Logic Networks Project

Prof. Fornaciari, Prof. Palermo, and Prof. Salice

Academic Year 2023 - 2024

SPECIFICATIONS for Project Execution

(Updated on December 22, 2023)

General Description

The specification for the *"Final Test (Logic Networks Project)"* for the Academic Year 2023/2024 requires the implementation of a hardware module (HW) described in VHDL, which interfaces with a memory and meets the specifications listed below.

At a general level, the system reads a message consisting of a sequence of `K` words (`W`) whose values range between 0 and 255. The value **0** in the sequence should not be considered as a numerical value but rather as the information *"the value is not specified."*

The sequence of `K` words `W` to be processed is stored starting from a specified address (`ADD`), with each word occupying 2 bytes (e.g., `ADD`, `ADD+2`, `ADD+4`, ..., `ADD+2*(K-1)`). Any missing byte must be completed as described below. The module to be designed is responsible for completing the sequence by replacing zeros with the last valid (non-zero) value read in the sequence and inserting a **credibility** value (`C`) into the missing byte for each word in the sequence.

The replacement of zeros is performed by copying the last valid value (not zero) read previously and belonging to the sequence. The credibility value (`C`) is set to **31** whenever the sequence value `W` is nonzero, while it is decremented relative to the previous value each time a zero is encountered in `W`. The `C` value associated with each word `W` is stored in memory in the immediately following byte (i.e., `ADD+1` for `W` in `ADD`, `ADD+3` for `W` in `ADD+2`, ...). The `C` value is always greater than or equal to **0** and is reinitialized to **31** each time a `W` value different from zero is encountered. When `C` reaches **0**, it is no longer decremented.

Example (values in decimal)

- Initial sequence (bold W values):

  128 0 **64** 0 **0** 0 **0** 0 **0** 0 **0** 0 **0** 0 **100** 0 **1** 0 **0** 0 **5** 0 **23** 0 **200** 0 **0** 0

- Final sequence:

  128 31 **64** 31 **64** 30 **64** 29 **64** 28 **64** 27 **64** 26 **100** 31 **1** 31 **1** 30 **5** 31 **23** 31 **200** 31 **200** 30

Functionality

The module to be implemented has **3 primary inputs**: one **1-bit** (`START`), one **16-bit** (`ADD`), and one **10-bit** (`K`), and one primary **1-bit output** (`DONE`). Additionally, the module has a **clock signal** (`CLK`), unique for the entire system, and a **RESET signal**, which is also unique for the entire system. All signals are synchronous and must be interpreted on the rising edge of the clock. The only exception is `RESET`, which is asynchronous.

At the initial instant, which corresponds to the **reset** of the system, the `DONE` output must be **0**. When `RESET` returns to zero, the module will start processing when a `START` input signal is set to **1**. The `START` signal remains high until the `DONE` signal is also set high. At the end of the computation (and once the result is written into memory), the module to be designed must raise (set to **1**) the `DONE` signal, which notifies the end of processing. The `DONE` signal must remain high until the `START` signal is reset to **0**. A new `START` signal cannot be given until `DONE` has been reset to zero.

The module must be designed considering that before the first `START=1` (and before requesting the correct operation of the module), `RESET (RESET=1)` will always be given. Before the first reset of the system, the operation of the module to be implemented is unspecified. A second (or subsequent) processing with `START=1` should not require waiting for a module reset.

Each time the `RESET` signal (`RESET=1`) is given, the module is reinitialized.

When the `START` signal is set to **1** (and for the entire period in which it remains high), the `ADD` and `K` inputs are set as the starting address and the sequence length to be processed. Before raising the `DONE` signal, the module must update the sequence and the corresponding credibility values as described in the general section of the module.

Note: If the first sequence value is **0**, it remains zero, and its credibility value must be set to **0 (zero)**. The same applies until reaching the first sequence value different from zero.

Component Interface

The component to be described must have the following interface:

```
entity project_reti_logiche is
    port (
        i_clk   : in std_logic;
        i_rst   : in std_logic;
        i_start : in std_logic;
        i_add   : in std_logic_vector(15 downto 0);
        i_k     : in std_logic_vector(9 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_data : out std_logic_vector(7 downto 0);
        o_mem_we   : out std_logic;
        o_mem_en   : out std_logic
    );
end project_reti_logiche;
```

Figure 1: Component Interface Diagram

Details

The module name **must be 'project_reti_logiche'**, and there must be **only one architecture per entity**; failure to comply with this requirement makes it impossible to execute the Test Bench and results in an automatic grade of zero.

Signal Descriptions:

- **i_clk**: CLOCK input signal generated by the Test Bench.
- **i_rst**: RESET signal that initializes the machine, ready to receive the first START signal.
- **i_start**: START signal generated by the Test Bench.
- **i_k**: The vector signal **K** generated by the Test Bench, representing the sequence length.
- **i_add**: The vector signal **ADD** generated by the Test Bench, representing the starting address of the sequence to be processed.
- **o_done**: The DONE output signal that communicates the end of processing.

- **o_mem_addr**: The vector output signal that sends the address to the memory.

- **i_mem_data**: The vector input signal from memory, containing the data read after a read request.

- **o_mem_data**: The vector output signal that sends data to memory to be subsequently written.

- **o_mem_en**: The ENABLE signal that must be sent to memory to enable communication (both for reading and writing).

- **o_mem_we**: The WRITE ENABLE signal that must be sent to memory (‘=1‘) to allow writing. To read from memory, this signal must be ‘0‘.