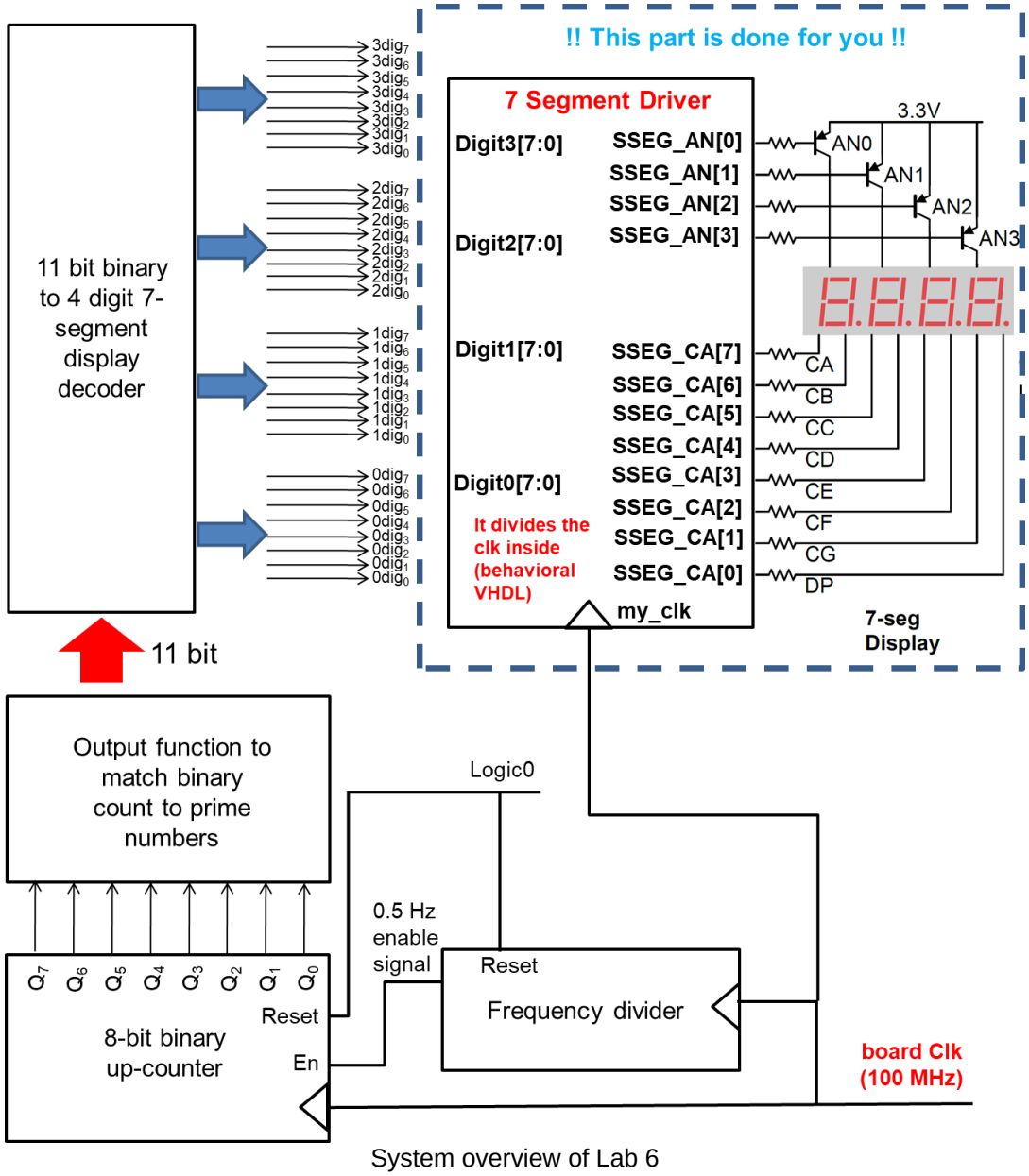
LAB6: 8-BIT COUNTER DESIGN TO DISPLAY THE FIRST 256 PRIME NUMBERS AND ITS FPGA IMPLEMENTATION

The purpose of this lab is to design a 4-digit BCD up-counter. Counting up is done automatically at every 2 second (0.5 Hz). You can do this by designing a state machine or an 8-bit binary up-counter. The outputs of this binary counter will be prime numbers. Instead of showing the count number, your design will show the prime number corresponding to that count. Prime numbers are the numbers that can be divided only by itself and 1. Number 1 is not a prime number. The 4 digit BCD will display the first 256 prime number values (from 2 to 1619). Then, it will roll over the beginning and continue to count. This counter is implemented on the FPGA board. Overall schematic for the system is shown below.



System overview of Lab 6

FPGA Implementation of the Up/Down Counter

The purpose of this laboratory assignment is to design, synthesize and implement a state machine, which has outputs to be displayed on 4-digit 7-segment LED display. In this assignment you are supposed to show that you can do up-counting, and implement an output function. Unfortunately, we cannot use push button switches on the Nexys3 board due to remote lab access. Hence, we cannot reset the system and put the circuit into a known state. Therefore, make sure that you also account for "don't care" states. If your design wakes in an unknown state (D flip-flop values which do not form desired next states), they will generate valid next state values in couple clock cycles. After loading your programming file to your FPGA board, the system will start randomly (maybe pseudo-randomly) at a count. The starting count is not important in this lab since we are not implementing any reset signal.

You should implement an 11-bit output function to map the current count state to given prime numbers. Then, you should implement a 11-bit binary to 4-digit BCD decoder and/our BCD to 7-segment decoder as you have done in Lab5. 4-digit BCD content should be displayed on the four digits of the seven-segment display of the Nexys3 board similar to what you have done in Lab5. The details are given below.

8-bit Up-Counter

Your counter should count up or your state machine should have 256 states that you automatically move from one to another in order at every clock edge. Every state should generate a specific 11-bit binary value. This value should be the corresponding prime number in the given order in the table below. For example, the first count state should generate output 1 and the 256th count state should generate output 1619 as 11-bit binary numbers. Then, you can design an 11-bit binary to BCD converter.

Count Order	Prime Number	Count Order	Prime Number	Count Order	Prime Number	Count Order	Prime Number
1	2	65	313	129	727	193	1171
2	3	66	317	130	733	194	1181
3	5	67	331	131	739	195	1187
4	7	68	337	132	743	196	1193
5	11	69	347	133	751	197	1201
6	13	70	349	134	757	198	1213
7	17	71	353	135	761	199	1217
8	19	72	359	136	769	200	1223
9	23	73	367	137	773	201	1229
10	29	74	373	138	787	202	1231
11	31	75	379	139	797	203	1237
12	37	76	383	140	809	204	1249
13	41	77	389	141	811	205	1259
14	43	78	397	142	821	206	1277
15	47	79	401	143	823	207	1279
16	53	80	409	144	827	208	1283
17	59	81	419	145	829	209	1289
18	61	82	421	146	839	210	1291
19	67	83	431	147	853	211	1297
20	71	84	433	148	857	212	1301

21	73	85	439	149	859	213	1303
22	79	86	443	150	863	214	1307
23	83	87	449	151	877	215	1319
24	89	88	457	152	881	216	1321
25	97	89	461	153	883	217	1327
26	101	90	463	154	887	218	1361
27	103	91	467	155	907	219	1367
28	107	92	479	156	911	220	1373
29	109	93	487	157	919	221	1381
30	113	94	491	158	929	222	1399
31	127	95	499	159	937	223	1409
32	131	96	503	160	941	224	1423
33	137	97	509	161	947	225	1427
34	139	98	521	162	953	226	1429
35	149	99	523	163	967	227	1433
36	151	100	541	164	971	228	1439
37	157	101	547	165	977	229	1447
38	163	102	557	166	983	230	1451
39	167	103	563	167	991	231	1453
40	173	104	569	168	997	232	1459
41	179	105	571	169	1009	233	1471
42	181	106	577	170	1013	234	1481
43	191	107	587	171	1019	235	1483
44	193	108	593	172	1021	236	1487
45	197	109	599	173	1031	237	1489
46	199	110	601	174	1033	238	1493
47	211	111	607	175	1039	239	1499
48	223	112	613	176	1049	240	1511
49	227	113	617	177	1051	241	1523
50	229	114	619	178	1061	242	1531
51	233	115	631	179	1063	243	1543
52	239	116	641	180	1069	244	1549
53	241	117	643	181	1087	245	1553
54	251	118	647	182	1091	246	1559
55	257	119	653	183	1093	247	1567
56	263	120	659	184	1097	248	1571
57	269	121	661	185	1103	249	1579
58	271	122	673	186	1109	250	1583
59	277	123	677	187	1117	251	1597
60	281	124	683	188	1123	252	1601
61	283	125	691	189	1129	253	1607
62	293	126	701	190	1151	254	1609
63	307	127	709	191	1153	255	1613
64	311	128	719	192	1163	256	1619

A behavioral VHDL code for an up-counting 8-bit binary counter is given below. You can modify this code to realize the desired counting function.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;


entity my_count8 is
  port( D: in std_logic_vector(7 downto 0);
        Q: out std_logic_vector(7 downto 0);
        CLK, clk_enable, RESET : in std_logic );
end count4;

ARCHITECTURE behavioral OF my_count8 IS
SIGNAL count : std_logic_vector (7 downto 0);
BEGIN

my_counter: PROCESS(clk, reset)
BEGIN

IF (reset = '0') THEN
  count <= (others => '0');
--This assigns ‘0’ to all bits of count signal. “Others” means other values of count
--This is a generic usage since signal assignment becomes independent of the
--vector size of count. i.e. you do not need to change this line if count becomes 16-bit wide
elsif(clock'event and clock='1') then
            if clock_enable='1' then
                  count <= count + “00000001”;
            else count <= count;
            end if;
end if;
END PROCESS my_counter;
Q <= count; -- assign count signal to output
END behavioral;
```

Writing the Counter Content to the LED Display

As done before in Lab 5, the contents of the counter are displayed on the four-digit-seven-segment LED display on the FPGA board using Digit0 through Digit3. In order to do that, an 11-bit binary to 4-digit BCD decoder and a BCD-to-7-segment decoder are necessary. Alternatively, you can do 11-bit binary to 4-digit 7-segment display conversion inside a single block.

We want to display four digits. Thus, your design should generate 32 bit output. The first 8 bit (least significant digit) is connected to Digit0. The second 8 bit is connected to Digit1, so on, so forth. The entity of your design should have four sets of 8-bit output ports. These outputs are connected to the seven-segment display driver, *nexys3_sseg_driver.vhd*, given and explained in Lab 5.

Overall Design

As explained above your design will implement an 8-bit up-counter. This will a slow counter with a frequency of 0.5 Hz. You will, then, map these counter’s current state values to 11-bit binary values which represent prime numbers in order. Finally, you will display these ordered prime number values as 4 decimal digit on 7-segment displays. Your design will generate 8-bit digit3 output, 8-bit digit2 output, 8-bit digit1 output and 8-

bit digit0 output. Then, these outputs will be inputs to the design given in the *nexys3_sseg_driver.vhd* file using structural VHDL style. This design will generate the SSEG_CA(7 downto 0) and SSEG_AN(3 downto 0) outputs using the clock signal it generates from the board clock. As a starting point your VHDL code for this design can have a structure as shown below. You should complete its architecture. Your VHDL code can use structural and/or dataflow and/or behavioral style VHDL.

```
-- Synthesizable 4-digit prime number displayer, EE240 class Bogazici University
-- Implemented on Xilinx Spartan VI FPGA chip

LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_signed.all ;
USE ieee.std_logic_arith.all ;

entity prime_number_displayer is
        port (
                MY_CLK: in std_logic;
                SSEG_CA: out std_logic_vector(7 downto 0);
                SSEG_AN: out std_logic_vector(3 downto 0));
end;

architecture arch_displayer of prime_number_displayer is

--MY_CLK is coming from the board clock, connected to V10

--digit3(0) is displayed on Digit0 Segment CA
--digit3(1) is displayed on Digit0 Segment CB
--digit3(2) is displayed on Digit0 Segment CC
--digit3(3) is displayed on Digit0 Segment CD
--digit3(4) is displayed on Digit0 Segment CE
--digit3(5) is displayed on Digit0 Segment CF
--digit3(6) is displayed on Digit0 Segment CG
--digit3(7) is displayed on Digit0 Segment DP, which is always logic 1

--digit2(0) is displayed on Digit0 Segment CA
--digit2(1) is displayed on Digit0 Segment CB
--digit2(2) is displayed on Digit0 Segment CC
--digit2(3) is displayed on Digit0 Segment CD
--digit2(4) is displayed on Digit0 Segment CE
--digit2(5) is displayed on Digit0 Segment CF
--digit2(6) is displayed on Digit0 Segment CG
--digit2(7) is displayed on Digit0 Segment DP, which is always logic 1

--digit1(0) is displayed on Digit0 Segment CA
--digit1(1) is displayed on Digit0 Segment CB
--digit1(2) is displayed on Digit0 Segment CC
--digit1(3) is displayed on Digit0 Segment CD
--digit1(4) is displayed on Digit0 Segment CE
--digit1(5) is displayed on Digit0 Segment CF
--digit1(6) is displayed on Digit0 Segment CG
--digit1(7) is displayed on Digit0 Segment DP, which is always logic 1

--digit0(0) is displayed on Digit0 Segment CA
--digit0(1) is displayed on Digit0 Segment CB
```

```
--digit0(2) is displayed on Digit0 Segment CC
--digit0(3) is displayed on Digit0 Segment CD
--digit0(4) is displayed on Digit0 Segment CE
--digit0(5) is displayed on Digit0 Segment CF
--digit0(6) is displayed on Digit0 Segment CG
--digit0(7) is displayed on Digit0 Segment DP, which is always logic 1

--digit3(7 downto 0), digit2(7 downto 0), digit1(7 downto 0) and digit0(7 downto 0) will be
-- input to the design given in the nexys3_sseg_driver.vhd file using structural vhdl style.
-- This design will generate the SSEG_CA(7 downto 0) and SSEG_AN(3 downto 0)
-- outputs using the board clock signal.

end arch_displayer;
```

Pin Assignment

After successfully completing the design, compilation, ISim simulations and synthesis, you should show that your design work on the FPGA board. By looking at the manual of the Nexys3 Board, we can find which FPGA pins are connected to which components.

The .ucf file should be like this.

```
#
# Pin assignments for the Nexys3 Spartan VI Board.
#
Net SSEG_CA<0> LOC=T17  |  IOSTANDARD=LVCMOS33; # Connected to CA
Net SSEG_CA<1> LOC=T18  |  IOSTANDARD=LVCMOS33; # Connected to CB
Net SSEG_CA<2> LOC=U17  |  IOSTANDARD=LVCMOS33; # Connected to CC
Net SSEG_CA<3> LOC=U18  |  IOSTANDARD=LVCMOS33; # Connected to CD
Net SSEG_CA<4> LOC=M14  |  IOSTANDARD=LVCMOS33; # Connected to CE
Net SSEG_CA<5> LOC=N14  |  IOSTANDARD=LVCMOS33; # Connected to CF
Net SSEG_CA<6> LOC=L14  |  IOSTANDARD=LVCMOS33; # Connected to CG
Net SSEG_CA<7> LOC=M13  |  IOSTANDARD=LVCMOS33; # Connected to DP

Net SSEG_AN<0> LOC=N16  |  IOSTANDARD=LVCMOS33; # Connected to AN0
Net SSEG_AN<1> LOC=N15  |  IOSTANDARD=LVCMOS33; # Connected to AN1
Net SSEG_AN<2> LOC=P18  |  IOSTANDARD=LVCMOS33; # Connected to AN2
Net SSEG_AN<3> LOC=P17  |  IOSTANDARD=LVCMOS33; # Connected to AN3

Net MY_CLK LOC=V10  |  IOSTANDARD=LVCMOS33; # Connected to board clock
```

Preparation (Prelab)

For Frequency Divider and 8-bit Counter Circuit

- Write a VHDL code for these components.
- Run ISim and verify that they function as expected by running test sequences.

For 8-bit binary to 11-bit prime number values

- Write a VHDL code for this component.
- Run ISim and verify that it functions as expected by running a few test sequences.

For 11-bit binary to 4-digit BCD and then to 4-digit 7-Segment Conversion

- Write a VHDL code for this component.

- Run ISim and verify that it functions as expected by running a few test sequences.

For the Overall Design

- Combine the design components and run ISim on it and verify that it functions as expected by running a few test sequences.

In Lab

- Use Xilinx ISE to synthesize the design based on Xilinx Spartan VI family. Use either the given ucf file or enter pins manually to make pin assignment.
  - Generate the bit file and upload it to your FPGA.
  - On the hardware show that your design works.
  - See if the 4-digits of 7-segment LED display show correct prime number values in order every 2 second.

References

[1] "Nexys 3 Manual." (accessed 18 March 2021).