**EXERCISE 1:**

**FORWARDING CONTROL SIGNALS**:

We´ll do the analysis of the control signals for Source A (control signals for Source B are completely analogous):

1. Signal ***mpc\_aselres\_e*** controls one of the multiplexers shown in Figure 6. It decides if Source A comes from the RF (***mpc\_aselres\_e***=0) or Forwarding (***mpc\_aselres\_e***=1). This signal is computed as the OR of two signals:

assign mpc\_aselres\_e = (apsall\_e | apswr\_e) ^ cpz\_rslip\_e

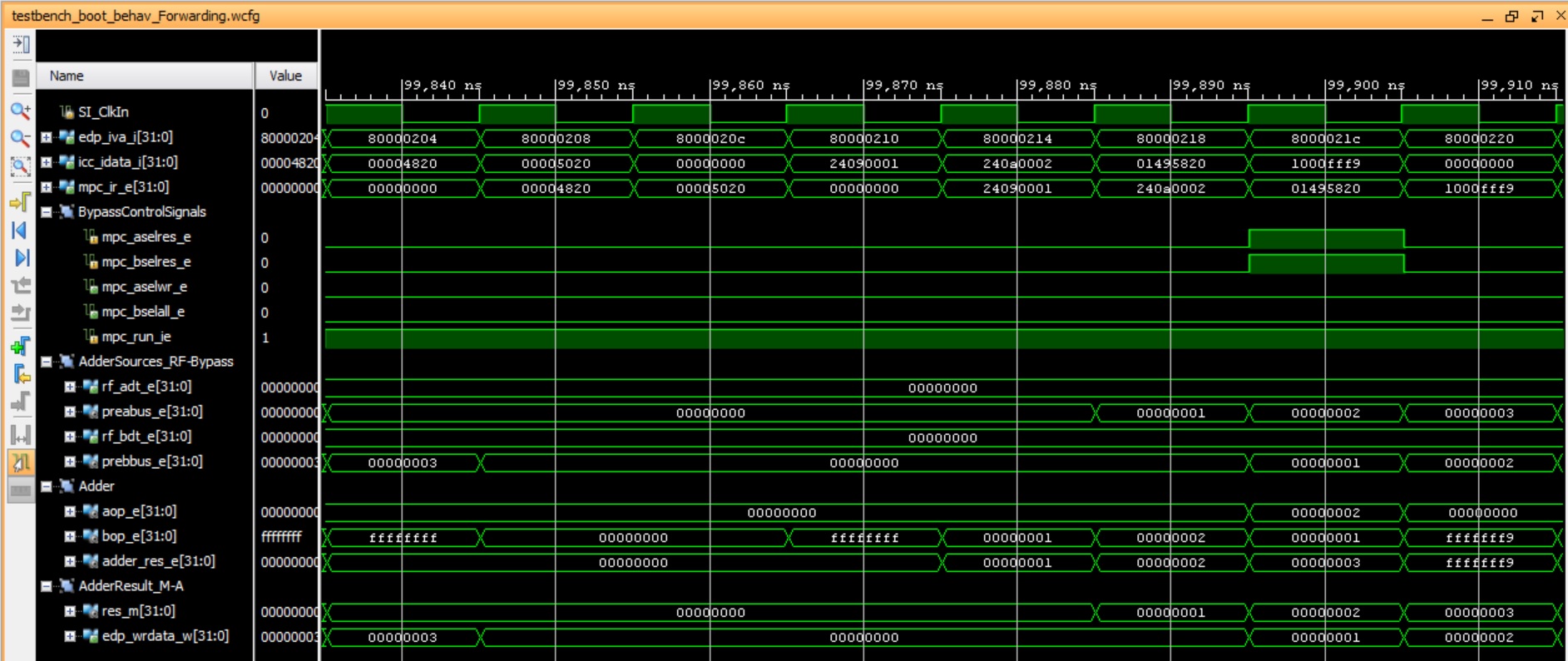
* 1. ***apsall\_e***=1 when data must be forwarded from the M-Stage. Computation of this signal is quite complex, but we can simplify it to the AND operation of 2 conditions:
     1. The instruction at the M-Stage will write to the RF (***vd\_m***=1): This signal comes from the Decoding of the instruction at the E-Stage, where ***vd\_e*** is computed.
     2. The destination register of the instruction at the M-Stage is the same as Source Operand A of the instruction at the E-Stage (***src\_a\_e*** == ***dest\_m[4:0]***).
  2. ***apswr\_e***==1 when data must be forwarded from the A-Stage. Again, computation of this signal is quite complex, but we can simplify it to the AND operation of 2 conditions:
     1. The instruction at the A-Stage will write to the RF (***vd\_w***=1).
     2. The destination register of the instruction at the A-Stage is the same as Source Operand A of the instruction at the E-Stage (***src\_a\_e*** == ***mpc\_dest\_w[4:0]***).

1. Signal ***mpc\_aselwr\_e*** controls the other multiplexers shown in Figure 6 related with Source A. It decides if the forwarding data must come from the M-Stage (***mpc\_aselwr\_e***=0) or from the A-Stage (***mpc\_aselwr\_e***=1). Simplifying, this is computed as follows:

assign mpc\_aselwr\_e = apswr\_e & ~apsall\_e …

Signals ***apswr\_e*** and ***apsall\_e*** are explained in the previous ítem. Note that the forwarding from the M-Stage has a higher priority than the forwarding from the A-Stage.

**SIMULATION**:



In the seventh cycle the add $t3, **$t2**, **$t1** instruction is executed. The first operand ($t2) is provided through forwarding (*mpc\_aselres\_a*=1) from the M-Stage (*mpc\_aselwr\_e*=0), and the second operand ($t1) is also provided through forwarding (*mpc\_bselres\_e*=1) from the A-Stage (*mpc\_bselall\_e*=0).

**EXERCISE 2:**

* + - **Bubble insertion**:
      1. When instruction (i-1) is a lw with destination register Rx, and instruction (*i*) is an Arithmetic-Logic instruction with source register Rx, one cycle must be inserted between both instructions.
      2. For creating a bubble at Stage-E, ***mpc\_run\_ie***=0. This signal is assigned from many signals and logic (***(prod\_cons\_stall\_req*** *&* ***ldst\_m) → load\_to\_use\_stall\_e*** → ***prod\_cons\_stall\_e*** → ***bubble\_e*** → ***bstall\_ie*** → ***stall\_ie*** → ***mpc\_run\_ie***). Simplifying, it is 0 when (***prod\_cons\_stall\_req*** *&* ***ldst\_m***) is 1.
         1. ***prod\_cons\_stall\_req***:It is set when there is a RAW dependency between the instruction at Stage-M and the instruction at Stage-E.
         2. ***ldst\_m***:It is set when the instruction at Stage-M is a load or a store.
      3. Signal ***mpc\_run\_ie***=0 has many effects, such as the following:
  + The pipeline registers from E-Stage to M-Stage (*\_ie\_pipe\_out\_50\_0\_*) do not change:

mvp\_cregister\_wide #(51) \_ie\_pipe\_out\_50\_0\_(ie\_pipe\_out[`M14K\_IE\_RANGE],gscanenable, mpc\_run\_ie, gclk, ie\_pipe\_in);

* + The IR (***mpc\_ir\_e*** ← ***int\_ir\_e***) is not updated:

mvp\_cregister\_wide #(32) \_int\_ir\_e\_31\_0\_(int\_ir\_e[31:0],gscanenable, (mpc\_run\_ie | mpc\_fixupi) & ~mpc\_atomic\_e

& ~mpc\_lsdc1\_e, gclk, icc\_idata\_i);

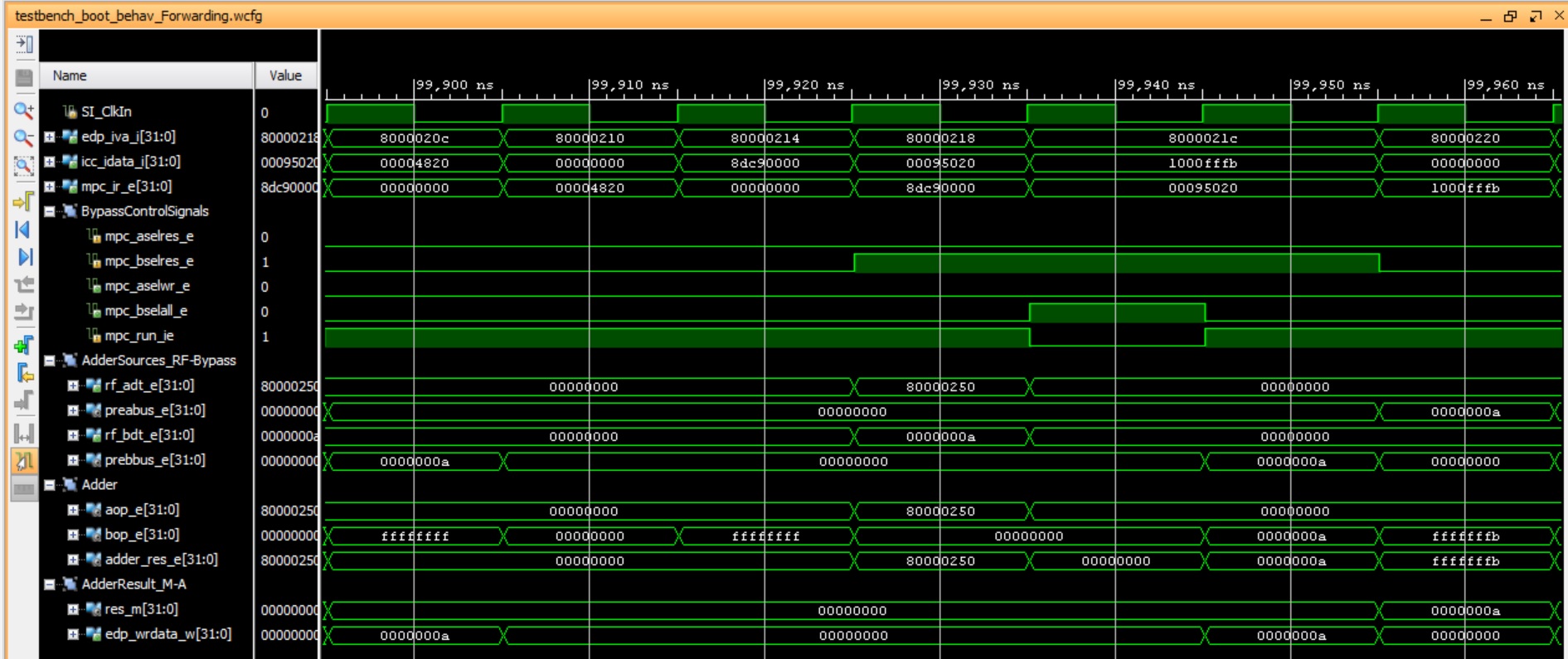
…

assign mpc\_ir\_e[31:0] = sel\_hw\_e ? hw\_ir\_e : int\_ir\_e;

* + The PC at Stage-I (***edp\_iva\_i***) is not updated:

mvp\_cregister\_wide #(32) \_edp\_iva\_i\_31\_0\_(edp\_iva\_i[31:0],gscanenable, (!icc\_umipsmode\_i & mpc\_run\_ie) ||( icc\_umipsmode\_i & !(icc\_imiss\_i & ~(mpc\_hw\_ls\_i | mpc\_chain\_hold)) & (mpc\_run\_ie | mpc\_fixupi)), gclk, edp\_iva\_p\_exit);

**SIMULATION**:



In the fifth cycle a dependency is detected between the lw instruction at the M-Stage and the second operand of the add instruction at the E-Stage. Given that the result of the lw instruction is not yet available, the add instruction is stalled at the E-Stage.

In the sixth cycle, the second operand of the add instruction is provided through forwarding (*mpc\_bselres\_e*=1) from the A-Stage (*mpc\_bselall\_e*=0).