Digital Logic Design
Report

Gabriele Santandrea

POLITECNICO
MILANO 1863

Anno Accademico: 2023-2024
Codice Persona: 10807746
Docente Referente: Gianluca Palermo

Contents

Introduction **2**

1 Model Architecture **5**
1.1 Counter . . . 5
1.2 Address Controller . . . 5
1.3 Sequence Controller . . . 6
1.4 Credibility Counter . . . 6
1.5 Word Manager . . . 8
1.6 Whole Module . . . 10

2 Finite State Automata **11**
2.1 First Read Path . . . 11
2.2 Write Path . . . 12
2.3 Remaining States . . . 13

3 Experimental Results **14**
3.1 Synthesis Report . . . 14
3.2 Components' Testing . . . 15
3.2.1 Counter's Test . . . 15
3.2.2 Address Controller's Test . . . 15
3.2.3 Sequence Controller's Test . . . 17
3.2.4 Credibility Counter's Test . . . 17
3.2.5 Word Manager's Test . . . 17
3.3 Complete Tests . . . 17
3.3.1 Base Testbench . . . 17
3.3.2 Multiple sequences Testbench . . . 19
3.3.3 Max Count Testbench . . . 19
3.3.4 Credibility Testbench . . . 19
3.3.5 Null Read Testbench . . . 19

Final Remarks **20**

Introduction

Project Requirements

The project consists of designing a circuit at *RT* level. The circuit interacts with a synchronous memory, processing a sequence of K *words* ($0 =< K < 1023$) evaluating their value as follows:

- if not zero, it inserts a well defined value, called *credibilità* (credibility), in the adjacent memory space;
- if zero, the circuit replaces the value at the current memory address with the one of the last *word* read and writes in the adjacent memory address the credibility value, decremented for each subsequent iteration performed in this case ($\forall t(31 > cred >= 0)$).

If ADD is the address of the first *word*, from which the module starts processing, then a possible state of the memory before and after elaboration is shown in the following tables.

ADD	ADD+1	ADD+2	ADD+3	ADD+4	ADD+5
218	0	17	0	0	31
ADD+6	ADD+7	ADD+8	ADD+9	ADD+10	ADD+11
77	0	0	0	0	0

ADD	ADD+1	ADD+2	ADD+3	ADD+4	ADD+5
218	31	17	31	17	30
ADD+6	ADD+7	ADD+8	ADD+9	ADD+10	ADD+11
77	31	77	30	77	29

Table 1: State of the memory before and after the elaboration of a general sequence

The module is responsible for signaling the end of the elaboration to the memory.

If a zero is read at the beginning of the sequence, the circuit writes zeros until a read of a positive value occurs. Afterwards, the processing proceeds as normal. This defines the edge case shown in Table 2.

ADD	ADD+1	ADD+2	ADD+3	ADD+4	ADD+5
0	31	0	2	0	0
ADD+6	ADD+7	ADD+8	ADD+9	ADD+10	ADD+11
77	0	0	12	0	0

ADD	ADD+1	ADD+2	ADD+3	ADD+4	ADD+5
0	0	0	0	0	0
ADD+6	ADD+7	ADD+8	ADD+9	ADD+10	ADD+11
77	31	77	30	77	29

Table 2: State of the memory before and after elaboration of a sequence starting with a 0

Component's Interface

The memory and the module communicate through the signals shown in Figure 1. The interface of the module is the following:

```
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);
        -- Write Enable signal: 0 read - 1 write
        o_mem_we: OUT std_logic;
        -- Enable signal
        o_mem_en: OUT std_logic
    );
END project_reti_logiche;
```

- **i_clk**: input clock signal;
- **i_rst**: input reset signal. If reset is high then the output signal **o_done** is low;
- **i_start** input start signal;
- **i_k** input sequence length signal;
- **i_add** input starting address signal[1];

[1] As a convention is called ADD

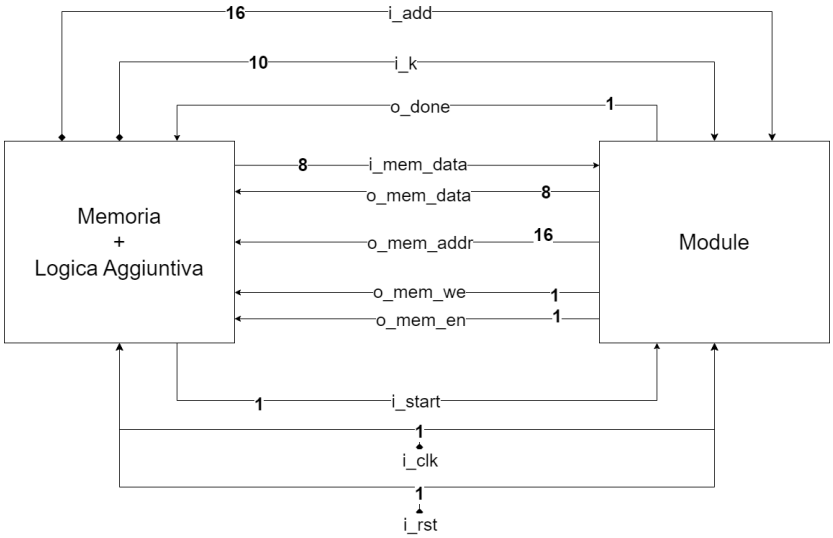


Figure 1: General schematic of the connections between the memory and the designed circuit.

- **o_done** output signal communicating the end of the elaboration;
- **o_mem_addr** output address signal to index memory;
- **i_mem_data** input signal containing the data following a read request;
- **o_mem_data** output signal containing data to write;
- **o_mem_en** *enable* signal to activate communication with the memory (in both read/write state);
- **o_mem_we** *write enable* signal: it allows write access to the memory if high (`o_mem_we = '1'`) or read access if low (`o_mem_we = '0'`).

Chapter 1

Model Architecture

1.1 Counter

The circuit has been designed using a bottom-up approach. Therefore, the first designed submodule is the *counter*. This component's inputs are constant, and thus the count is defined as an offset from the constant input signal.

It is formed by a multiplexer controlled by the signal **counter-init** whose function is to initialize the register whenever necessary (`counter_init <= '1'`); otherwise it selects the signal carrying the current value incremented by one (`counter_init <= '0'`).

The register has a reset value of `0x0` and it is controlled through an asynchronous signal **counter-load**, which allows the next count value (or zero) to be stored on the clock's rising edge.

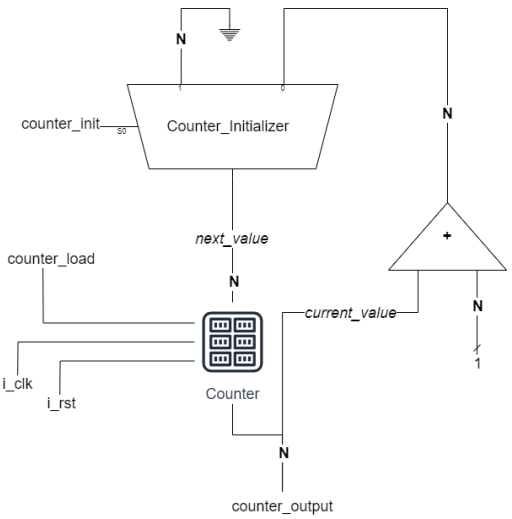
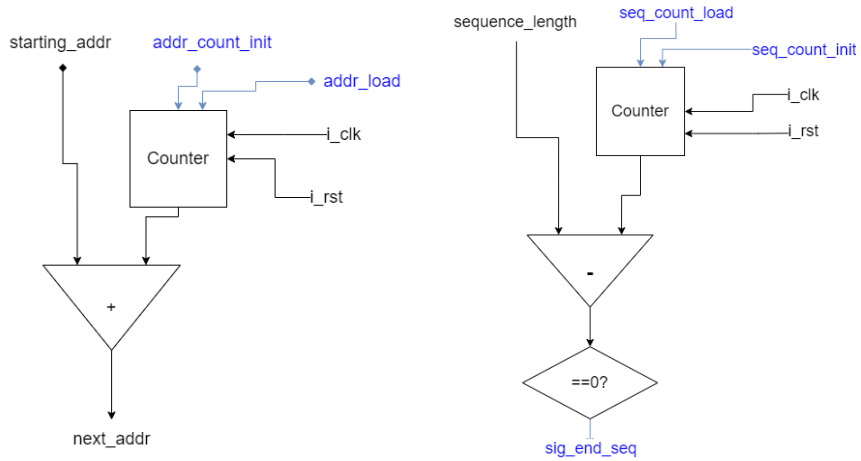


Figure 1.1: Schematic of a counter with a signal-controlled registry

1.2 Address Controller

The *address controller* (Figure 1.2a) is the component that manages address indexing.
As showing in Figure 1.5 its input (**starting-addr** => **i-mem-addr**) and output signals (**next-addr** => **o-mem-addr**) are those that manage the memory addresses.
As noted above, the counter operates as an offset from the **starting-addr**:

$next\_addr = starting\_addr + (offset\_addr + 1)$, where *offset_addr* is the counted amount.



(a) Address controller's schematic (b) Sequence controller's schematic

Figure 1.2: Controller schematics: components implementing the counter

addr-count-init and **addr-load** signals map to the counter signals. The first one is raised (`addr_count_ini <= '1'`) only in the reset state, in fact, only at the beginning of a new elaboration is it necessary to start counting again from the initial memory address. It is then lowered (`addr_count_init <= '0'`) if **addr-load** is high, i.e. when it is necessary to cycle to the next address.

1.3 Sequence Controller

The *sequence controller* (Figure 1.2b) has analogous mechanics to the address controller. It differs from the latter in that the input signal is the sequence value, i.e. the K count of words to process; the output signal generates an internal control signal.

The control signals **seq-count-load** and **seq-count-init** are connected to the counter's control signals and manage the read *word* count.

The output signal (**sig-end-seq**) notifies the module that the sequence has ended, thus performing termination steps, preparing for a new elaboration.

It may be noted that only one counter instance could have been used to handle both modules, however this would impose more complex design choices regarding the FSM, as the address offset is incremented at different instants than the sequence offset.

1.4 Credibility Counter

The *credibility counter* is the component that tracks the credibility value. The component could be implemented incorporating the counter, however, it would add complexity to the FSM's signal handling. This sub-module does not lend itself easily to implementation over a constant signal to which an offset is added.

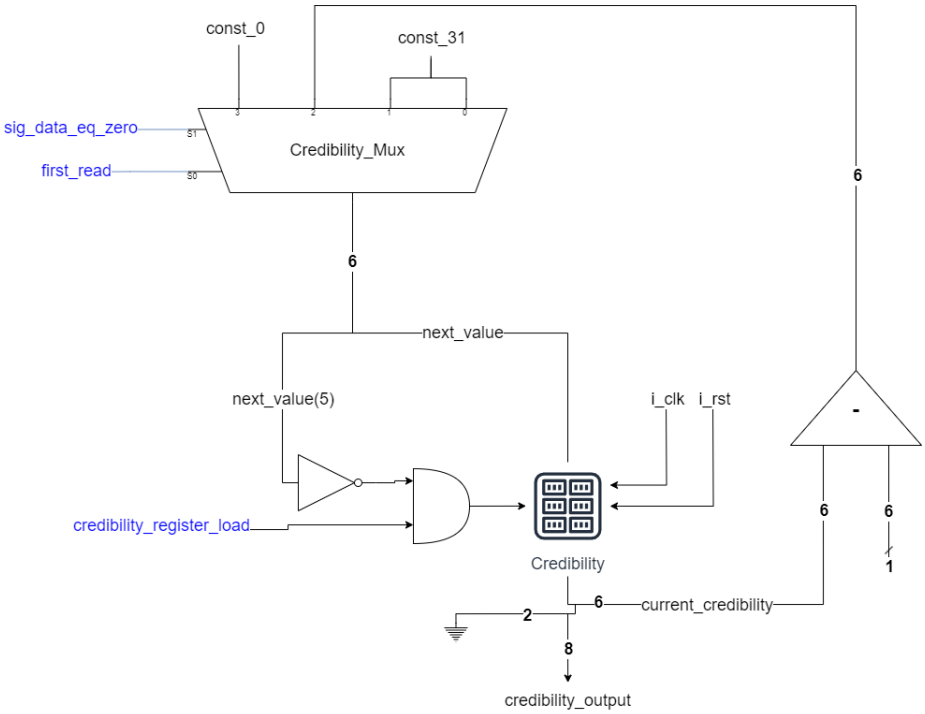


Figure 1.3: Credibility counter schematic

The component is composed of a two-bit multiplexer controlled by the signals:

- **sig-data-eq-zero** signals if the read is equal to zero;
- **first-read** signals if at least a read cycle has been completed.

The first signal selects the credibility initialization value (`sig_data_eq_zero = '0'`), `0x1f`, or the next credibility value (`sig_data_eq_zero = '1'`).
The second signal selects the initalization value (`first_read = '0'`) or the edge case starting value `0x0`.

This encodes four well-defined options, two of which are identical, as shown in Table 1.1.

sig-data-eq-zero	**first-read**	mutex selection	next value
0	0	0b00	31
0	1	0b01	31
1	0	0b10	curr_value - 1
1	1	0b11	0

Table 1.1: Multiplexer output values

The register has a reset value of `0x0` and is controlled by an *AND* gate; the output value is shown in Table 1.2.

By displaying this table, it is easier to comprehend those cases in which it is possible to write on the register. First and foremost, the write is controlled

credibility_register_load	next_value(5)	sig_load
0	-	0
1	0	1
1	1	0

Table 1.2: Credibility counter's write load signal's truth table.

by **credibility-register-load**: if low, in any case, the register will not be overwritten, otherwise it depends on the next credibility value.
As the specification requires $\forall t(31 > cred >= 0)$, considering a two's complement representation of a 6-bit number, the maximum representable value is `0x1f` corresponding to `0b011111`, conversely the minimum representable number is `0x20` corresponding to `0b100000`. Moreover the requirement becomes `if next_value >= 0`; which can be evaluated analysing the most significant bit of the **next-value** signal: if zero the credibility value is still between 31 and 0, if not its value is higher than 31 (or in two's complement representation, less then zero).

1.5 Word Manager

The last module is the *word manager*. It tracks the last read word other than zero (or zero in the edge case). The input signal is the word read at the memory address specified by the *address controller*.

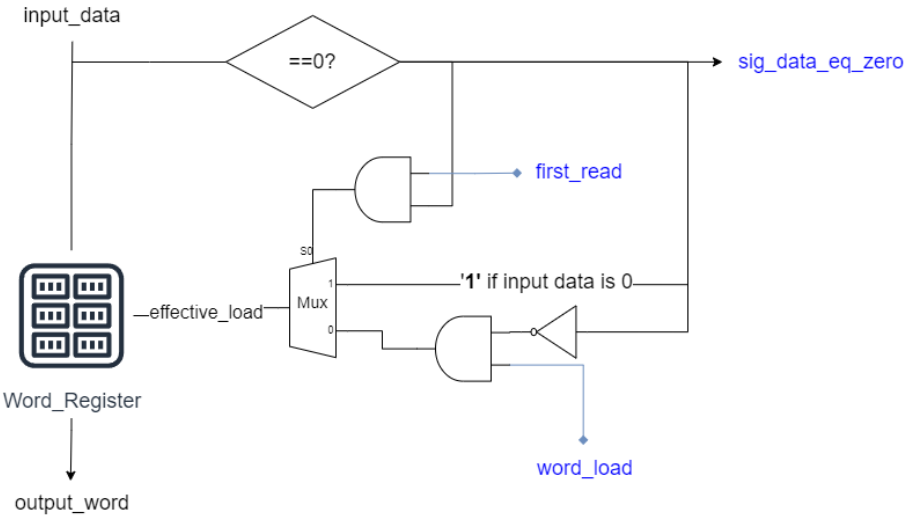


Figure 1.4: Word manager's schematic

The control logic of the word manager's register is the most complex piece of this component. In fact, except in special case, the last read word other than zero must be tracked. The register write signal is controlled by an *AND* gate which only allows writes if the input word is non-zero (`sig_data_eq_zero = '0'`) and the **word-load** signal is high.

Furthermore, to take account of the special case, it is necessary to allow writing a zero only in the first read cycle. If this is the case then both `sig_data_eq_zero = '1'` and `first_read = '1'`; this implies that the second output of the multiplexer is selected (`0b1`), which can be represented by a constant one, by construction.

1.6 Whole Module

Module Description

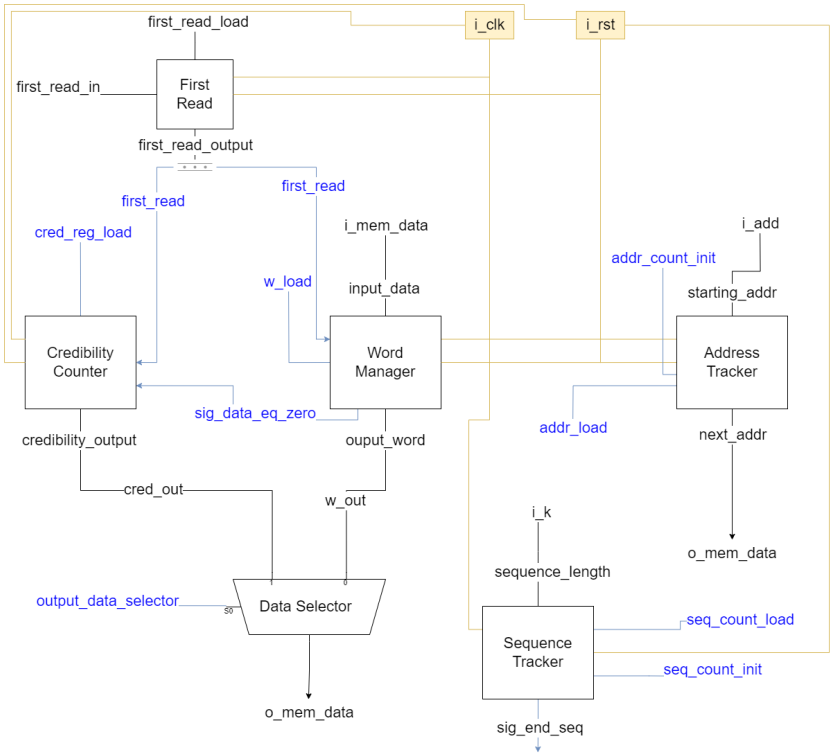


Figure 1.5: Overall schematic of the designed module, with external view of sub-components

Two other components that complete the module can also be seen in Figure 1.5:

- *first read*'s flip-flop that signals the first read cycle to the word manager and the credibility counter;
- *data selector* multiplexer that selects which signal to output.

Some Remarks

Although an attempt has always been made to maintain a modular approach to design and to keep the various components of the module independent of each other, it can be seen that at least as far as the *credibility counter* (Figure 1.3) and the *word manager* (Figure 1.4) are concerned, they are interdependent and are not really portable to other systems. This design choice was dictated by time and ease of design.[1]

[1]The two best-designed components are considered to be the two controllers.

Chapter 2

Finite State Automata

The finite state machine (Moore's) defined on this architecture consists of eight states, six of which form a cycle. The latter is the actual processing cycle consisting of two read states, two write states and two auxiliary states that allow memory scrolling. It manages the logic of all the signals within the system, but also the output signals, which are communication control signals with the *RAM*.

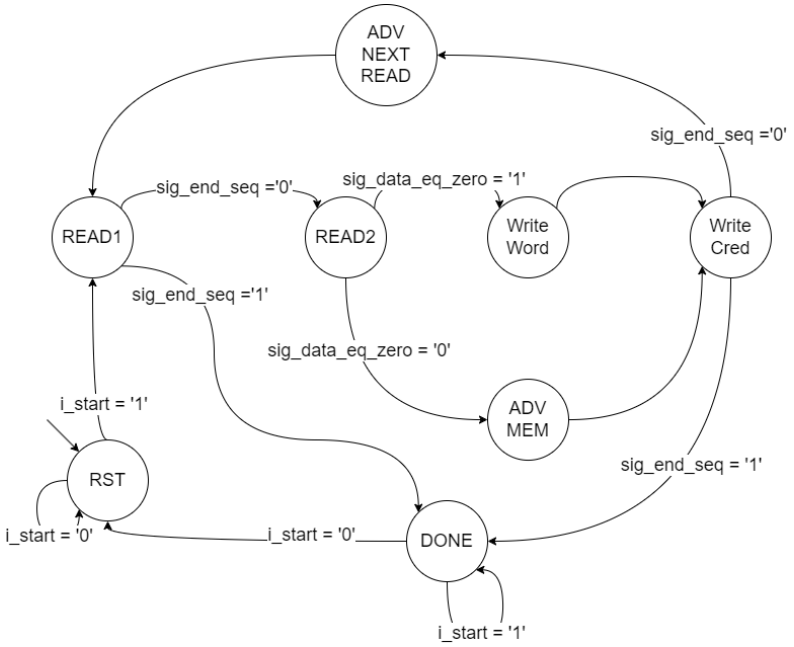


Figure 2.1: FSM's transitions scheme

2.1 First Read Path

From the reset state (*RST_S*), that lowers **o-done** and initializes all sub-components, i.e. zero the counters of the two controllers and rise the **first-read** signal, this path reaches for the first time the two read states.

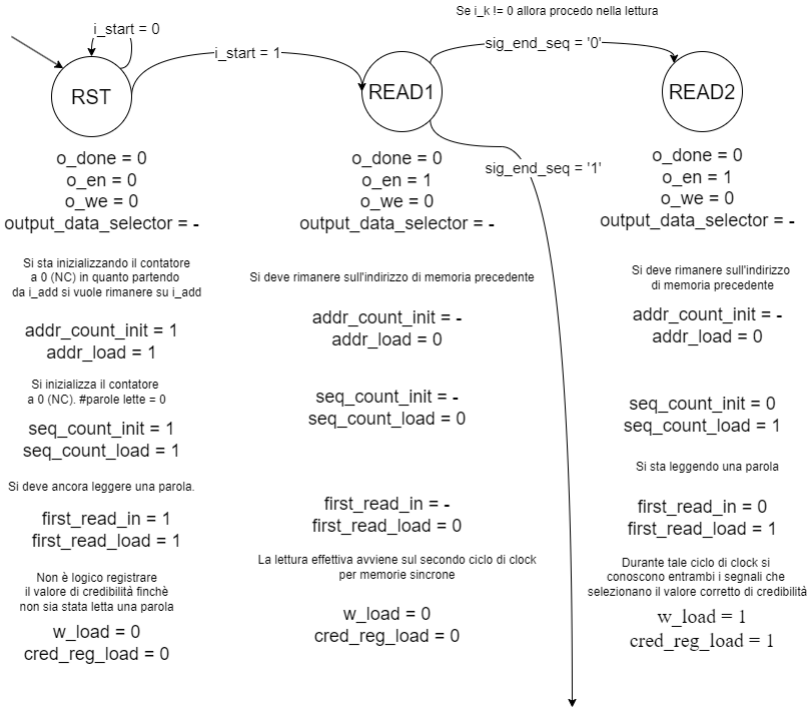


Figure 2.2: States traversed during first read

In the first state, the module requires read access to the memory, in the second state, it stores the received input, increments the sequence counter (read words + 1) and lowers the **first-read** signal.[1]

Moreover in the first read state there's already two branches: if the end sequence signal is high, then no words were to be processed; it is thus possible to terminate elaboration without effectively reading the input.

2.2 Write Path

The write path depends on the input value. If zero (`sig_data_eq_zero = '1'`) the value the current memory address must be overwritten before writing the credibility value (decremented). Alternatively, it is possible to directly write the credibility value[2].

WRITE WORD state overwrites the value in the memory with the last read word; to do so, the first output of the *output selector* is selected (`output data selector <= '0'`).

Similarly the *WRITE CRED* state selects the second output of *output selector* corresponding to the current credibility value.

If the sequence has ended (`sig_end_seq = '1'`) the next state selected is the final state *DONE*, otherwise the module readies for another read.

[1] At this state credibility counter's initialization signals are available

[2] Table 1.1 shows the credibility values

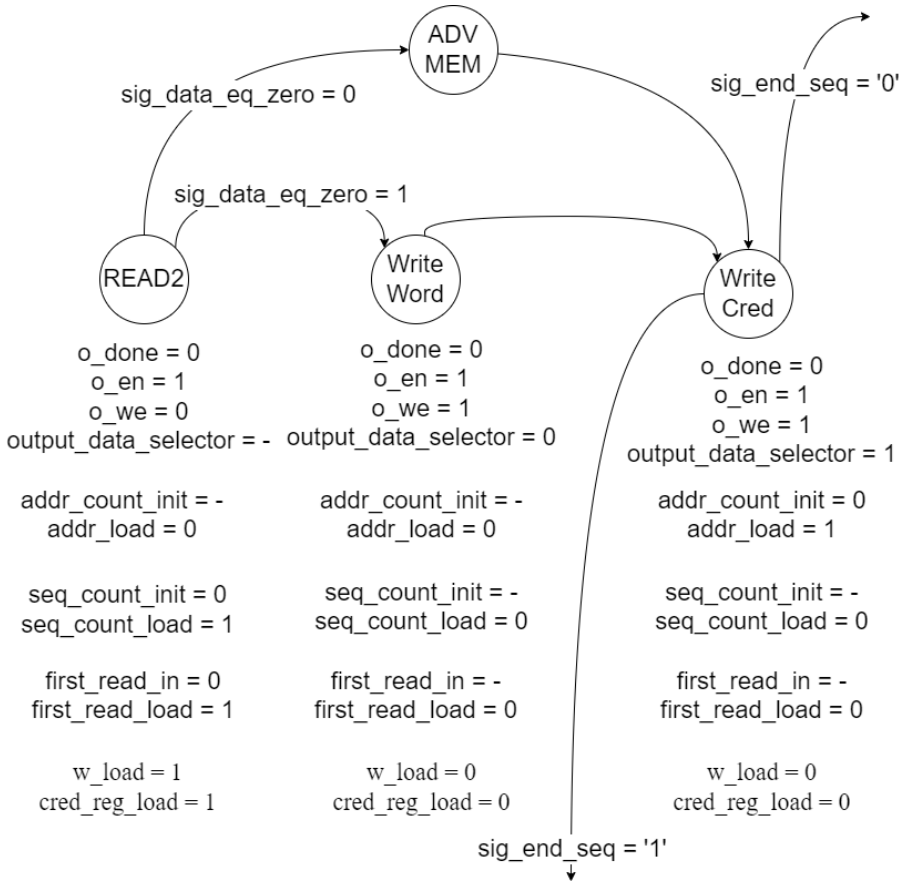


Figure 2.3: States traversed during write

2.3 Remaining States

DONE state lowers all signals and rises the **o-done** signal, until **i-start** is lowered.
ADV MEM and *ADV NEXT READ* states increment the address controller's counter, thus advancing the memory.

Chapter 3

Experimental Results

3.1 Synthesis Report

From the synthesis report, it is possible to acquire some information regarding the synthesis of the module. First of all, in Table 3.1[1] is shown the mapping of the states of the FSM in *sequential* mode.[2]

State	Encoding
rst-s	000
read-1	001
read-2	010
ww	011
advmem	100
wc	101
adv-next-read	110
done	111

Table 3.1: Transposition of states in sequential encoding

Next, all information regarding the sub-modules is reported. Specifically, information with respect to the synthesized registers is reported in Table 3.2.[3]

Registers:	10 Bit Registers := 1
	8 Bit Registers := 1
	6 Bit Registers := 1
	1 Bit Registers := 1

Table 3.2: Table of synthesized registers

By comparing the schematic on the FPGA and using the in-depth report, the mapping of the registers to the described sub-modules has been made. Indeed,

[1] See Figure 3.2

[2] Mode's name directly copied from Vivado's report. Another name for such encoding is *gray coding*.

[3] See Figure 3.3

there is one register missing: the 16-bit counter of the address controller[4] has been replaced with a group of LUTs that implement the count.

The 1-bit register is the **first-read** flip-flop, the 8-bit register is the one in the word manager and lastly, the 6-bit register is the credibility counter register.

These registers are all implemented through flip-flops, as shown in Table 3.3.[5]

Site Type	Used	Util%
Slice Registers	44	0.05
Registers as Flip Flop	44	0.05
Registers as Latch	0	0.00

Table 3.3: Table of register types

Finally, as shown in Figure 3.1, the module operates within the imposed 20 ns clock limit .

```
Timing Report

Slack (MET) :             17.368ns  (required time - arrival time)
  Source:                 SeqTrack/SeqCount/current_value_reg[1]/C
                            (rising edge-triggered cell FDCE clocked by clock  {rise@
  Destination:            FSM_sequential_current_state_reg[0]/D
                            (rising edge-triggered cell FDCE clocked by clock  {rise@
  Path Group:             clock
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            20.000ns  (clock rise@20.000ns - clock rise@0.000ns)
  Data Path Delay:        2.489ns  (logic 1.174ns (47.168%)  route 1.315ns (52.832%))
  Logic Levels:           5  (CARRY4=2 LUT2=1 LUT5=1 LUT6=1)
```

Figure 3.1: Slack's report

3.2 Components' Testing

The drawing and design of the components were followed by implementation (with schematic control) and both logic-behavioural and post-synthesis testing.

3.2.1 Counter's Test

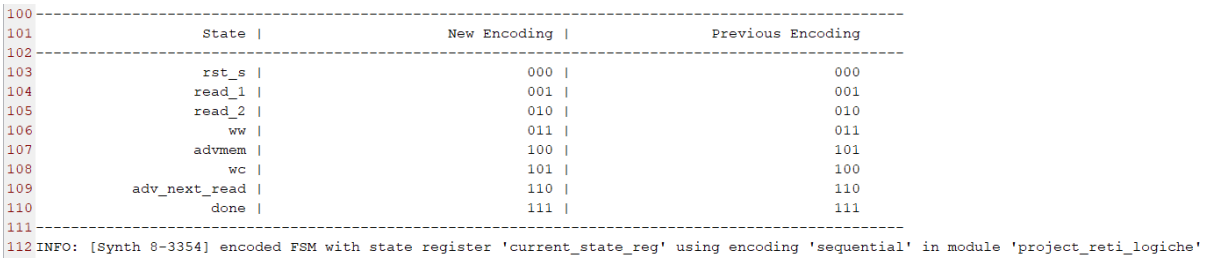
The counter test is a small test that simply checks that register is correctly incremented once the correct multiplexer input has been chosen. This test was immediately positive and laid the foundation for all others.

3.2.2 Address Controller's Test

Immediately after the counter, the address controller was tested. It was checked that it ran over the memory in the correct manner and that the module therefore fulfilled its requirements.

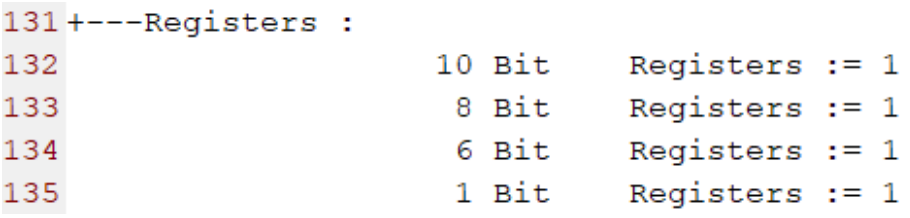
Alongside the testing of the actual module, a write test was also carried out to assess how the defined module and memory interacted with respect to

[4]See Figure 1.2a
[5]See Figure 3.4



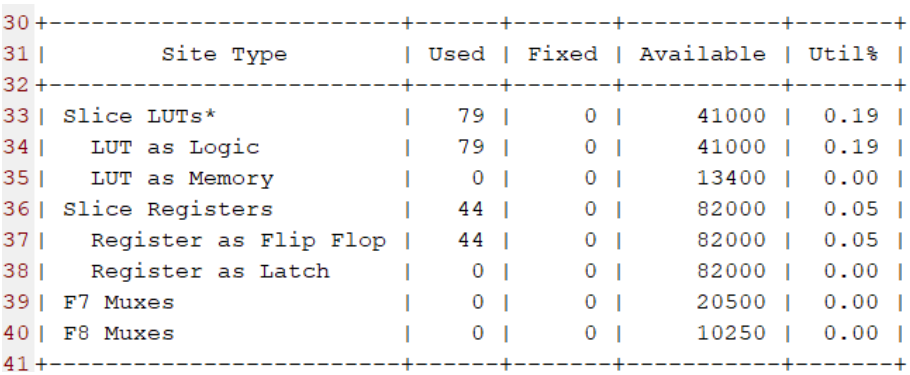
```
---------------------------------------------------------------------------------------------------
               State |                     New Encoding |                Previous Encoding
---------------------------------------------------------------------------------------------------
               rst_s |                              000 |                              000
              read_1 |                              001 |                              001
              read_2 |                              010 |                              010
                  ww |                              011 |                              011
              advmem |                              100 |                              101
                  wc |                              101 |                              100
      adv_next_read |                              110 |                              110
                done |                              111 |                              111
---------------------------------------------------------------------------------------------------
INFO: [Synth 8-3354] encoded FSM with state register 'current_state_reg' using encoding 'sequential' in module 'project_reti_logiche'
```

Figure 3.2: State encoding table



```
+---Registers :
                   10 Bit    Registers := 1
                    8 Bit    Registers := 1
                    6 Bit    Registers := 1
                    1 Bit    Registers := 1
```

Figure 3.3: Table of synthesized registers



```
+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs*             |   79 |     0 |     41000 |  0.19 |
|   LUT as Logic          |   79 |     0 |     41000 |  0.19 |
|   LUT as Memory         |    0 |     0 |     13400 |  0.00 |
| Slice Registers         |   44 |     0 |     82000 |  0.05 |
|   Register as Flip Flop |   44 |     0 |     82000 |  0.05 |
|   Register as Latch     |    0 |     0 |     82000 |  0.00 |
| F7 Muxes                |    0 |     0 |     20500 |  0.00 |
| F8 Muxes                |    0 |     0 |     10250 |  0.00 |
+-------------------------+------+-------+-----------+-------+
```

Figure 3.4: Slice logic table

the given interface. The wrong conclusions were first drawn: it was assumed, from a reading of the provided testbench, that the memory address change was faster than the write, and that therefore the value would actually be written in the next clock cycle and to the memory address calculated later. This error of judgment was later corrected in the final stage of completion of the module, as it was noted that the waste could not actually be used in this way.

3.2.3 Sequence Controller's Test

In order to properly control the sequence controller, the first FSMs were described, so that the correct decrement and evaluation of the end of the sequence could be checked.

Through this test, the actual behaviour of the flip-flops was started to be noted and how this would impact the operation of the module.

3.2.4 Credibility Counter's Test

The credibility controller was the most complex component to design. This was due to the implementation by means of a multiplexer controlled by two internal control signals.

During the first write-up of the test, before evaluating the results, it was noted that the desired behaviour could not be met by the first version of the component designed[6]. A re-design of the component had then to be carried out, which produced the sub-module described[7].

Of this component, it was tested that all multiplexer inputs functioned as desired and that the register writing was consistent with what was required.

3.2.5 Word Manager's Test

The word manager was not tested in a stand-alone, customized testbench. However, its correct functioning was evaluated both by checking the schematic, which mirrored the desired circuit, and by testbenches made later for the complete module.

3.3 Complete Tests

Once the complete module was composed, it was tested in its entirety, starting with the provided testbench, which immediately showed the evaluation error made during the testing of the address controller.

3.3.1 Base Testbench

The testbench provided revealed that writing to memory compared to changing addresses is faster, and therefore it was necessary to move the memory advance action to a different state. However, this result also showed how it was possible to write to the current address and move to the next memory address within the same state: since the counter needs to wait for the rising edge to

[6] See Figure 3.5
[7] See Figure 1.3

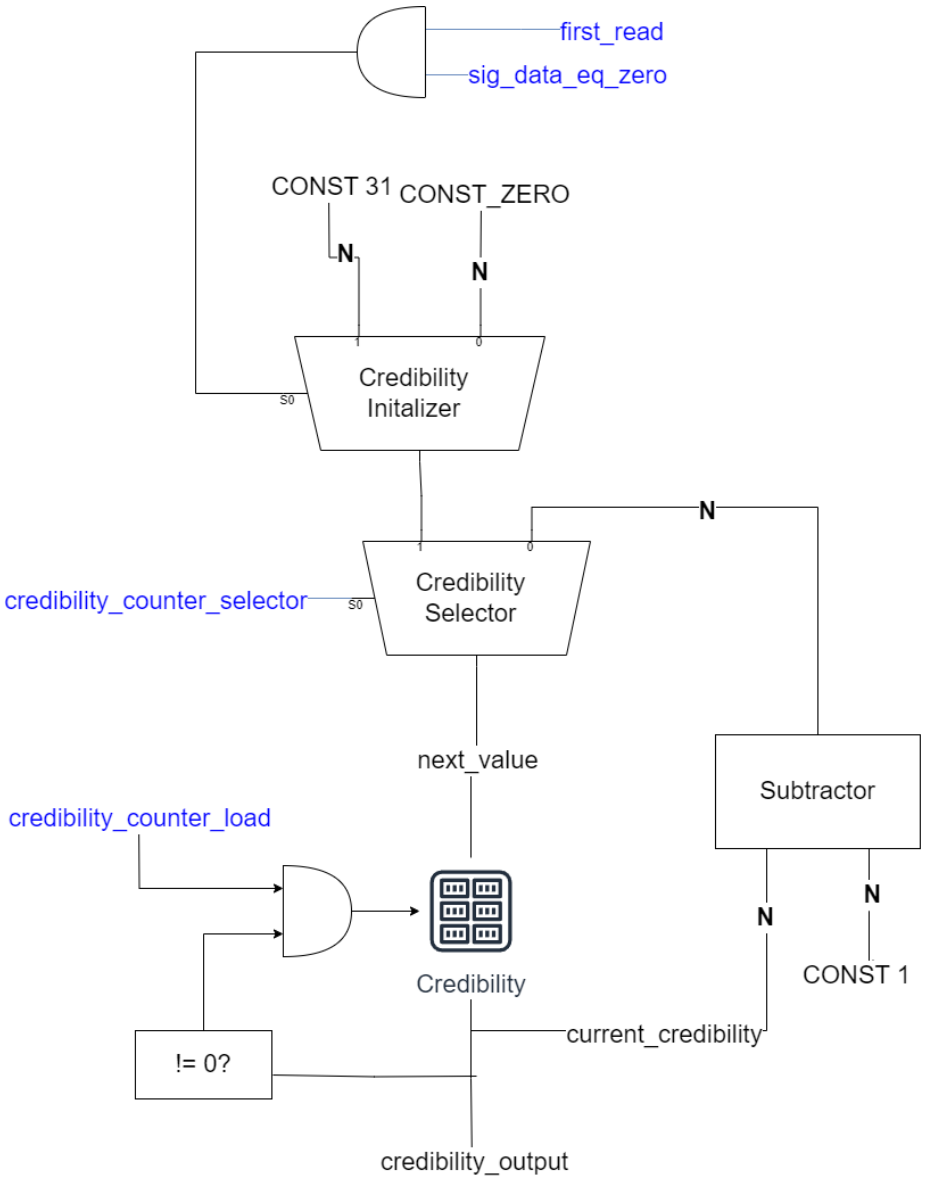


Figure 3.5: Credibility counter: first design

register an increment, then preparing the counter write in the word write state (`addr_count_init <= '0'` and `addr_load <= '1'`), it was found possible to write to the next state in the memory space of the credibility value.

3.3.2 Multiple sequences Testbench

This testbench tests two consecutive sequences. It verifies that the signal **o-done** is correctly lowered and that the module correctly prepares for a new processing.

It also tests the special case of reading a series of zeros at the start of processing:

```
constant SCENARIO_LENGTH : integer := 14;
type scenario_type is array (0 to SCENARIO_LENGTH*2-1) of integer;

signal scenario_input : scenario_type := (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);

constant SCENARIO2_LENGTH : integer := 15;
type scenario2_type is array (0 to SCENARIO2_LENGTH*2-1) of integer;
signal scenario2_input : scenario2_type := (0, 0, 0, 0, 64, 0, 0, 0, 0, 0,
                                            0, 0, 0, 0, 0, 0, 150, 0, 1, 0,
                                            0, 0, 5, 0, 23, 0, 0, 0, 0, 0);
```

A variation tests a reset signal in between the two sequences.

3.3.3 Max Count Testbench

This test evaluates the maximum number of words that can be counted, i.e. 1023 (**i-k** is a 10-bit signal).

3.3.4 Credibility Testbench

A test was defined to check whether the form limits credibility to zero.

3.3.5 Null Read Testbench

The testbench imposes a number of words to be processed equal to zero, at the beginning of the sequence but also in a sequence of processing. It serves to verify that the component does not perform a write if the affected signal is zero.

Final Remarks

The design of the module described above moved in successive steps according to a bottom-up procedure, first identifying the behaviours required to meet the requirements and then proceeding with the design of the components.

Finally, the design of the FSM and the testbenches made on each component brought to light some of their design flaws. In particular, it was a challenge to design the credibility counter without using an independent process or signal, as all the processing complexity is reduced to this component.

A negative aspect was the abandonment of a modular design philosophy for the sake of simplicity of construction.