A Methodology to Remove Unwanted Delays in Outputs and Pre and Post-Synthesis Simulation Mismatches in Implicit State Machines

Shahriyar M. Rizvi
American International University-Bangladesh,
Dhaka, Dhaka-1213, Bangladesh

Jerry J. Cupal
University of Wyoming,
Laramie, WY 82070, USA

Explicit Coding Style:

Models hardware with 2 (or 3) always blocks

Reference: HDL Chip Design,
Douglas Smith, Doone Publications, ISBN 0-9651934-3-8

Implicit Coding Style:

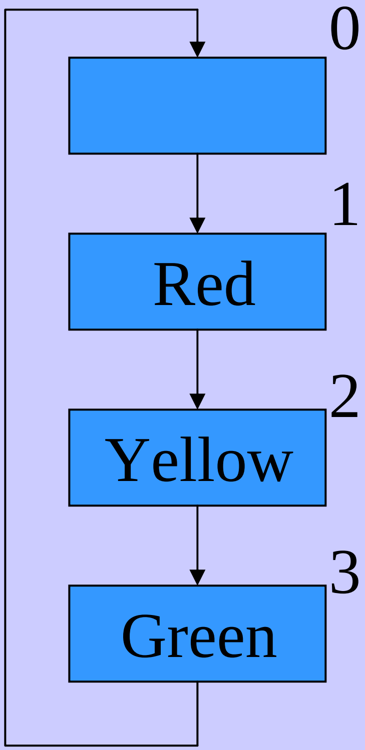
Models the algorithm with one always block

Reference: Verilog Digital Computer Design, Algorithms into Hardware,
Mark Arnold, Prentice Hall PTR, ISBN 0-13-639253-9

Tools

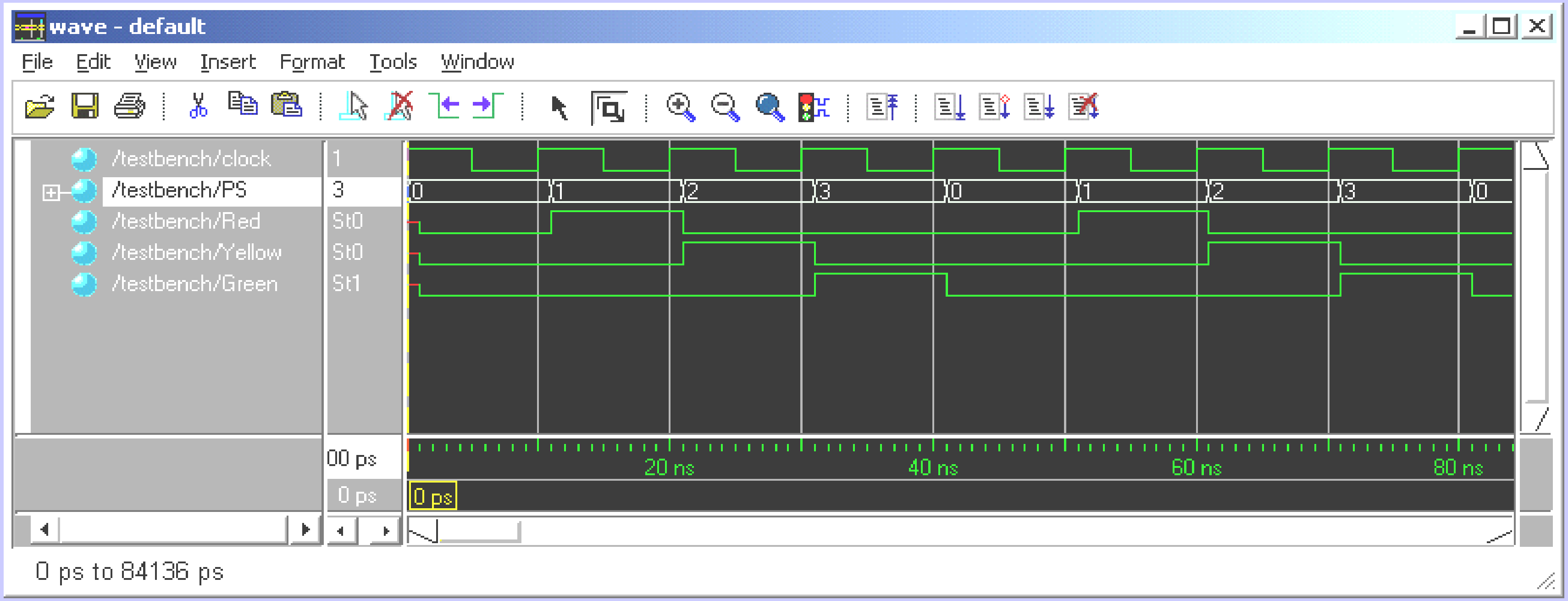
Design environment:	Xilinx ISE v3.1, v5.1.03i
Simulator:	ModelSim XE v3.4d, XE v5.6a
Synthesis tool:	Synopsys FPGA Express

Implicit Code Style of States

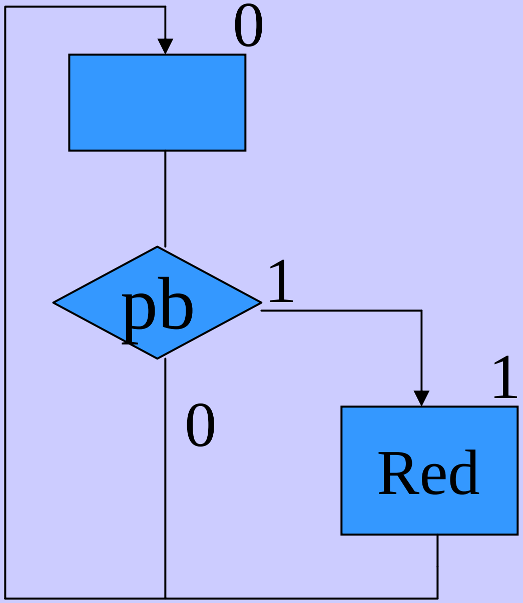


```
always
begin
            @(posedge clock) #1 PS=0;
                        Red=0; Yellow=0; Green=0;

            @(posedge clock) #1 PS=1;
                        Red=1; Yellow=0; Green=0;

            @(posedge clock) #1 PS=2;
                        Red=0; Yellow=1; Green=0;

            @(posedge clock) #1 PS=3;
                        Red=0; Yellow=0; Green=1;
end
```

Functional Simulation:



Implicit Coding Style for Branches

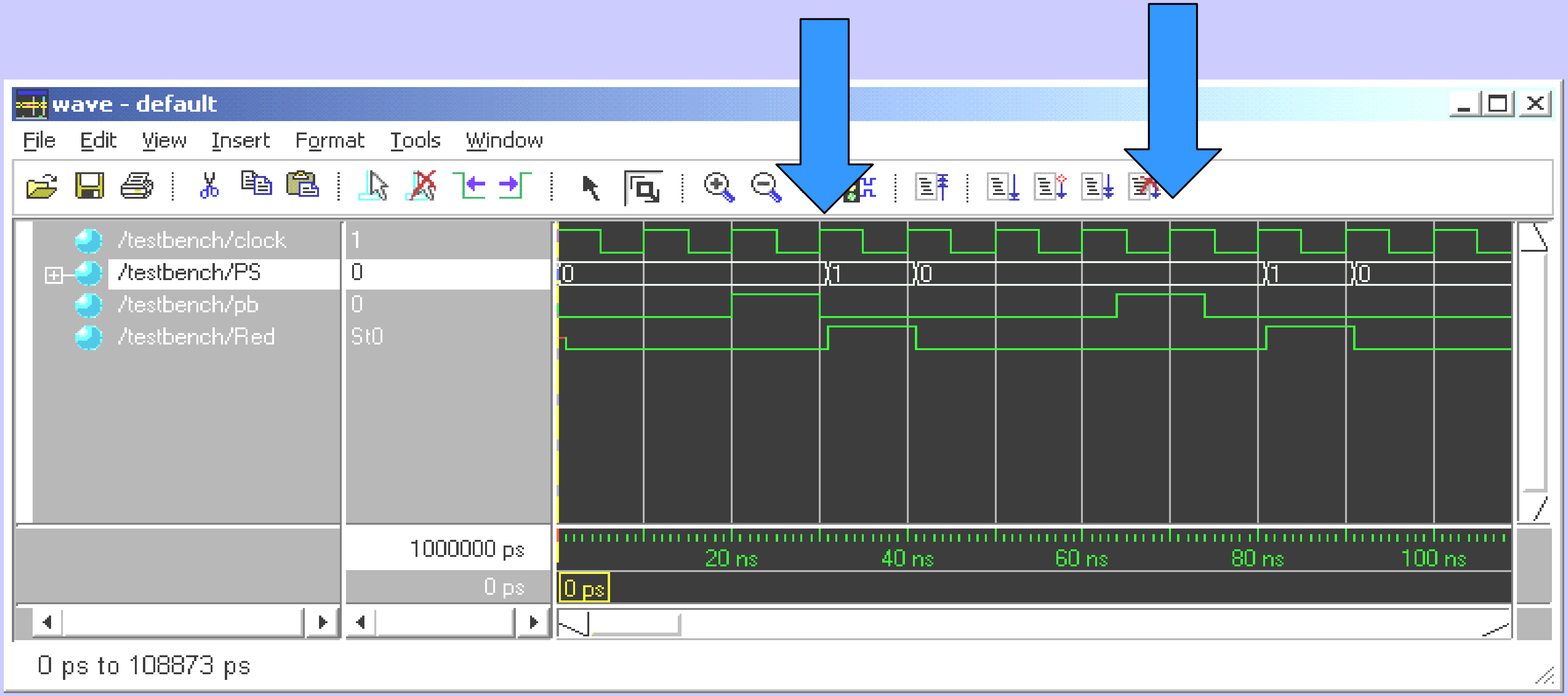


```
always
begin
 @(posedge clock) #1 PS=0; Red=0;

                   if(pb==1)
                   begin
                       @(posedge clock) #1 PS=1; Red=1;
                   end
end
```

Functional Simulation:

Ideal signals

Real signals



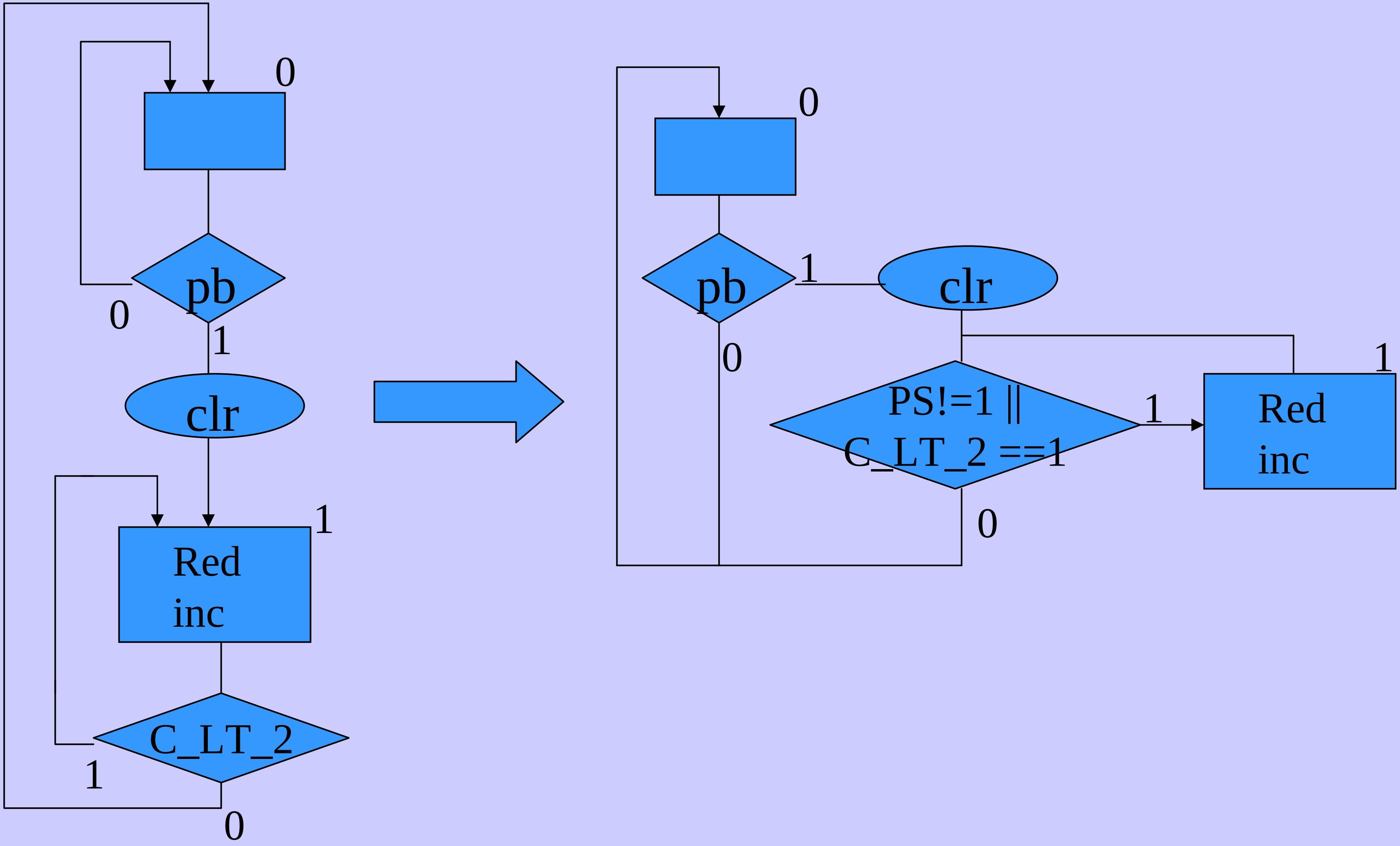
/testbench/clock	1
/testbench/PS	0
/testbench/pb	0
/testbench/Red	St0

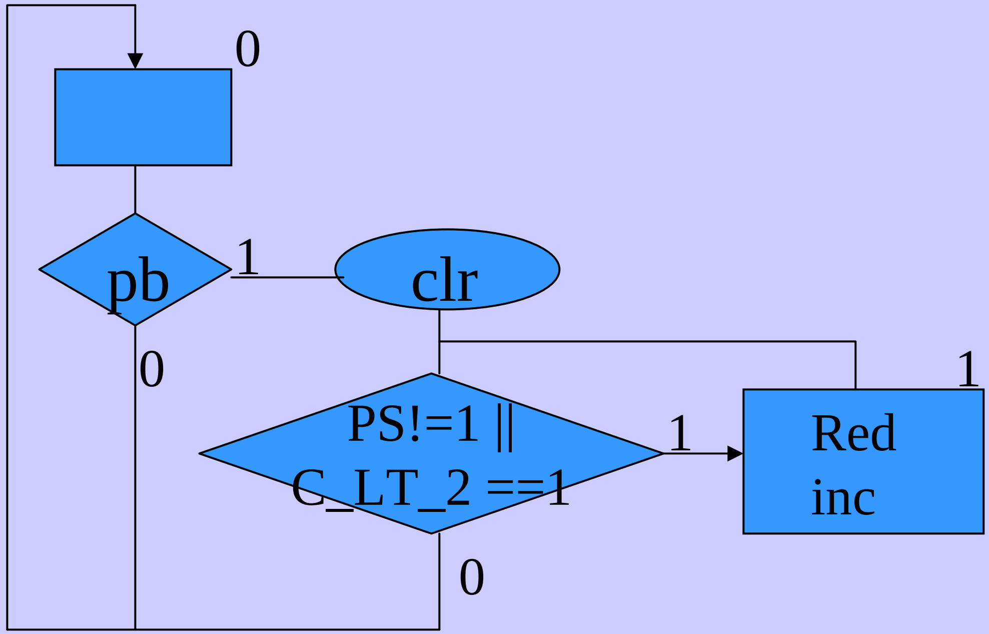
1000000 ps

0 ps

0 ps to 108873 ps

Implicit Coding Style for Loops

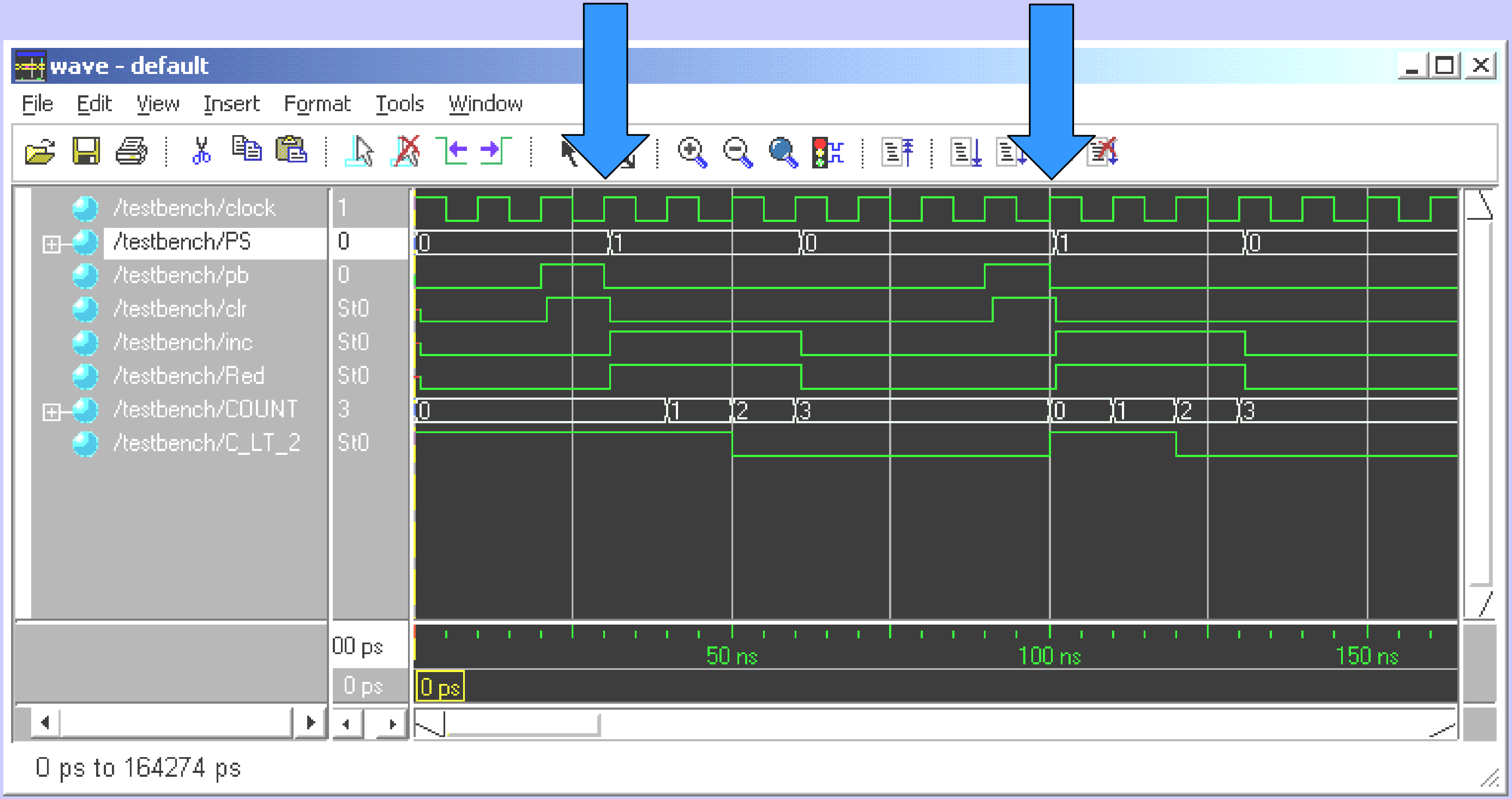




```
always
begin
@(posedge clock) #1 PS=0; clr=0; inc=0; Red=0;
if (pb==1)
        begin
        clr=1;
        while(PS!=1 || C_LT_2==1)
                begin
                @(posedge clock) #1 PS=1; clr=0; inc=1; Red=1;
                end
        end
end
```

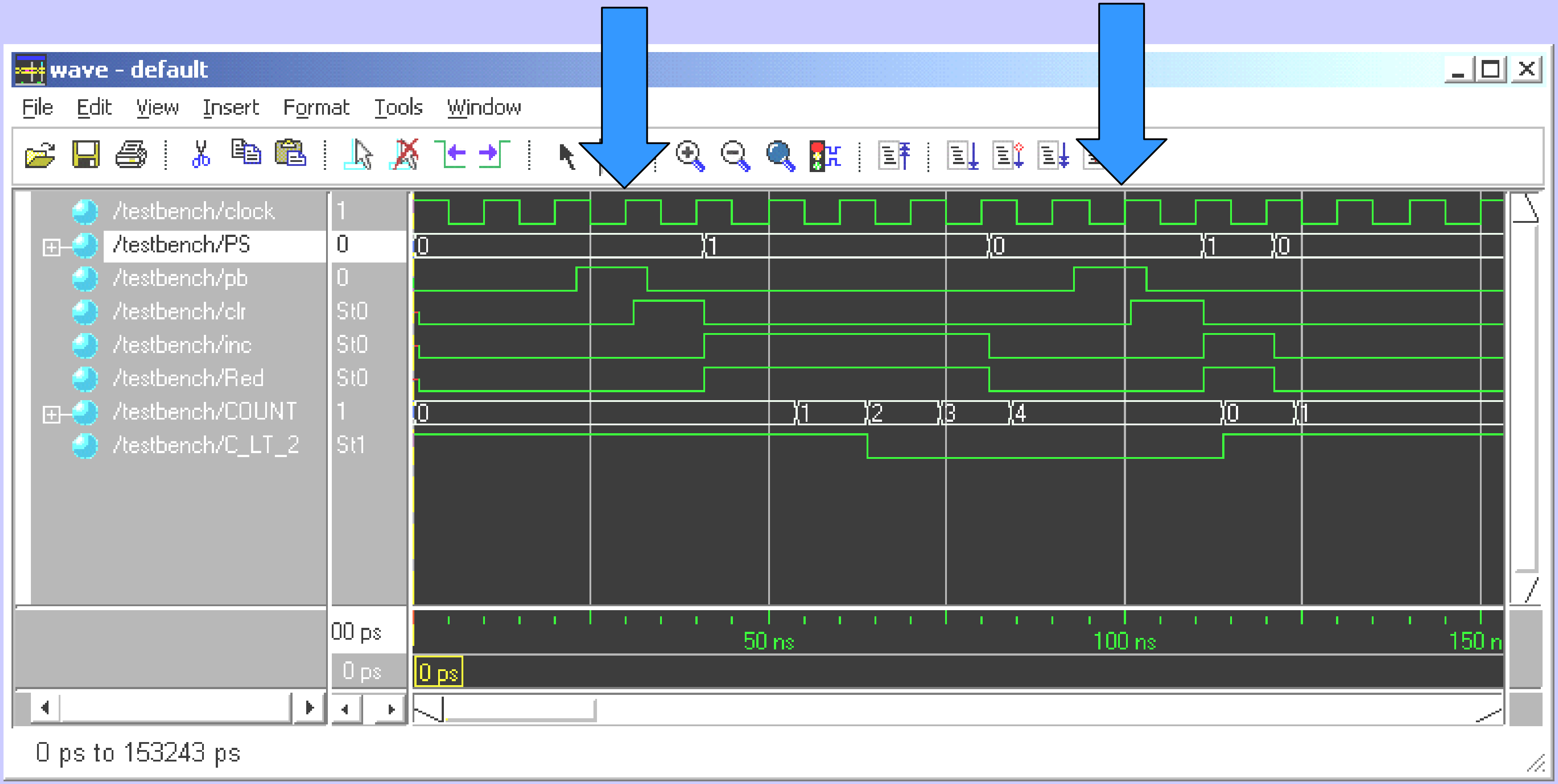
Functional Simulation

Ideal signals



Functional Simulation

Real signals

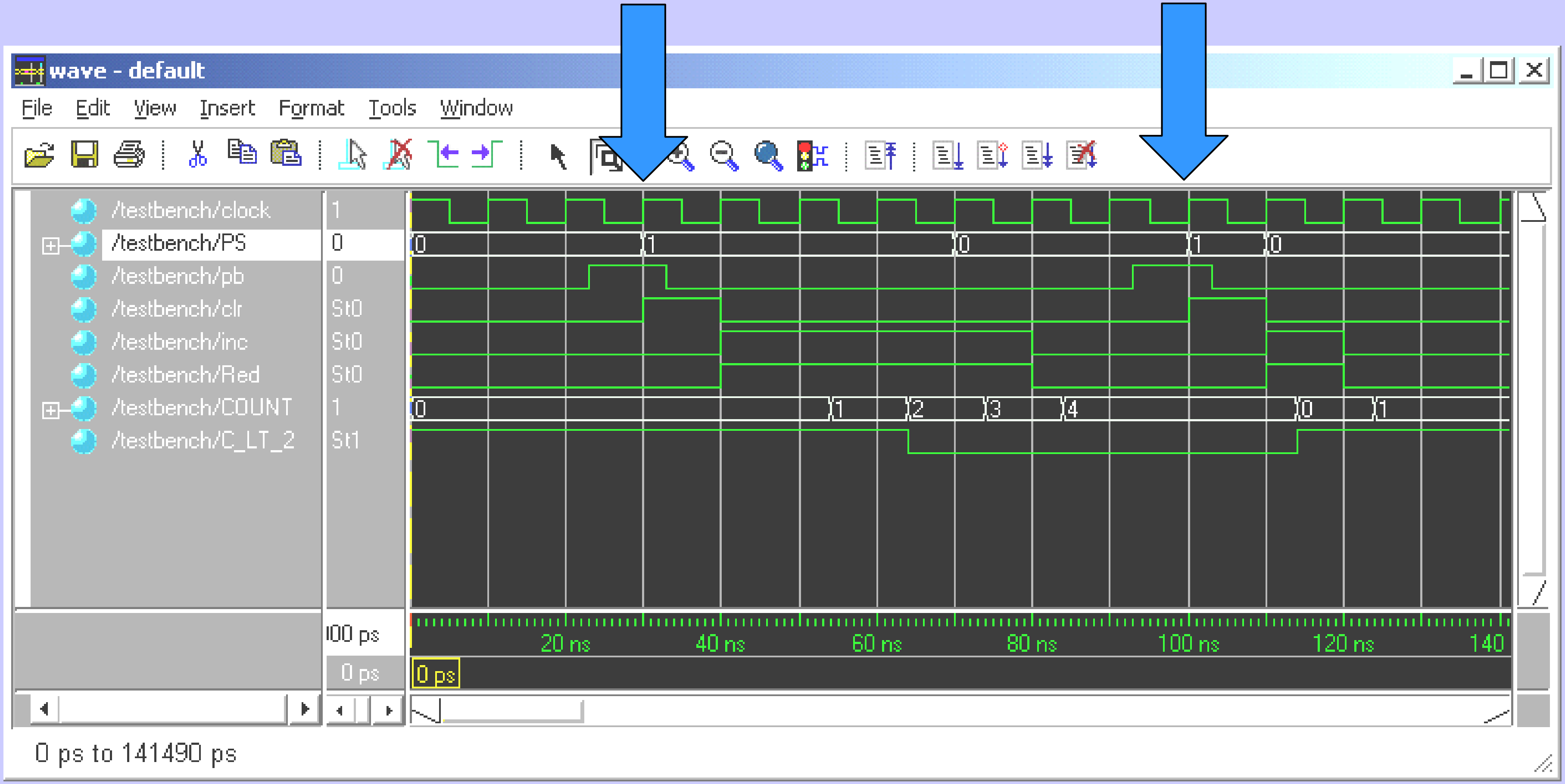


/testbench/clock	1
/testbench/PS	0
/testbench/pb	0
/testbench/clr	St0
/testbench/inc	St0
/testbench/Red	St0
/testbench/COUNT	1
/testbench/C_LT_2	St1

0 ps to 153243 ps

Explicit Style Coding with Registered Outputs

Real signals



/testbench/clock	1
/testbench/PS	0
/testbench/pb	0
/testbench/clr	St0
/testbench/inc	St0
/testbench/Red	St0
/testbench/COUNT	1
/testbench/C_LT_2	St1

0 ps to 141490 ps

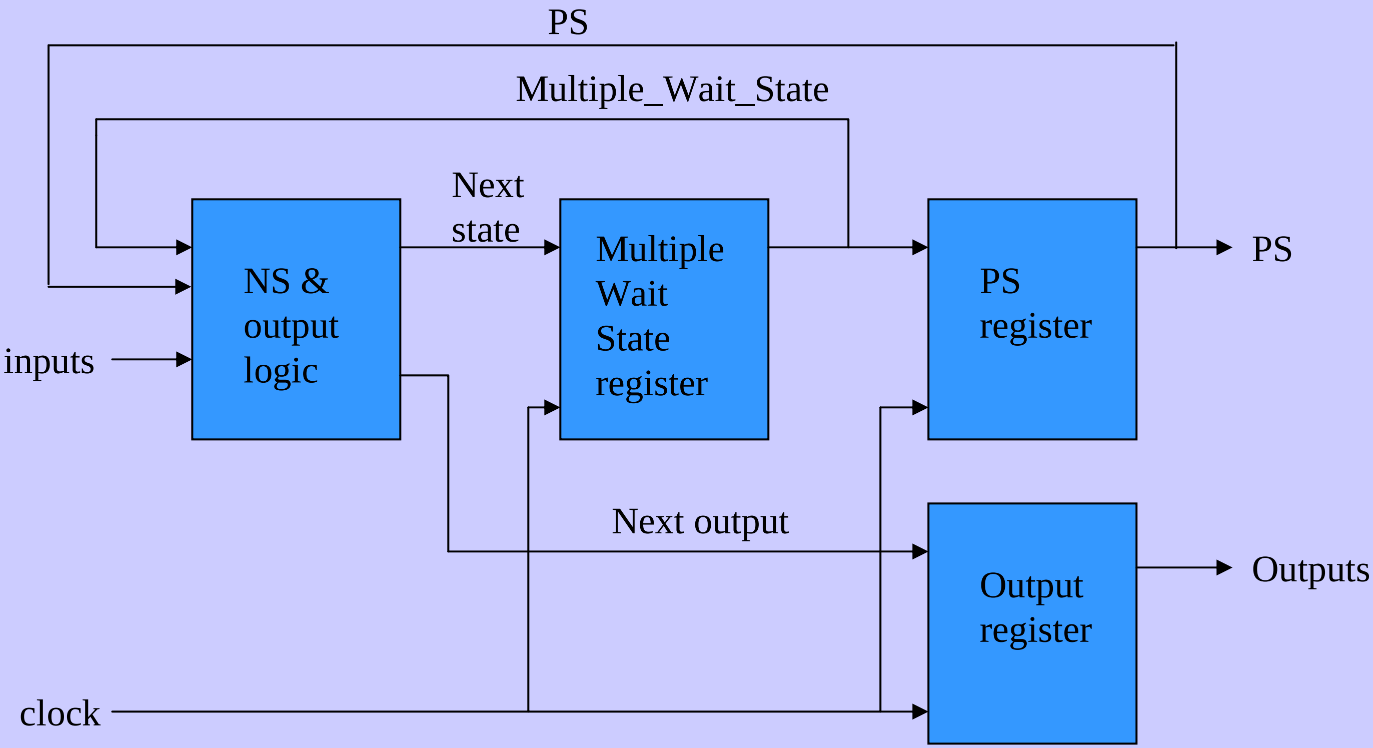
Conclusion 1

Use ideal signals when doing functional simulations of state machines coded in implicit style.

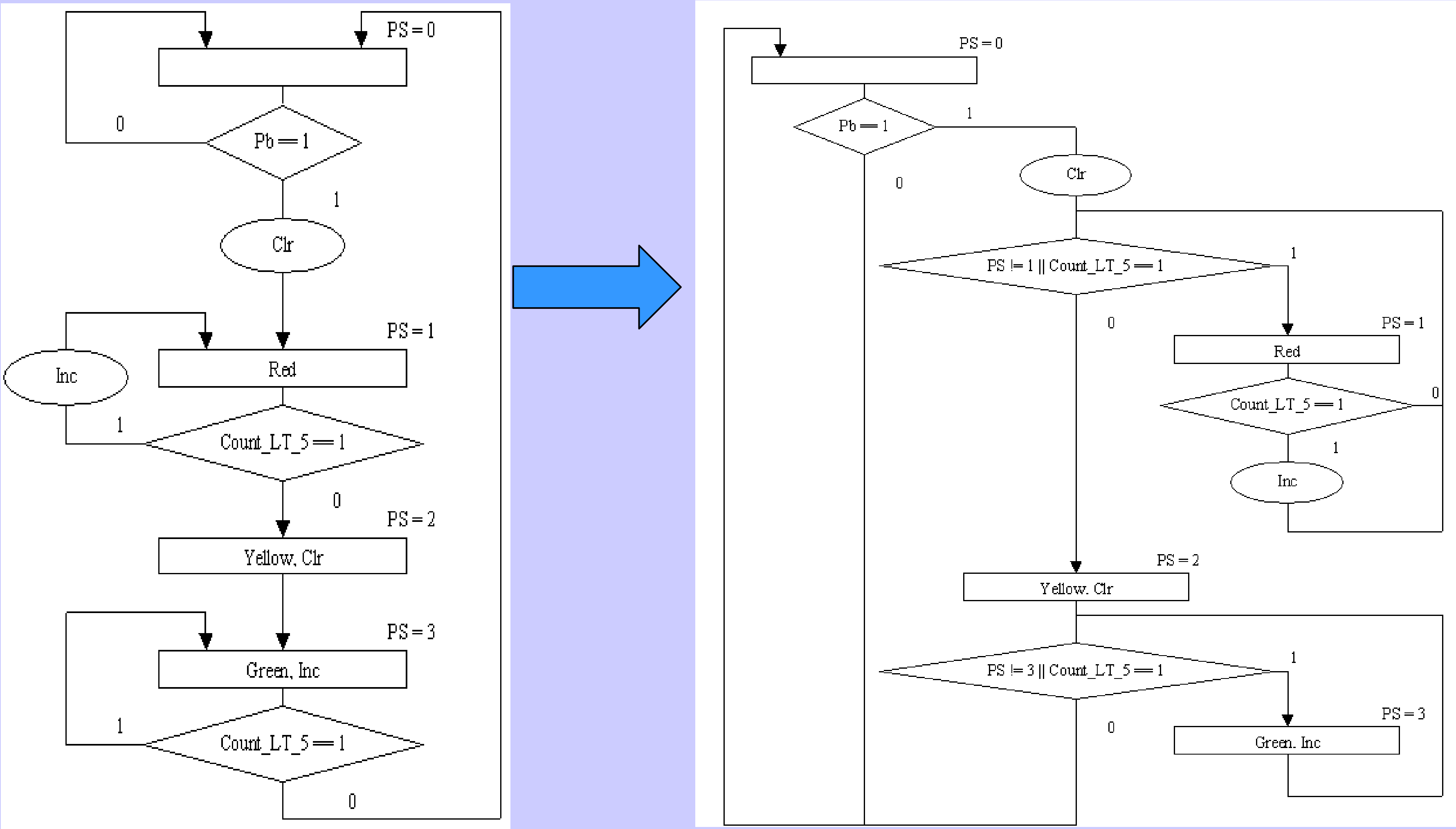
ie, Synchronized inputs and no delays in data path elements.

Be aware that non-ideal timing will produce incorrect behavior in the functional simulation. In fact, timing will be like state machines coded in explicit style with registered outputs

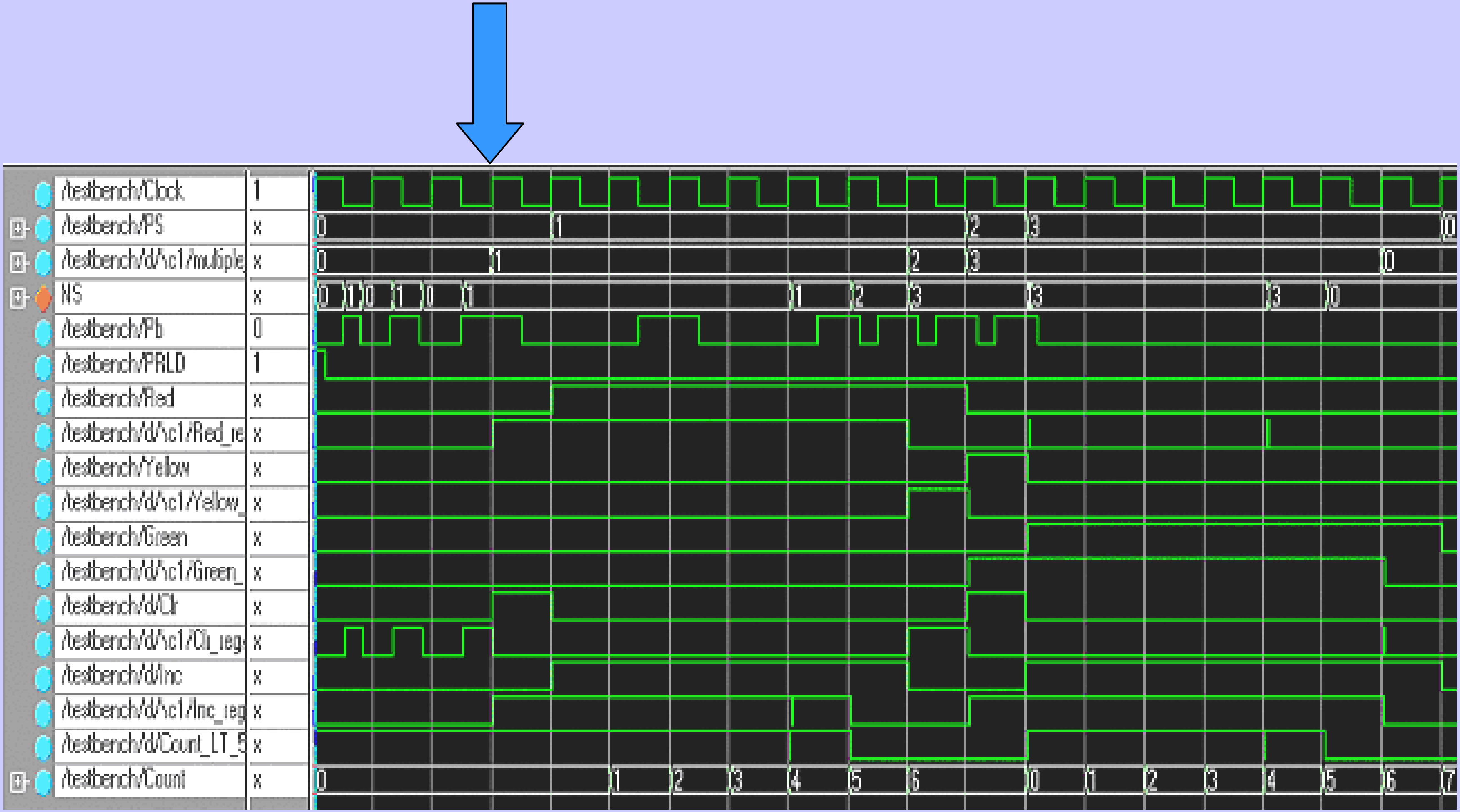
Hardware Generated in the Synthesis Process



State Machine Example



Operation after Place-and-Route



Modification Process:
Replace output resisters with buffers

1. Insert simple buffer library cell if it does not exist in file

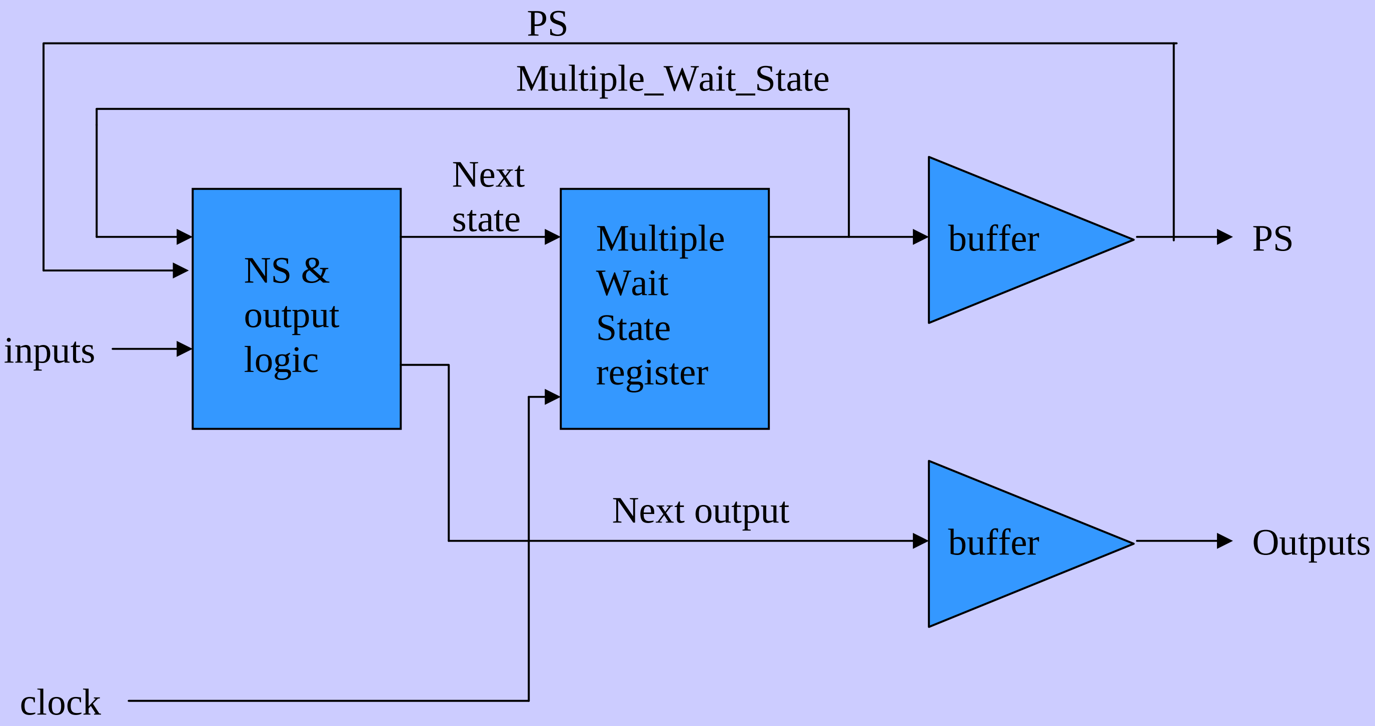
2. Remove the clock signal from the output flip-flops

3. Remove the clear and preset signals from the output flip-flops

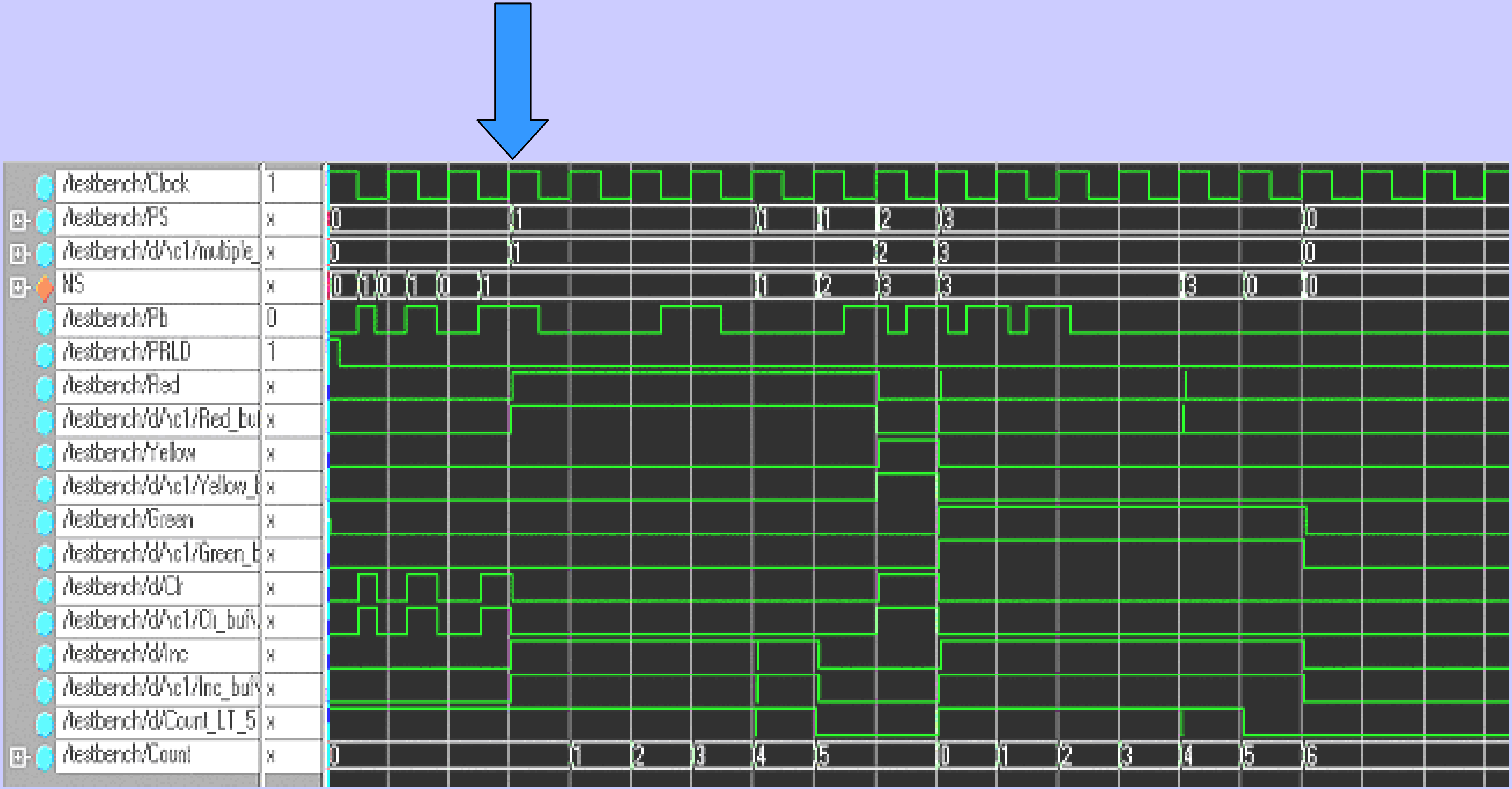
4. Replace the output flip-flops with the buffers

5. Route the input and outputs of the output flip-flops
to the inputs and outputs of the buffers

Hardware After Modification Process



Operation after Place-and-Route of Modified Hardware



Conclusion 2

Synthesized hardware straight out of FPGA Express will not function as expected. In addition, there are extra and un-necessary flip-flops in the implementation.

If the output of FPGA Express is modified by the process describe here, the hardware works as expected.

Modification Process

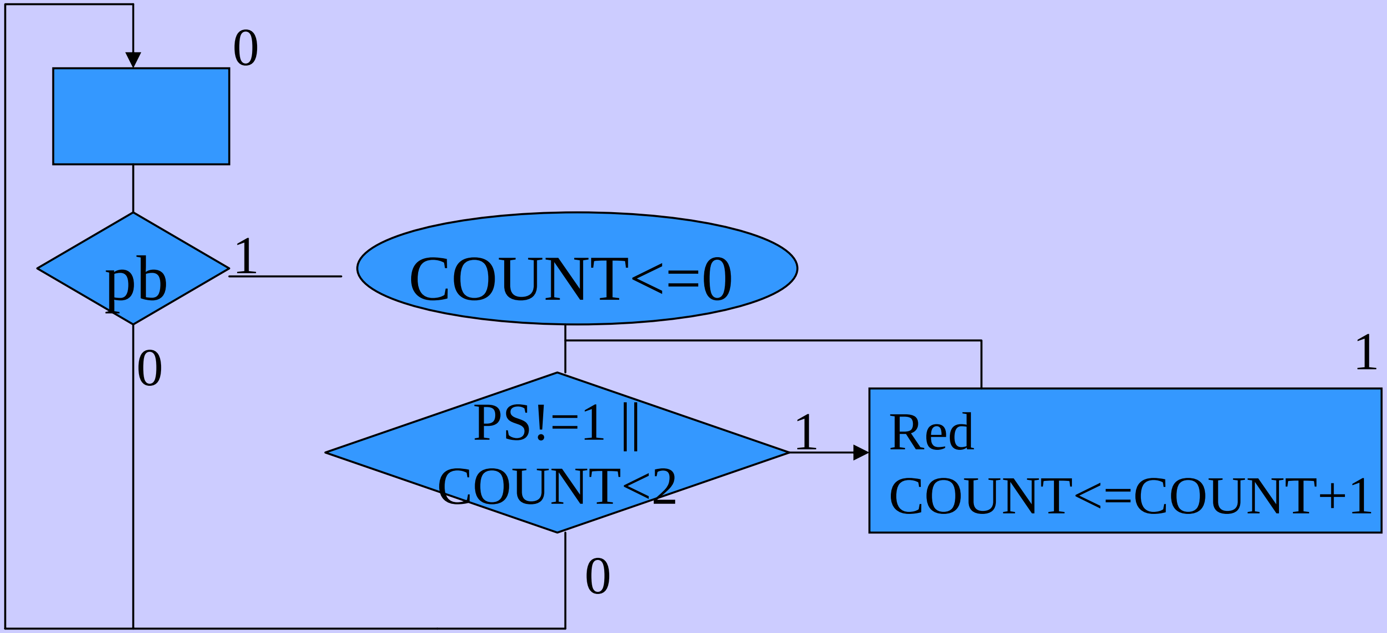
Currently, the modification process is done by manually editing the .edn file out of FPGA Express. It is necessary to implement this operation in a program.

Ideally, it would be nice to have this process done as an option in FPGA Express.

Future Work

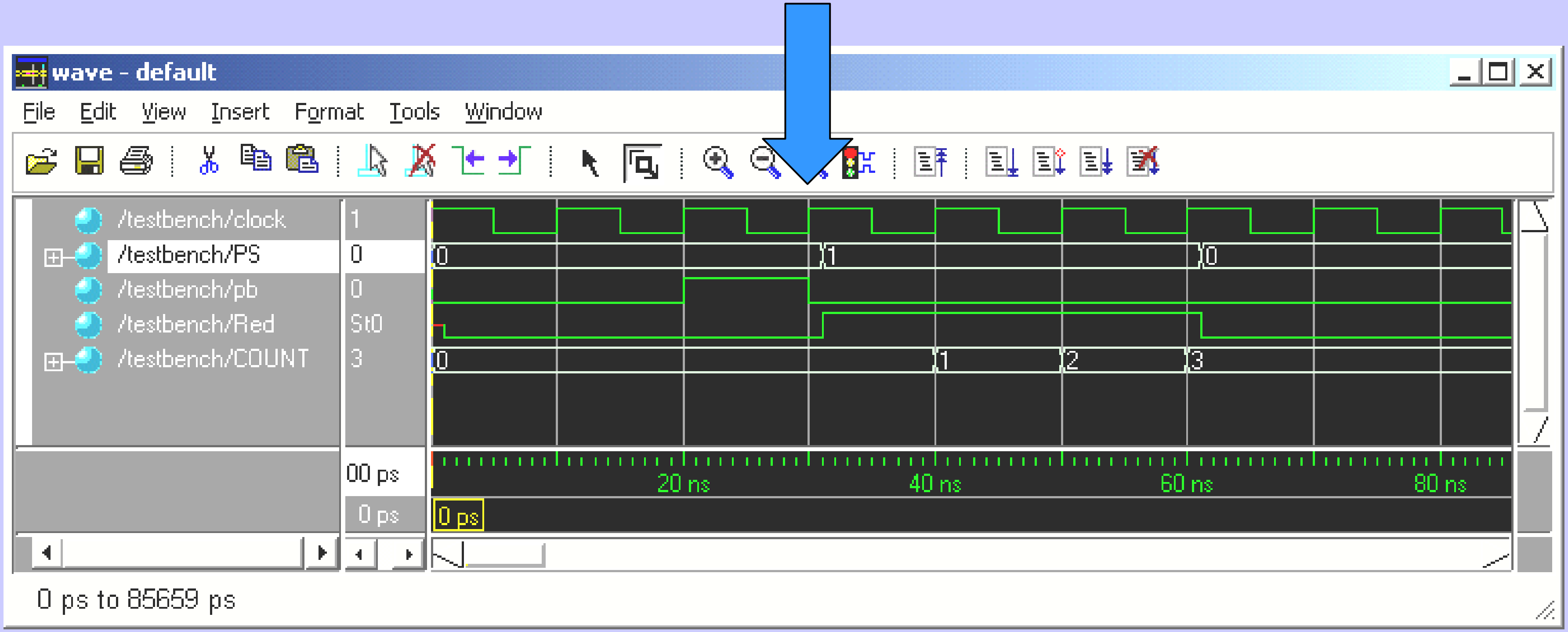
In explicit designs, it is possible to model the data path elements (registers as well as combinational logic) within the code for the state machine. This allows one to write at a higher, more abstract level.

This method can also be done in implicit coding, but gives improper behavior.



```
always
begin
@(posedge clock) #1 PS=0;  Red=0;
if (pb==1)
            begin
            COUNT <= @(posedge clock) 0;
            while(PS!=1 || COUNT<2)
                        begin
                        @(posedge clock) #1 PS=1;  Red=1;
                        COUNT <= @(posedge clock) COUNT +1;
                        end
            end
end
```

Functional Simulation



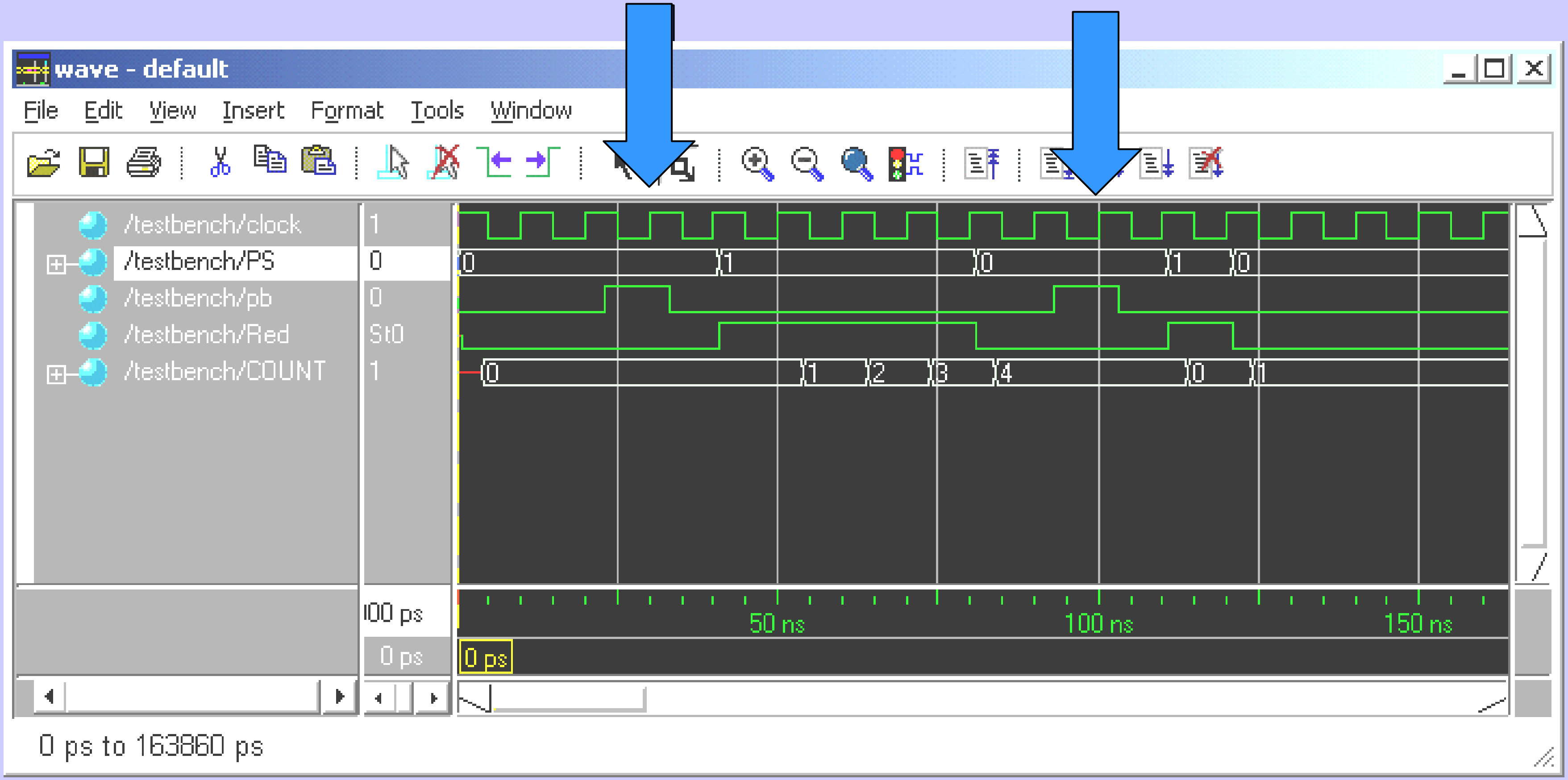
Ideal signals

/testbench/clock 1
/testbench/PS 0
/testbench/pb 0
/testbench/Red St0
/testbench/COUNT 3

0 ps to 85659 ps

Functional Simulation

Real signals



Final Comments

Explicit style coding has a “goto“ look. It is difficult to read the algorithm by looking at the code.

Implicit style coding follows the algorithm closely. There is a higher level of abstraction which allows the designer to concentrate on the algorithm rather than the architectural details.

If one codes in implicit style, you should do functional simulations with ideal signals, then synthesize and remove the output flip-flops in the process described here. The real hardware will work as a design done in explicit style.