 output_wip (0) <= input_wip(0);
  end process p_gf_xor_9x;

end behavior;
```

A.2.5 gf_multiplier.vhd

```
--===========================================================================
-- File        : gf_multiplier.vhd
-- Author      : José María Nadal
-- Date        : Mar, 2001
-- Description : Connection of the lower levels in the hierarchy

-- Copyright (C) 2001  José María Nadal <chema@scouts-es.org>

-- This program is free software; you can redistribute it and/or
-- modify it under the terms of the GNU General Public License
-- as published by the Free Software Foundation; either version 2
-- of the License, or (at your option) any later version.

-- This program is distributed in the hope that it will be useful,
-- but WITHOUT ANY WARRANTY; without even the implied warranty of
-- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
-- GNU General Public License for more details.

-- You should have received a copy of the GNU General Public License
-- along with this program; if not, write to the Free Software
-- Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
--===========================================================================

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.all;

entity gf_multiplier is

  port (
    reset  : in  std_logic;            -- #RESET
    phi1   : in  std_logic;
    phi2   : in  std_logic;
    input  : in  std_logic_vector(31 downto 0);
                                       -- Input to the Galois Field multiplier.
                                       -- It comes from the feedback of the FCS
    output_fcs : out std_logic_vector(15 downto 0);  -- LS Word
    -- "inout" to be able to read the signal (feedback)
    output_xor : out std_logic_vector(15 downto 0));   -- MS Word

end gf_multiplier;



architecture structural of gf_multiplier is

  -- The output register is half the size of the rest
  component gf_phi1_register_out
    port (
      reset        : in  std_logic;          -- #RESET
      phi1         : in  std_logic;          -- Clock
      input_wip    : in  std_logic_vector(31 downto 0);
      output_final : out std_logic_vector(31 downto 0));
  end component;
```

```
component gf_phi1_register
  port (
    reset      : in  std_logic;                      -- #RESET
    phi1       : in  std_logic;                      -- Clock
    input_wip  : in  std_logic_vector(31 downto 0);  -- The incoming WIP
    input_fcs  : in  std_logic_vector(31 downto 0);
                  -- The original data for that step. We are using pipelining
                  -- so we have to grant that we will have the original FCS
                  -- available.
    output_wip : out std_logic_vector(31 downto 0);
                  -- The modified data -our "WIP"-
    output_fcs : out std_logic_vector(31 downto 0));
                  -- The original data is kept untouched
end component;

component gf_phi2_register
  port (
    reset      : in  std_logic;                      -- #RESET
    phi2       : in  std_logic;                      -- Clock
    input_wip  : in  std_logic_vector(31 downto 0);  -- The incoming WIP
    input_fcs  : in  std_logic_vector(31 downto 0);
                  -- The original data for that step. We are using pipelining
                  -- we have to grant that we will have the original FCS data
                  -- available.
    output_wip : out std_logic_vector(31 downto 0);
                  -- The modified data -our "WIP"-
    output_fcs : out std_logic_vector(31 downto 0));
                  -- The original data is kept untouched
end component;

-- These components below are the best example of bad VHDL coding

component gf_xor_2x
  port (
    input_wip  : in  std_logic_vector(31 downto 0);
    input_fcs  : in  std_logic_vector(31 downto 0);
    output_wip : out  std_logic_vector(31 downto 0));
end component;

component gf_xor_3x
  port (
    input_wip  : in  std_logic_vector(31 downto 0);
    input_fcs  : in  std_logic_vector(31 downto 0);
    output_wip : out  std_logic_vector(31 downto 0));
end component;

component gf_xor_4x
  port (
    input_wip  : in  std_logic_vector(31 downto 0);
    input_fcs  : in  std_logic_vector(31 downto 0);
    output_wip : out  std_logic_vector(31 downto 0));
end component;

component gf_xor_5x
  port (
    input_wip  : in  std_logic_vector(31 downto 0);
    input_fcs  : in  std_logic_vector(31 downto 0);
    output_wip : out  std_logic_vector(31 downto 0));
end component;

component gf_xor_6x
  port (
    input_wip  : in  std_logic_vector(31 downto 0);
    input_fcs  : in  std_logic_vector(31 downto 0);
    output_wip : out  std_logic_vector(31 downto 0));
end component;

component gf_xor_7x
  port (
    input_wip  : in  std_logic_vector(31 downto 0);
    input_fcs  : in  std_logic_vector(31 downto 0);
    output_wip : out  std_logic_vector(31 downto 0));
end component;

component gf_xor_8x
  port (
    input_wip  : in  std_logic_vector(31 downto 0);
    input_fcs  : in  std_logic_vector(31 downto 0);
    output_wip : out  std_logic_vector(31 downto 0));
end component;

component gf_xor_9x
  port (
    input_wip  : in  std_logic_vector(31 downto 0);
    input_fcs  : in  std_logic_vector(31 downto 0);
    output_wip : out  std_logic_vector(31 downto 0));
end component;

-- The XOR right after the input is smaller
component gf_xor_input
  port (
    input_fcs  : in  std_logic_vector(31 downto 0);
    output_wip : out std_logic_vector(31 downto 0));
end component;

-- We now declare all the signals needed to comunicate the
-- different components
signal btw2_3   : std_logic_vector(31 downto 0);  -- Original data
signal btw3_4   : std_logic_vector(31 downto 0);  -- Original data
signal btw4_5   : std_logic_vector(31 downto 0);  -- Original data
signal btw5_6   : std_logic_vector(31 downto 0);  -- Original data
signal btw6_7   : std_logic_vector(31 downto 0);  -- Original data
signal btw7_8   : std_logic_vector(31 downto 0);  -- Original data
signal btw8_9   : std_logic_vector(31 downto 0);  -- Original data
signal btw9_10  : std_logic_vector(31 downto 0);  -- Original data

signal btw1x_2   : std_logic_vector(31 downto 0);  -- WIP data
signal btw2_2x   : std_logic_vector(31 downto 0);  -- WIP data
signal btw2x_3   : std_logic_vector(31 downto 0);  -- WIP data
signal btw3_3x   : std_logic_vector(31 downto 0);  -- WIP data
signal btw3x_4   : std_logic_vector(31 downto 0);  -- WIP data
signal btw4_4x   : std_logic_vector(31 downto 0);  -- WIP data
signal btw4x_5   : std_logic_vector(31 downto 0);  -- WIP data
```

```
  signal btw5_5x   : std_logic_vector(31 downto 0);  -- WIP data
  signal btw5x_6   : std_logic_vector(31 downto 0);  -- WIP data
  signal btw6_6x   : std_logic_vector(31 downto 0);  -- WIP data
  signal btw6x_7   : std_logic_vector(31 downto 0);  -- WIP data
  signal btw7_7x   : std_logic_vector(31 downto 0);  -- WIP data
  signal btw7x_8   : std_logic_vector(31 downto 0);  -- WIP data
  signal btw8_8x   : std_logic_vector(31 downto 0);  -- WIP data
  signal btw8x_9   : std_logic_vector(31 downto 0);  -- WIP data
  signal btw9_9x   : std_logic_vector(31 downto 0);  -- WIP data
  signal btw9x_10  : std_logic_vector(31 downto 0);  -- WIP data

begin  -- structural

  GF1x : gf_xor_input port map (input_fcs => input, output_wip => btw1x_2);
  GF2  : gf_phi1_register port map (reset => reset, phi1 => phi1,
                                   input_wip  => btw1x_2, input_fcs  => input,
                                   output_wip => btw2_2x, output_fcs => btw2_3);

  GF2x : gf_xor_2x port map (input_wip  => btw2_2x, input_fcs  => btw2_3,
                          output_wip => btw2x_3);
  GF3  : gf_phi2_register port map (reset => reset, phi2 => phi2,
                                   input_wip  => btw2x_3, input_fcs  => btw2_3,
                                   output_wip => btw3_3x, output_fcs => btw3_4);

  GF3x : gf_xor_3x port map (input_wip  => btw3_3x, input_fcs  => btw3_4,
                          output_wip => btw3x_4);
  GF4  : gf_phi1_register port map (reset => reset, phi1 => phi1,
                                   input_wip  => btw3x_4, input_fcs  => btw3_4,
                                   output_wip => btw4_4x, output_fcs => btw4_5);

  GF4x : gf_xor_4x port map (input_wip  => btw4_4x, input_fcs  => btw4_5,
                          output_wip => btw4x_5);
  GF5  : gf_phi2_register port map (reset => reset, phi2 => phi2,
                                   input_wip  => btw4x_5, input_fcs  => btw4_5,
                                   output_wip => btw5_5x, output_fcs => btw5_6);

  GF5x : gf_xor_5x port map (input_wip  => btw5_5x, input_fcs  => btw5_6,
                          output_wip => btw5x_6);
  GF6  : gf_phi1_register port map (reset => reset, phi1 => phi1,
                                   input_wip  => btw5x_6, input_fcs  => btw5_6,
                                   output_wip => btw6_6x, output_fcs => btw6_7);

  GF6x : gf_xor_6x port map (input_wip  => btw6_6x, input_fcs  => btw6_7,
                          output_wip => btw6x_7);
  GF7  : gf_phi2_register port map (reset => reset, phi2 => phi2,
                                   input_wip  => btw6x_7, input_fcs  => btw6_7,
                                   output_wip => btw7_7x, output_fcs => btw7_8);

  GF7x : gf_xor_7x port map (input_wip  => btw7_7x, input_fcs  => btw7_8,
                          output_wip => btw7x_8);
  GF8  : gf_phi1_register port map (reset => reset, phi1 => phi1,
                                   input_wip  => btw7x_8, input_fcs  => btw7_8,
                                   output_wip => btw8_8x, output_fcs => btw8_9);

  GF8x : gf_xor_8x port map (input_wip  => btw8_8x, input_fcs  => btw8_9,
                          output_wip => btw8x_9);
  GF9  : gf_phi2_register port map (reset => reset, phi2 => phi2,
                                  input_wip  => btw8x_9, input_fcs  => btw8_9,
                                  output_wip => btw9_9x, output_fcs => btw9_10);

  GF9x : gf_xor_9x port map (input_wip  => btw9_9x, input_fcs  => btw9_10,
                         output_wip => btw9x_10);
  GF10 : gf_phi1_register_out port map (reset => reset, phi1 => phi1,
                                  input_wip  => btw9x_10,
                                  output_final(15 downto 0) => output_xor,
                                  output_final(31 downto 16)  => output_fcs);

end structural;


configuration cfg_gf_multiplier of gf_multiplier is

  for structural
    for GF1x : gf_xor_input use entity work.gf_xor_input(behavior); end for;
    for GF2  : gf_phi1_register
             use entity work.gf_phi1_register(behavior); end for;
    for GF2x : gf_xor_2x
             use entity work.gf_xor_2x(behavior); end for;
    for GF3  : gf_phi2_register
             use entity work.gf_phi2_register(behavior); end for;
    for GF3x : gf_xor_3x
             use entity work.gf_xor_3x(behavior); end for;
    for GF4  : gf_phi1_register
             use entity work.gf_phi1_register(behavior); end for;
    for GF4x : gf_xor_4x
             use entity work.gf_xor_4x(behavior); end for;
    for GF5  : gf_phi2_register
             use entity work.gf_phi2_register(behavior); end for;
    for GF5x : gf_xor_5x
             use entity work.gf_xor_5x(behavior); end for;
    for GF6  : gf_phi1_register
             use entity work.gf_phi1_register(behavior); end for;
    for GF6x : gf_xor_6x
             use entity work.gf_xor_6x(behavior); end for;
    for GF7  : gf_phi2_register
             use entity work.gf_phi2_register(behavior); end for;
    for GF7x : gf_xor_7x
             use entity work.gf_xor_7x(behavior); end for;
    for GF8  : gf_phi1_register
             use entity work.gf_phi1_register(behavior); end for;
    for GF8x : gf_xor_8x
             use entity work.gf_xor_8x(behavior); end for;
    for GF9  : gf_phi2_register
             use entity work.gf_phi2_register(behavior); end for;
    for GF9x : gf_xor_9x
             use entity work.gf_xor_9x(behavior); end for;
    for GF10 : gf_phi1_register_out
             use entity work.gf_phi1_register_out(behavior); end for;
  end for;

end cfg_gf_multiplier;
```

A.2.6 big_xor.vhd

```
--=============================================================================
-- File        : big_xor.vhd
-- Author      : José María Nadal
-- Date        : Mar, 2001
-- Description : Defines the output stage

-- Copyright (C) 2001  José María Nadal <chema@scouts-es.org>

-- This program is free software; you can redistribute it and/or
-- modify it under the terms of the GNU General Public License
-- as published by the Free Software Foundation; either version 2
-- of the License, or (at your option) any later version.

-- This program is distributed in the hope that it will be useful,
-- but WITHOUT ANY WARRANTY; without even the implied warranty of
-- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
-- GNU General Public License for more details.

-- You should have received a copy of the GNU General Public License
-- along with this program; if not, write to the Free Software
-- Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
--=============================================================================


library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity big_xor is
  port (
    reset       : in  std_logic;
    -- This reset is more a "set" to the initial FCS value
    phi2        : in  std_logic;
    input_input : in  std_logic_vector(15 downto 0);
    fcs_input   : in  std_logic_vector(15 downto 0);  -- LS Word
    gf_input    : in  std_logic_vector(15 downto 0);  -- MS Word
    output      : out std_logic_vector(31 downto 0));
end big_xor ;

architecture behavior of big_xor is

  signal output_xor : std_logic_vector(15 downto 0);
  -- Intermediate signal between the XOR and the FCS register

begin  -- behavior

  -- purpose: This is the final part of the generator. The input and the
  -- output from the multiplier are XOR-ed in order to obtain the final value.
  -- type   : sequential
  -- inputs : phi2, reset, input_input, fcs_input, gf_input
  -- outputs: output
  p_big_xor_register: process (phi2, reset)
  begin  -- process p_big_xor_register
    if reset = '0' then                 -- asynchronous reset (active low)
      output <= X"46AF6449";
    elsif phi2'event and phi2 = '1' then  -- rising clock edge
      output(15 downto 0)  <= output_xor(15 downto 0);
      output(31 downto 16) <= fcs_input(15 downto 0);
    end if;
  end process p_big_xor_register;

  p_big_xor_combinational: process (gf_input, input_input)
  begin  -- process p_big_xor_combinational
      output_xor <= gf_input xor input_input;
  end process p_big_xor_combinational;

end behavior;
```

A.2.7 crc_top.vhd

```
--============================================================================
-- File        : CRC_top.vhd
-- Author      : José María Nadal
-- Date        : Mar, 2001
-- Description : Top level file

-- Copyright (C) 2001  José María Nadal <chema@scouts-es.org>

-- This program is free software; you can redistribute it and/or
-- modify it under the terms of the GNU General Public License
-- as published by the Free Software Foundation; either version 2
-- of the License, or (at your option) any later version.

-- This program is distributed in the hope that it will be useful,
-- but WITHOUT ANY WARRANTY; without even the implied warranty of
-- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
-- GNU General Public License for more details.

-- You should have received a copy of the GNU General Public License
-- along with this program; if not, write to the Free Software
-- Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
--============================================================================


library IEEE;
use IEEE.std_logic_1164.all;

entity CRC_top is

  port (
    phi1    : in  std_logic;
                                       -- We will use the two phase discipline
                                       -- which we don't generate.
    phi2    : in  std_logic;
    reset   : in  std_logic;           -- #RESET
    input   : in  std_logic_vector(15 downto 0);
                                       -- The serial/parallel conversion has
                                       -- been made somewhere else
    fcs_out : out std_logic_vector(31 downto 0));
    -- "inout" because we have to read this value (feedback to the multiplier)

end CRC_top;
```

```
architecture structural of CRC_top is

  component input_wait
    port (
      reset  : in  std_logic;
      phi1   : in  std_logic;
      phi2   : in  std_logic;
      input  : in  std_logic_vector(15 downto 0);
      output : out std_logic_vector(15 downto 0));
  end component;

  component gf_multiplier
    port (
      reset      : in  std_logic;
      phi1       : in  std_logic;
      phi2       : in  std_logic;
      input      : in  std_logic_vector(31 downto 0);
      output_fcs : out std_logic_vector(15 downto 0);     -- LS Word
      -- "inout" since we have to read this value (feedback)
      output_xor : out std_logic_vector(15 downto 0));    -- MS Word
  end component;

  component big_xor
    port (
      reset       : in  std_logic;
      phi2        : in  std_logic;
      input_input : in  std_logic_vector(15 downto 0);
      fcs_input   : in  std_logic_vector(15 downto 0);
      gf_input    : in  std_logic_vector(15 downto 0);
      output      : out std_logic_vector(31 downto 0));
  end component;

  component ff_reset is
    port (
      phi2         : in  std_logic;
      reset_glitch : in  std_logic;
      reset_clean  : out std_logic);
  end component;

  signal wait_intermediate : std_logic_vector(15 downto 0);
  -- Connects the input_wait component with the big_xor one
  signal fcs_intermediate : std_logic_vector(15 downto 0);
  -- Connects the multiplier with the output register
  signal xor_intermediate : std_logic_vector(15 downto 0);
  -- Connects the multiplier with the final XOR
  signal fcs_out_read : std_logic_vector (31 downto 0);
  -- This signal will avoid the use of "inout" ports
  signal reset_intermediate : std_logic;
  -- Clean reset to feed the whole circuit

begin

  ff_reset_1 : ff_reset port map (phi2 => phi2, reset_glitch => reset,
                                  reset_clean => reset_intermediate);

  input_wait_1 : input_wait port map (reset => reset_intermediate, phi1 => phi1,
                                    phi2 => phi2, input  => input,
                                    output => wait_intermediate);

  gf_multiplier_1 : gf_multiplier port map (reset => reset_intermediate,
                                          phi1 => phi1, phi2=> phi2,
                                          input => fcs_out_read,
                                          output_xor => xor_intermediate,
                                          output_fcs => fcs_intermediate);

  big_xor_1 : big_xor port map (reset => reset_intermediate, phi2 => phi2,
                                input_input => wait_intermediate,
                                fcs_input   => fcs_intermediate,
                                gf_input    => xor_intermediate,
output      => fcs_out_read);

  fcs_out <= fcs_out_read;

end structural;

configuration cfg_CRC_top_structural of CRC_top is

  for structural
    for input_wait_1 : input_wait use entity work.input_wait(structural);
    end for;
    for gf_multiplier_1 : gf_multiplier use entity work.gf_multiplier(structural);
    end for;
    for big_xor_1 : big_xor use entity work.big_xor(behavior); end for;
    for ff_reset_1 : ff_reset use entity work.ff_reset(behavior); end for;
  end for;

end cfg_CRC_top_structural;
```

A.2.8 Test bench: input_wait_tb.vhd

```
--============================================================================
-- File        : input_wait_tb.vhd
-- Author      : José María Nadal
-- Date        : Mar, 2001
-- Description : Test bench file for input_wait.vhd

-- Copyright (C) 2001  José María Nadal <chema@scouts-es.org>

-- This program is free software; you can redistribute it and/or
-- modify it under the terms of the GNU General Public License
-- as published by the Free Software Foundation; either version 2
-- of the License, or (at your option) any later version.

-- This program is distributed in the hope that it will be useful,
-- but WITHOUT ANY WARRANTY; without even the implied warranty of
-- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
-- GNU General Public License for more details.

-- You should have received a copy of the GNU General Public License
-- along with this program; if not, write to the Free Software
-- Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
--============================================================================
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

-- synopsys synthesis on
library STD;
use STD.TEXTIO.all;
-- synopsys synthesis off

entity input_wait_tb is end input_wait_tb;

architecture structural of input_wait_tb is

component input_wait
  port (
    phi1    : in    std_logic;
    phi2    : in    std_logic;
    reset   : in    std_logic;
    input   : in    std_logic_vector(15 downto 0);
    output  : out   std_logic_vector(15 downto 0));
end component;

signal phi1, phi2, reset : std_logic;
signal input             : std_logic_vector(15 downto 0);
signal output            : std_logic_vector(15 downto 0);

-- We also specify the files used to get the data and to put the data back
file file_read : text is in "/var/tmp/data/data_input_in.dat";
file file_out  : text is out "/var/tmp/data/data_input_out.dat";

begin  -- structural


  -- Important: These parameters have been calculated for a 500MHz
  -- relative clock -2 ns per stage-
  p_phi1: process
  begin  -- process p_phi1
    phi1 <= '1', '0' after 2.1 ns;
    wait for 4.2 ns;
  end process p_phi1;

  p_phi2: process
  begin  -- process p_phi2
    phi2 <= '1', '0' after 0.3 ns, '1' after 1.8 ns;
    wait for 4.2 ns;
  end process p_phi2;

  p_reset: process
  begin  -- process p_reset
    reset <= '0' after 10 ns, '1' after 30 ns;
    wait;
  end process p_reset;

  p_input: process (phi1)
    variable input_file : bit_vector(15 downto 0);
    variable line : line;
  begin  -- process p_input
   if phi1'event and phi1='1' then
    if (not (endfile(file_read))) then
      readline (file_read, line);
      read(line, input_file);
      input <= To_StdLogicVector(input_file);
    end if;
   end if;
  end process p_input;

  p_output: process (phi1)
    variable output_file : bit_vector(15 downto 0);
    variable line : line;
  begin  -- process p_output
    if phi1'event and phi1 = '1' then  -- rising clock edge
      output_file:=To_BitVector(output);
      write(line,output_file);
      writeline(file_out,line);
    end if;
  end process p_output;

  input_wait_1 : input_wait port map (phi1 => phi1, phi2 => phi2,
      reset => reset, input => input,
                                  output => output);

end structural;

configuration cfg_input_wait_structural of input_wait_tb is

  for structural
    for input_wait_1: input_wait use entity work.input_wait(structural);
    end for;
  end for;

end cfg_input_wait_structural;
```

A.2.9 Test bench: gf_multiplier_tb.vhd

```
--=============================================================================
-- File        : gf_multiplier_tb.vhd
-- Author      : José María Nadal
-- Date        : Mar, 2001
-- Description : Test bench file for gf_multiplier.vhd

-- Copyright (C) 2001  José María Nadal <chema@scouts-es.org>

-- This program is free software; you can redistribute it and/or
-- modify it under the terms of the GNU General Public License
-- as published by the Free Software Foundation; either version 2
-- of the License, or (at your option) any later version.

-- This program is distributed in the hope that it will be useful,
-- but WITHOUT ANY WARRANTY; without even the implied warranty of
-- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
-- GNU General Public License for more details.

-- You should have received a copy of the GNU General Public License
-- along with this program; if not, write to the Free Software
-- Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
--=============================================================================

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

-- synopsys synthesis on
library STD;
use STD.TEXTIO.all;
-- synopsys synthesis off

entity gf_multiplier_tb is end gf_multiplier_tb;

architecture structural of gf_multiplier_tb is

component gf_multiplier
  port (
    phi1       : in    std_logic;
    phi2       : in    std_logic;
    reset      : in    std_logic;
    input      : in    std_logic_vector(31 downto 0);
    output_fcs : inout std_logic_vector(15 downto 0);
    output_xor : out   std_logic_vector(15 downto 0));
end component;

signal phi1, phi2, reset : std_logic;
signal input             : std_logic_vector(31 downto 0);
signal output_fcs        : std_logic_vector(15 downto 0);
signal output_xor        : std_logic_vector(15 downto 0);

-- We also specify the files used to get the data and to put the data back
file file_read     : text is in "/var/tmp/data/data_multiplier_input.dat";
file file_out_fcs  : text is out "/var/tmp/data/data_multiplier_fcs_out.dat";
file file_out_xor  : text is out "/var/tmp/data/data_multiplier_xor_out.dat";

begin  -- structural


  -- Important: These parameters have been calculated for a 500MHz
  -- relative clock -2 ns per stage-
  p_phi1: process
  begin  -- process p_phi1
    phi1 <= '1', '0' after 2.1 ns;
    wait for 4.2 ns;
  end process p_phi1;

  p_phi2: process
  begin  -- process p_phi2
    phi2 <= '1', '0' after 0.3 ns, '1' after 1.8 ns;
    wait for 4.2 ns;
  end process p_phi2;

  p_reset: process
  begin  -- process p_reset
    reset <= '0' after 10 ns, '1' after 30 ns;
    wait;
  end process p_reset;

  p_input: process (phi2)
    variable input_file : bit_vector(31 downto 0);
    variable line : line;
  begin  -- process p_input
   if phi2'event and phi2='1' then
    if (not (endfile(file_read))) then
      readline (file_read, line);
      read(line, input_file);
      input <= To_StdLogicVector(input_file);
    end if;
   end if;
  end process p_input;

  p_output: process (phi1)
    variable output_file_fcs : bit_vector(15 downto 0);
    variable output_file_xor : bit_vector(15 downto 0);
    variable line_fcs : line;
    variable line_xor : line;
  begin  -- process p_output
    if phi1'event and phi1 = '1' then  -- rising clock edge
      output_file_fcs:=To_BitVector(output_fcs);
      write(line_fcs,output_file_fcs);
      writeline(file_out_fcs,line_fcs);

      output_file_xor:=To_BitVector(output_xor);
      write(line_xor,output_file_xor);
      writeline(file_out_xor,line_xor);
    end if;
  end process p_output;

  gf_multiplier_1 : gf_multiplier port map (phi1 => phi1, phi2 => phi2,
          reset => reset, input => input,
                                            output_fcs => output_fcs,
                                            output_xor => output_xor);

end structural;

configuration cfg_gf_multiplier_tb_structural of gf_multiplier_tb is

  for structural
    for gf_multiplier_1: gf_multiplier use entity work.gf_multiplier(structural);
    end for;
  end for;

end cfg_gf_multiplier_tb_structural;
```

A.2.10 Test bench: big_xor_tb.vhd

```
--=========================================================================
-- File        : big_xor_tb.vhd
-- Author      : José María Nadal
-- Date        : Mar, 2001
-- Description : Test bench file for big_xor.vhd
-- Copyright (C) 2001  José María Nadal <chema@scouts-es.org>

-- This program is free software; you can redistribute it and/or
-- modify it under the terms of the GNU General Public License
-- as published by the Free Software Foundation; either version 2
-- of the License, or (at your option) any later version.

-- This program is distributed in the hope that it will be useful,
-- but WITHOUT ANY WARRANTY; without even the implied warranty of
-- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
-- GNU General Public License for more details.

-- You should have received a copy of the GNU General Public License
-- along with this program; if not, write to the Free Software
-- Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
--=============================================================================

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

-- synopsys synthesis on
library STD;
use STD.TEXTIO.all;
-- synopsys synthesis off

entity big_xor_tb is end big_xor_tb;

architecture structural of big_xor_tb is

component big_xor
  port (
    phi2        : in std_logic;
    reset       : in std_logic;
    input_input : in std_logic_vector(15 downto 0);
    fcs_input   : in std_logic_vector(15 downto 0);
    gf_input    : in std_logic_vector(15 downto 0);
    output      : out std_logic_vector(31 downto 0));
end component;

signal phi2, reset : std_logic;
signal input_input : std_logic_vector(15 downto 0);
signal fcs_input   : std_logic_vector(15 downto 0);
signal gf_input    : std_logic_vector(15 downto 0);
signal output      : std_logic_vector(31 downto 0);

signal phi1        : std_logic;
-- Necessary to be able to run the process that generates phi1

-- We also specify the files used to get the data and to put the data back
file file_read_input : text is in "/var/tmp/data/data_xor_input.dat";
file file_read_fcs   : text is in "/var/tmp/data/data_xor_fcs.dat";
file file_read_gf    : text is in "/var/tmp/data/data_xor_gf.dat";
file file_out        : text is out "/var/tmp/data/data_xor_out.dat";

begin  -- structural
```

```
-- Important: These parameters have been calculated for a 500MHz
-- relative clock -2 ns per stage-
p_phi1: process
begin  -- process p_phi1
  phi1 <= '1', '0' after 2.1 ns;
  wait for 4.2 ns;
end process p_phi1;

p_phi2: process
begin  -- process p_phi2
  phi2 <= '1', '0' after 0.3 ns, '1' after 1.8 ns;
  wait for 4.2 ns;
end process p_phi2;

p_reset: process
begin  -- process p_reset
  reset <= '0' after 10 ns, '1' after 30 ns;
  wait;
end process p_reset;

p_input: process (phi1)
  variable line_input : line;
  variable line_fcs : line;
  variable line_gf : line;
  variable file_input : bit_vector(15 downto 0);
  variable file_fcs : bit_vector(15 downto 0);
  variable file_gf : bit_vector(15 downto 0);
begin  -- process p_input
 if phi1'event and phi1='1' then
  -- We just verify the end of a file (the files will be done of the same length)
  if (not (endfile(file_read_input))) then
    readline (file_read_input, line_input);
    read(line_input, file_input);
    input_input <= To_StdLogicVector(file_input);

    readline (file_read_fcs, line_fcs);
    read(line_fcs, file_fcs);
    fcs_input <= To_StdLogicVector(file_fcs);

    readline (file_read_gf, line_gf);
    read(line_gf, file_gf);
    gf_input <= To_StdLogicVector(file_gf);
  end if;
 end if;
end process p_input;

p_output: process (phi2)
  variable output_file : bit_vector(31 downto 0);
  variable line : line;
begin  -- process p_output
  if phi2'event and phi2 = '1' then  -- rising clock edge
    output_file:=To_BitVector(output);
    write(line,output_file);
    writeline(file_out,line);
  end if;
end process p_output;
```

```
  big_xor_1 : big_xor port map (phi2 => phi2, reset => reset, gf_input => gf_input,
                                input_input => input_input, fcs_input => fcs_input,
                                output => output);

end structural;

configuration cfg_big_xor_structural of big_xor_tb is

  for structural
    for big_xor_1: big_xor use entity work.big_xor(behavior); end for;
  end for;

end cfg_big_xor_structural;
```

A.2.11 Test bench: CRC_top_tb.vhd

```
--============================================================================
-- File        : CRC_top_tb.vhd
-- Author      : José María Nadal
-- Date        : Mar, 2001
-- Description : Test bench file for CRC_top.vhd

-- Copyright (C) 2001  José María Nadal <chema@scouts-es.org>

-- This program is free software; you can redistribute it and/or
-- modify it under the terms of the GNU General Public License
-- as published by the Free Software Foundation; either version 2
-- of the License, or (at your option) any later version.

-- This program is distributed in the hope that it will be useful,
-- but WITHOUT ANY WARRANTY; without even the implied warranty of
-- MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
-- GNU General Public License for more details.

-- You should have received a copy of the GNU General Public License
-- along with this program; if not, write to the Free Software
-- Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
--============================================================================

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

-- synopsys synthesis on
library STD;
use STD.TEXTIO.all;
-- synopsys synthesis off

entity CRC_top_tb is end CRC_top_tb;

architecture structural of CRC_top_tb is

component CRC_top
  port (
    phi1    : in    std_logic;
    phi2    : in    std_logic;
    reset   : in    std_logic;
    input   : in    std_logic_vector(15 downto 0);
    fcs_out : out std_logic_vector(31 downto 0));
end component;

signal phi1, phi2, reset : std_logic;
signal input             : std_logic_vector(15 downto 0);
signal fcs_out           : std_logic_vector(31 downto 0);

-- We also specify the files used to get the data and to put the data back
file file_read : text is in "/var/tmp/data/data_in.dat";
file file_out  : text is out "/var/tmp/data/data_out.dat";

begin  -- structural


  -- Important: These parameters have been calculated for a 500MHz
  -- relative clock -2 ns per stage-
  p_phi1: process
  begin  -- process p_phi1
    phi1 <= '1', '0' after 2.1 ns;
    wait for 4.2 ns;
  end process p_phi1;

  p_phi2: process
  begin  -- process p_phi2
    phi2 <= '1', '0' after 0.3 ns, '1' after 1.8 ns;
    wait for 4.2 ns;
  end process p_phi2;

  p_reset: process
  begin  -- process p_reset
    reset <= '0' after 10 ns, '1' after 30 ns;
    wait;
  end process p_reset;

  p_input: process (phi1)
    variable input_file : bit_vector(15 downto 0);
    variable line : line;
  begin  -- process p_input
   if phi1'event and phi1='1' then
    if (not (endfile(file_read))) then
      readline (file_read, line);
      read(line, input_file);
      input <= To_StdLogicVector(input_file);
    end if;
   end if;
  end process p_input;

  p_output: process (phi2)
    variable output_file : bit_vector(31 downto 0);
    variable line : line;
  begin  -- process p_output
    if phi2'event and phi2 = '1' then  -- rising clock edge
      output_file:=To_BitVector(fcs_out);
      write(line,output_file);
      writeline(file_out,line);
    end if;
  end process p_output;

  CRC_1 : CRC_top port map (phi1 => phi1, phi2 => phi2, reset => reset,
                            input => input, fcs_out => fcs_out);

end structural;

configuration cfg_CRC_top_tb_structural of CRC_top_tb is

  for structural
    for CRC_1: CRC_top use entity work.CRC_top(structural); end for;
  end for;

end cfg_CRC_top_tb_structural;
```

B I/O interface

B.1 Interfaces of the different blocks

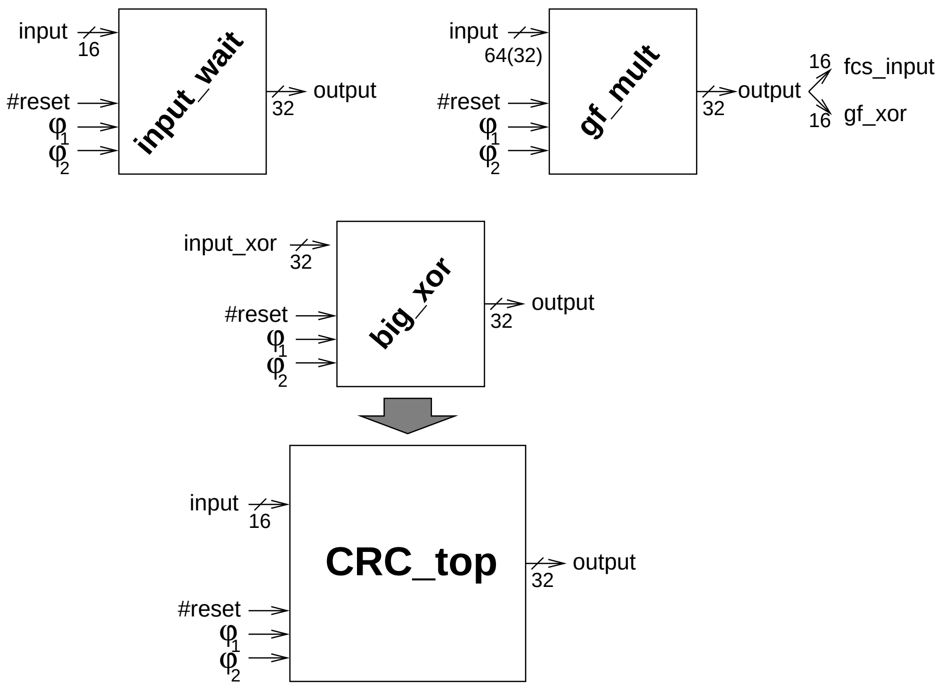

Figure 16: The three main blocks and their interfaces

B.2 I/O interface of the chip

The PADs have been selected so that a maximum I/O speed could be reached. In this way, we will have 57 PADs:

- 51 of them will be I/O PADs, 16 for the data input, 3 for reset and the clock (two phases) and other 32 for the output.
- 6 PADs will be devoted to power (1) and ground (5).

Special care has been put on the placement of the PADs. The inputs are sorrounded by DC lines, in an attempt to minimize the coupling between the rest of the PADs and them.

When it comes to the choice of the PADs, we have to say that it is remarkable the fact that be will use analog PADs for the inputs, since they can reach much better performance in speed. However, this is achieved at the cost of the reduction of protective elements and buffers, so we will have to hadle the chip with care to avoid destruction due to static discharges.

The output PADs have been kept digital, since they just delay the output and we do need the buffers within them to be able to measure the outputs with reasonable levels.

The PADs are named as follows:

- *A_PAD[0..15]*: Input PADs (analog).
- *B_PAD[0..31]*: Output PADs (digital).
- *PWR1*: Input, Vdd.

- *PWR2*: Input, GND.
- *PWR3*: Input, GND.
- *I1*: Input, φ_1.
- *I2*: Input, φ_2.
- *I3*: Input, reset.
- *GND1*: Input, GND.
- *GND2*: Input, GND.
- *GND3*: Input, GND.

The location of the PADs in the circuit can be seen in figure 17.

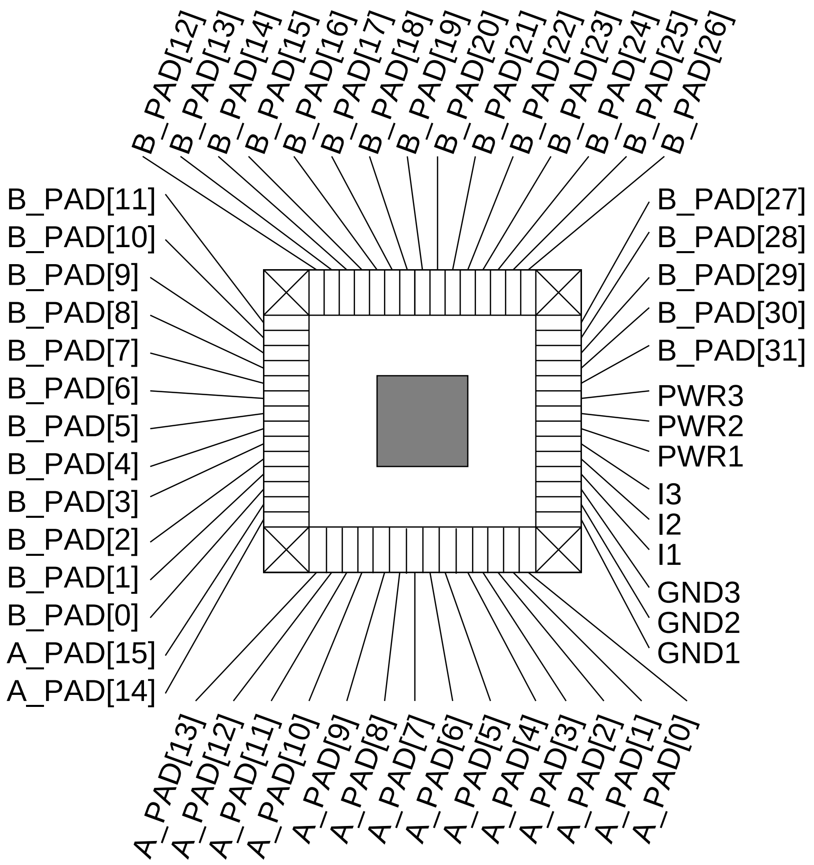

Figure 17: Distribution of the PADs in the chip. I/O interface

B.3 Package

In order not to decrease much the performance of the chip we will have to choose carefully the package of the chip. In this way, we chose a standard QFP package, which was the best one available for this kind of purpose. However, some chips were also requested not to have package. This way we will be able to test the chip in different conditions, from “full speed” (with special probes in the lab and no package) to the lower speed but easier to test packaged version.

In any case, if we ever planned to release this circuit as a conventional one, we would have to choose a better packaging, for example ball grid array packages.

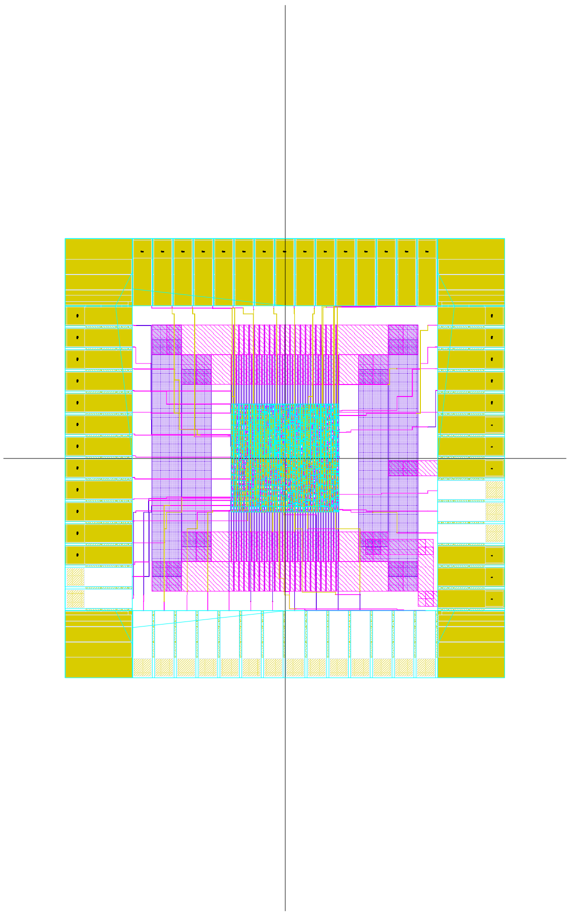

Figure 18: Layout of the physical chip

C License of this work

C.1 GNU GENERAL PUBLIC LICENSE

C.1.1 Preamble

C.1.2 TERMS AND CONDITIONS..

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserv-

ing the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

C.1.3 NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

C.2 GNU Free Documentation License

Version 1.1, March 2000

Copyright © 2000 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

C.2.1 Preamble

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

C.2.2 Applicability and Definitions

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below,

C.2.3 Verbatim Copying

C.2.4 Copying in Quantity

C.2.5 Modifications

may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

- In any section entitled “Acknowledgements” or “Dedications”, preserve the section’s title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- Delete any section entitled “Endorsements”. Such a section may not be included in the Modified Version.
- Do not retitle any existing section as “Endorsements” or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties – for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

C.2.6 Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled “History” in the various original documents, forming one section entitled “History”; likewise combine any sections entitled “Acknowledgements”, and any sections entitled “Dedications”. You must delete all sections entitled “Endorsements.”

C.2.7 Collections of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

C.2.8 Aggregation With Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

C.2.9 Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

C.2.10 Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

C.3 Future Revisions of This License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

C.3.1 ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

LTH, Elektrovetenskap, Lund 2002