# Lab 2

**Top file:**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity LogicalStep\_Lab2\_top

is port (

clkin\_50 : in std\_logic;

pb : in std\_logic\_vector(3 downto 0); --Push Button Inputs

sw : in std\_logic\_vector(7 downto 0); --Switch Inputs

leds : out std\_logic\_vector(7 downto 0); --Displaying the switch content

seg7\_data : out std\_logic\_vector(6 downto 0); --7-bit outputs to a 7-Segment

seg7\_char1 : out std\_logic; --Seg7 digit1 selector

seg7\_char2 : out std\_logic --Seg7 digit2 selector

);

end LogicalStep\_Lab2\_top;

architecture difficultCircuit of LogicalStep\_Lab2\_top is

--COMPONENTS USED --

-------------------------------------------------------------------------------------------

--COMPONENT SevenSegment

component SevenSegment port(

hex : in std\_logic\_vector(3 downto 0);

sevenseg: out std\_logic\_vector(6 downto 0)

);

end component;

--COMPONENT Segment7\_mux

component segment7\_mux port (

clk : in std\_logic := '0';

DIN2 : in std\_logic\_vector(6 downto 0);

DIN1 : in std\_logic\_vector(6 downto 0);

DOUT : out std\_logic\_vector(6 downto 0);

DIG2 : out std\_logic;

DIG1 : out std\_logic

);

end component;

-- COMPONENT new\_mux

component new\_mux port (

conc\_hex : in std\_logic\_vector(7 downto 0); -- concatenated 8 bit hex\_A and hex\_B

sum: in std\_logic\_vector(7 downto 0); --8 bit sum from adder

pb3 : in std\_logic\_vector(3 downto 0); --control variable

mux\_out : out std\_logic\_vector(7 downto 0) --8 bit output

);

end component;

--COMPONENT adder

component adder port (

A : in std\_logic\_vector(3 downto 0);

B : in std\_logic\_vector(3 downto 0);

sum: out std\_logic\_vector(7 downto 0)

);

end component;

--COMPONENT logic\_processor

component logic\_processor port (

HexA : in std\_logic\_vector(3 downto 0);

HexB : in std\_logic\_vector(3 downto 0);

pb : in std\_logic\_vector(3 downto 0);

logic\_out : out std\_logic\_vector(3 downto 0)

);

end component;

--COMPONENT digitsplit

component digitsplit port (

mux\_vector : in std\_logic\_vector(7 downto 0);

tens : out std\_logic\_vector(3 downto 0);

ones : out std\_logic\_vector(3 downto 0)

);

end component;

--COMPONENT led

component led port (

mux : in std\_logic\_vector (7 downto 0);

led\_out : out std\_logic\_vector (7 downto 0)

);

end component;

--SIGNAL CREATION--

----------------------------------------------------------------------------

signal seg7\_A : std\_logic\_vector(6 downto 0);

signal hex\_A : std\_logic\_vector(3 downto 0);

signal seg7\_B : std\_logic\_vector(6 downto 0);

signal hex\_B : std\_logic\_vector(3 downto 0);

signal pb\_bar : std\_logic\_vector(3 downto 0);

signal sum\_adder: std\_logic\_vector(7 downto 0);

signal mux\_result : std\_logic\_vector(7 downto 0);

signal ten : std\_logic\_vector(3 downto 0);

signal one : std\_logic\_vector(3 downto 0);

signal logic\_result : std\_logic\_vector(3 downto 0);

signal led\_muxout : std\_logic\_vector(7 downto 0);

--Here the circuit begins

begin

hex\_A <= sw(3 downto 0);

hex\_B <= sw(7 downto 4);

pb\_bar <= NOT(pb);

-- Component Hookup

INST1: adder port map (hex\_A, hex\_B, sum\_adder);

INST2: new\_mux port map ((hex\_A & hex\_B), sum\_adder, pb\_bar, mux\_result);

INST3: digitsplit port map (mux\_result, ten, one);

INST4: SevenSegment port map (ten, seg7\_A);

INST5: SevenSegment port map (one, seg7\_B);

INST6: segment7\_mux port map (clkin\_50, seg7\_A, seg7\_B, seg7\_data, seg7\_char1, seg7\_char2);

INST7: logic\_processor port map (hex\_A, hex\_B, pb\_bar, logic\_result);

INST8: new\_mux port map (("0000" & logic\_result), sum\_adder, pb\_bar, led\_muxout);

INST9: led port map (led\_muxout, leds);

end difficultCircuit;

**Seven Segment:**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

-------------------------------------------------------------------------

-- 7-segment display driver. It displays a 4-bit number on a 7-segment

-- This is created as an entity so that it can be reused many times easily

--

entity SevenSegment is port (

hex : in std\_logic\_vector(3 downto 0); -- The 4 bit data to be displayed

sevenseg : out std\_logic\_vector(6 downto 0) -- 7-bit outputs to a 7-segment

);

end SevenSegment;

architecture Behavioral of SevenSegment is

--

-- The following statements convert a 4-bit input, called dataIn to a pattern of 7 bits

-- The segment turns on when it is '1' otherwise '0'

--

begin

with hex select --GFEDCBA 3210 -- data in

sevenseg <= "0111111" when "0000", -- [0]

"0000110" when "0001", -- [1]

"1011011" when "0010", -- [2] +---- a -----+

"1001111" when "0011", -- [3] | |

"1100110" when "0100", -- [4] | |

"1101101" when "0101", -- [5] f b

"1111101" when "0110", -- [6] | |

"0000111" when "0111", -- [7] | |

"1111111" when "1000", -- [8] +---- g -----+

"1101111" when "1001", -- [9] | |

"1110111" when "1010", -- [A] | |

"1111100" when "1011", -- [b] e c

"1011000" when "1100", -- [c] | |

"1011110" when "1101", -- [d] | |

"1111001" when "1110", -- [E] +---- d -----+

"1110001" when "1111", -- [F]

"0000000" when others; -- [ ]

end architecture Behavioral;

**Led:**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

--LEDS - Displaying binary value of logical operations or arithmetic addition

entity led is port (

mux : in std\_logic\_vector (7 downto 0); -- Result of Logical Operations OR Result of Binary Addtion of Operands

led\_out : out std\_logic\_vector (7 downto 0) -- LED Display of value

);

end entity led;

architecture led of led is

begin

led\_out(7 downto 0) <= mux(7 downto 0);

end led; --LEDS

**Adder:**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

--ADDER - Arithmetic Addition of OP1 And OP2 Displayed on Dual Seven Segment Display

entity adder is

port(

A : in std\_logic\_vector(3 downto 0);--Operand 1

B : in std\_logic\_vector(3 downto 0);--Operand 2

sum: out std\_logic\_vector(7 downto 0) --Output: Sum of Operands 1 and 2

);

end adder;

architecture adder of adder is

begin

sum (7 downto 0)<= std\_logic\_vector(unsigned("0000" & A) + unsigned("0000" & B));

end adder;

--ADDER

**DigitSplit**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

--DIGITSPLIT - Splitting 8-bit value into 4-bit values (splitting decimal value into tens and ones)

entity digitsplit is port (

mux\_vector : in std\_logic\_vector(7 downto 0); --Concantenated HexA, HexB OR Sum of Operands

tens : out std\_logic\_vector(3 downto 0); --Decimal tens value OR HexA

ones : out std\_logic\_vector(3 downto 0) --Decimal ones value OR HexB

);

end entity digitsplit;

architecture digitsplit of digitsplit is

begin

tens <= mux\_vector(7 downto 4);

ones <= mux\_vector(3 downto 0);

end digitsplit;

--DIGITSPLIT

**Logic processor:**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

--LOGIC\_PROCESSOR - Executing logical operations on operands

entity logic\_processor is

port(

HexA : in std\_logic\_vector(3 downto 0); -- Operand 1

HexB : in std\_logic\_vector(3 downto 0); -- Operand 2

pb : in std\_logic\_vector(3 downto 0); -- Logic Operation Selection Variable

logic\_out : out std\_logic\_vector(3 downto 0) -- Result of logical operations on operands 1 and 2

);

end entity logic\_processor;

architecture logic\_processor of logic\_processor is

begin

with pb(3 downto 0) select

logic\_out <= HexA AND HexB when "0001",

HexA OR HexB when "0010",

HexA XOR HexB when "0100",

"0000" when others; --Don't Cares, since its not required to excuted operations for other combinations of 4-bit binary 0s and 1s

end logic\_processor;

--LOGIC\_PROCESSOR

**New\_mux:**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

--NEW\_MUX - Selection of operations to be executed

entity new\_mux is port

(

conc\_hex : in std\_logic\_vector(7 downto 0); -- concatenated 8 bit hex\_A and hex\_B

sum: in std\_logic\_vector(7 downto 0); --8 bit sum from adder

pb3 : in std\_logic\_vector(3 downto 0); --control variable

mux\_out : out std\_logic\_vector(7 downto 0) --8 bit output

);

end entity new\_mux;

architecture new\_mux of new\_mux is

begin

with pb3(3 downto 0) select

mux\_out <= sum when "1000",

conc\_hex when "0001",

conc\_hex when "0010",

conc\_hex when "0100",

conc\_hex when others;

end new\_mux;

--NEW\_MUX

**Segment Seven:**

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- \* Entity \*

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

entity segment7\_mux is

port (

clk : in std\_logic := '0';

DIN2 : in std\_logic\_vector(6 downto 0);

DIN1 : in std\_logic\_vector(6 downto 0);

DOUT : out std\_logic\_vector(6 downto 0);

DIG2 : out std\_logic;

DIG1 : out std\_logic

);

end entity segment7\_mux;

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-- \* Architecture \*

-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

architecture syn of segment7\_mux is

signal toggle : std\_logic;

signal DOUT\_TEMP : std\_logic\_vector(6 downto 0);

begin

----------------------------------------------

-- Register File

----------------------------------------------

clk\_proc:process(CLK)

variable COUNT :unsigned(10 downto 0) := "00000000000";

begin

if (rising\_edge(CLK)) then

COUNT := COUNT + 1;

else

COUNT := COUNT;

end if;

toggle <= COUNT(10);

end process clk\_proc;

DIG1 <= NOT toggle;

DIG2 <= toggle;

DOUT\_TEMP(0) <= (DIN2(0)) WHEN (toggle = '1') ELSE (DIN1(0));

DOUT\_TEMP(1) <= (DIN2(1)) WHEN (toggle = '1') ELSE (DIN1(1));

DOUT\_TEMP(2) <= (DIN2(2)) WHEN (toggle = '1') ELSE (DIN1(2));

DOUT\_TEMP(3) <= (DIN2(3)) WHEN (toggle = '1') ELSE (DIN1(3));

DOUT\_TEMP(4) <= (DIN2(4)) WHEN (toggle = '1') ELSE (DIN1(4));

DOUT\_TEMP(5) <= (DIN2(5)) WHEN (toggle = '1') ELSE (DIN1(5));

DOUT\_TEMP(6) <= (DIN2(6)) WHEN (toggle = '1') ELSE (DIN1(6));

-- DOUT\_TEMP(7) <= (DIN2(7)) WHEN (toggle = '1') ELSE (DIN1(7));

DOUT(0) <= '0' WHEN (DOUT\_TEMP(0) = '0') ELSE '1';

DOUT(1) <= '0' WHEN (DOUT\_TEMP(1) = '0') ELSE 'Z'; --open drain

DOUT(2) <= '0' WHEN (DOUT\_TEMP(2) = '0') ELSE '1';

DOUT(3) <= '0' WHEN (DOUT\_TEMP(3) = '0') ELSE '1';

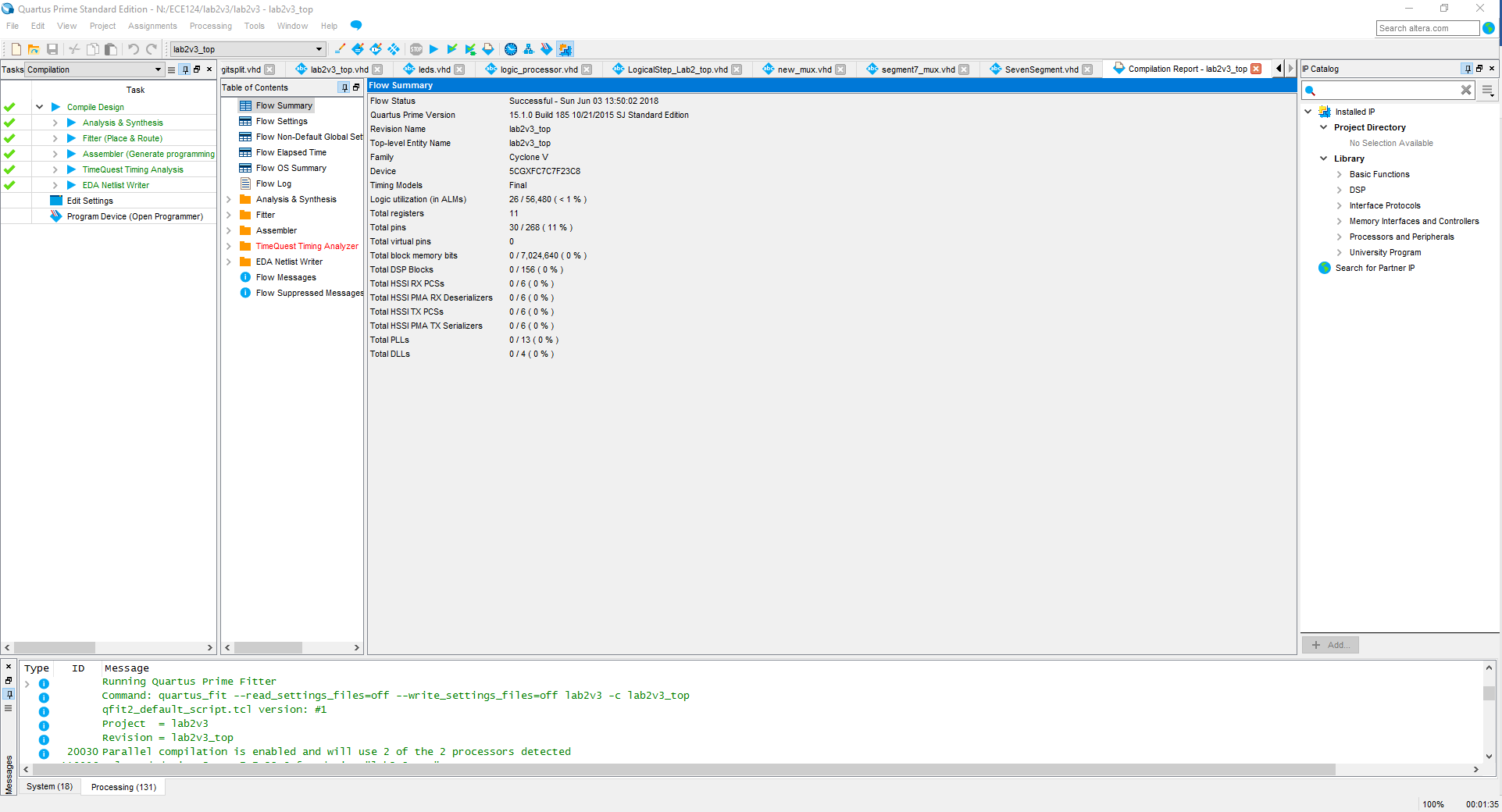
DOUT(4) <= '0' WHEN (DOUT\_TEMP(4) = '0') ELSE '1';

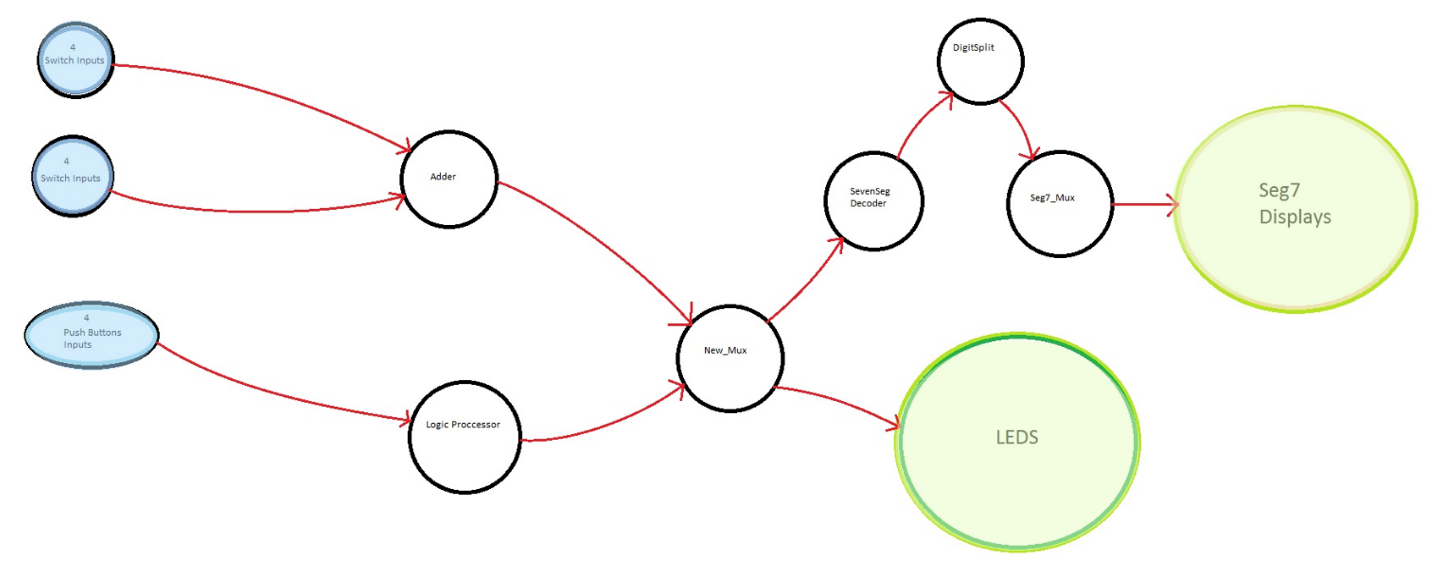
DOUT(5) <= '0' WHEN (DOUT\_TEMP(5) = '0') ELSE 'Z'; --open drain

DOUT(6) <= '0' WHEN (DOUT\_TEMP(6) = '0') ELSE 'Z'; --open drain

-- DOUT(7) <= '0' WHEN (DOUT\_TEMP(7) = '0') ELSE '1';

end architecture syn;

Compilation report



State Diagram 1

# Simulations

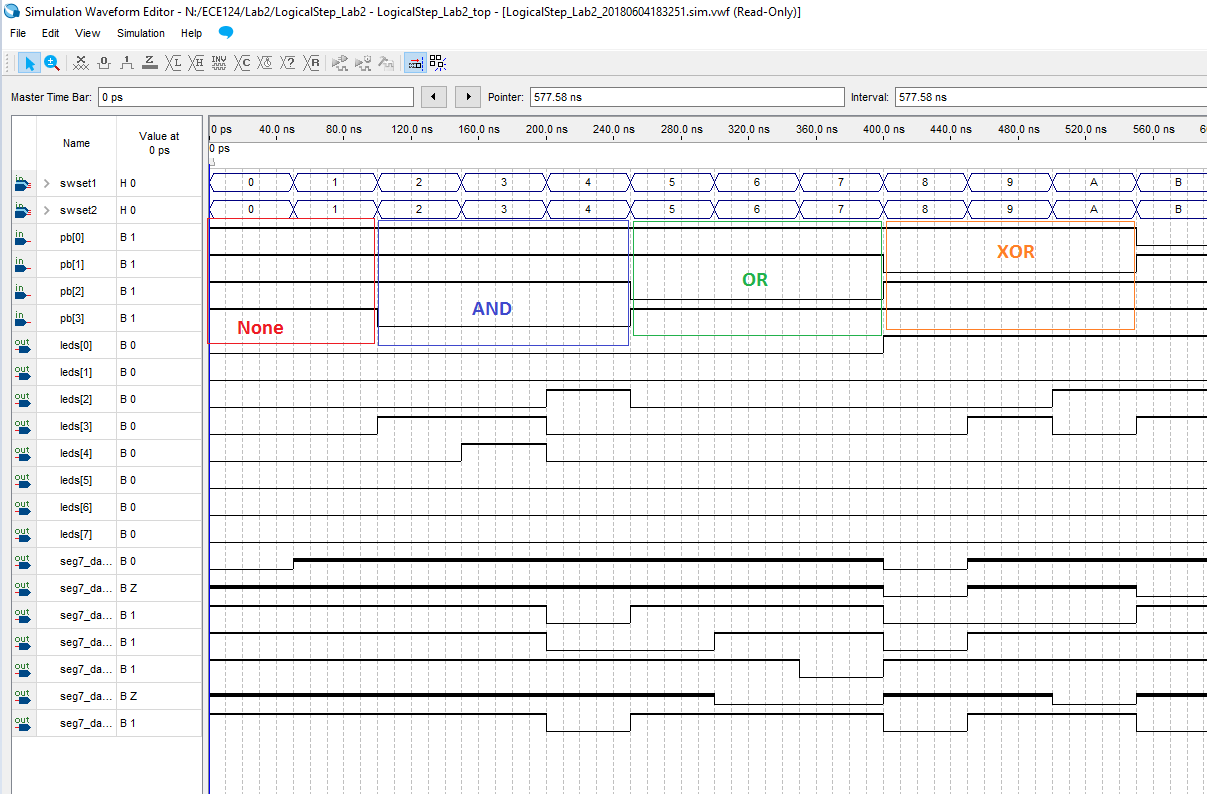


Figure Logical Operations Simulations

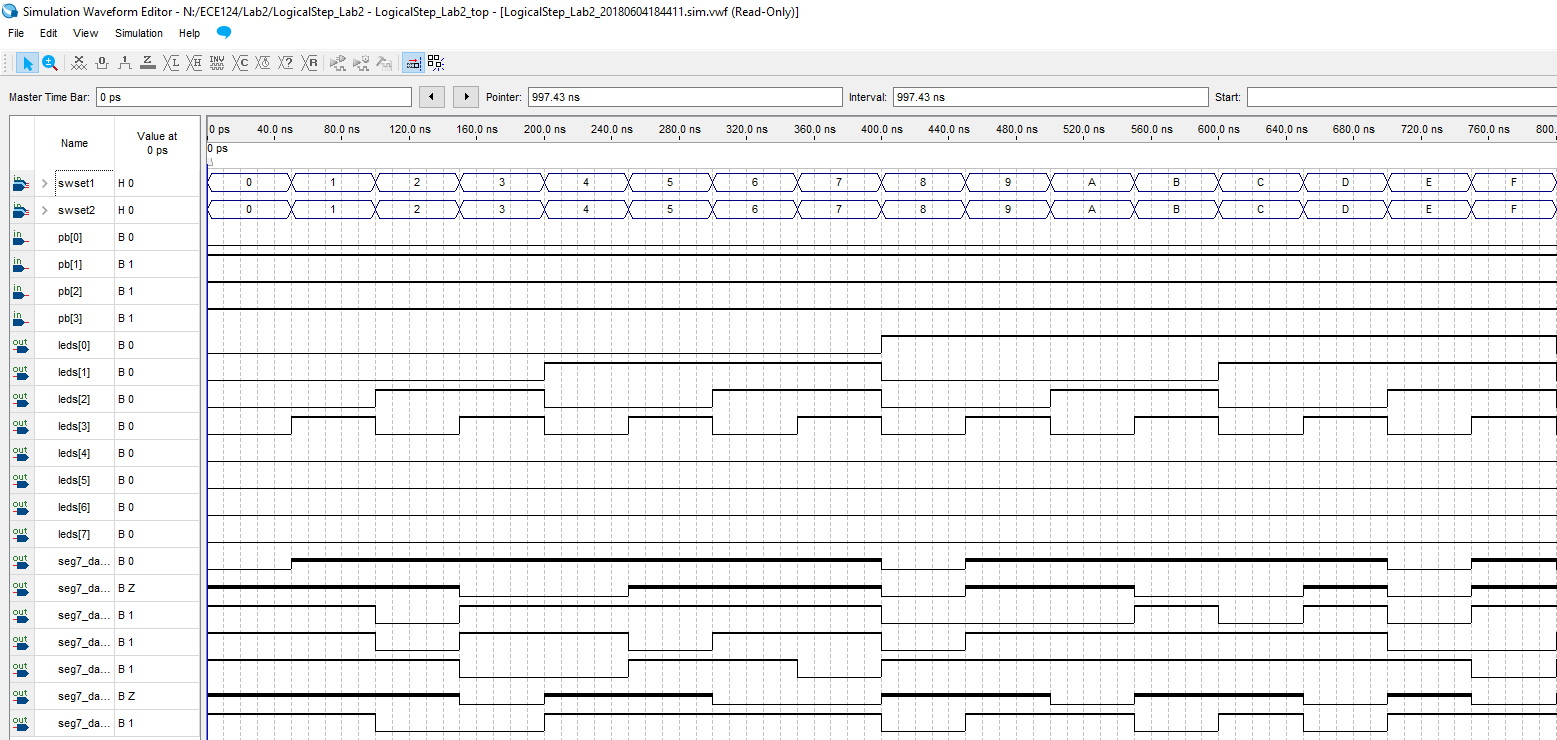


Figure Add Simulations Non-Increment

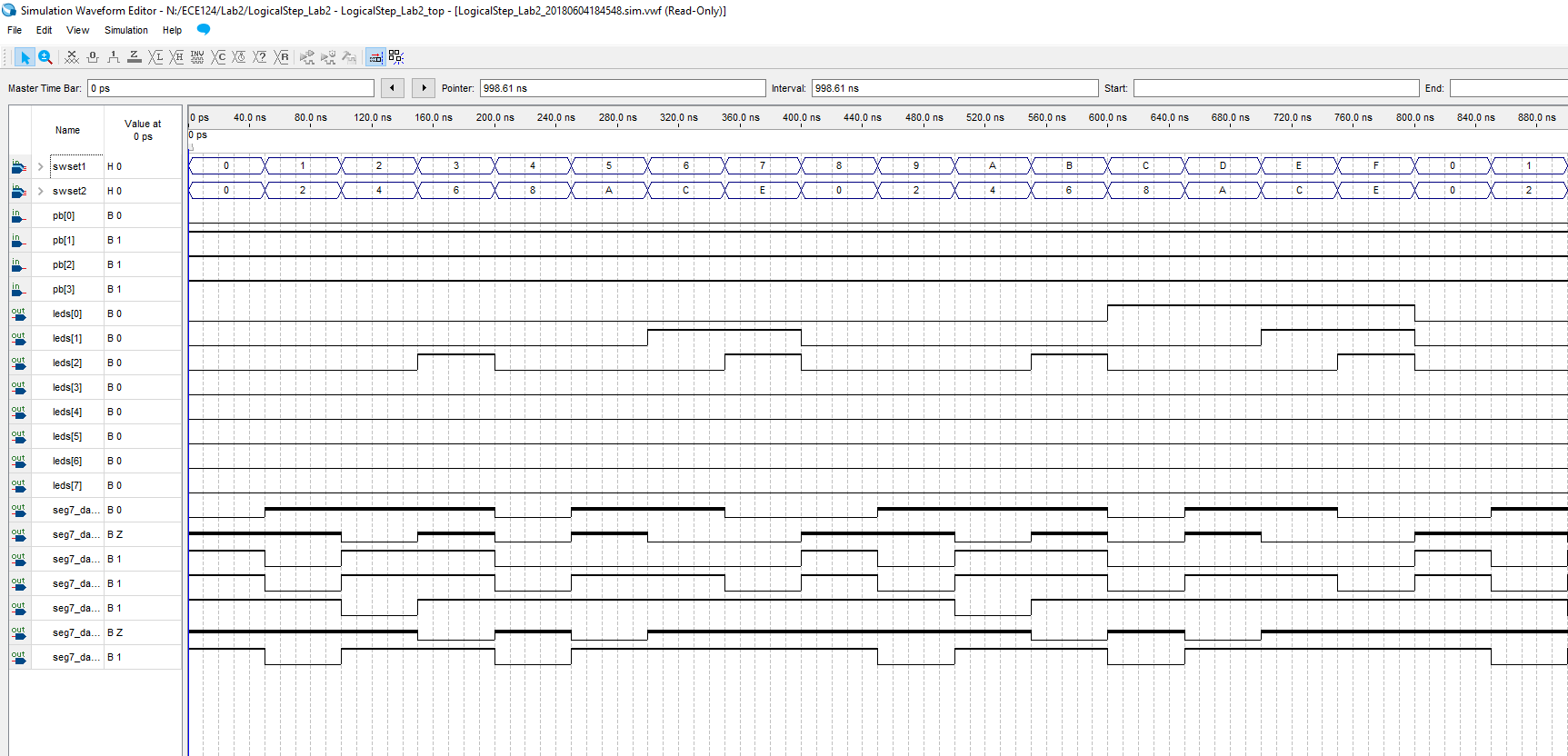


Figure Add Simulations Increments Swtiches