Digital Design Laboratory

Dr. Oniga István
University of Debrecen, Faculty of Informatics

This work was supported by the construction EFOP-3.4.3-16-2016-00021.
The project was supported by the European Union, co-financed by the European Social Fund.

8. Laboratory assignments

- Counters
  - Simulations
    - 4-bit synchronous up counter with synchronous or asynchronous clear
    - 4-bit synchronous down counter with synchronous clear
    - 4-bit synchronous up/down counter
    - Decimal up counter with load
    - N-bit synchronous up counter with asynchronous clear
  - Implementations
    - Clock divider
    - 8-bit counter with LED output
    - 8-bit counter with 7 segment display

Lab8_1-Lab8_5: Synchronous counters simulation

Next description is related to Lab8_x (x = 1:5).

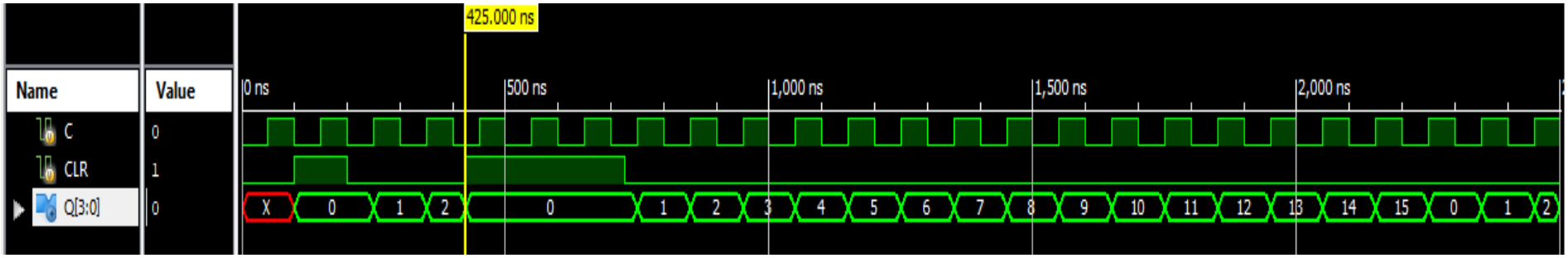
- Create a new HDL project (Lab8_x).
- Add a new Verilog source file Lab8_x.v.
- Describe the counter behavior using Verilog language .
- Add a Verilog test fixture file.
- Specify the stimulus signals.
- Simulate the circuit.

Lab8_1a: 4-bit synchronous up counter with asynchronous clear

```
//
// 4-bit up counter with an asynchronous clear.
//
module counter_1 (input C, CLR, output reg
[3:0] Q);

     always @(posedge C or posedge CLR)
     begin
          if (CLR)
                    Q <= 4'b0000;
          else
                    Q <= Q + 1'b1;
     end

endmodule
```

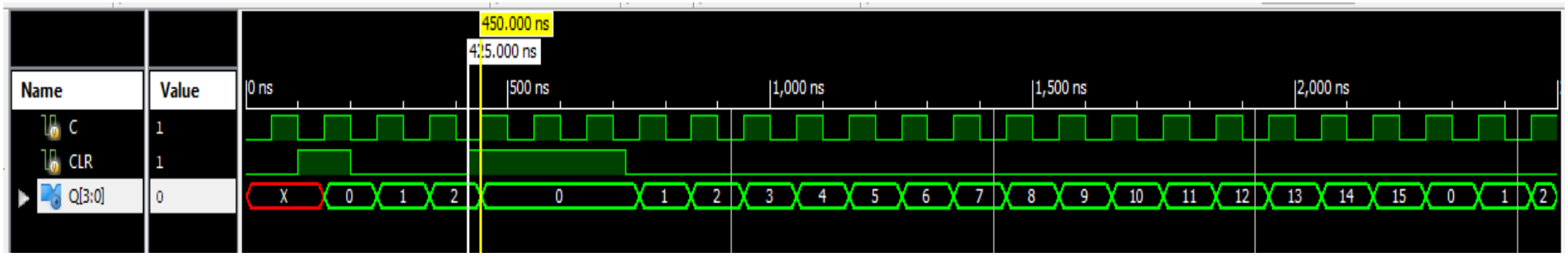
Gerjesztő jelek specifikálása

```
initial begin // Initialize Inputs
C = 0;
CLR = 0;
// Wait 100 ns for global reset to finish
#100;
// Add stimulus here
CLR = 1;
#100;
CLR = 0;
# 225
CLR = 1;
# 300
CLR = 0 ;
end
always #50
C <= ~ C;
```



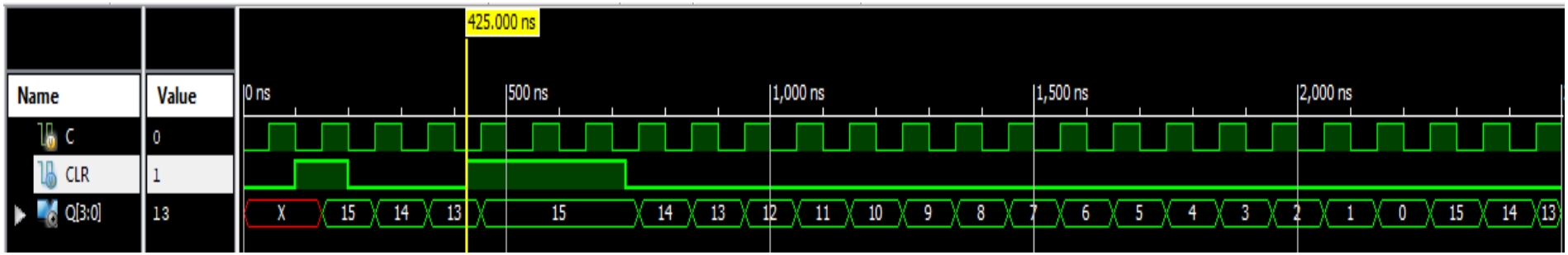
Lab8_1b: 4-bit synchronous up counter with synchronous clear

```
//
// 4-bit up counter with an synchronous clear.
//
module counter_2 (input C, CLR, output reg [3:0] Q);

     always @(posedge C)
     begin
          if (CLR)
                    Q <= 4'b0000;
          else
                    Q <= Q + 1'b1;
     end

endmodule
```



Lab8_2: 4-bit synchronous down counter with synchronous clear

```
module counter_3 (input C, CLR, output reg [3:0] Q);
     always @(posedge C)
     begin
          if (CLR)
                    Q <= 4'b1111;
          else
                    Q <= Q - 1'b1;
     end
endmodule
```

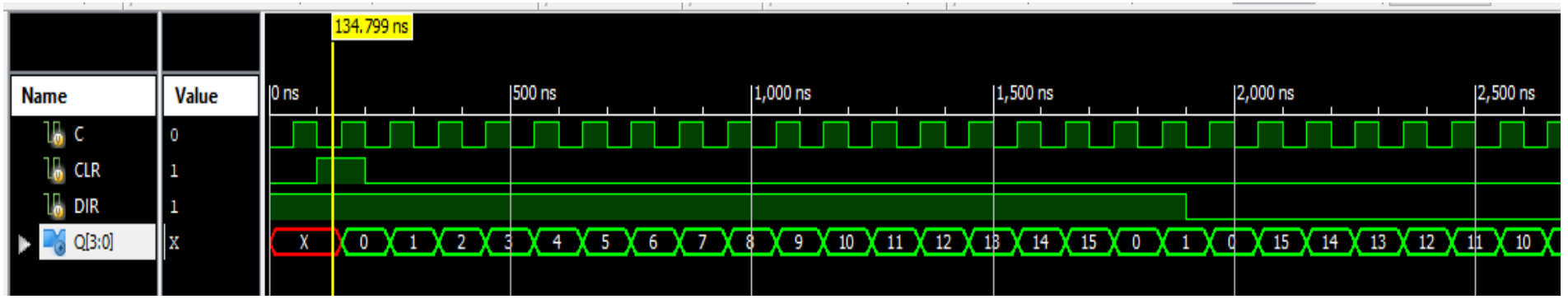


Lab8_3: 4-bit synchronous up/down counter

```
module counter_3 (input C, CLR, DIR, output
reg [3:0] Q);
     always @(posedge C)
     begin
          if (CLR)
                    Q <= 4'b0000;
          else if (DIR)
                    Q <= Q + 1'b1;
          else
                    Q <= Q - 1'b1;
     end
endmodule
```

Gerjesztő jelek specifikálása

```
initial begin
    // Initialize Inputs
    C = 0;
    CLR = 0;
    DIR = 1;

    // Wait 100 ns for global reset to finish
    #100;
    // Add stimulus here
    CLR = 1;
    #100;
    CLR = 0;
    #1700;
    DIR=0;

end

always #50
    C <= ~ C;
```

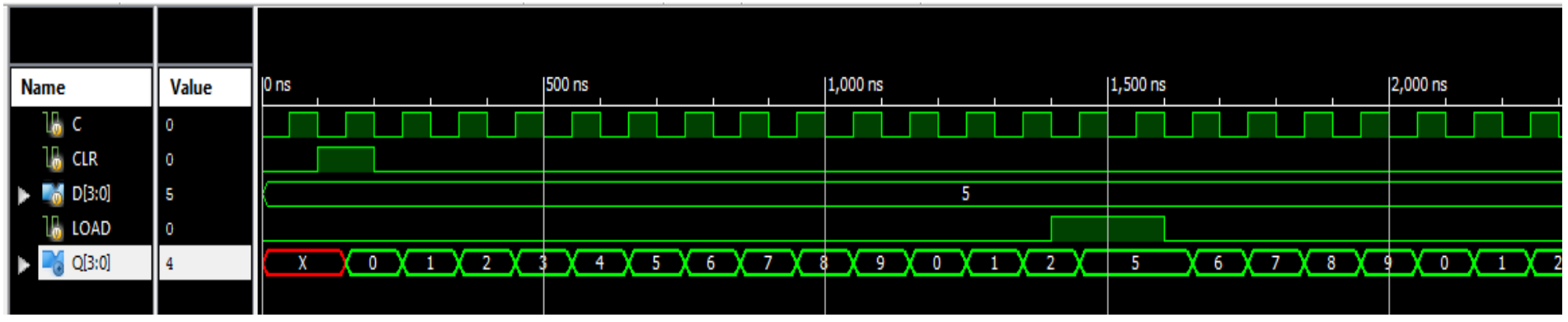


Lab8_4: Decimal up counter with load

```
module counter_5 (input C, CLR, LOAD, input [3:0]
D, output reg [3:0] Q);

assign q9 = (Q== 4'd9); //assign q12 = (Q== 4'd12);

always @(posedge C)
          begin
          if (CLR | q9)
                    Q <= 4'b0000;
          else if (LOAD) // (LOAD==1)
                    Q <= D; //vagy konstanssal;
          else
                    Q <= Q + 1'b1;
          end
endmodule
```

Gerjesztő jelek specifikálása

```
initial begin
    // Initialize Inputs
    C = 0;
    CLR = 0;
    LOAD = 0;
    D = 4'b0101;;

    // Wait 100 ns for global reset
    #100;
    // Add stimulus here
    CLR = 1;
    #100;
    CLR = 0;
    #1200;
    LOAD = 1;
    # 200;
    LOAD = 0;

end

always #50
C <= ~ C;
```

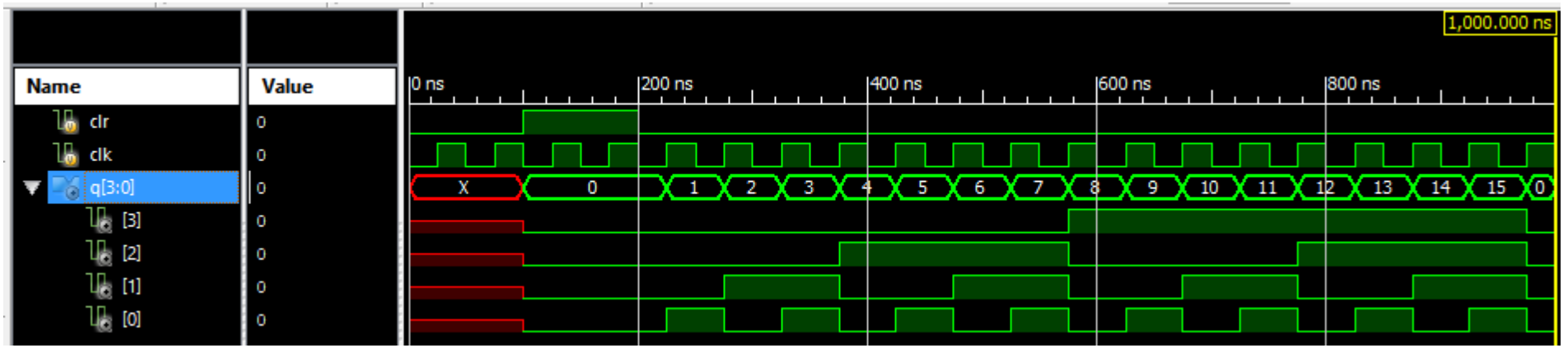


Lab8_5: N-bit synchronous up counter with asynchronous clear

```
module counter
#(parameter N = 4)
   (input wire clr , clk ,
   output reg [N-1:0] q );
// N-bit counter

always @(posedge clk or posedge clr)
   begin
      if(clr == 1)
         q <= 0;
      else
         q <= q + 1;
   end
endmodule
```

Gerjesztő jelek specifikálása


```
initial begin
   // Initialize Inputs
   clr = 0;
   clk = 0;

   // Wait 100 ns for global reset to finish
   #100 clr = 1;
   #100 clr = 0;

   // Add stimulus here

end
   always #25
   clk <= ~clk;
```



The simulation of the N-bit counter shows that the outputs *q*[*i*] of a counter are square waves where the output *q*[0] has a frequency half of the clock frequency, the output *q*[1] has a frequency half of *q*[0], etc.

Lab8_6: Clock divider

- The Nexys-4 board has an onboard 100 MHz clock (clk100mhz)
- In this example we will show how to design an N-bit counter in Verilog and how to use a counter to generate clock signals of lower frequencies
- Each FF from the componence of the counter will divide the input frequency by 2:
  - q[0]= clk/2; q[1]= q[0]/2 …
  - We will use a 27-bit counter to divide the clock

```
module clkdiv ( input clk, clr, output
clk1, clk2, clk3 );
reg [26:0] q;
// 27-bit counter
always @(posedge clk or posedge clr)
     begin
          if(clr == 1)
               q <= 0;
          else
               q <= q + 1;
          end
assign clk1 = q[26]; // ~0.75 Hz
assign clk2 = q[25]; // ~1.5
assign clk3 = q[24]; // ~3 Hz
endmodule
```

Q(i)	Frecquency (Hz)	Period (ms)
0	50000000.00	0.00002
1	25000000.00	0.00004
2	12500000.00	0.00008
3	6250000.00	0.00016
4	3125000.00	0.00032
5	1562500.00	0.00064
6	781250.00	0.00128
7	390625.00	0.00256
8	195312.50	0.00512
9	97656.25	0.01024
10	48828.13	0.02048
11	24414.06	0.04096
12	12207.03	0.08192
13	6103.52	0.16384
14	3051.76	0.32768
15	1525.88	0.65536
16	762.94	1.31072
17	381.47	2.62144
18	190.73	5.24288
19	95.37	10.48576
20	47.68	20.97152
21	23.84	41.94304
22	11.92	83.88608
23	5.96	167.77216
24	2.98	335.54432
25	1.49	671.08864
26	0.745	1342.17728

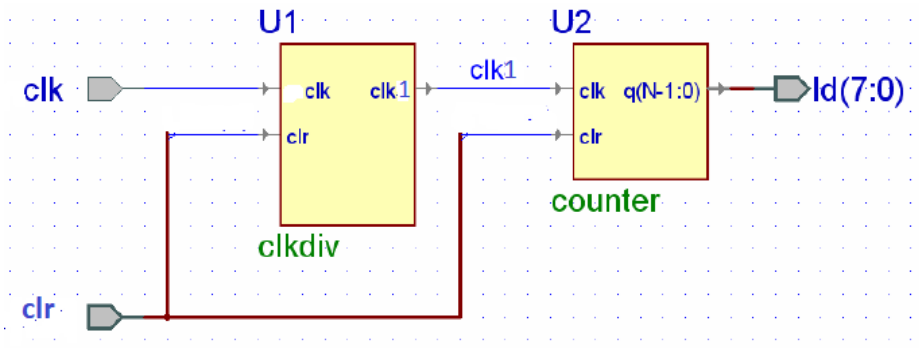
Lab8_6: Clock divider implementation

- Add an ucf file to the project
  - Inputs
    - clk -> clk100mhz, clr -> btnc,
  - Outputs
    - clk1 -> led<0>, clk2 -> led<1>, clk3 -> led<2>.

```
## Clock signal
NET "clk"   LOC = "E3"  | IOSTANDARD = "LVCMOS33"; #Bank = 35, Pin name =
NET "clk" TNM_NET = sys_clk_pin;
TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 100 MHz HIGH 50%;

## Buttons
NET "clr"           LOC=N17 | IOSTANDARD=LVCMOS33; #IO_L9P_T1_DQS_14

## LEDs
NET "clk1"         LOC=H17 | IOSTANDARD=LVCMOS33; #IO_L18P_T2_A24_15
NET "clk2"         LOC=K15 | IOSTANDARD=LVCMOS33; #IO_L24P_T3_RS1_15
NET "clk3"         LOC=J13 | IOSTANDARD=LVCMOS33; #IO_L17N_T2_A25_15
```

- Generate the configuration file, download to board and test.

Lab8_7: 8-bit counter with LED output




```
module count8_top (input clk, input  clr,
output [7:0] led) ;

wire clk1;

clkdiv
U1(.clk1(clk1),.clr(clr),.clk(clk));

counter #( .N(8)) U2
(.clk(clk1), .clr(clr), .q(led[7:0]));
endmodule
```

- Create a new project (Lab8_7)
- Add the modules clkdiv and counter created in the previous assignment.
- Add a new VERILOG file (Lab8_7.v), this will be the top module that will connect the 2 modules as in the figure.
- Add the ucf file specifying the
  - Inputs: clk -> clk100mhz, clr -> btnc,
  - Outputs: led -> led.
- Generate the configuration file, download to board and test.
  - Observe the binary counting sequence on the leds.
  - Try to clear the counter using btnc

Lab8_8: 8-bit counter with 7-segments output

- Create a new project (Lab8_8)
- Add the previously created clkdiv and counter modules.
- Add a copy of the hex7seg module created in the previous assignment.
- Add a new VERILOG file (Lab8_8.v), this will be the top module that will connect the 3 modules added to the project.
- Add the user constraints file specifying the
  - Inputs: clk, clr,
  - Outputs: an, a_to_g, dp, led.
- Generate the configuration file, download to board and test.
  - Observe the binary counting sequence on the leds.
  - Try to clear the counter using btnc

```
module count4_top (input clk, input  clr, output [7:0] led,
          output [6:0] a_to_g, output [7:0] an, output dp ) ;

wire clk1;

clkdiv U1(.clr(clr),.clk(clk), .clk1(clk1));
counter #( .N(8)) U2 (.clk(clk1), .clr(clr), .q(led[7:0]));
hex7seg U3 (.x(led[3:0]), .a_to_g(a_to_g));

assign an = 8'b11111110;
assign dp = 1; // dp off

endmodule
```