DIGITAL LOGIC WITH VHDL
(Fall 2013)
Unit 7

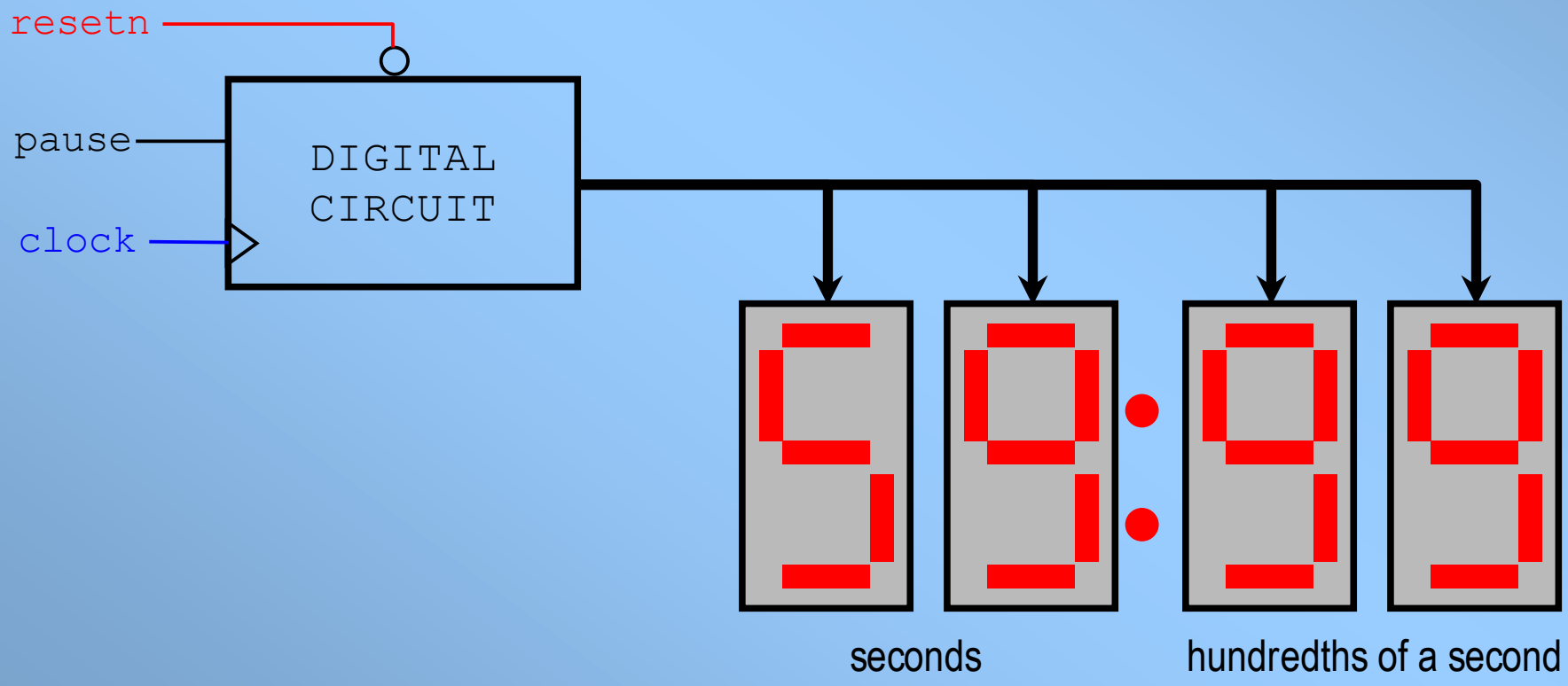
✓*INTRODUCTION TO DIGITAL LOGIC DESIGN*:

- Digital System Model
- Example: Stopwatch

Instructor: Daniel Llamocca

✓*DIGITAL CIRCUIT DESIGN*

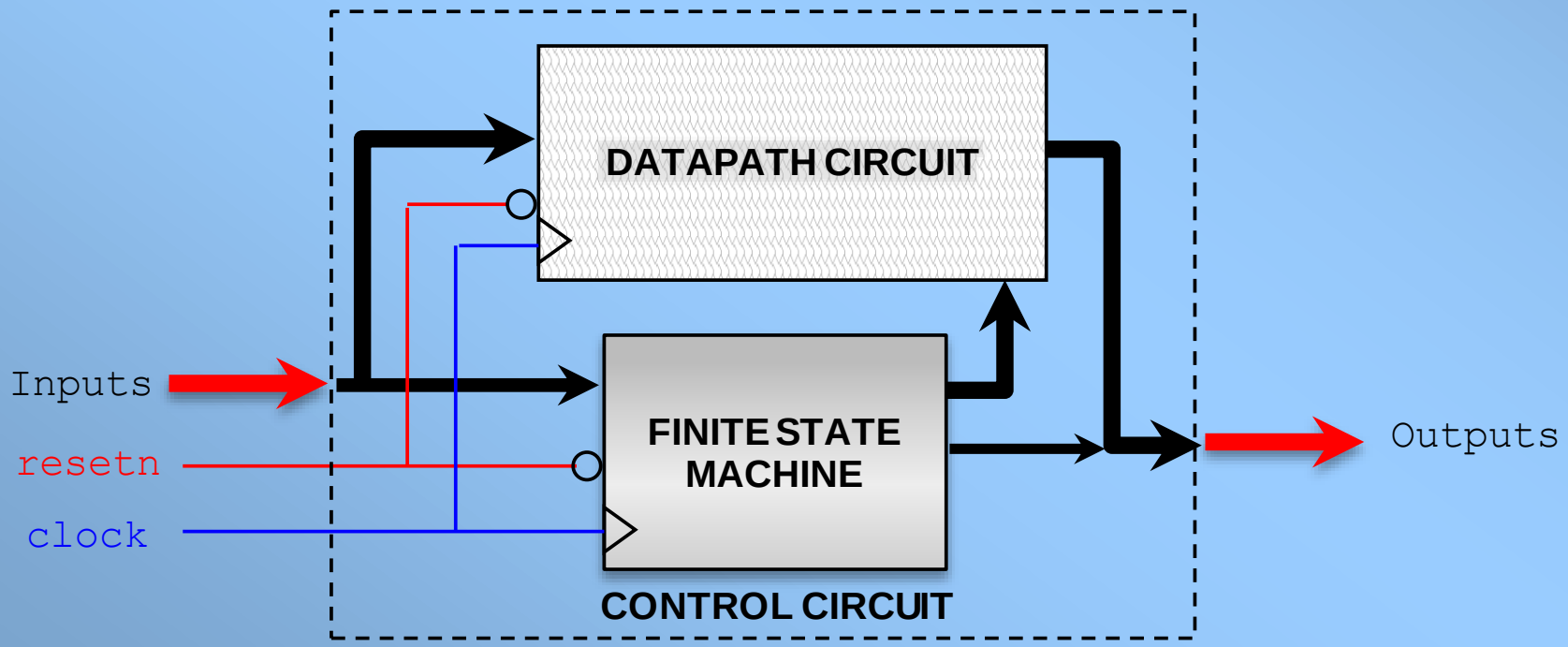
- **Selected example**: Stopwatch that counts in increments of 1/100th of a second. *Circuit design and VHDL implementation*.
  - **Inputs**: Pause, resetn, clock
  - **Outputs**: Count on four 7-segment displays
  - **Materials**: DIGILENT NEXYS3 Board



✓*IMPLEMENTATION STEPS*

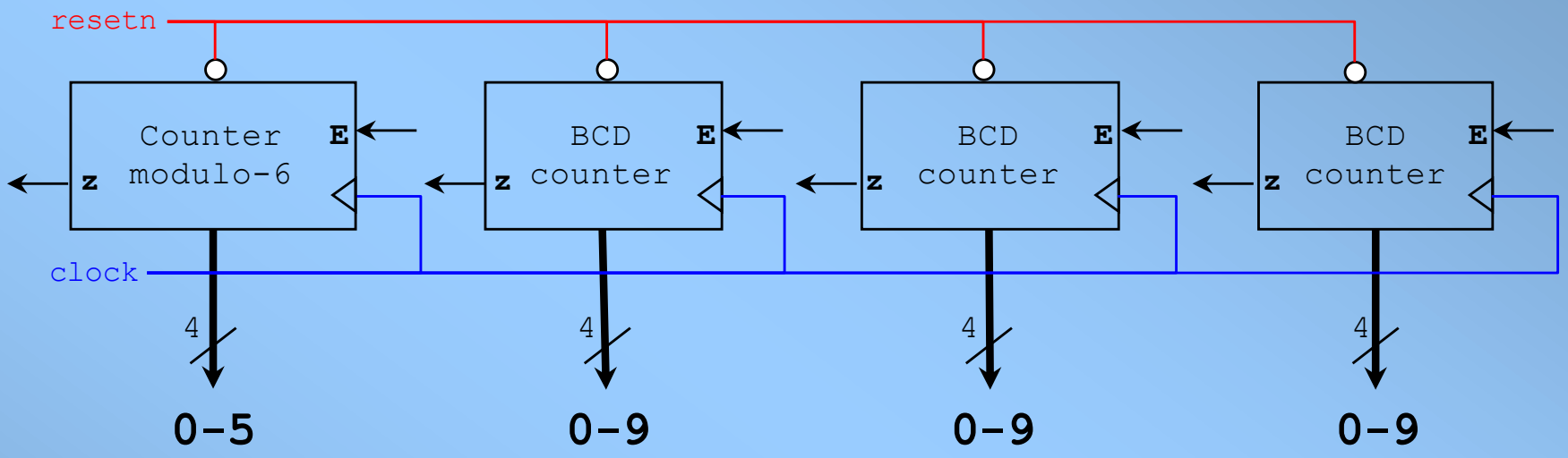
- Circuit Design
- VHDL coding
- Synthesis
- Simulation
- Place and route (also pin assignment)
- FPGA Programming and Testing

- *CIRCUIT DESIGN*: We follow the Digital System Model that requires a **Finite State Machine** and a **Datapath Circuit.**

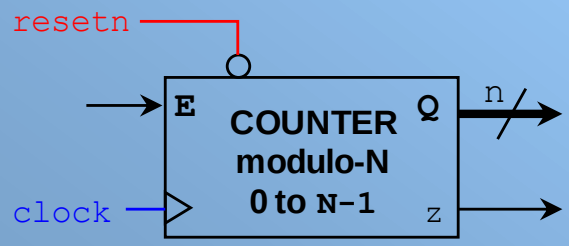


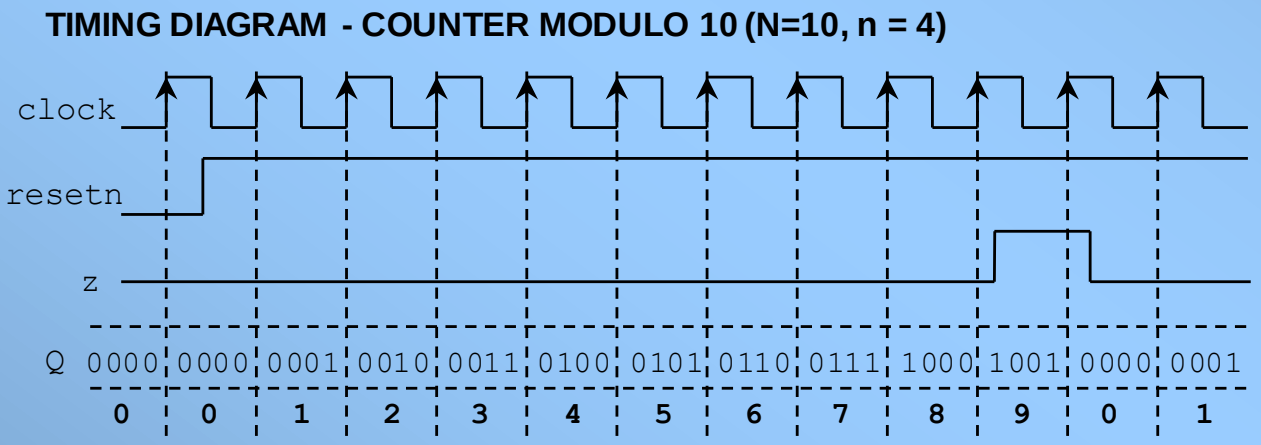
✓DATAPATH CIRCUIT

- Brainstorming: we need four counters. Tree counters modulo-10 and one counter modulo-6.



- The figure depicts a generic modulo-N, where n = ⌈log2(N)⌉





✓DIGITAL COUNTER DESIGN

- Counters are usually designed as State Machines. Then, one just needs to write the HDL code for that FSM. Problem: For different counts, we need a different state machine.
- More efficient manner: Think of them as accumulators. This way, the HDL code is easier to read and modify (if we require a different count).
- Example: BCD counter

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity my_bcdcount is
 port ( clock, resetn, E: in std_logic;
        Q: out std_logic_vector(3 downto 0);
        z: out std_logic);
end my_bcdcount;

architecture bhv of my_bcdcount is
  signal Qt: std_logic_vector(3 downto 0);
begin
  process (resetn,clock,E)
  begin
    if resetn = '0' then Qt <= "0000";
    elsif (clock'event and clock='1') then
       if E = '1' then
           if Qt = "1001"' then
               Qt <= "0000";
           else
               Qt <= Qt + "0001";
           end if;
       end if;
    end if;
  end process;
  z <= '1' when Qt = "1001" else '0';
  Q <= Qt;
end bhv;
```

✓DIGITAL COUNTER DESIGN: PARAMETRIC CODE

- The previous VHDL code allows for easy parameterization so that we can create counters with arbitrary counts.
- Parametric VHDL code: use of '*generics*'. The following VHDL code has a parameter COUNT that the user can modify at will without modifying anything else.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use ieee.math_real.log2.all;
use ieee.math_real_ceil.all;

entity my_gencount is
 generic (COUNT: INTEGER:= 10);
 port ( clock, resetn, E: in std_logic;
   Q: out std_logic_vector(integer(ceil(log2(real(COUNT))))-1 downto 0);
   z: out std_logic);
end my_gencount;

architecture bhv of my_gencount is
  constant nbits:= INTEGER:= integer(ceil(log2(real(COUNT))));
  signal Qt: std_logic_vector(nbits - 1 downto 0);
...
```

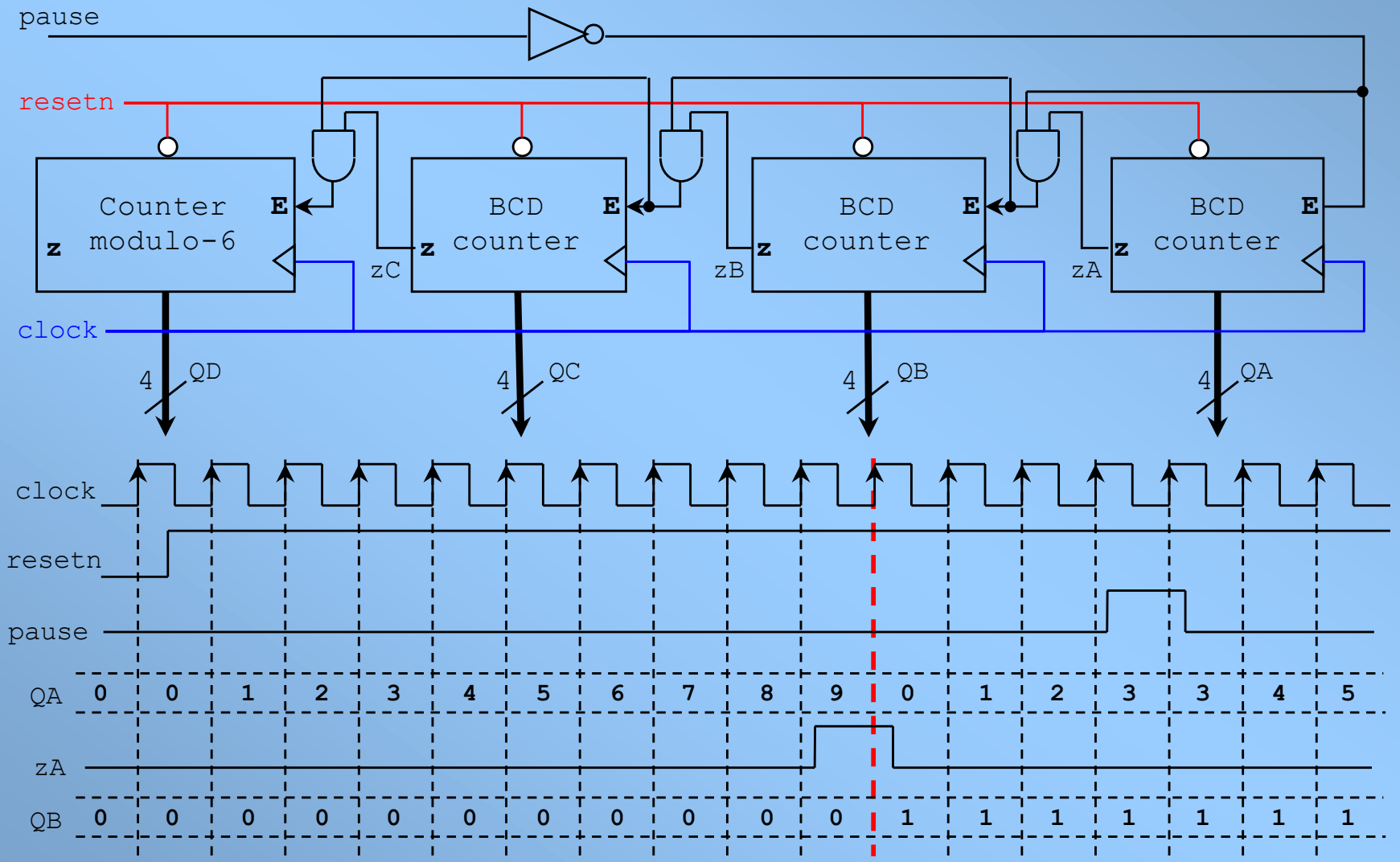
✓DIGITAL COUNTER DESIGN: PARAMETRIC CODE

- This generic architectural description allows for a different counter to be created every time we modify the parameter COUNT.

```
...
begin
  process (resetn,clock,E)
  begin
    if resetn = '0' then Qt <= (others => '0');
    elsif (clock'event and clock='1') then
       if E = '1' then
           if Qt = conv_std_logic_vector(COUNT-1,nbits) then
               Qt <= (others => '0');
           else
               Qt <= Qt + conv_std_logic_vector(1,nbits);
           end if;
       end if;
    end if;
  end process;
  z <= '1' when Qt = conv_std_logic_vector (COUNT-1, nbits) else '0';
  Q <= Qt;
end bhv;
```

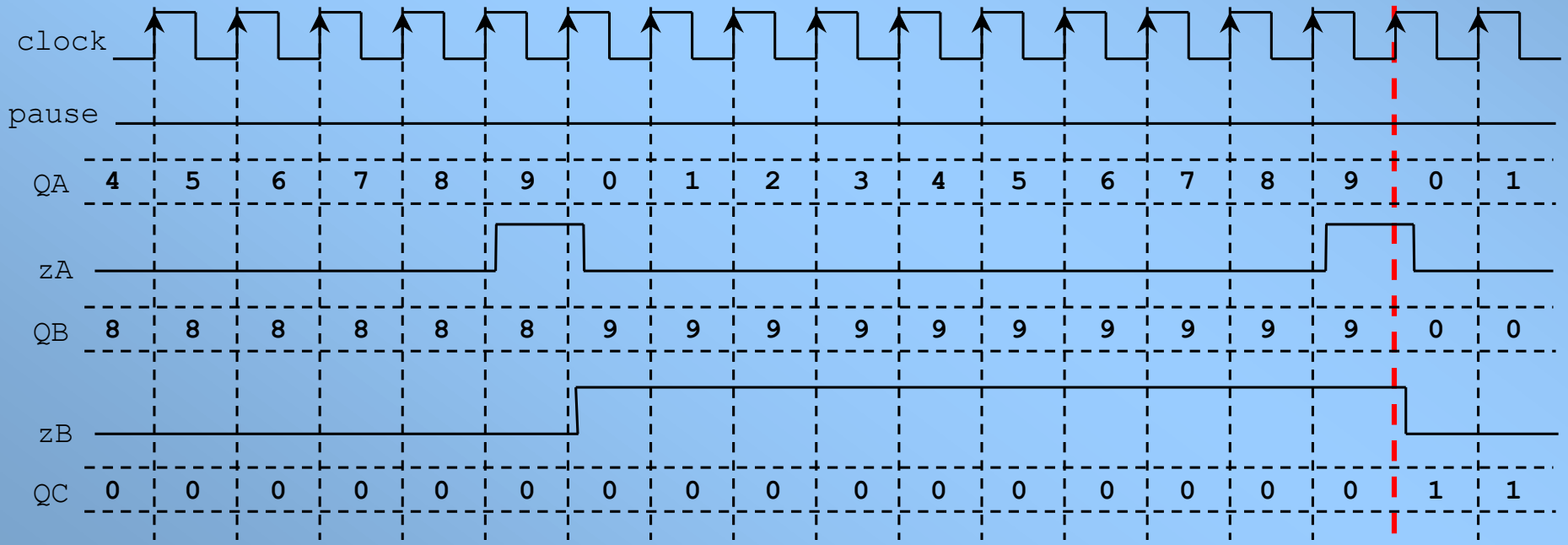
✓DATAPATH CIRCUIT

- Cascade interconnection: This allows the counters to behave as desired.



✓DATAPATH CIRCUIT

- '*pause*' input: If the user asserts this input, the count must freeze. This is achieved by using *~pause* to enable all the counters.
- Note that it is possible to come up with this circuit by designing an FSM that controls the four enable inputs of the counters.
- In the figure, note what needs to happen so that the third counter (QC) increments its count.

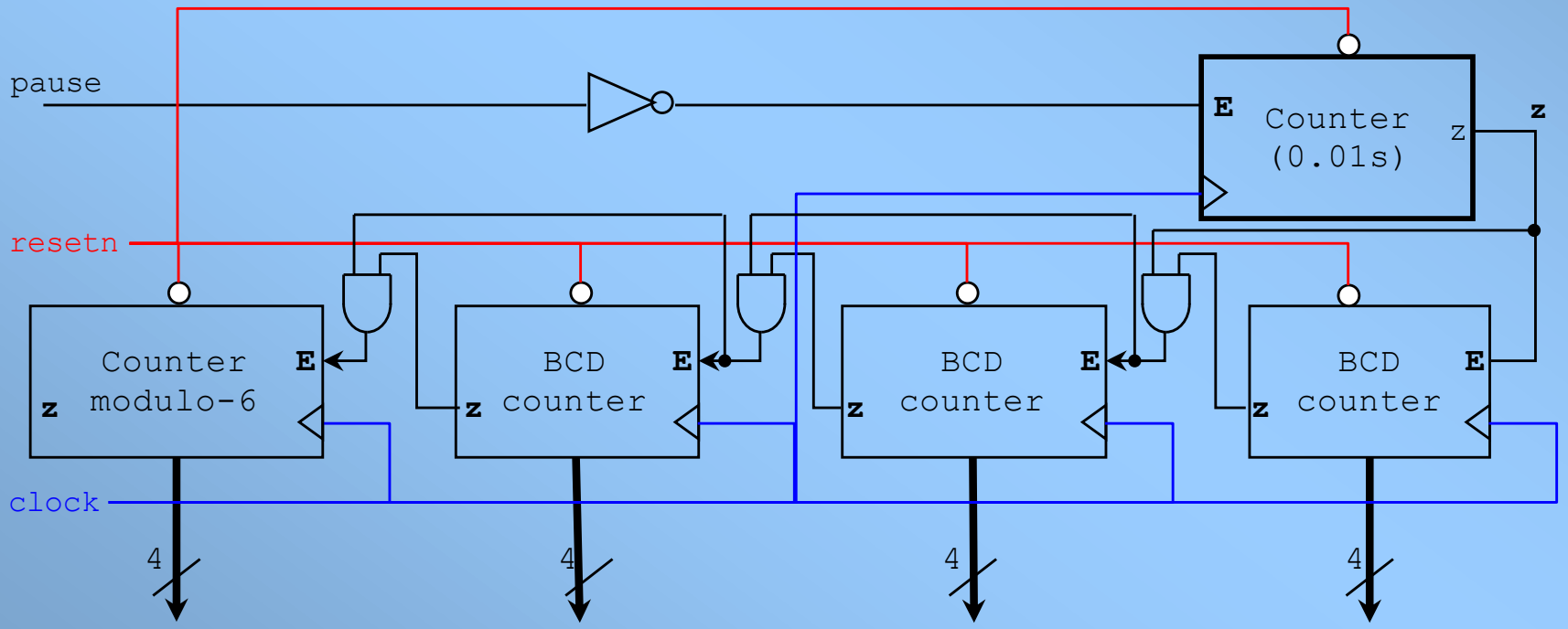


✓*STOPWATCH DESIGN*

- The NEXYS 3 boards features an input clock of 100 MHz. Thus, the counter QA will increment its count every 10 ns.
- We want QA to increment its count every 0.01s (10 ms).
- **Straightforward solution**: Change the input clock to 100 Hz (period 10 ms). This can be a hard problem if a precise input clock is required (FPGA clock skew, 100 Hz clock will not go in the clock tree, etc)
- **Efficient solution**: We create a circuit that generates a pulse every 10 ms. This output is then connected to every enable input of the counters. This way, we get the same effect as modifying the clock frequency to 100 Hz.
- The pulse is of duration of the input clock period (10 ns). To generate a pulse every 10 ms, we need a counter that counts up to (10ms/10ns) -1 = 10^6 -1. Note that our generic counter with COUNT=10^6 allows for a straightforward implementation of this circuit.

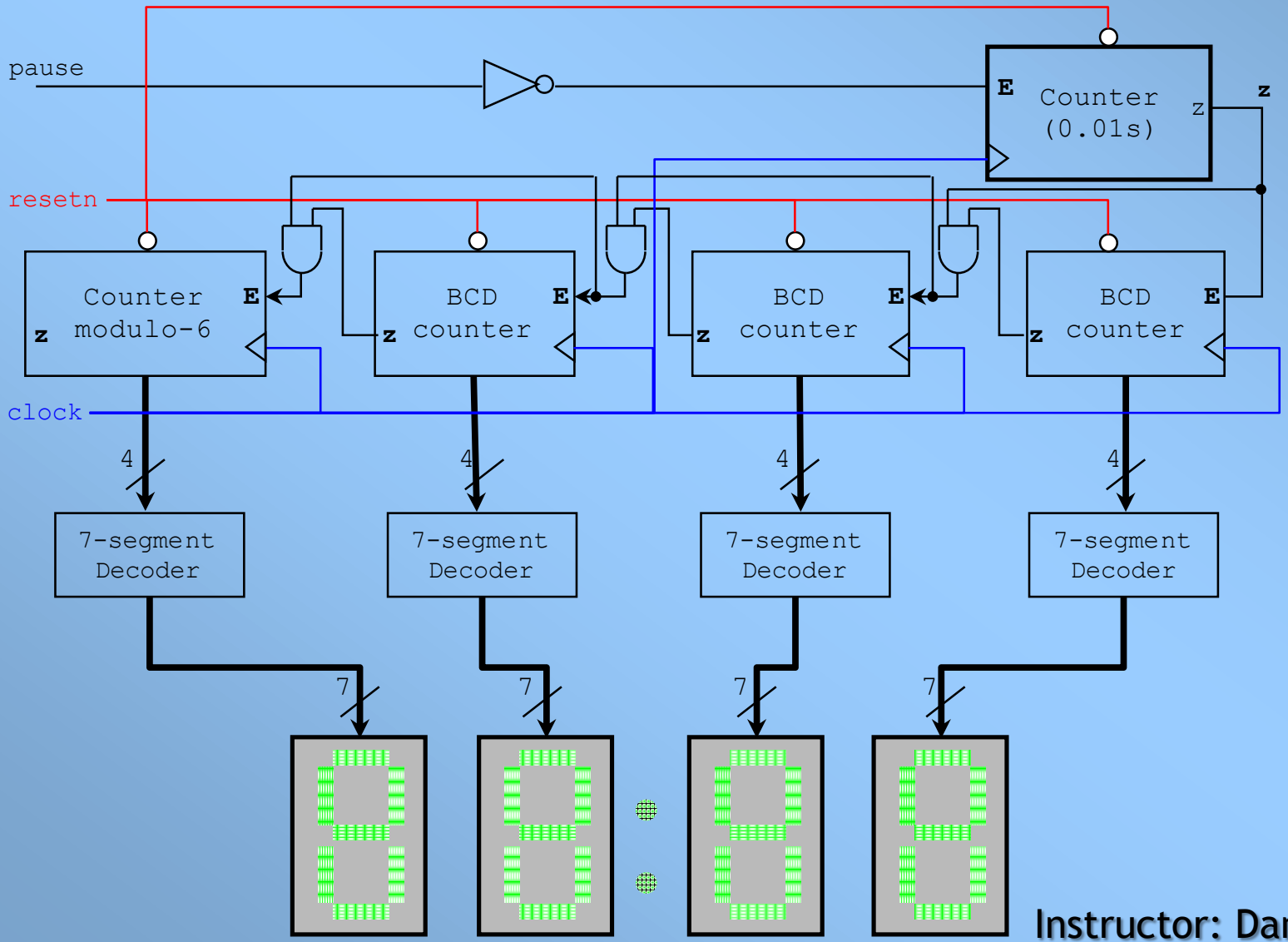
✓STOPWATCH DESIGN

- The figure shows the stopwatch with the counter up to 10^6 (named 0.01s counter).
- Note how the output ‘z’ of the 0.01s counter is now connected to all the enables in the counters. This way we make sure that every transition in the counter only occurs every 0.01 s (10 ms)



✓STOPWATCH DESIGN

- Final system with four 7-segment decoders so that the counter outputs can be seen in the displays.



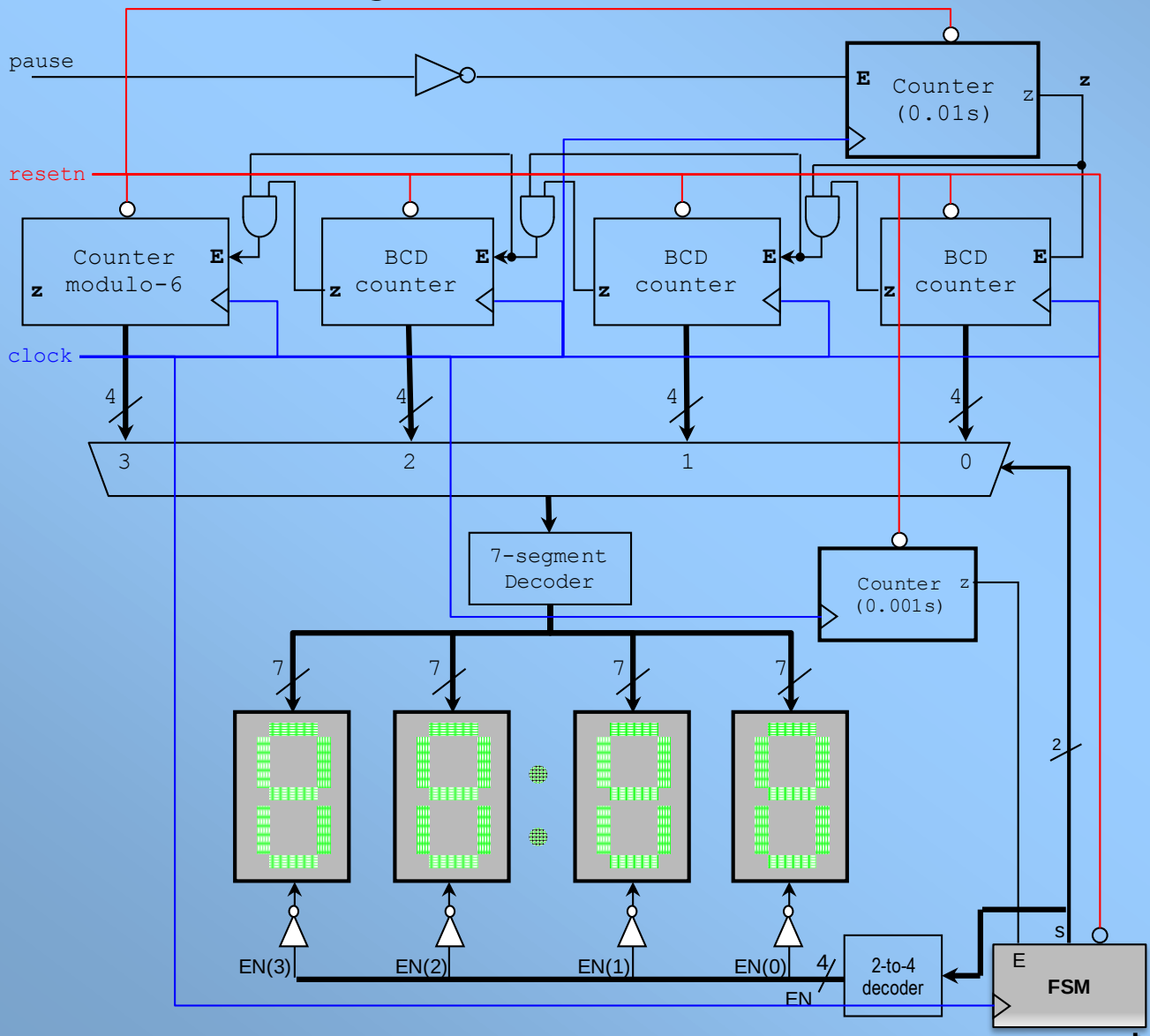
✓*STOPWATCH DESIGN*

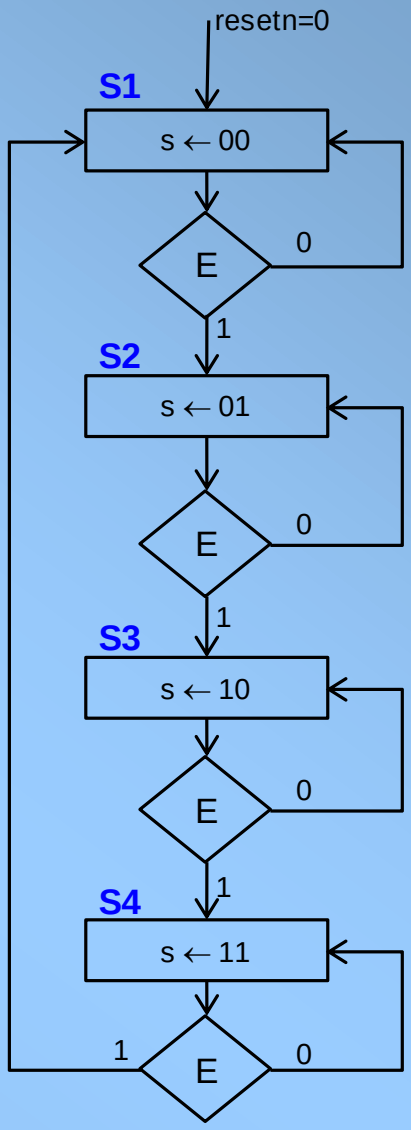
- NEXYS3 Board: It has four 7-segment displays, but all the displays share the same inputs. We can also enable (negative logic) a particular 7-segment display via the common anode (see *Seven[segment Display section* in NEXYS3 datasheet).
- *Why do we have then four 7-seg displays, if apparently all of them will display the same pattern?*

- *OUTPUT SERIALIZATION*: With an enable for each 7-segment display, we are able to use one 7-segment display at a time. In order for each digit to appear bright and continuously illuminated, we illuminate each digit for only 1ms every 4 ms.
- We need only one 7-segment decoder, a Multiplexor, and an FSM that control the selector of the multiplexor.
- If we want the multiplexor selector to transition only 1 ms every 4 ms, we feed the output ‘z’ of a new counter (to 0.001s, COUNT = 10^5) to the enable input of the FSM.

✓STOPWATCH DESIGN

- Final Design

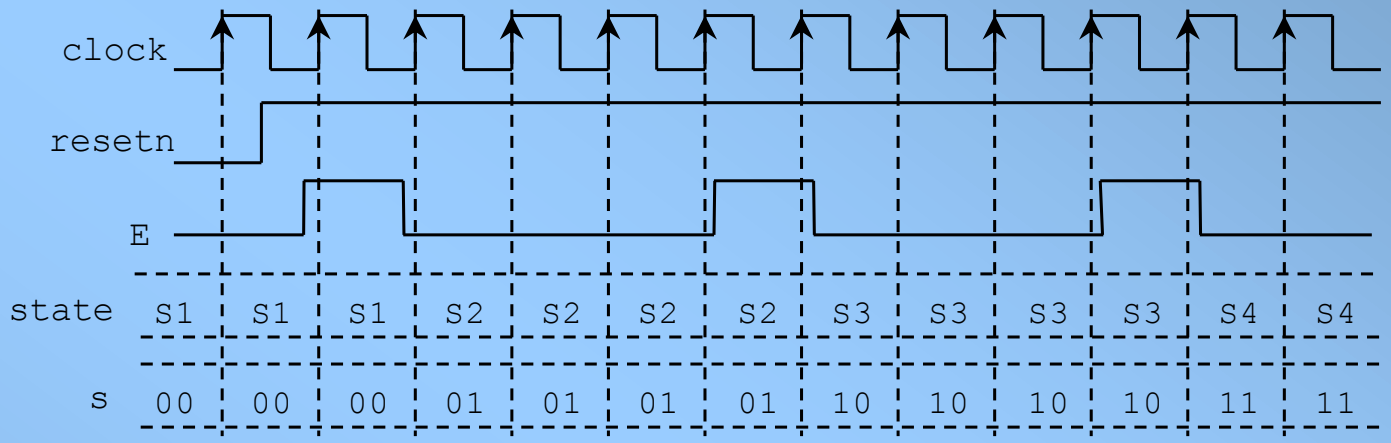




Instructor: Daniel Llamocca

✓*STOPWATCH DESIGN*

- FSM Timing Diagram:



- **Xilinx ISE Project:**
  - VHDL code: Instantiation (port map, generic map). For VHDL styling, *see other units in VHDL Workshop*.
  - Simulation: Testbench. If 0.01s counter is omitted and z = E, we can easily simulate the circuit.
  - Place-and-Route (pin assignment
  - FPGA programming and testing
  - ***More examples (with VHDL code) available at dllamocca.org***

`dig_stopwatch.vhd`

`my_genpulse.vhd`

`sevenseg.vhd`