Final Test (Digital Logic Design Project)

Prof. Fabio Salice – Academic Year 2019/2020

Daniele Giorgianni

Index

1 Introduction 3
1.1 Project aim 3
1.2 Overview 3
1.3 Interface of the component 3
1.4 Data and memory description 4
2 Design properties 5
2.1 Machine States 5
2.2 Designed picks 6
3 Testbench 8
4 Conclusions 8
4.1 Results of the synthesis 8
4.2 Optimizations 9

1 Introduction

1.1 Project aim

The purpose of the project is to develop a hardware component in VHDL that can identify the membership of a memory address to one of the frequently accessed memory areas, called Working Zone, so that we can reduce the power dissipated by the machine.

1.2 Overview

Working Zones are a range of addresses with a fixed size and are specified within memory through their base address.

The addresses to be treated are composed of 8 bits of which only 7 significant bits, so that we can access memory addresses ranging from 0 to 127.

If the address to be encoded was part of one of the Working Zones, which are eight, it will be properly encoded before exiting the synthesized hardware component to be written in memory, if it were not to be part of it, it will have written as it was read.

Address encoding provides that the most significant bit is set to 1, the three preceding bits will indicate the number of the Working Zone to which it belongs, and the four least significant will indicate the offset from the base address of the One-Hot-encoded Working Zone.

1.3 Interface of the component

The component interface, as per specification, is as follows:

```
entity project_reti_logiche is
    port(
        i_clk       :   in std_logic;
        i_start     :   in std_logic;
        i_rst       :   in std_logic;
        i_data      :   in std_logic_vector(7 downto 0);
        o_address   :   out std_logic_vector(15 downto 0);
        o_done      :   out std_logic;
        o_en        :   out std_logic;
        o_we        :   out std_logic;
        o_data      :   out std_logic_vector(7 downto 0)
        );
end project_reti_logiche;
```

Especially:

- i_clk indicates the incoming clock signal;
- i_start if it is brought to 1 it starts the processing and will have to remain high throughout the encoding, when this is finished it can be brought back to 0 to give the module the possibility to start with another processing;
- i_rst is the reset signal, which brings the module to its starting state in whatever situation it is in;
- i_data contains the address that is read from memory;
- o_address is the address of the memory cell to which the module points;
- o_done if it is 1 represents the encoding termination, 0 when the processing process is in progress;
- o_en is a bit that must be worth 1 in order to interact with memory, 0 otherwise;
- o_we is a bit indicating the possibility of writing to memory, must be worth 1 to write and 0 to read;
- o_data contains the value that was processed by the module at the end of the process.

1.4 Data and memory description

The memory is divided as follows:

address	contents
0	Basic address WZ_0
1	Basic address WZ_1
2	Basic address WZ_2
3	Basic address WZ_3
4	Basic address WZ_4
5	Basic address WZ_5
6	Basic address WZ_6
7	Basic address WZ_7
8	Address to encode
9	Address (possibly) coded

Each Working Zone consists of four addresses starting from the basic one.

The address to be treated is present in address 8 of the memory, while the address (possibly) encoded will be written to address 9.

2 Design properties

The hardware component developed is based on a synchronous finite state machine with respect to the clock signal ascent front.

The FSM consists of six states, the first is the initialization one, where we remain until the i_start signal is increased to 1. Once the processing is complete, it will be in the final state where it will raise the o_done termination signal to 1 and remain in it until the i_start signal is brought to 0, that is the condition for which the current state will return to be the first. The FSM will return to the initialization state at the time the i_rst signal is raised (as required by specification) and if the i_start signal should drop, by mistake, during computation.

2.1 Machine States

The set of states constituting the FSM is {S0, S1, S2, S3, S4, S5}, in detail:

- S0 is the initialization state, where variables are set to the default values and expects the i_start switch to 1. Whenever the i_rst will raised or i_start will lowered during processing, it will return to this state;
- S1 is the state in which signals are set for reading and then the read data is stored in a variable;
- S2 is the state in which it is verified that the address read belongs to one of the eight Working Zones, if so the next state will be S3, otherwise S4;
- S3 is the state in which the address is encoded, as specified, and will later be written to memory;
- S4 is the state in which the read address is copied outbound, as it does not belong to any Working Zone;
- S5 is the termination state, where you communicate the end of the processing by raising the signal o_done and expect the i_start be lowered.

2.2 Designed picks

The hardware component, based on the FPGA xc7a200tfbg484-1, has been described through a single process, which takes care of the entire processing.

It was decided not to store the basic address of the Working Zones in the component in order to reduce the memory elements required for it.

Once the address to be encoded has been read, it will be compared with the basic addresses of the WZ so that it can determine whether it belongs.

The comparison between the address to be encoded and the base address of the WZ is by subtraction of the second from the first, after converting both to integers. If the result of this operation is between 0 and 3 then the address will be part of the WZ and the result will indicate the offset from the base, otherwise you will compare it with the base address of the next one.

The comparison of the address to be encoded with the base address of the WZ starts with the first WZ, specified in memory cell 0. If a conclusion of membership is reached, the remaining (any) WZ, i.e. not compared, will be ignored and the machine will go to the next state for encoding the address, otherwise, then exhausted the WZ to be compared, the machine will go to the next state where it will copy the address to be encoded in the output one.

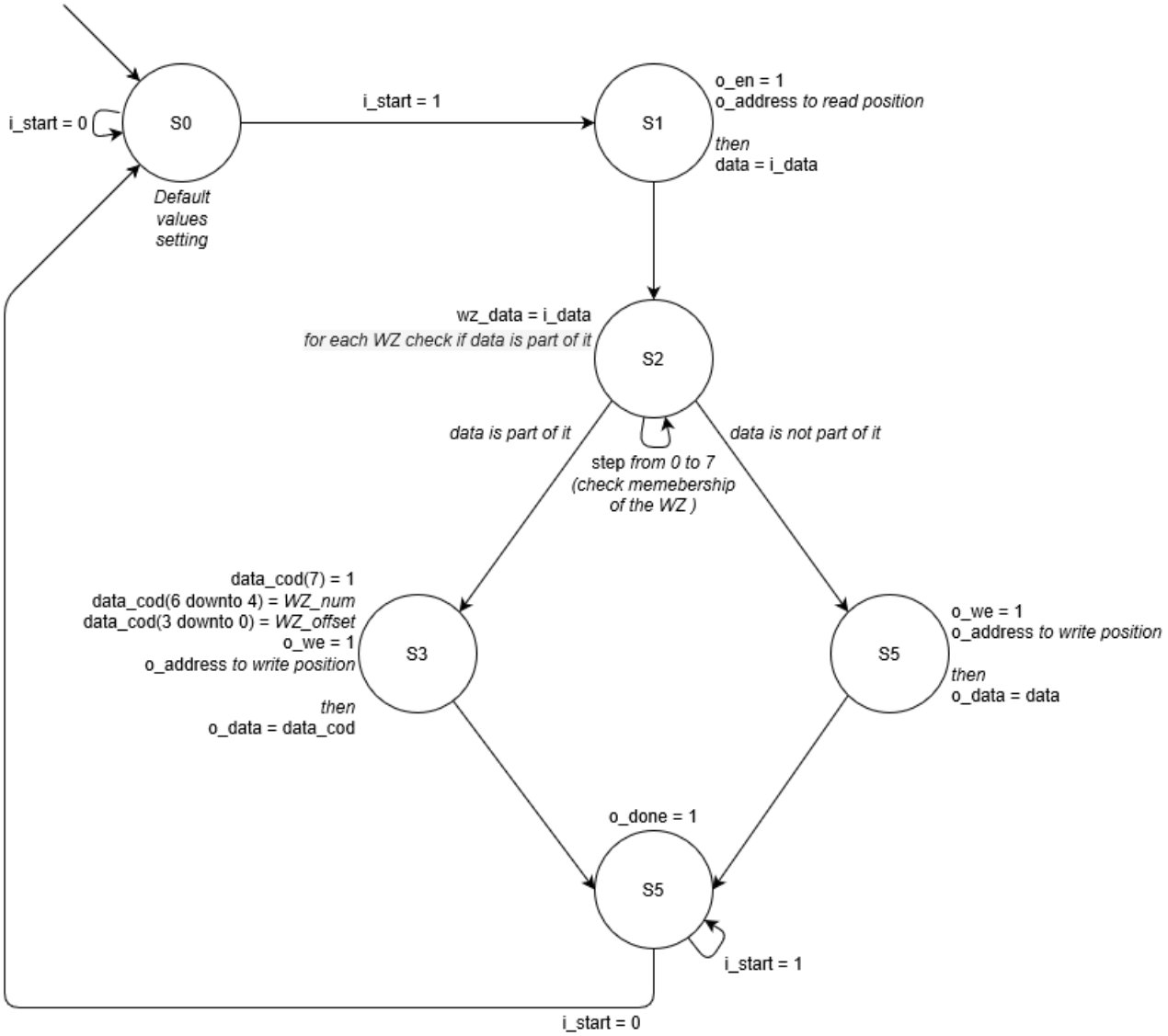
The following page shows the image of the FSM.

The image is deprived of two types of links to facilitate reading, which are:

- The connections coming out of each state directed to the initial state S0 due to the presence of the reset signal;
- Outgoing links from states {S1, S2, S3, S4, S5} directed to the initial state S0 due to the fact that each state proceeds to run only if i_start remains high, otherwise the state will become S0 (as specified at the beginning of this chapter).

Variables used by the process:

```
type state_type is (S0, S1, S2, S3, S4, S5);
signal State : state_type;
shared variable step: integer range 0 to 7;
shared variable clk_cycle: integer range -1 to 1;
shared variable data, wz_data, data_cod : std_logic_vector(7 downto 0);
```



3 Testbench

The synthesized component positively passes both the tests provided by the teachers and those independently generated to go and cover as many cases as possible.

The tests developed, obviously synchronous with the clock, can be grouped according to the cases covered by them:

- Various memory configurations, so as to test a large number of addresses to been coded, both in the case of WZ membership and in the opposite case, and assign various basic addresses to the WZ contained in memory (including boundary cases such as address 0 and 124);
- Multiple start, so as to evaluate both the correct start of a processing and its correct execution throughout the entire coding process;
- Multiple resets, so that the machine checks the correct transaction at the initial state when the reset signal is raised in any available state.

These three types have often been combined to generate as comprehensive tests as possible.

4 Conclusions

4.1 Results of the synthesis

The implemented hardware component, as described above, positively passes tests in both Behavioral Simulation and Post – Synthesis Functional Simulation, using a clock period of 100ns.

Using the "Schematic" button on the menu under the heading "Synthesized Design" you can observe the number of various resources used by the component, which are:

- 176 Cells;
- 213 Nets;
- 81 LUT (corresponding to 0.06% of the available quantity);
- FF 47 (corresponding to 0.02% of the available quantity).

4.2 Optimizations

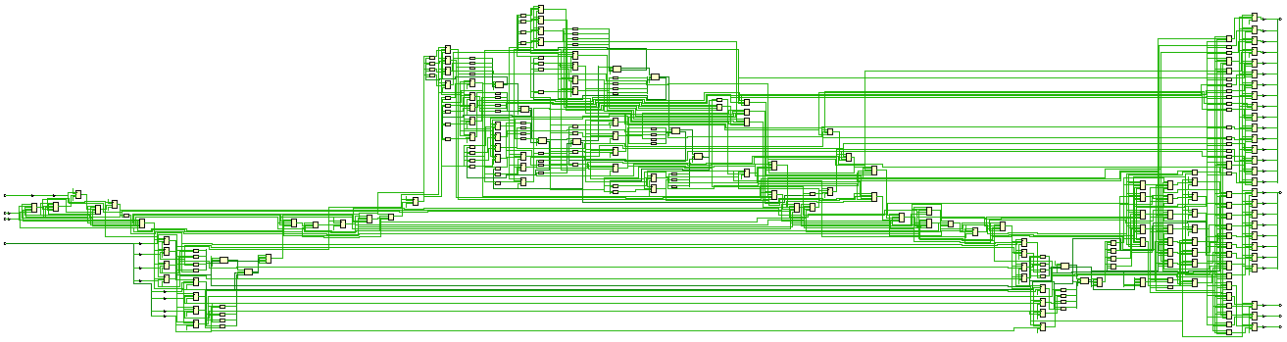
The FSM implemented in its first version used resources as follows:

- 321Cells;
- 411Nets;
- 175LUT;
- FF 92.

Following careful analysis of the written code, it was possible to reduce the number of variables used, eliminate unnecessary assignments, and optimize the same by inserting appropriate conditions "if – then – else ". This allowed us to have:

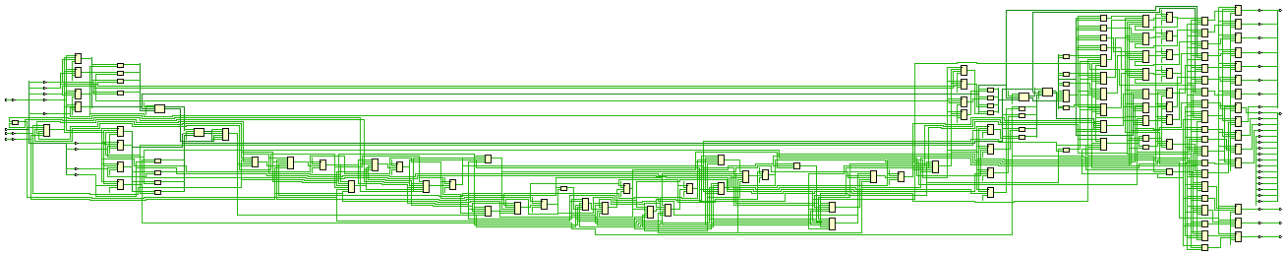
- 299 Cells;
- 389 Nets;
- 157 LUT;
- FF 88.

With a scheme as follows:



A change in the number of resources used was allowed, in addition to a further reduction in the variables used (which led to the rewriting of some parts of the code), by the insertion of a range for the integer variables used. Allowing you to have the values indicated in Chapter 4.1.

The diagram of the latter optimization is below in order to compare it with the previous one in terms of the area used.



On the following page you canobserve rand the scheme inits entirety.