Relazione Prova Finale, Progetto Reti Logiche

Andrea Cioccarelli 10713858

Alberto Cantele 10766393

Prof. William Fornaciari, Federico Reghenzani
A.A. 2022/2023

Contents

POLITECNICO
MILANO 1863

1 Introduction

1.1 Project Overview

The project's goal is to design a hardware module (with memory access) acting as an address dereferencer, with stateful output capabilities. The circuit is implemented using VHDL to design the component and to enable its synthesis on FPGA (the project specifics impose the `Artix-7 xc7a200tfbg484-1` board)

1.2 Component Specifications

On a high level, the component receives (through an input bus) an output selector (two bits) and an address (up to 16 bits). The selector is used to choose which output port is considered as active for the duration of the current component operation. The component should interface with the external memory component, transmitting the received address and receiving the 8-bit data word associated with the given input address. Finally, the component should output the received information on one of its four designated output (8-bit) buses, and it should do so while also retaining any eventual previous output data in the same state, unless explicitly overridden.

The specifications describe a synchronous component, clocked at a period of $100ns$. It follows that `i_clk` is the clock signal, while `i_rst` is the reset signal. We receive the input signal bit by bit from `i_w`. `i_start` represents an input read enabler. The `o_z0`, `o_z1`, `o_z2` and `o_z3` are the four output ports, `o_done` is the done bit, and the remaining ports are for memory access.

```vhdl
entity project_reti_logiche is
        port(
                i_clk:          in  std_logic;
                i_rst:          in  std_logic;
                i_start:        in  std_logic;
                i_w:            in  std_logic;

                o_z0:           out std_logic_vector(7  downto 0);
                o_z1:           out std_logic_vector(7  downto 0);
                o_z2:           out std_logic_vector(7  downto 0);
                o_z3:           out std_logic_vector(7  downto 0);
                o_done:         out std_logic;

                o_mem_addr:     out std_logic_vector(15 downto 0);
                i_mem_data:     in  std_logic_vector(7  downto 0);
                o_mem_we:       out std_logic;
                o_mem_en:       out std_logic
        );
end project_reti_logiche;
```

Figure 1: The component's VHDL entity

1.2.1 Input Pipeline

The input data is provided through two different 1-bit bus connections:

- The start signal (`i_start`) marks whether the input reading phase should start, and thus the input data read. It is treated as an input enable;
- The data signal (`i_w`) carries all the address and selection information.

According to specifics, letting c be the number of clock cycles the `i_start` signal can be continuously be raised for, it is always true that $c \in [2, 18]$ (this means that we don't need to consider either cases where the selection bits are not complete or when the address exceeds the allowed size for the RAM component used in the project).

It follows that the amount of pure-address bits available are $k := c - 2,\ k \in [0, 16]$

When the `i_start` signal is raised, with every clock period the component should process the incoming data value (from the `i_start` bus).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	0	1	1	1	0	1	0	1	0	0	1	1	0	0	1	1	1	0

Figure 2: Maximum-length (18 bits) input address acquired through `i_w`.

The first two bits, S_0 and S_1, represent the output selection, which determines which output lane is chosen. There are 4 possible output 8-bit lanes, `z_0` through `z_3`, so the joined bits S_0S_1 uniquely identify an active output port (using a demultiplexer). The next zero to 16 bits are the address. They are ordered LSB to MSB, thus they have to be stored right-to-left, and finally padded with zeros up to 16 bits if $k < 16$.

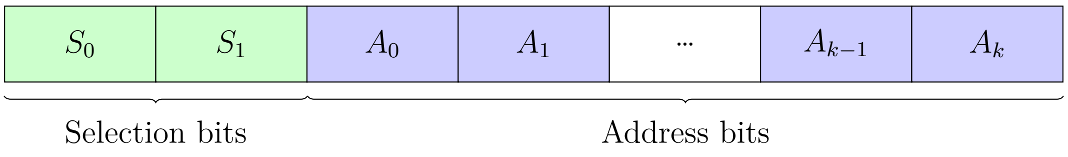


Figure 3: Dividing the input signal in two types, *select* and *address*. $k \in [0, 16]$.

1.2.2 Memory Access

The component can interface with a RAM component (Single-Port Block RAM Write-First Mode) to do the actual fetching of the address and return an 8-bit word.

The main component can interact with the memory via a set of pins:

- `o_mem_addr` is the memory address input, where the received address should be transmitted. It is a 16-bit port.
- `i_mem_data` is the return data lane from the memory, where the requested data will be received. It is an 8-bit port.
- `o_mem_we` and `o_mem_en` are enable bits for reading and writing operations. The write enable should always be off, the read enable should be on when actively interacting with the component.

0000000000000000	D_0
0000000000000001	D_1
0000000000000010	D_2
0000000000000011	D_3
0000000000000100	D_4
000000000000010	D_5
. . . .	
1111111111111101	$D_{2^{16}-3}$
1111111111111110	$D_{2^{16}-2}$
1111111111111111	$D_{2^{16}-1}$

Figure 4: Memory representation. Each line represents a mapping between an address and a data word.

1.2.3 Output and State Preservation

The component's output signals are divided as follows:

- `o_done` is the done status bit, notifying that the component has completed its work cycle and is not active;
- `o_z0` through `o_z3` are the output ports to send the actual output signal(s) to. They are 8-bit ports, since the data contained in the RAM memory is at 8 bit. Those ports should be hooked to an internal register to allow stateful output processing.

Once the main component receives the data word from memory, the specifications impose that the following must happen:

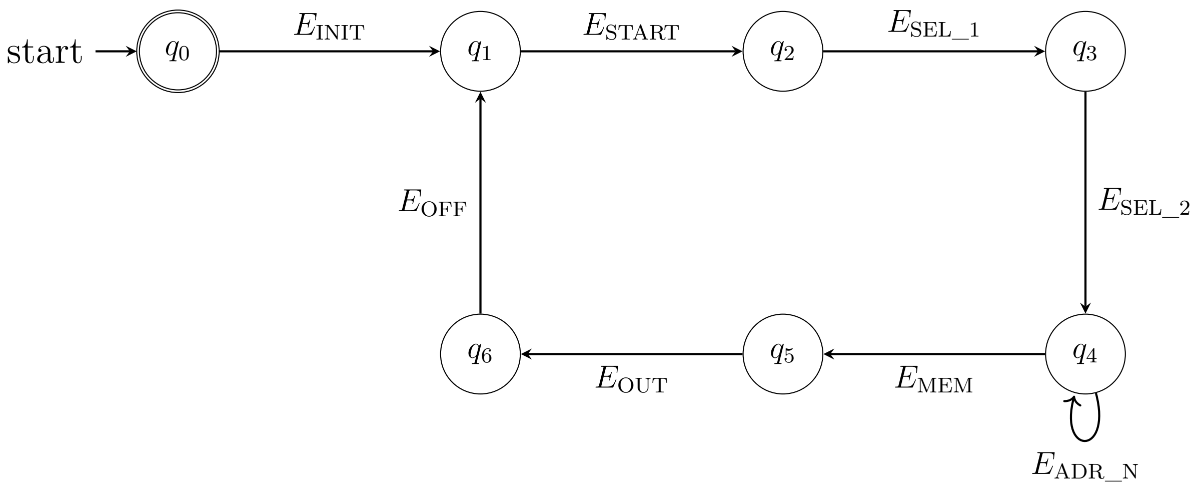
- For `o_done=0`, all the output ports should transmit an all-zero signal. This should be the default behaviour;
- For `o_done=1`, which should happen for exactly one clock cycle per component operation:
  - ⋆ The active output port (chosen by the selection bits S_0S_1) must output the data word D_i extracted from memory. The word must match the address that has been given as input during the input acquisition state;
  - ⋆ All the remaining ports should retain their previous output state. *By default*, all exits start out at null, but if an output port has already been used (read as: selected with S_0S_1, set as active and had a data word D_j output through it) in the past, then it should remember the last value it has used as output and retain it for the clock cycle when `i_done=1`.

Thus, the component must keep an internal state of the last exit data word to actually produce that output once an operation is completed. Also, at most one register's content is overridden with a single operation. The component does not allow for multiple (batched) operations.
When the component is brought in the reset state (via enabling `i_rst`, all the state information is lost (or rather, set to zero, like at the beginning).

2 Architecture

The design for the described component fits very well for a finite state machine pattern. We described the model using a fsm and implemented that in VHDL.

Note that a single-fsm design has been chosen as the current solution for its increased simplicity and consistency (for the problem at hand), when compared to multi-fsm VHDL components.



The machine has n states, each one with its specific purpose, and each state transition is marked by an event.

The states are cyclic, meaning that one component operation is translated into a full cycle of the normal states.

2.1 Reset State (q_0)

This is the fsm's initial state, and it allows for proper component initialization and output setting. This state is supposed to initialize all the internal signals and register contents. Once that is done, the machine automatically transitions to the q_1 (Idle) state

This state is reached when the component is first turned on (it is hardcoded in the component's logic), and every time the `i_rst` signal is asserted on a rising clock edge.

2.2 Idle State (q_1)

The idling state is used when there is no ongoing operation, and the `i_start` signal has not been asserted yet. This state just waits for the start signal to be asserted, and switches to the q_2 state. The state is indeed labeled throughout the code as `WAIT_START`, since its sole purpose is to detect a rising edge on the start signal. Once that happens, the fsm transitions to q_2

2.3 Selection Bits Acquisition State (q_2 and q_3)

Those two states are used for reading the first and second output selection bits, S_0 and S_1, which are stored in appropriate signals for output selection later on.

We are guaranteed to have `i_start` asserted for at least two clock cycles by the project specifications, letting us ignore the case where the signals falls before.

2.4 Address Bits Acquisition State (q_4)

This state is used to acquire a variable number (from 0 to 16) of address bits, until the `i_start` signal goes low. The first address bit is b_0 (the 0-th bit, with address value 2^0). Each new bit is stored as the next significant bit. The i-th bit has a relative address-value of 2^i.

This state can be repeated for up to 16 times.

Once the start signal goes low, we move to the q_5 state.

2.5 Memory Access State (q_5 and q_5')

This state is actually two, sequentially executed states. The received address is sent out to the memory. Its read bit is enabled. Then in the next state we wait for the return data to be transmitted by the RAM component, and once we receive them we can go ahead to q_6.

2.6 Output State (q_6)

The received data word is written in the expected exit's register (through a demux), the `o_done` bit is set, and then all the registers will output their content on `O_z0` through `o_z3`.

The done bit will be reset to zero on the next clock cycle, and the state will become idle again.

```vhdl
--| FSM state enumeration
 type fsm_main_state is (
    --| Reset state:        brings the circuit in its initial state
    RESET,

    --| Idling state:       Waiting for the `i_start` signal to go high.
    WAIT_START,

    --| Acquisition states: necessary for the full bit-by-bit acquisition of the
    --|                     selection bits and memory address. The `i_w` signal
    --|                     is read.
    ACQUIRE_SEL_BIT_1, ACQUIRE_SEL_BIT_2, ACQUIRE_ADDR_BIT_N,

    --| Fetching RAM state: sends the processed address to the RAM component and
    --|                     waits for the operation to finish.
    ASK_MEM, WAITING_STATE,

    --| Output:             sends the data to the correct output pins and returns
    --|                     to WAIT_START, storing the 8-bit word in the register
    --|                     corresponding to the selected exit.
    OUTPUT_Z
)

signal fsm_state: fsm_main_state;
```

Figure 5: The component's VHDL status enumeration

3 Results

3.1 Synthetized Component

The component can be logically divided in the flowing blocks:

- A fsm module handling input acquisition, RAM interaction and clock/reset logic;
- An output module with a demultiplexer and registers for holding information.

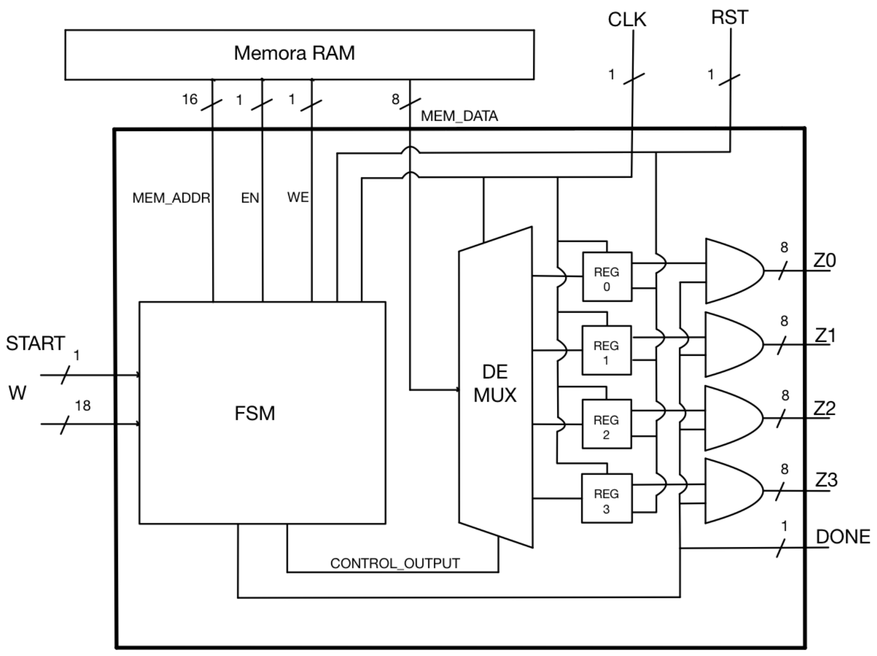


Figure 6: The component's simplified schematics

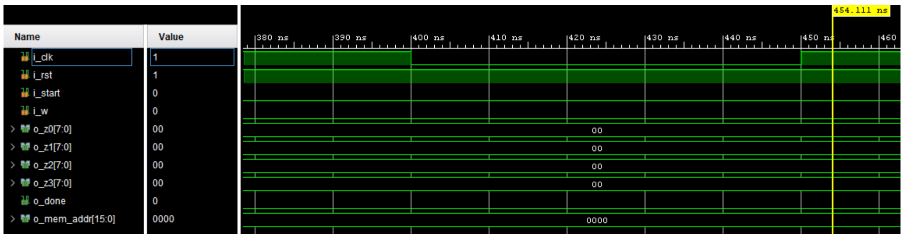
3.2 Output Logic

For the testing phase of the project, it was chosen to implement the following three main features:

- `o_done` is the done status bit, notifying that the component has compleated its work cycle;
- `o_z0` through `o_z3` are the output ports to send the actual output signal(s) to. 8-bit ports.

3.2.1 Reset Signal

An important constraint imposed by the specifics is the reset signal, which bring the fsm to the initial state independently from the current state.

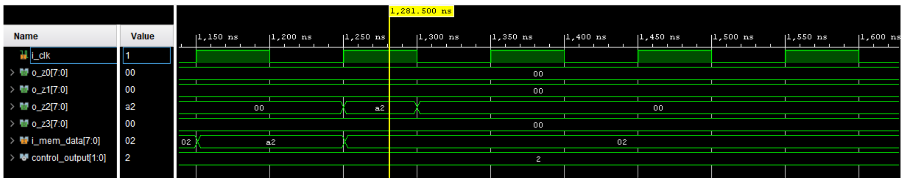


Name	Value
i_clk	1
i_rst	1
i_start	0
i_w	0
o_z0[7:0]	00
o_z1[7:0]	00
o_z2[7:0]	00
o_z3[7:0]	00
o_done	0
o_mem_addr[15:0]	0000

Figure 7: The fsm signal being reset to default after the reset signal has been asserted

3.2.2 Done Signal

The done signal is used to communicate that a result has been produced. The output signals will be activated and produce their data, and after one clock cycle everything goes back to zero.

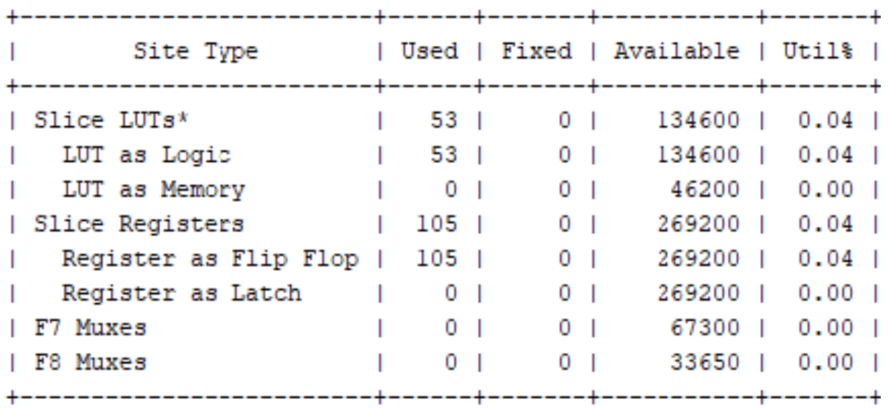


Name	Value
i_clk	1
o_z0[7:0]	00
o_z1[7:0]	00
o_z2[7:0]	a2
o_z3[7:0]	00
i_mem_data[7:0]	02
control_output[1:0]	2

Figure 8: All the output signals going back to an 8 bit zero exit value after the done bit becomes low

3.3 Post-Synthesis Analysis

The designed component is successfully synthesizable and produces the following utilization report. No latches are generated.



```
+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs*             |   53 |     0 |    134600 |  0.04 |
|   LUT as Logic          |   53 |     0 |    134600 |  0.04 |
|   LUT as Memory         |    0 |     0 |     46200 |  0.00 |
| Slice Registers         |  105 |     0 |    269200 |  0.04 |
|   Register as Flip Flop |  105 |     0 |    269200 |  0.04 |
|   Register as Latch     |    0 |     0 |    269200 |  0.00 |
| F7 Muxes                |    0 |     0 |     67300 |  0.00 |
| F8 Muxes                |    0 |     0 |     33650 |  0.00 |
+-------------------------+------+-------+-----------+-------+
```

Figure 9: Component usage table, from TLC's `report_utilization` command in Vivado

4 Conclusions

The designed component meets the project's specifics and has been tested with the provided test benches, along with some external tests as well to ensure it works under all conditions.

We analyzed our solution and a single circuit operation takes between 5 and 21 clock cycles.

An interesting aspect for the current circuit is that it is mostly stateless. Apart from holding the memory words which have been previously extracted and output through a designated exit bus, the entire component functionality and logic is completely stateless. It relies on the external memory component for data fetching and on the input ports for address

We mentioned that the component's core logic resembles an address dereferencer, given that it takes an address and gives out (mostly) a data word. This kind of circuit would model a *load* assembly instruction, given that it takes a memory address and puts in a register the corresponding word

An important note is that the component makes a few assumptions about the environment around it to work correctly, and it not resilient to changes

The list of assumptions

- The start signal is kept active long enough for the fsm to read the first two selection bits, and it is stopped after the maximum address size is reached:
  - ⋆ If the `i_start` signal were to be interrupted before the 2 clock cycle necessary to acquire the selection bits and address, zeros are used instead;
  - ⋆ If the `i_start` signal were to be kept high after 18 clock cycles, the incoming bit would become the MSB and shift right, removing the previous LSB.
- The memory takes exactly 1 clock cycle to lookup the data and return it on the `i_mem_data` port.

We have also explored different solutions for this specific project: specifically one which involved splitting the combinatory and synchronous logic in two different VHDL processes to decouple the state changes and the circuit logic in different units, but we preferred to stick with a single process given the project specifications did not require that kind of approach.