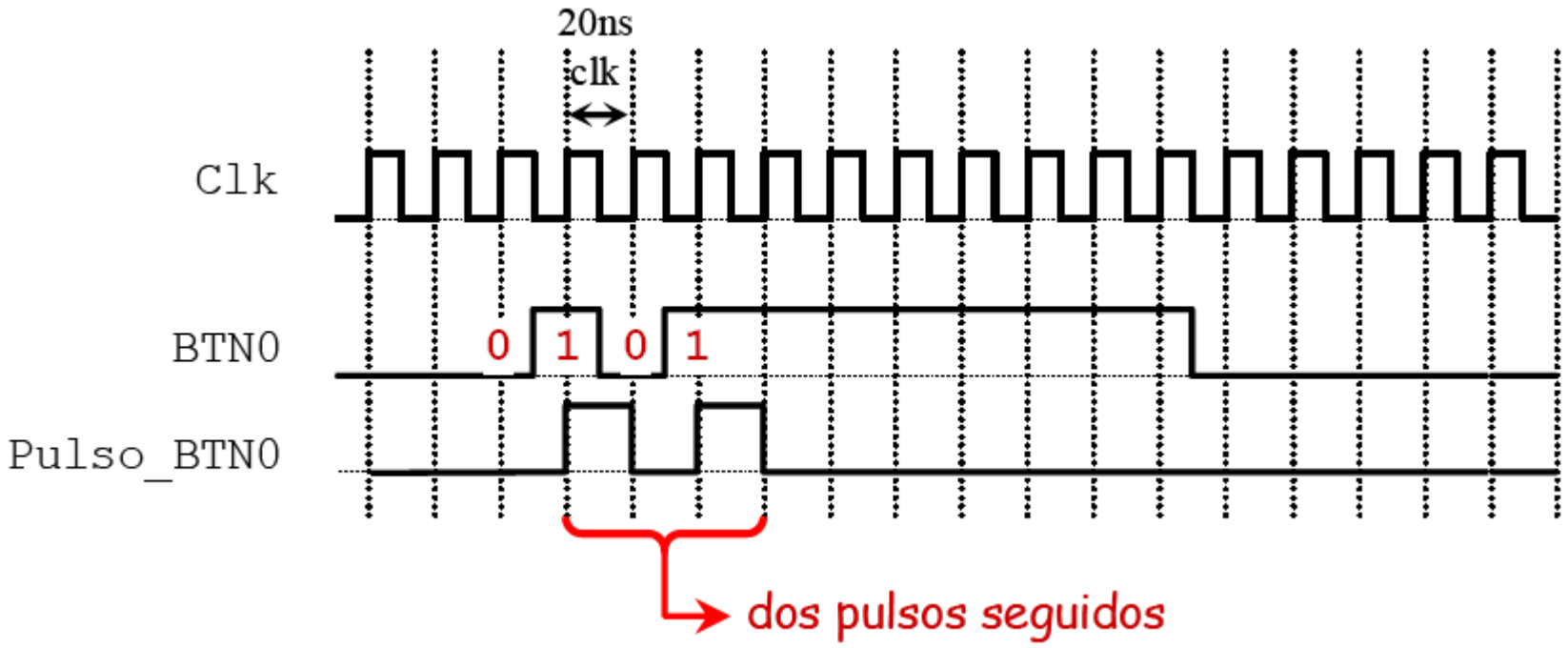
Práctica5. Diseño de máquinas de estado en VHDL

5.1. Circuito antirrebotes

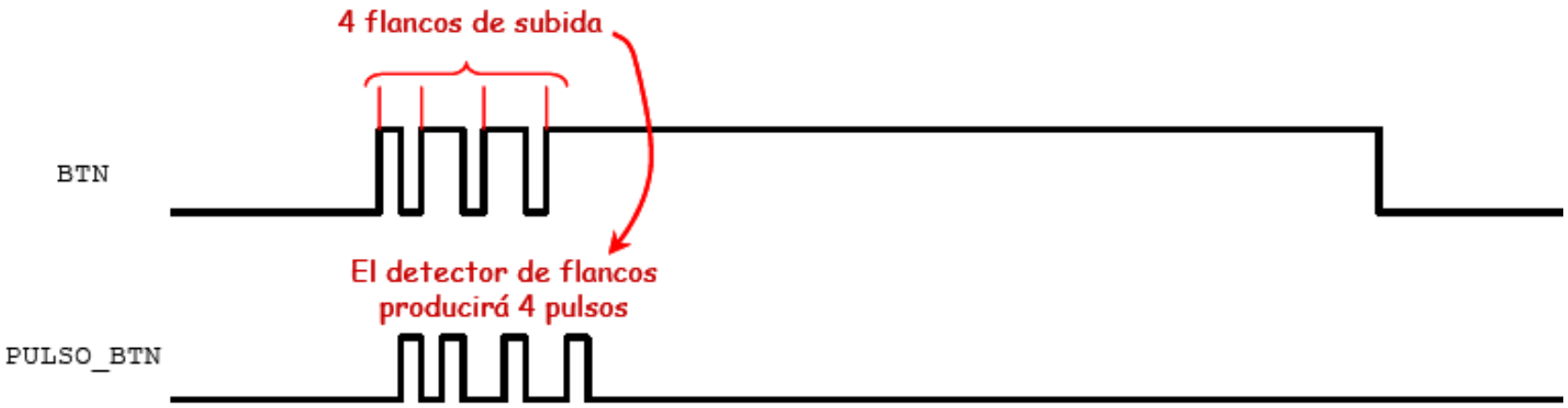
Circuito antirrebotes

El detector de flanco envía un solo pulso por cada secuencia ‘01’, pero puede detectar otra ‘01’ de forma inmediata por culpa de la mecánica del pulsador. (Imposible dos pulsaciones humanas tan rápidas.)



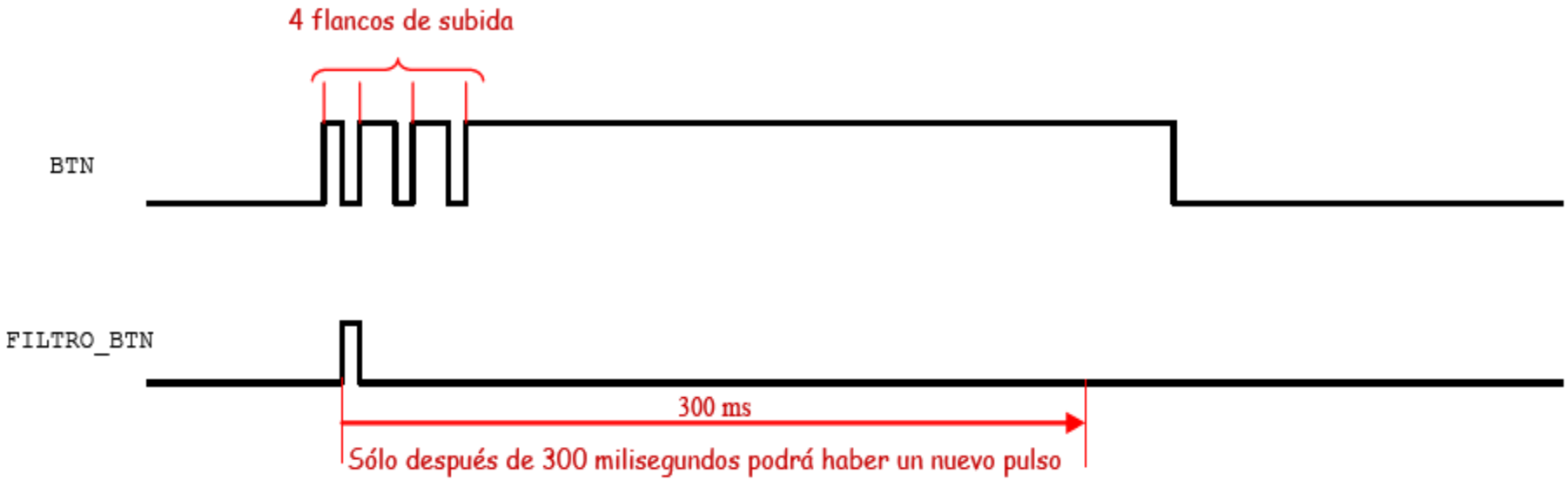
Circuito antirrebotes

El detector de flancos no es suficiente, vemos cuatro secuencias “01” seguidas.

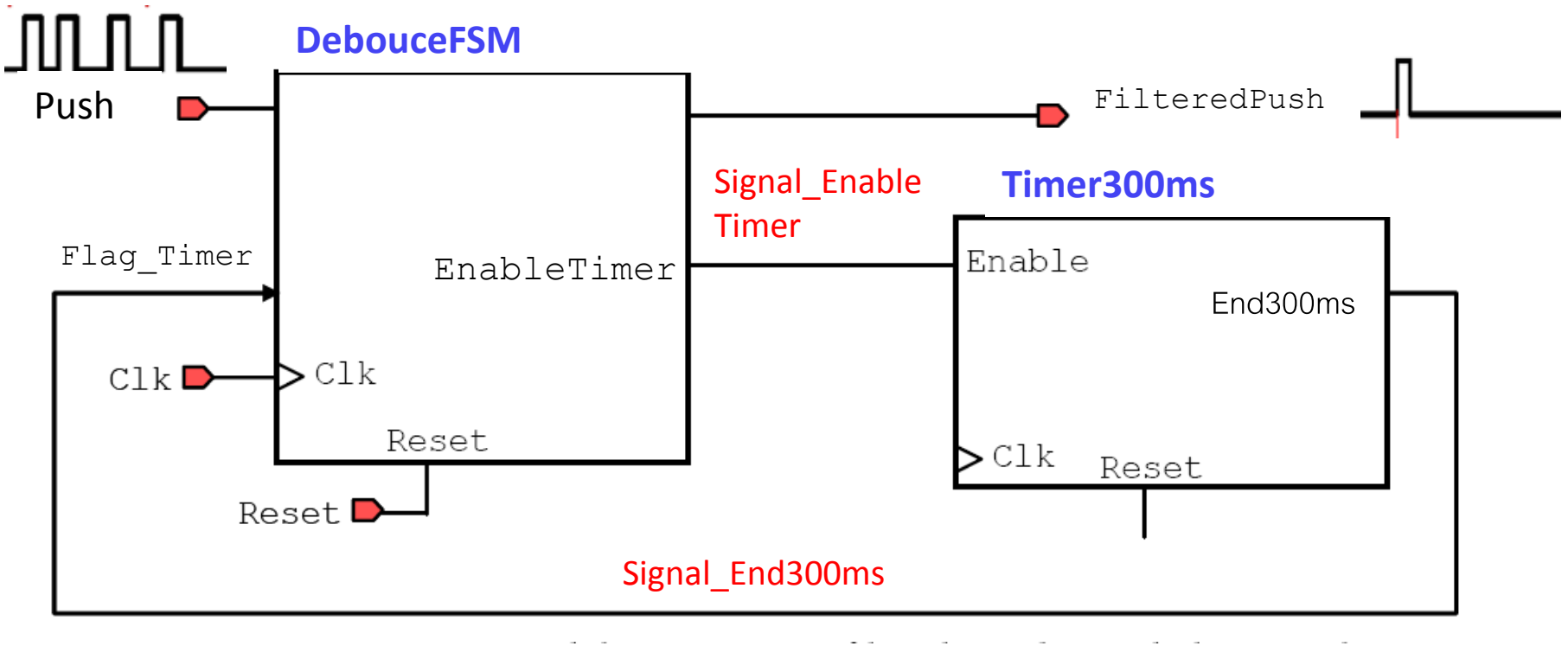


Circuito antirrebotes

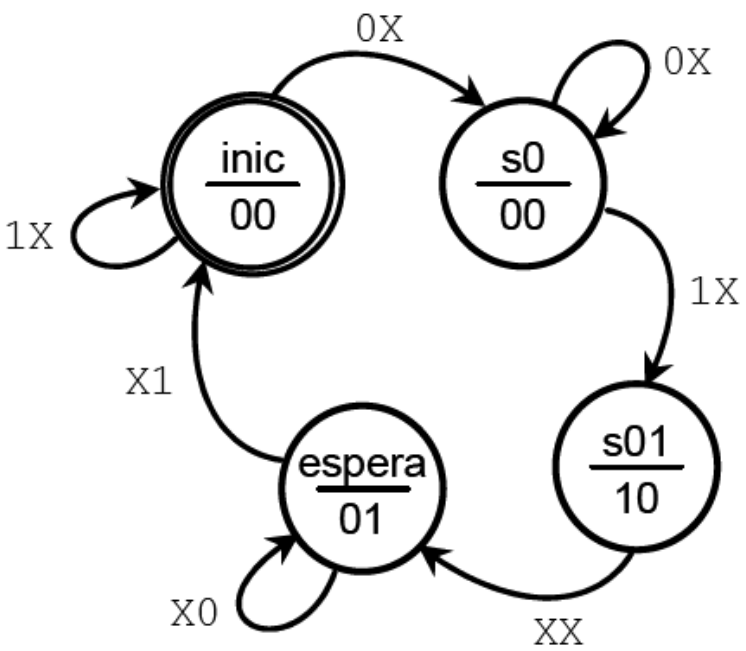
Solución: esperar un tiempo suficiente despues de detectar la secuencia “01” antes de aceptar otra.

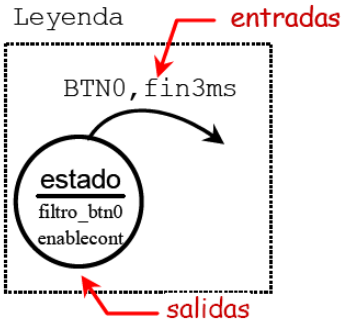


Proyecto 30 : Circuito antirrebotes



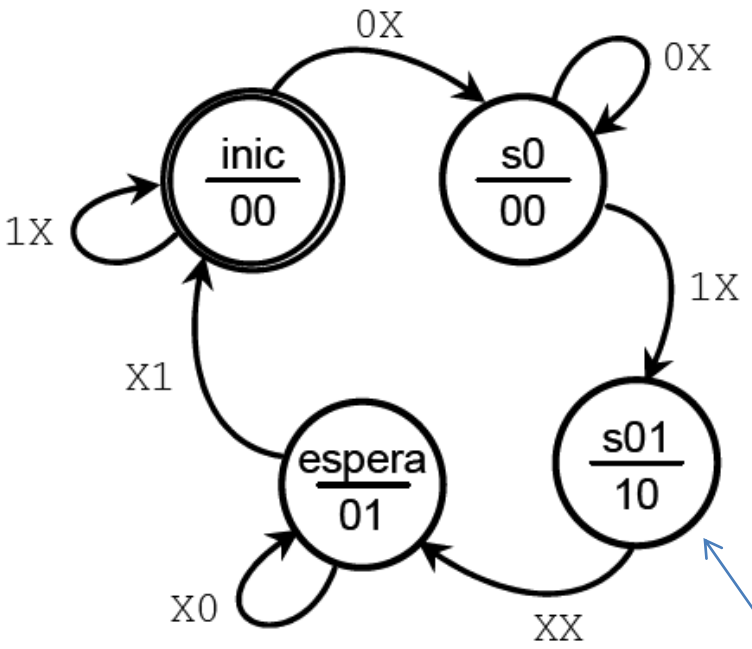
Proyecto 30 : Circuito antirrebotes





Push	Flag_Timer	Estado	Estado Futuro
0	X	Inic	S0
1	X		Inic
0	X	S0	S0
1	X		S01
X	X	S01	Espera
X	1	espera	Inic
X	0		Espera

Proyecto 30 : Circuito antirrebotes



Salida asociada

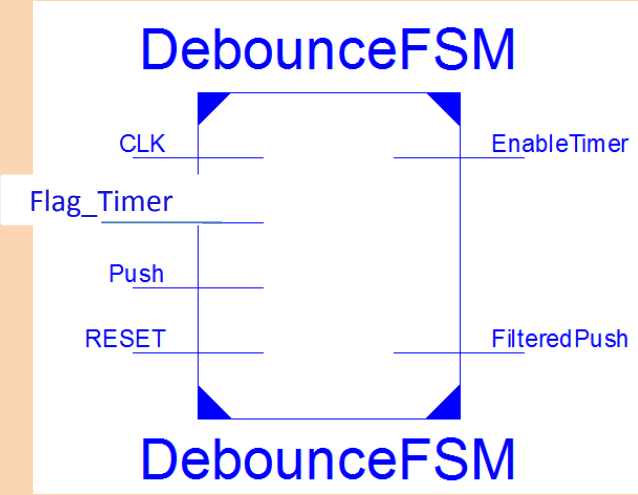
Estado	FilteredPush	EnableTimer
Inic	0	0
S0	0	0
S01	1	0
espera	0	1

Este estado (s01) solo dura un ciclo de reloj, independientemente del valor de las entradas.

Circuito antirrebotes

1. Crea un nuevo proyecto, **Proyecto30**, diseña e incluye los módulos **DebounceFSM** y **Timer300ms**.
2. **DebounceFSM** es una máquina de estado que cuando detecta la secuencia "01" realiza dos acciones:
   - Manda un '1' a la salida EnableTimer
   - Manda un pulso de un ciclo de duración a la salida FilteredPush.

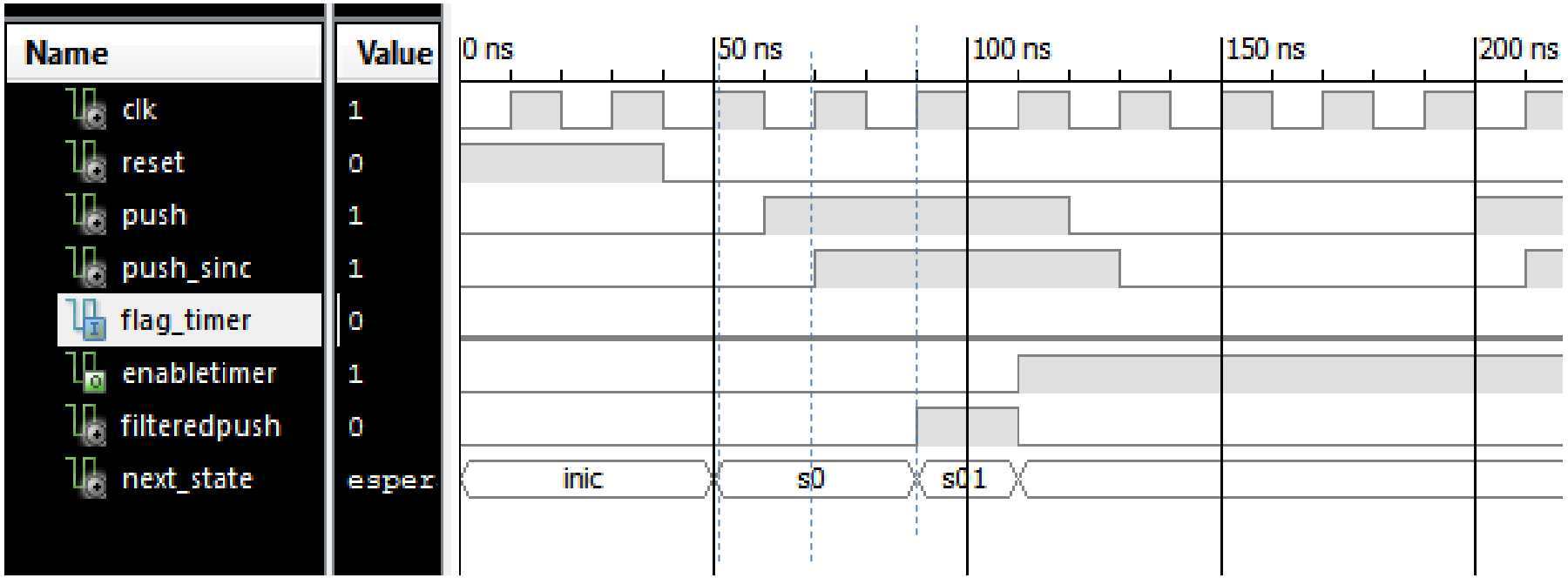
No olvides añadir un primer proceso para sincronizar/registrar la entrada "Push".



Circuito antirrebotes

3. **Timer300ms** en un divisor de frecuencia que se activa cuando su entrada Enable recibe un valor ‘1’. Entonces pone a ‘1’, su salida End300ms durante 20ns, cada 300 ms.
4. Crea un nuevo módulo que interconecte los dos anteriores **Debounce**.

Circuito antirrebotes



1 Ciclo de retraso por sincronizar Push

5.2. Unidad de Control

DidaComp

5. Crea el Proyecto32 e incluye en él un módulo denominado "ControlUnit_DidaComp". Este módulo deberá incluir la FSM capaz de ejecutar el juego de instrucciones completo (ISA) definido para DidaComp.

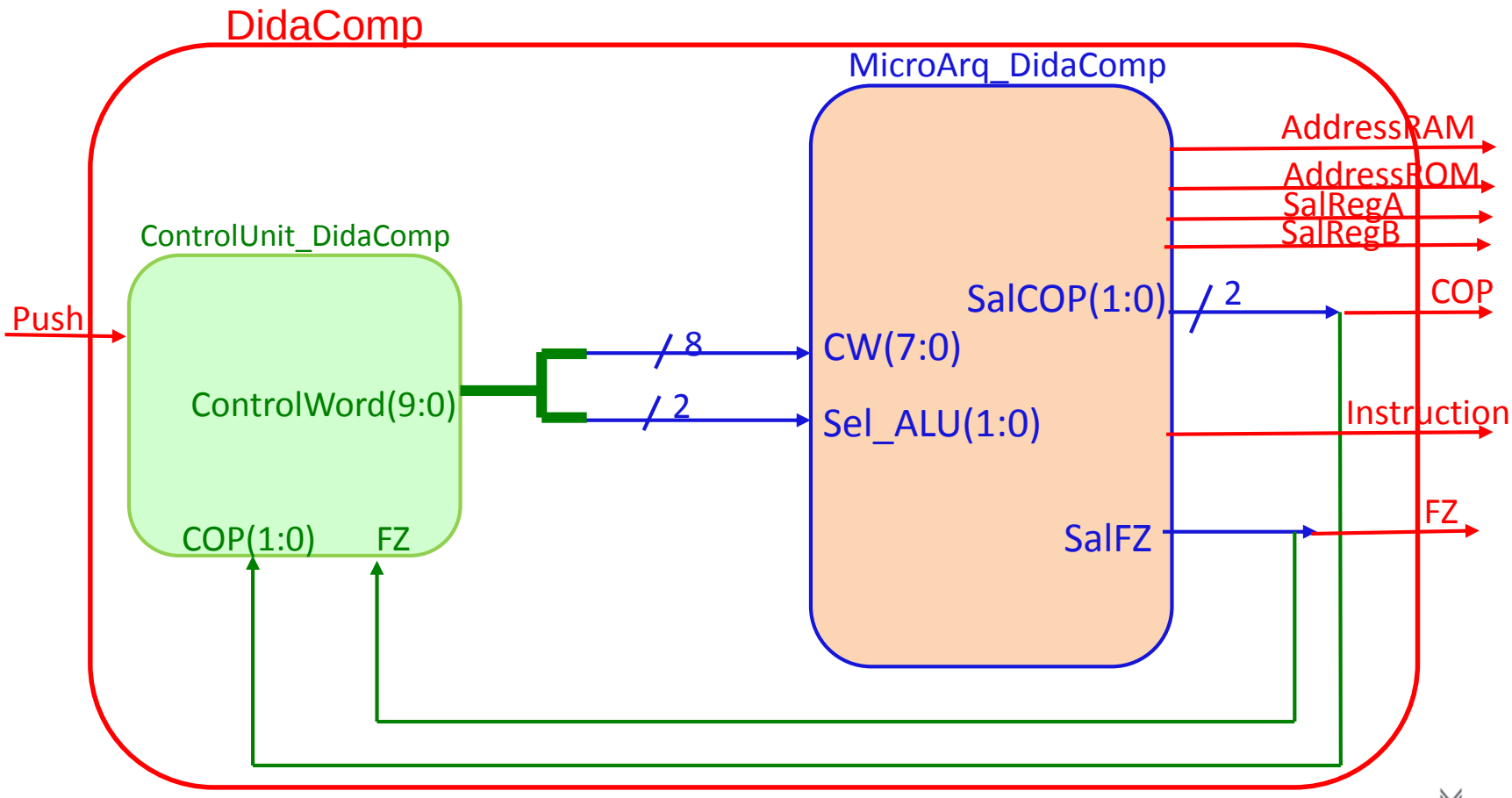
NOTA: Se recomienda usar constantes para identificar con nombre los valores de salida asociadas a cada estado

6. Añadir un testbench para comprobar el funcionamiento de la Unidad de Control. Debe mostrarse la señal interna Next_State.

5.3. Computador Simple (DidaComp)

DidaComp

7. Crea el Proyecto33 e incluye en él un módulo denominado "DidaComp". Este módulo deberá incluir la unidad de control de DidaComp, ControlUnit_DidaComp, y la microarquitectura de DidaComp, MicroArq_DidaComp. Interconecte ambos.



DidaComp

NOTA: Elimine el módulo RisingEdge usado en TOP04 para filtrar CW7 para MicroArq_DidaComp.

8. Añadir un testbench para comprobar el correcto, o no correcto, funcionamiento de DidaComp.

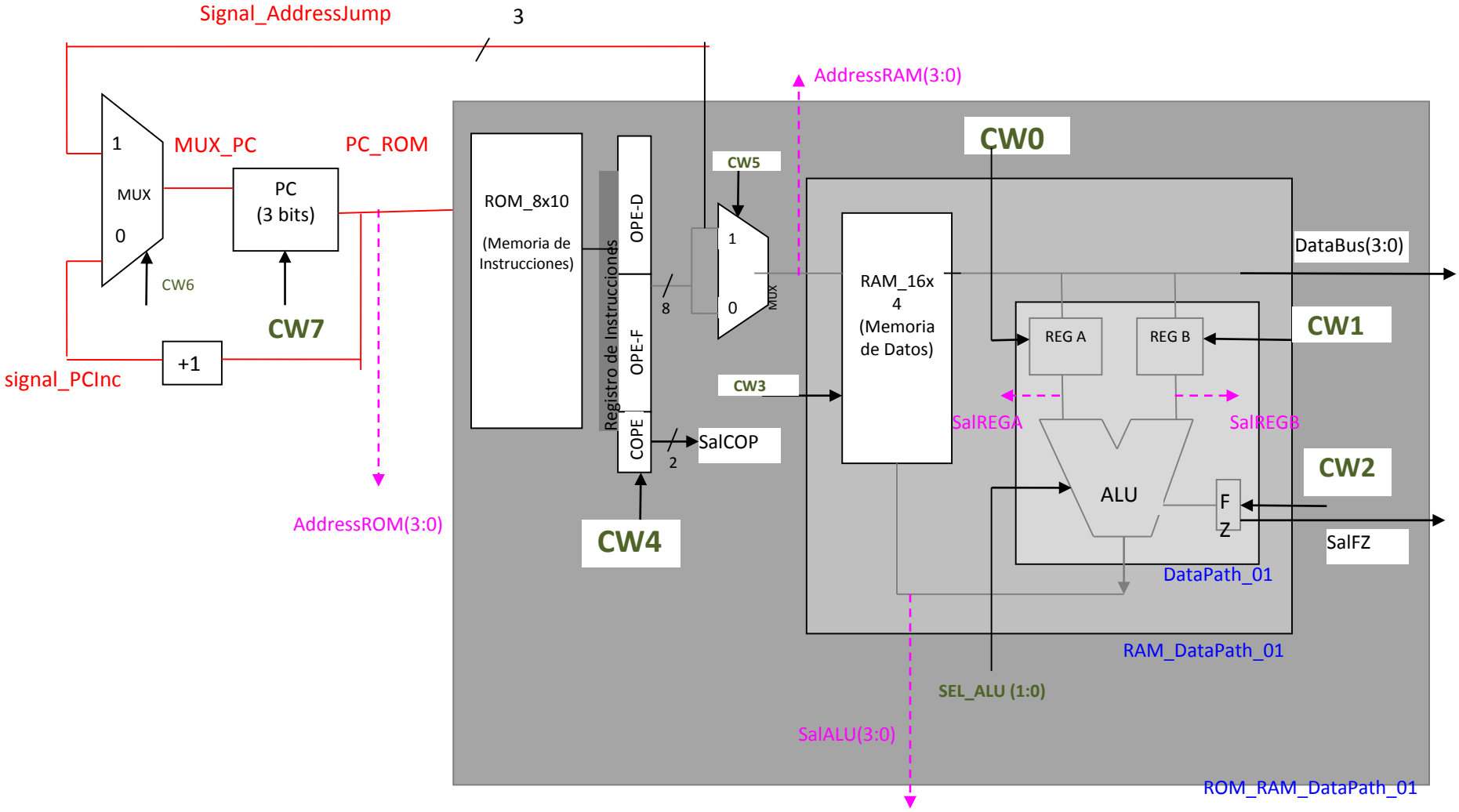
DidaComp

9. Proyecto33: Añade Didacomp y el módulo Disp7Seg_4ON y cree un nuevo módulo "TOP05" que interconecte DidaComp con Disp7Seg_4ON. Verifique el correcto funcionamiento en Basys2.

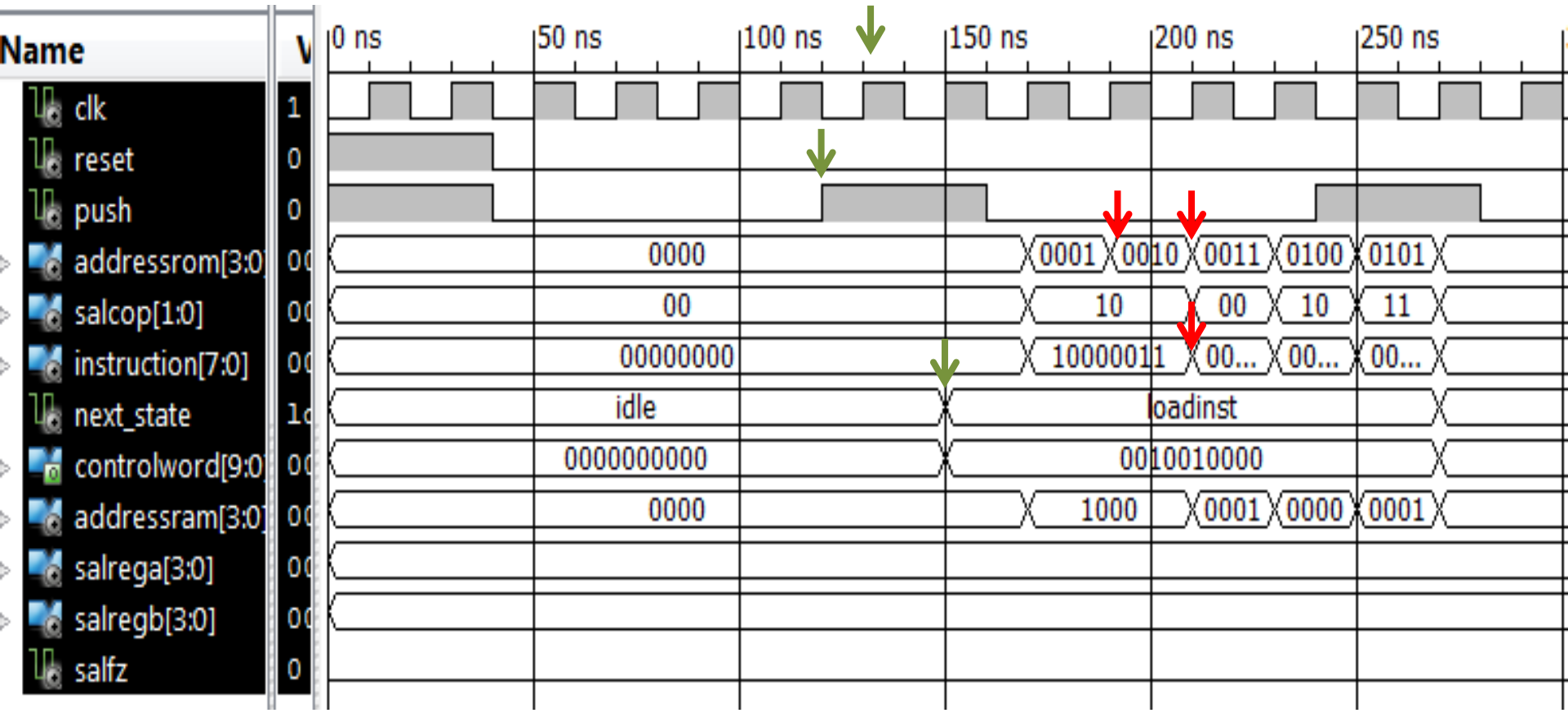
- BTN3 → Push
- BTN0 → Reset
- Led7-0 → Instruction(7:0)
- Display3 → AddressROM (3:0)
- Display2 → AddressRAM(3:0)
- Display1 → RegA(3:0)
- Display0 → RegB(3:0)
- Led Externo (B5,B7) → SalCOP (1:0)
- Led Externo (D12) → SalFZ

Anexo. Control de Push y ControlWord

DidaComp

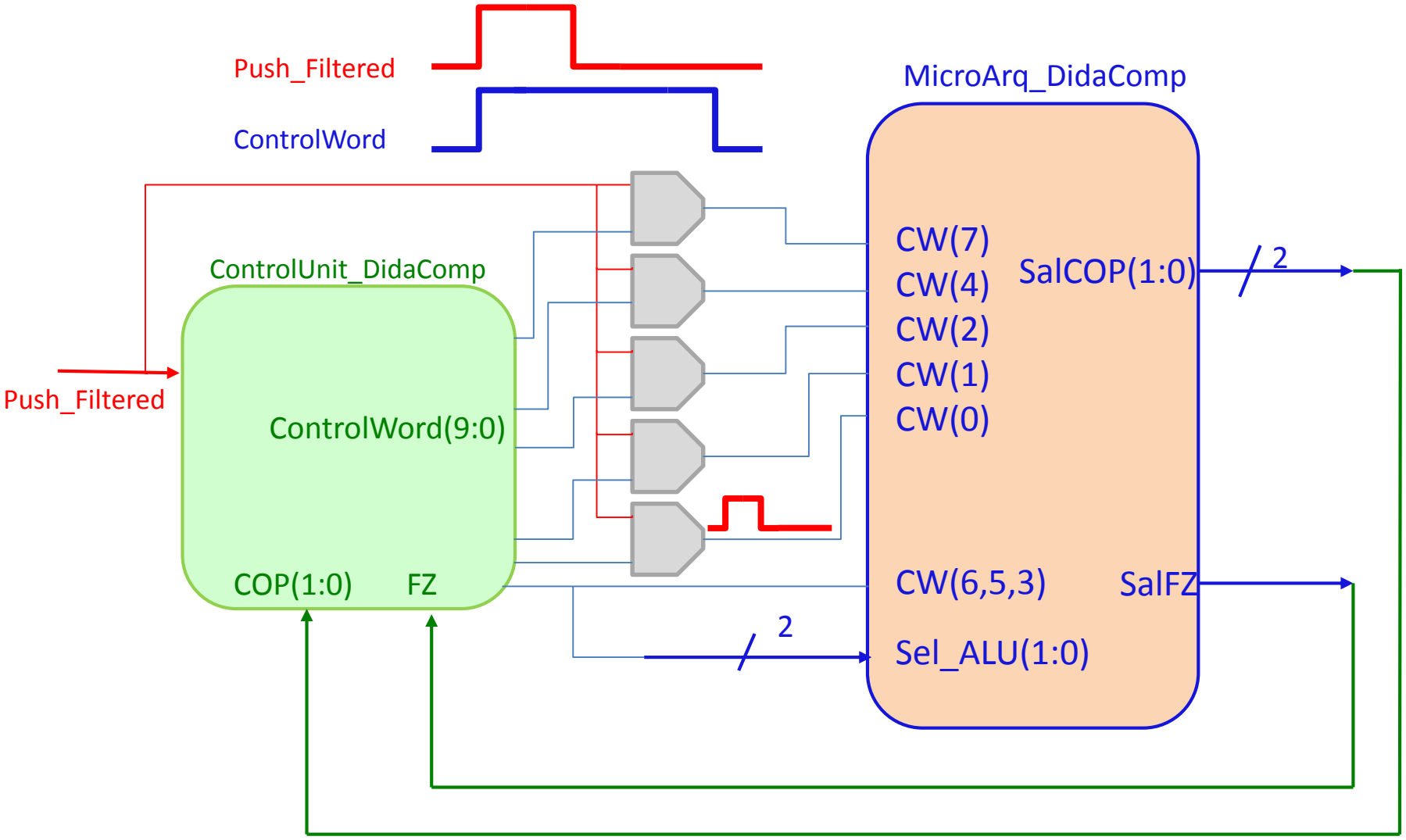

Signal_AddressJump
3
AddressRAM(3:0)
1
MUX
0
MUX_PC
PC_ROM
PC
(3 bits)
CW6
CW7
+1
signal_PCInc
AddressROM(3:0)
ROM_8x10
(Memoria de Instrucciones)
Registro de Instrucciones
OPE-D
OPE-F
COPE
8
2
SalCOP
CW4
CW5
1
0
MUX
CW0
RAM_16x
4
(Memoria
de Datos)
CW3
DataBus(3:0)
CW1
REG A
REG B
SalREGA
SalREGB
ALU
F
Z
CW2
SalFZ
DataPath_01
RAM_DataPath_01
SEL_ALU (1:0)
SalALU(3:0)
ROM_RAM_DataPath_01

DidaComp: “Push” controla la actividad

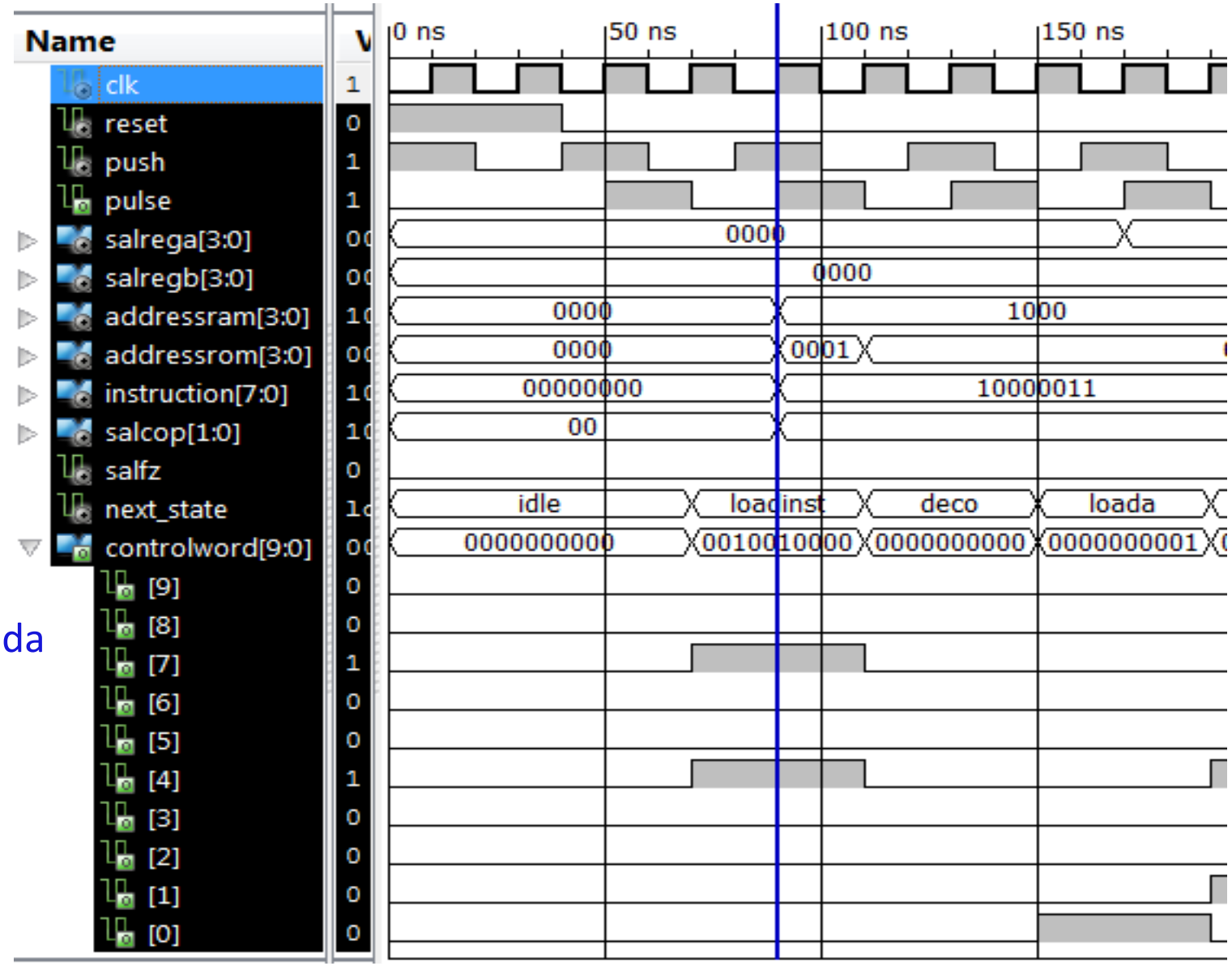


ACTIVIDAD POR FLANCO Flanco “**CLK**” AND Flanco “**Push**”

DidaComp: “Push” controla la actividad



DidaComp: “Push” dura 1 ciclo de reloj



Sin entrada “Push”

DidaComp: Retraso de un ciclo