Reti Logiche - Prof. Fornaciari William
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

Implementing a Working-Zone based Encoding of Addresses

Piersilvio De Bartolomeis (887260)
Ian Di Dio Lavore (891500)

10-03-2020

Contents

1. Introduction

1.1 Abstract

Multiple studies have shown that a Working-Zone approach can lower the power consumption of Microprocessors Address Buses [1][2]. This project focuses on implementing such technique on an FPGA, providing a design that works properly during pre-synthesis and post-synthesis while keeping the code as simple as possible for future developments.

1.2 Component Interface

Two main actors can be seen in this project, the actual component to develop and the memory with which it interacts. The memory was provided within the testbench as a Single-Port Block RAM in Write-First mode. The component to be created had to comply with the following interface:

```
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;
```

In particular:

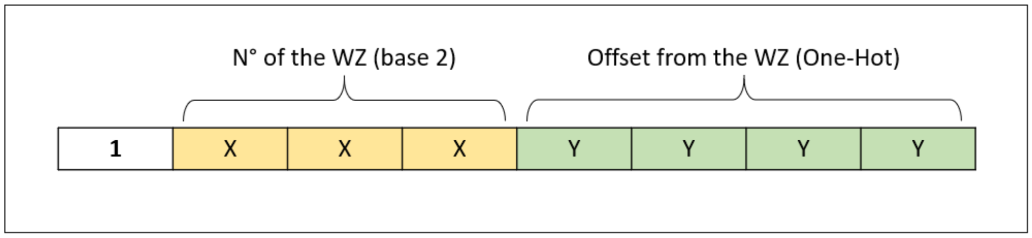
- **i_clk** is the CLOCK input signal generated in the testbench.
- **i_start** is the START signal generated in the testbench.
- **i_rst** is the RESET signal that initializes the machines, making it ready to receive the first START signal.
- **i_data** is the signal coming from the memory after a read request.
- **o_address** is the output signal (vector) that sends the address to the memory.
- **o_en** is the ENABLE signal for the memory; it must be high for the memory to operate.
- **o_we** is the WRITE ENABLE signal that we send to the memory in order to write; when this signal is high the memory writes the content in 'o_data', when this signal is low the memory read the address in 'o_address'.
- **o_data** is the output signal that goes into the memory.

1.3 Component Behaviour

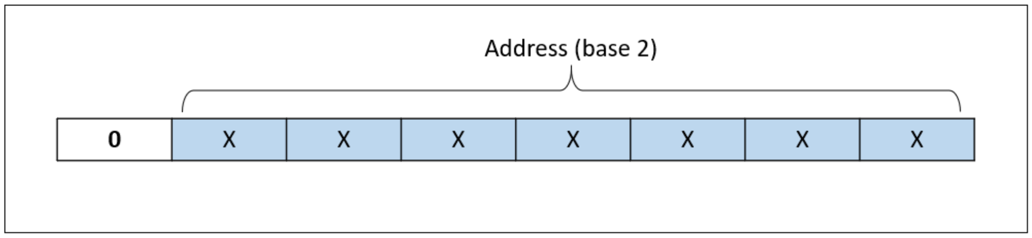
We can describe the basic functioning of our component with the steps below:

1. The component positions itself in a waiting stage, looking for a start signal.
2. As the start signal arrives it requests the address to encode from the RAM.
3. It saves this address, and simultaneously requests the first Working-Zone.
4. The Working-Zone is read and the component checks if the address belongs to it.
5. If the address belongs to the Working-Zone, it writes to the memory the encoding 1, otherwise it will ask the RAM for the next Working-Zone and repeat from the step 4 or, if this was the last Working-Zone to be checked it will save with the encoding 2.

The specifications stated that an initial reset signal is provided, so we used this step to initialize all the variables.



Encoding 1 – «IN A WORKING ZONE»



Encoding 2 – «NOT IN A WORKING ZONE»

1.4 Speed limitations

The upper bound for the number of clocks required for the computations was limited mainly by the RAM. It required one clock cycle for one read or write instruction. So a maximum speed objective was set. To obtain that result during the computation, we simultaneously elaborate the data presented by the RAM and we also ask the memory for the data necessary for the upcoming cycle.
Another possible optimization was found in caching the previously fetched Working-Zones, by doing so, when another computation comes in without a reset signal, our component would not be limited by the RAM. Since we had no additional information regarding the frequency of the reset signal, we made the assumption that the Working-Zones might change frequently, creating no need to implement a caching system through registers.

2. Architecture

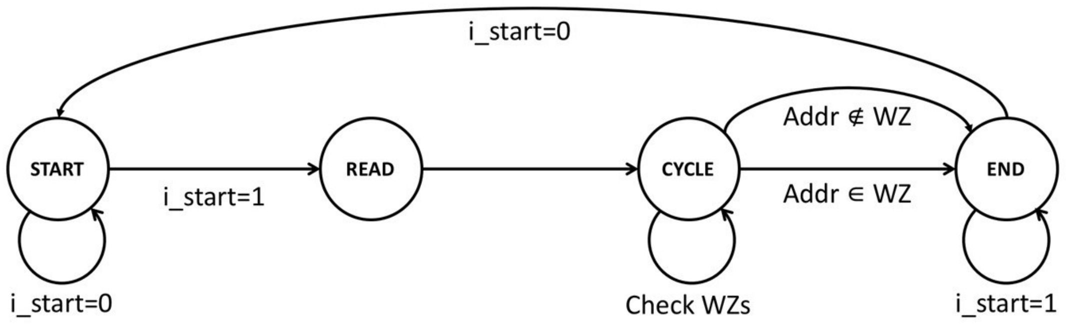


Figure 2.1: *Schematic of the FSM*

2.1 Finite State Machine

Given the specifications of the project (especially i_reset, i_start and i_done signal) we opted for a FSM (Finite State Machine) design for the component. The structure follows figure 2.1, please note that we decided to remove the asynchronous reset signal for a better readability of the schematic. We will now present the four states of the FSM:

- START: the machine is idle while the **i_start** signal is set to LOW, it wakes when the **i_start** signal is set to HIGH. When **i_rst** is set to HIGH we go back to this state.
- READ: the address to be encoded is loaded from the RAM, the first Working-Zone is requested and then the machine moves to the next state unconditionally.
- CYCLE: the address loaded in **i_data** is compared with the eight Working-Zones. There are two possible outcomes for this computation and they determine how **o_data** is composed:
  - The address belongs to one of the Working-Zones: the resulting encoding is **1 & WZ_NUM & WZ_OFFSET**. Where WZ_NUM is the number of the working zone encoded in binary and WZ_OFFSET is the offset from the base address of the working zone encoded as one-hot.
  - The address does not belong to any Working-Zone: the resulting address is **0 & ADDRESS**.
- END: the encoded address is written on the RAM. Once **i_start** is set to LOW the machine goes back to the "START" state.

When the signal i_start is set to HIGH, the component starts its computation moving from START to READ. Once the computation is done, after the encoded address is written in memory, the component sets the o_done signal to HIGH. Then, the user sets i_start to LOW and the component's response is to set o_done back to LOW. The component goes back to its idle state (START), waiting for i_start to be set HIGH again.

2.2 RAM

As shown in Figure 2.2 our component is using the first 10 slots of the RAM:

- **Slot 0 to 7:** these slots are used to store the Working-Zones.
- **Slot 8:** this slot contains the address that will be encoded.
- **Slot 9:** this slot contains the result of the elaboration, i.e. the encoded address.

Different optimizations have taken place during the FSM design process. One of the biggest limits was the memory with which the component is connected. The type of RAM required by the specifications requires at least one clock cycle to present the data required or to write the data into it.

In our first implementation, the machine required two clock cycles to read from the RAM:

- Request: we set **o_en** to HIGH and **o_address** to the slot of RAM we wanted to read.
- Read: we read the data received from the RAM in **i_data**.

Later, we optimized this process sending a new request during the read phase, this reduced the number of clock cycles required to one and shortened the execution time. Furthermore, we were able to reduce the states of the FSM, since a "request state" was no longer necessary.

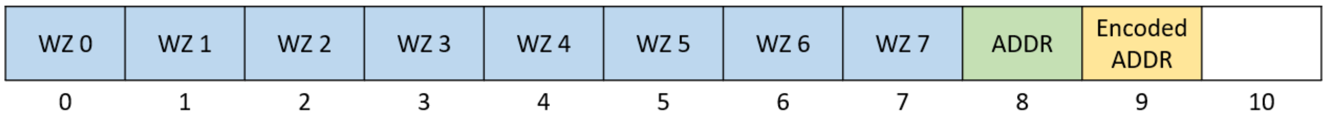


Figure 2.2: RAM values

2.3 VHDL Implementation

We present a single process implementation with two signals in his sensitivity list: **i_clk** and **i_rst**. The former provides a correct implementation of the asynchronous reset while the latter activates the process on its falling edge and thus, the execution of the current state.

We decided to save data as little as possible. Indeed, the component reads the Working-Zones from RAM for each encoding. This approach provides better scalability: if the number of working zones should increase we would not need major modifications in the source code.

3. Synthesis

As previously said, we achieved a design that works both in pre-synthesis and post-synthesis. In this section we will present only the meaningful statistics extracted from the Vivado synthesis report.

3.1 Occupied Area

The project uses Vivado v2019.2 for the synthesis phase which offers a "Report Utilization" feature. While no effort in optimizing the area used by the component was made (as it was not required by the specifications provided), we can see from this report that the usage of the target FPGA was of various order of magnitude lower than the availability.

Table 3.1: Utilization report

Resource	Utilization	Available	Utilization %
LUT	39	134600	0.03
FF	28	269200	0.01

3.2 Timing

We created a constraint file containing a clock period of 100ns. This allowed us to obtain a timing report. In this report we can see how fast the synthesized design is related to a single clock cycle. We obtained a Worst Negative Slack of 95,415ns, combining this result with the delay created by the RAM, we can calculate the minimum clock period for our design:

$$T_{min} = T_{curr} - WNS + T_{ram} = 100ns - 95,415ns + 1ns = 5,585ns \tag{3.1}$$

The results in 3.1 provide us with a maximum clock frequency of:

$$f_{max} = 1/T_{min} \approx 180Mhz \tag{3.2}$$

3.3 Post-Synthesis Warnings

One warning was presented after the synthesis of the component:

- "Port driven by constant 0":

This message is due to the fact that "o_address" is composed by 16 bits, but we actually use only the four less significant bits. Indeed, "o_address" is used to access the RAM (indexed with 16 bits) and as we just access the first 10 cells of it, this warning should be expected.

4. Simulations

We created specific testbenches, in order to verify the correct functioning of our component in normal scenarios as well as in extreme cases. Apart from the specific tests, we decided to simulate a huge number of random tests (450 thousands) in order to increase the likelihood of finding bugs in our implementation. Here, we report the most significant test cases.

4.1 Extreme cases

We verified that our component works properly in the following cases:

- Testbench in which the address to encode is **111111** (Corner Case).
- Testbench in which the address to encode is **0000000** (Corner Case).
- Testbench in which we evaluate the correct functioning of the component when successive elaborations without a reset are presented.
- Testbench in which we evaluate the correct functioning of the component when a reset signal is received during elaboration.
- Testbench in which we evaluate the correct functioning of the component when multiple reset signals are received.

4.2 Testbench 1

In this testbench a normal case is being tested: it can be a good baseline for our project. The i_rst signal stays low during the elaboration. We have only one address to encode and it belongs to the third Working-Zone. The address is encoded correctly and then written on the RAM.

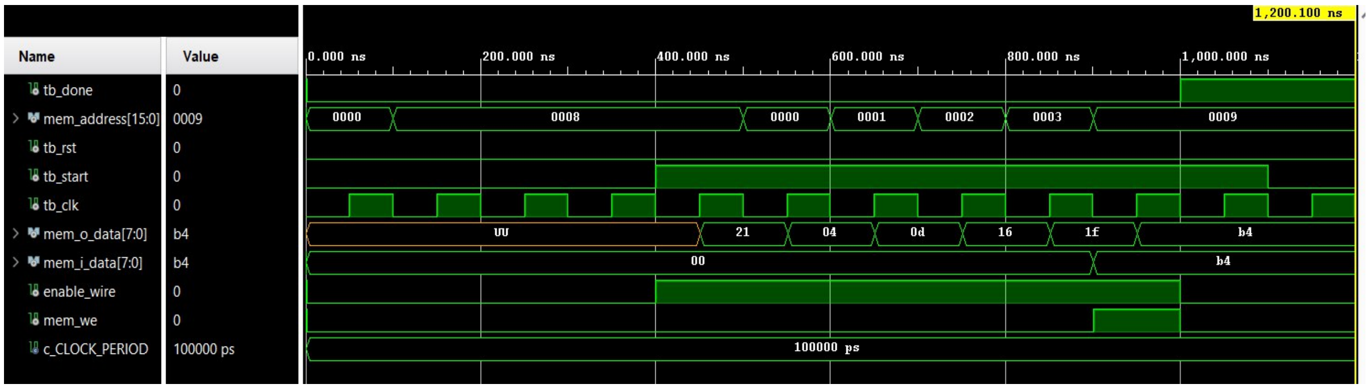


Figure 4.1: *Post-synthesis Simulation*

4.3 Testbench 2

In this testbench a more relevant case is being tested. There is a reset preceding the start signal and the address does not belong to any Working-Zone. The component loops through all the working zone, then it encodes the address and writes it on the RAM.

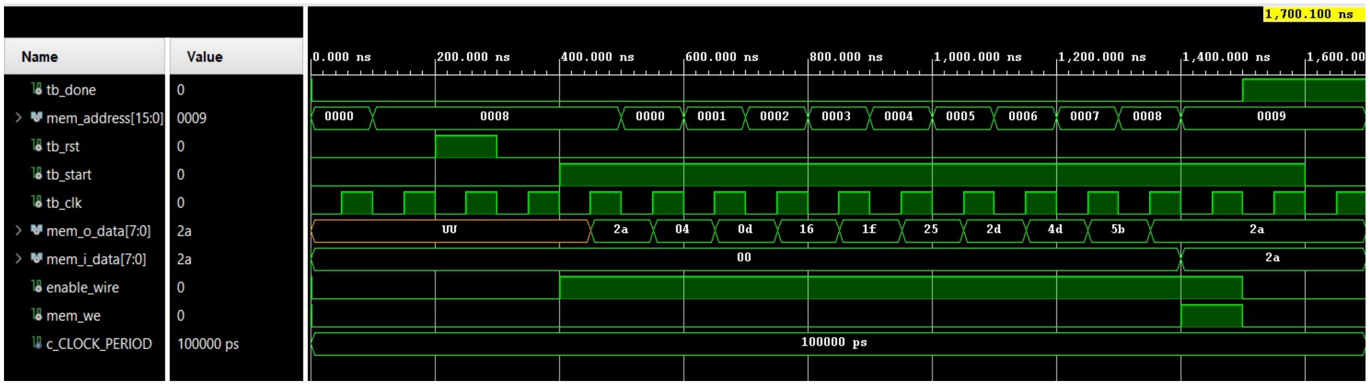


Figure 4.2: *Post-synthesis Simulation*

5. Conclusions

To summarized we created a design with the following characteristic:

- Functioning properly in pre-synthesis and post-synthesis.
- Optimized in order to take full advantage of the RAM.
- Maximum clock frequency can be set to 180Mhz.
- LUT utilization is 0.03%.
- FF utilization is 0.01%.

Bibliography

[1] Enric Musoll, Tomás Lang, and Jordi Cortadella. Exploiting the locality of memory references to reduce the address bus energy. In *Proceedings of 1997 International Symposium on Low Power Electronics and Design*, pages 202–207. IEEE, 1997.

[2] Enric Musoll, Tomás Lang, and Jordi Cortadella. Working-zone encoding for reducing the energy in microprocessor address buses. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 6(4):568–572, 1998.