* **NAND INSTRUCTION:**

**EXPLANATION**:

* + - Instruction encoding: In this example we are going to employ a reserved opcode / field code:
      1. Specifically, we are going to use the *SPECIAL* opcode / function field *001110*, which is an opcode / function field that is reserved for future use (marked with a “\*” in Table A.3 of document “Architecture for Programmers, Volume II-A”).
      2. Being reserved, this operation / field code will “cause a *Reserved Instruction Exception*”, which we must inhibit:
         1. signal ***ri\_e*** is assigned in m14k\_mpc\_dec, from the OR function of several signals.
         2. Among those signals, one is for *SPECIAL* instruction Reserved Exceptions: signal ***spec\_ri\_e***. This signal is 1 for all Reserved Function Fields in the SPECIAL Opcode. We just need to comment line *(****mpc\_ir\_e[5:0] == 6'o16****)* when computing that signal, inhibiting a Reserved Exception for that opcode / field code.
    - Inclusion of the functionality for the new instruction in the processor. Figure 1 shows the original Logic Unit, while Figure 2 shows the modified Logic Unit after including support for the new instruction. All changes related to the new instruction are tagged with comment: *// NAND* in the soft-core. We must perform the following actions:
      1. We must include support for the new instruction in the following two signals:
         1. ***sel\_logic\_e***:

This signal controls registration of the output of logic operations from Stage-E to Stage-M.

This signal is assigned in module m14k\_mpc\_dec. We just include the opcode / field code of the new instruction: *(mpc\_ir\_e[5:0] == 6'b001\_110)*.

* + - * 1. ***alu\_sel\_e***:

This signal must be 1 for Arithmetic-Logic operations, for selecting, at Stage-M, the output of the ALU.

This signal is assigned in module mpc\_dec. Like in the previous signal, we just include the opcode / field code of the new instruction: *special\_e && (mpc\_ir\_e[5:0] == 6'o16)*.

* + - * 1. What about the RF write at W-Stage? For Special Instructions (note that NOP is also an special instruction), only for some of them the RF write is not enabled (line 1250 in module “m14k\_mpc\_dec”). For the remaining Special instructions, it´s always enabled. So for the new instruction we just do nothing.
      1. We must include a new Functional Unit for calculating the Logic NAND. This is simply achieved by including the following line in the m14k\_edp module:

*assign a\_nand\_b\_e [31:0] = ~a\_and\_b\_e; // NAND*

* + - 1. We must include a new signal (***nand\_instruction*** is 1 if we have the new instruction and 0 if we have another instruction) for selecting the nand operation at the output of the Logic Unit.
         1. ***nand\_instruction***:

*assign nand\_instruction = special\_e & (mpc\_ir\_e[5:0] == 6'o16); // NAND*

* + - * 1. The following MUX has been modified:

*mvp\_mux4 #(32) \_logic\_out\_e\_partial\_31\_0\_(*

*logic\_out\_e\_partial[31:0], // Output*

*mpc\_alufunc\_e, // Control Signal*

*a\_and\_b\_e, // Input-1*

*a\_or\_b\_e, // Input-2*

*a\_xor\_b\_e, // Input-3*

*a\_nor\_b\_e); // Input-4*

* + - * 1. The following MUX has been included:

*mvp\_mux2 #(32) \_logic\_out\_e\_31\_0\_(*

*logic\_out\_e[31:0], // Output*

*nand\_instruction, // Control Signal*

*logic\_out\_e\_partial, // Input-1*

*a\_nand\_b\_e); // Input-2*

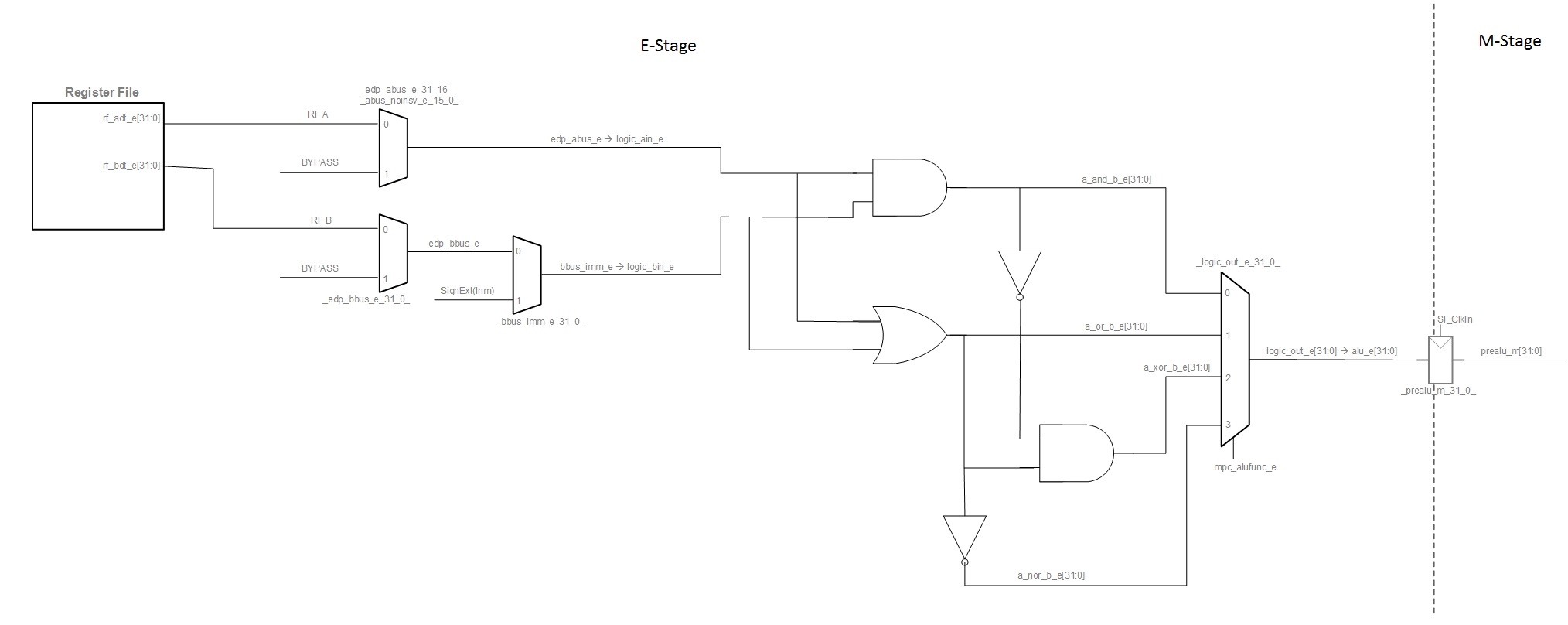


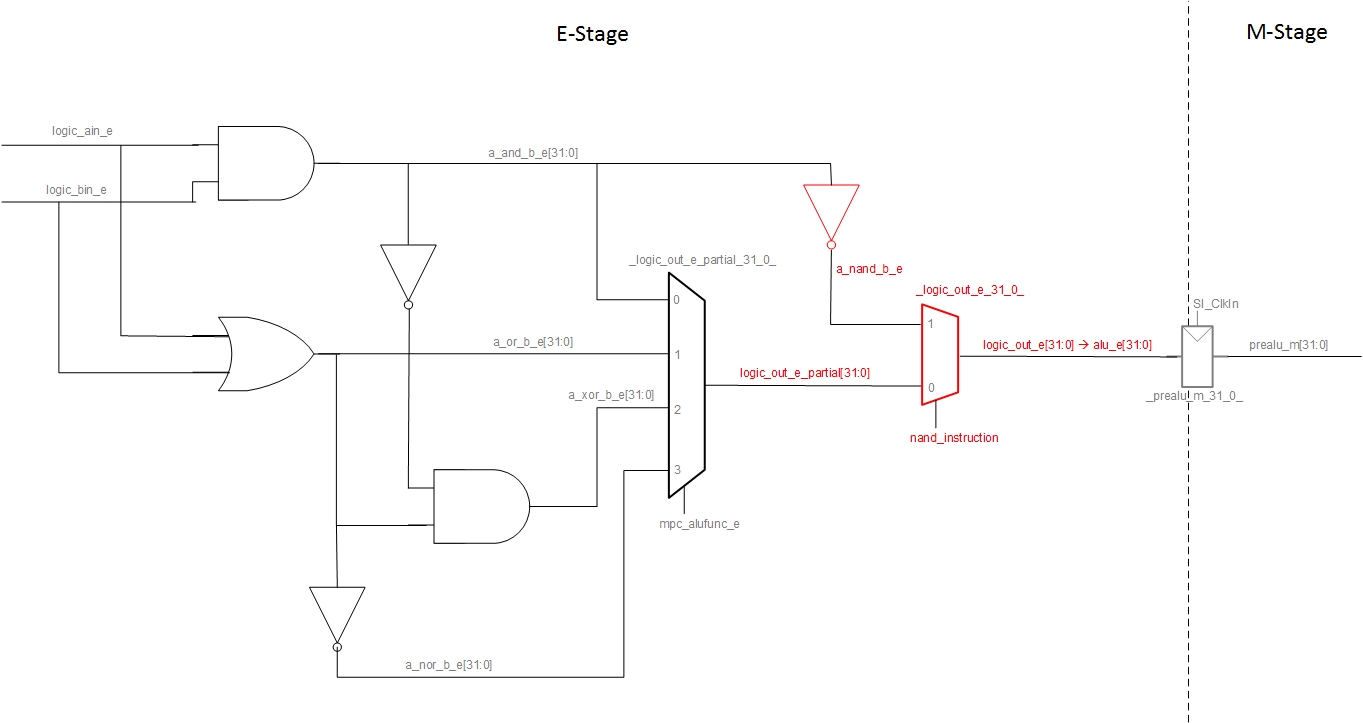
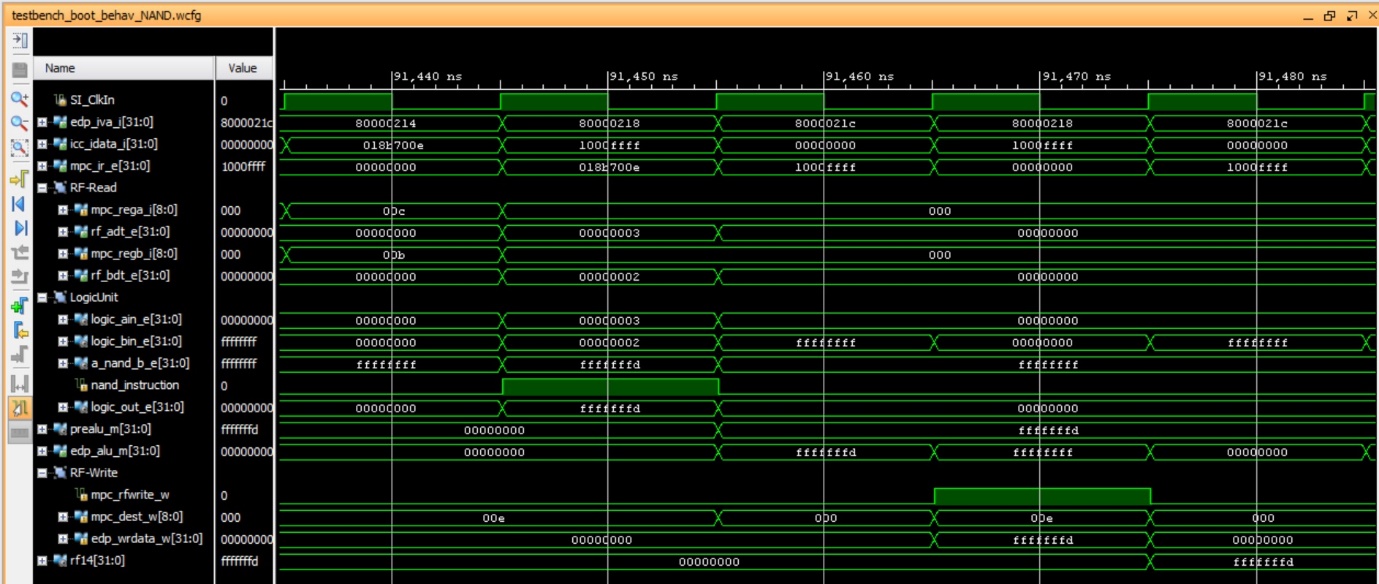
FIGURE 1. Original Logic Unit.

FIGURE 2. Modified Logic Unit. The new Hw and the new Signals are highlighted in red. Note that we have renamed some signals or muxes (also highlighted).

**EXAMPLE - SIMULATION**:



Observe that, in the 5th cycle, the destination register (rf14=$t6) is written with the result of the nand operation (0xfffffffd).

**EXAMPLE - IMPLEMENTATION IN THE FPGA**:

When the program is downloaded on the board, you should see on the 7-seg displays:

* 7-seg displays=$t6, which in our example is 0xfffffffd

Then, when you debug the program following the steps stated in the document, you should observe the following:

