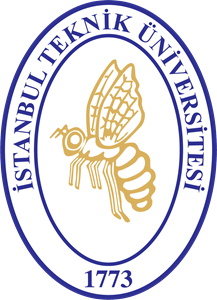
**DIGITAL SYSTEM DESIGN APPLICATIONS**

**(CRN: 11275)**

**THE REPORT OF EXPERIMENT – 2**



Faculty of Electrical and Electronics Engineering

Electronics and Communication Engineering

Yusuf Tekin – 040200043

**1.Decoder**

Decoder Verilog Code

`timescale 1ns **/** 1ps

**module** DECODER**(**

**input** **[**3**:**0**]** IN**,**

**output** **reg** **[**15**:**0**]** OUT

**);**

**always** **@(\*)** **begin**

**case(**IN**)**

4'b0000**:** OUT **=** 16'b0000\_0000\_0000\_0001**;**

4'b0001**:** OUT **=** 16'b0000\_0000\_0000\_0010**;**

4'b0010**:** OUT **=** 16'b0000\_0000\_0000\_0100**;**

4'b0011**:** OUT **=** 16'b0000\_0000\_0000\_1000**;**

4'b0100**:** OUT **=** 16'b0000\_0000\_0001\_0000**;**

4'b0101**:** OUT **=** 16'b0000\_0000\_0010\_0000**;**

4'b0110**:** OUT **=** 16'b0000\_0000\_0100\_0000**;**

4'b0111**:** OUT **=** 16'b0000\_0000\_1000\_0000**;**

4'b1000**:** OUT **=** 16'b0000\_0001\_0000\_0000**;**

4'b1001**:** OUT **=** 16'b0000\_0010\_0000\_0000**;**

4'b1010**:** OUT **=** 16'b0000\_0100\_0000\_0000**;**

4'b1011**:** OUT **=** 16'b0000\_1000\_0000\_0000**;**

4'b1100**:** OUT **=** 16'b0001\_0000\_0000\_0000**;**

4'b1101**:** OUT **=** 16'b0010\_0000\_0000\_0000**;**

4'b1110**:** OUT **=** 16'b0100\_0000\_0000\_0000**;**

4'b1111**:** OUT **=** 16'b1000\_0000\_0000\_0000**;**

**default:** OUT **=** 16'b0000\_0000\_0000\_0000**;**

**endcase**

**end**

**endmodule**

top\_module Testbench Code

`timescale 1ns **/** 1ps

**module** top\_modul\_tb**();**

**reg** **[**7**:**0**]** sw**;**

**reg** **[**3**:**0**]** btn **=** 4'b0000**;**

**wire** **[**7**:**0**]** led**;**

**wire** **[**6**:**0**]** cat**;**

**wire** **[**3**:**0**]** an**;**

**wire** dp**;**

top\_module uut**(**

**.**sw**(**sw**),**

**.**btn**(**btn**),**

**.**led**(**led**),**

**.**cat**(**cat**),**

**.**an**(**an**),**

**.**dp**(**dp**)**

**);**

**initial** **begin**

sw **=** 8'b0000\_0000**;** **#**10**;**

sw **=** 8'b0000\_0001**;** **#**10**;**

sw **=** 8'b0000\_0010**;** **#**10**;**

sw **=** 8'b0000\_0011**;** **#**10**;**

sw **=** 8'b0000\_0100**;** **#**10**;**

sw **=** 8'b0000\_0101**;** **#**10**;**

sw **=** 8'b0000\_0110**;** **#**10**;**

sw **=** 8'b0000\_0111**;** **#**10**;**

sw **=** 8'b0000\_1000**;** **#**10**;**

sw **=** 8'b0000\_1001**;** **#**10**;**

sw **=** 8'b0000\_1010**;** **#**10**;**

sw **=** 8'b0000\_1011**;** **#**10**;**

sw **=** 8'b0000\_1100**;** **#**10**;**

sw **=** 8'b0000\_1101**;** **#**10**;**

sw **=** 8'b0000\_1110**;** **#**10**;**

sw **=** 8'b0000\_1111**;** **#**10**;**

sw **=** 8'b0000\_0000**;** **#**10**;**

$finish**();**

**end**

**endmodule**

top\_module Verilog Code

`timescale 1ns **/** 1ps

**module** top\_module**(**

**input** **[**7**:**0**]** sw**,**

**input** **[**3**:**0**]** btn**,**

**output** **[**7**:**0**]** led**,**

**output** **[**6**:**0**]** cat**,**

**output** **[**3**:**0**]** an**,**

**output** dp

**);**

**wire** **[**15**:**0**]** decoder\_out**;**

**wire** **[**3**:**0**]** decoder\_in**;**

**assign** dp **=** decoder\_out**[**15**];**

**assign** cat **=** decoder\_out**[**14**:**8**];**

**assign** led **=** decoder\_out**[**7**:**0**];**

**assign** decoder\_in**[**3**:**0**]** **=** sw**[**3**:**0**];**

**assign** an **=** 4'b1110**;**

DECODER decoder1**(**

**.**IN**(**decoder\_in**),**

**.**OUT**(**decoder\_out**)**

**);**

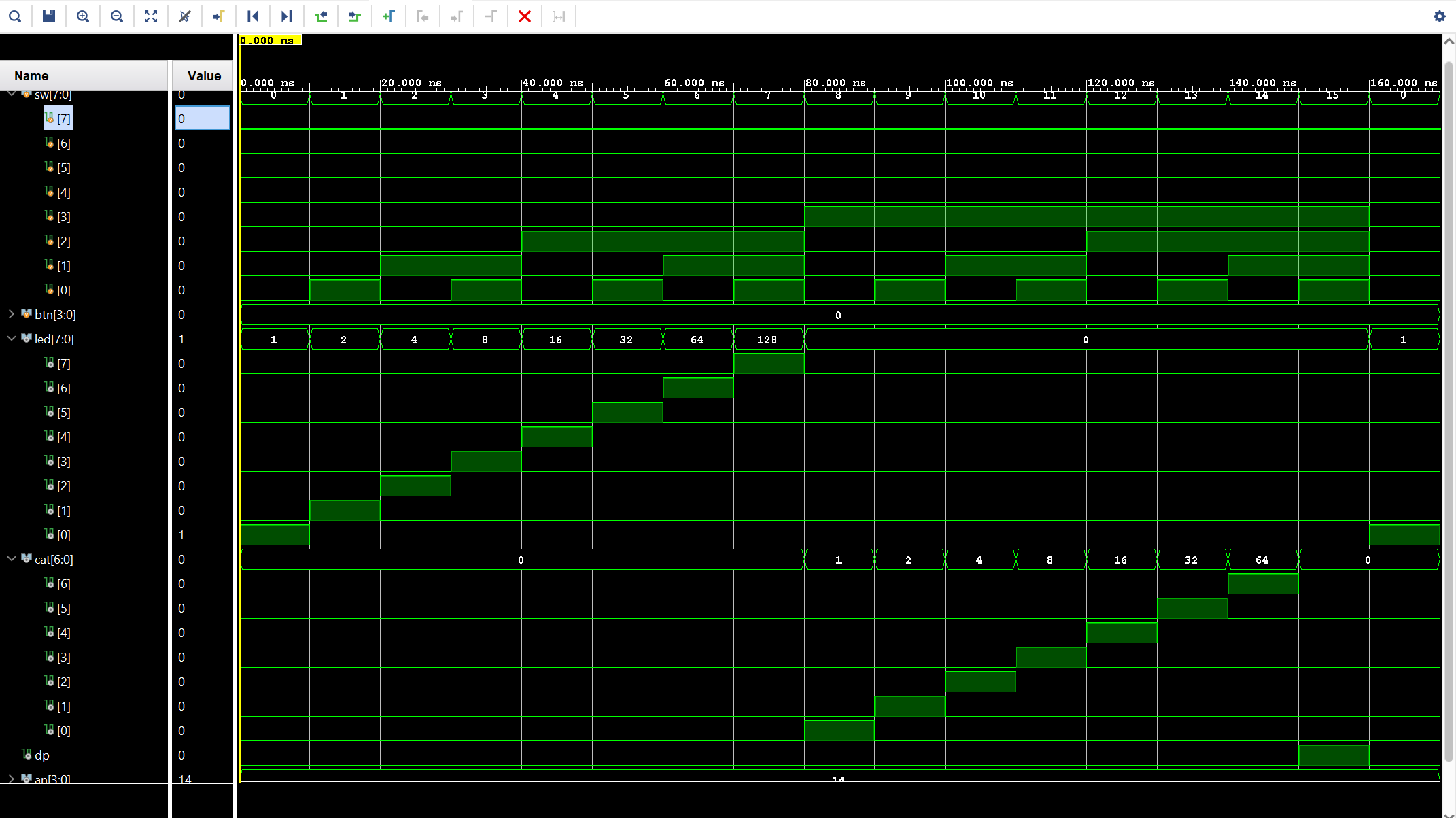
**endmodule**

A diagram of a decoder

Description automatically generated

Decoder RTL Schematic (at the top\_module)

The RTL synthesis tool interprets the pattern of 4-bit addressing as a ROM table where IN is the address and OUT is the data. That is the reason why there is a standalone block of ROM in the RTL Schematic. As it can be seen in the figure, besides the input “btn” all the inputs and outputs are arranged accordingly.



Decoder Behavioral Simulation (at the top\_module)

A diagram of a computer

Description automatically generated

Technology Schematic

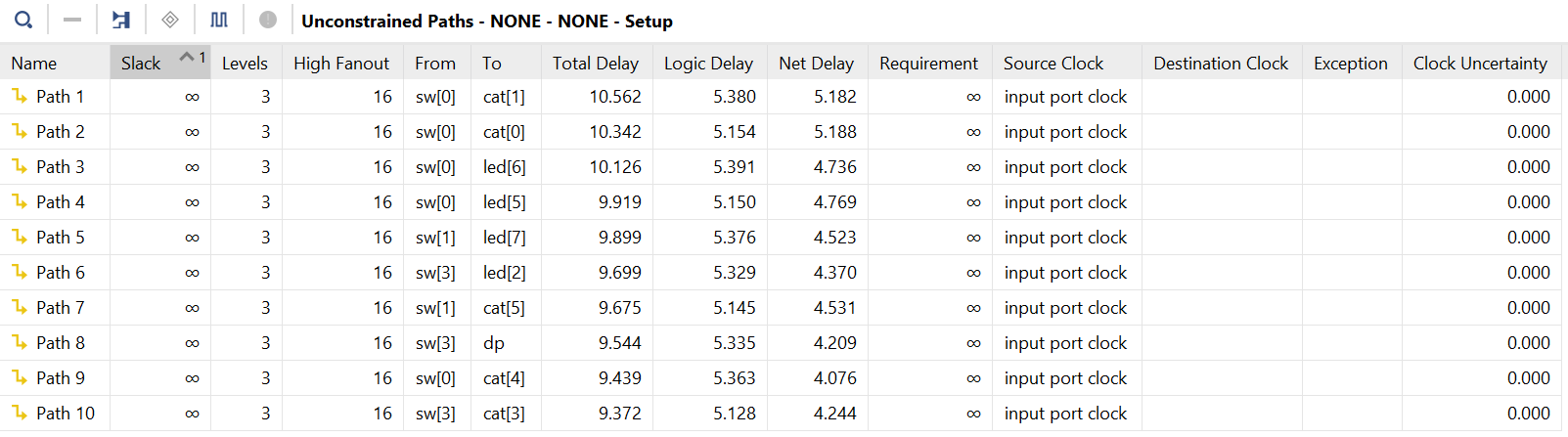
A diagram of electrical wiring

Description automatically generated

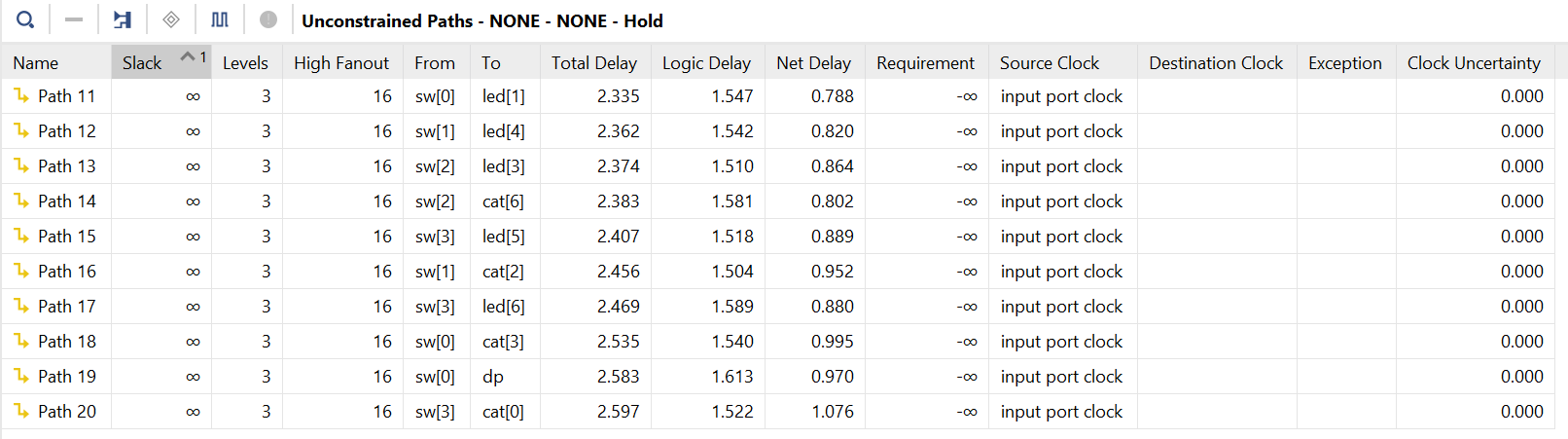
Technology Schematic (detailed)

As it can be seen in the detailed Technology Schematic *(p.6)*, there are sixteen LUTs used in the design. The LUTs represents seven “cat”, eight “led” and a “dp” outputs, in total of sixteen outputs. For each four bits value of “sw” input from 0000 to 1111, there is a LUT entry. Since it is required to be unique outputs for each input value, the design is required to be use sixteen LUTs for the decoder.

For instance, if led[0] is desired to be enable, the LUT named *led\_OBUF(0)\_inst\_i\_1* must has the input values of I[0:3] = {0,0,0,0}. (I = 4’b0000)



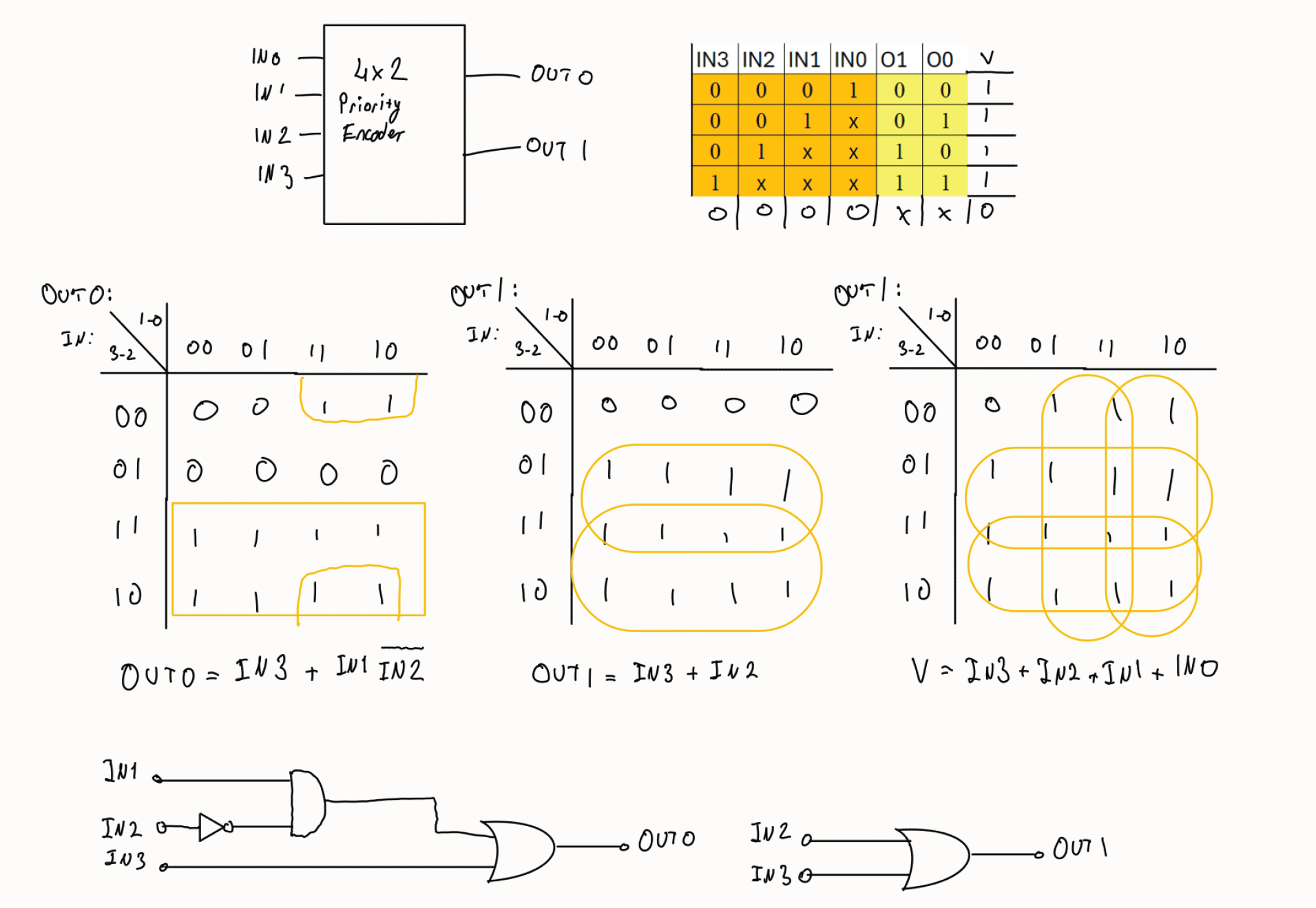
Timing Summary - Setup



Timing Summary - Hold

For the largest delay, setup paths are more commonly longer as these represents the delays from register to input or register to register or register to output. In this case there is no difference. It can be seen in the *Timing Summary – Setup* table the greatest delayed path is the **Path 1** which connects sw[0] to cat[1] with the total delay of 10.562.

**2. Priority Encoder**



Priorty Encoder truthtables, logic statements and schematics

`timescale 1ns **/** 1ps

**module** ENCODER**(**

**input** **[**3**:**0**]** IN**,**

**output** **[**1**:**0**]** OUT**,**

**output** V**);**

**assign** OUT**[**0**]** **=** IN**[**3**]** **|** **(**IN**[**1**]** **&** **!**IN**[**2**]);**

**assign** OUT**[**1**]** **=** IN**[**2**]** **|** IN**[**3**];**

**assign** V **=** IN**[**0**]** **|** IN**[**1**]** **|** IN**[**2**]** **|** IN**[**3**];**

**endmodule**

Priority Encoder Verilog Code

`timescale 1ns **/** 1ps

**module** top\_modul\_tb**();**

**reg** **[**7**:**0**]** sw**;**

**wire** **[**7**:**0**]** led**;**

top\_module uut**(**

**.**sw**(**sw**),**

**.**btn**(**btn**),**

**.**led**(**led**),**

**.**cat**(**cat**),**

**.**an**(**an**),**

**.**dp**(**dp**)**

**);**

**initial** **begin**

sw **=** 8'b0000\_0000**;** **#**10**;**

sw **=** 8'b0000\_0001**;** **#**10**;**

sw **=** 8'b0000\_0010**;** **#**10**;**

sw **=** 8'b0000\_0011**;** **#**10**;**

sw **=** 8'b0000\_0100**;** **#**10**;**

sw **=** 8'b0000\_0101**;** **#**10**;**

sw **=** 8'b0000\_0110**;** **#**10**;**

sw **=** 8'b0000\_0111**;** **#**10**;**

sw **=** 8'b0000\_1000**;** **#**10**;**

sw **=** 8'b0000\_1001**;** **#**10**;**

sw **=** 8'b0000\_1010**;** **#**10**;**

sw **=** 8'b0000\_1011**;** **#**10**;**

sw **=** 8'b0000\_1100**;** **#**10**;**

sw **=** 8'b0000\_1101**;** **#**10**;**

sw **=** 8'b0000\_1110**;** **#**10**;**

sw **=** 8'b0000\_1111**;** **#**10**;**

sw **=** 8'b0000\_0000**;** **#**10**;**

$finish**();**

**end**

**endmodule**

Priorty Encoder Verilog Code top\_module

`timescale 1ns **/** 1ps

**module** top\_module**(**

**input** **[**7**:**0**]** sw**,**

**input** **[**3**:**0**]** btn**,**

**output** **[**7**:**0**]** led**,**

**output** **[**6**:**0**]** cat**,**

**output** **[**3**:**0**]** an**,**

**output** dp

**);**

**wire** **[**1**:**0**]** encoder\_out**;**

**wire** **[**3**:**0**]** encoder\_in**;**

**wire** v\_out**;**

**assign** led**[**1**:**0**]** **=** encoder\_out**;**

**assign** led**[**7**]** **=** v\_out**;**

**assign** encoder\_in**[**3**:**0**]** **=** sw**[**3**:**0**];**

ENCODER encoder1**(**

**.**IN**(**encoder\_in**),**

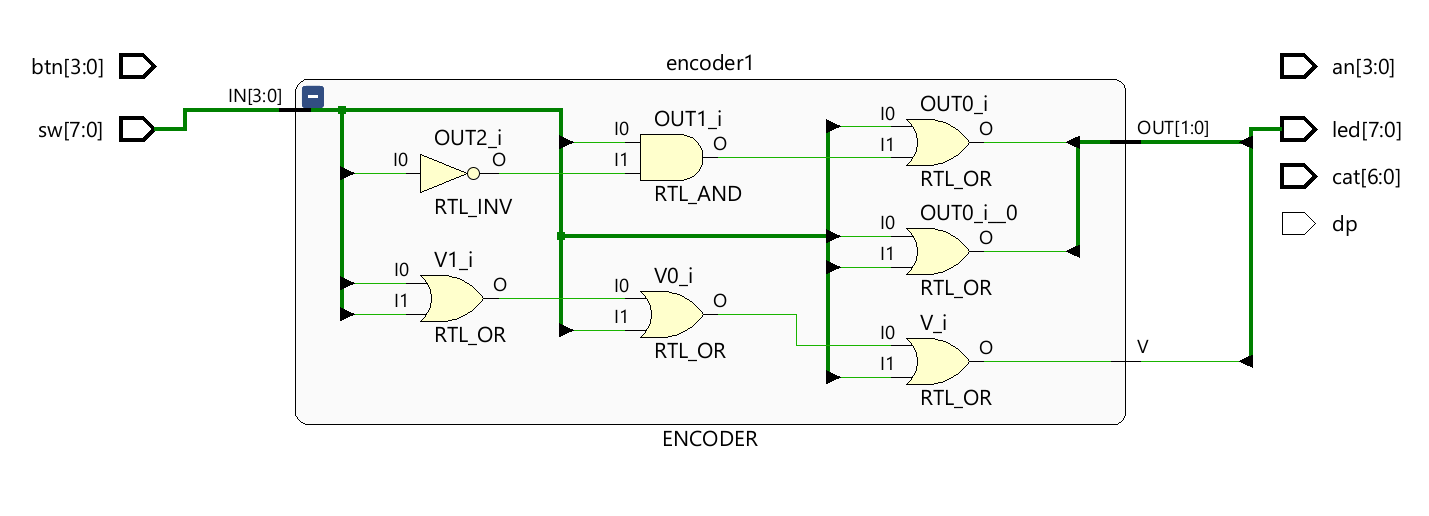
**.**OUT**(**encoder\_out**),**

**.**V**(**v\_out**)**

**);**

**endmodule**

Priority Encoder Testbench Code (top\_module)



Priority Encoder RTL Schematic

A screenshot of a computer

Description automatically generated

Priority Encoder Behavioral Simulation

A screenshot of a computer

Description automatically generated

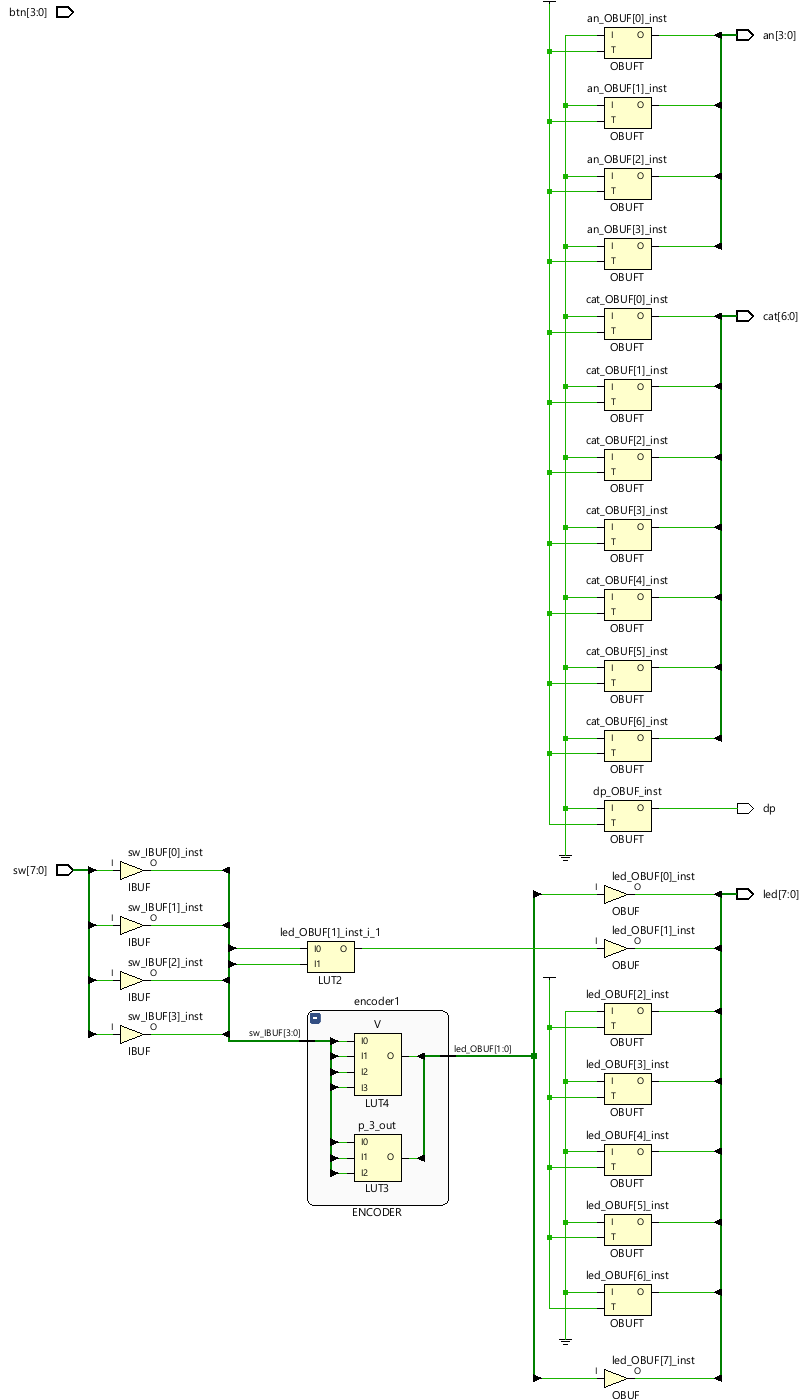
Priority Encoder Post-Synthesis Simulation

A screenshot of a computer

Description automatically generated

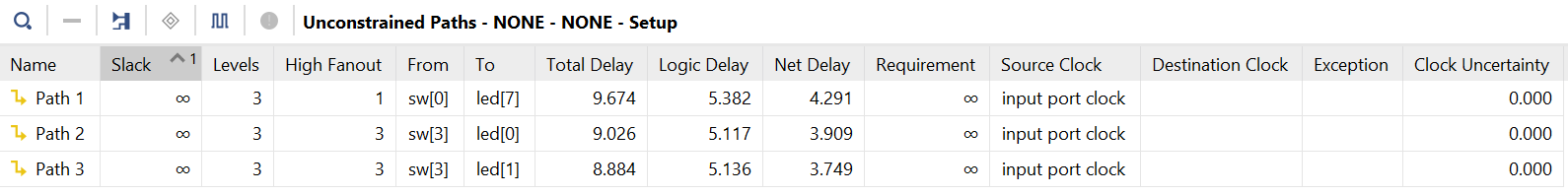
Priority Encoder Post-Synthesis Simulation (zoomed in)

In the simulation waves, it is clear that the encoder work as intended, although, there are visible delays at led[0] and led[7] in the post-synthesis simulation. These delays are caused by the different lengths of the paths.

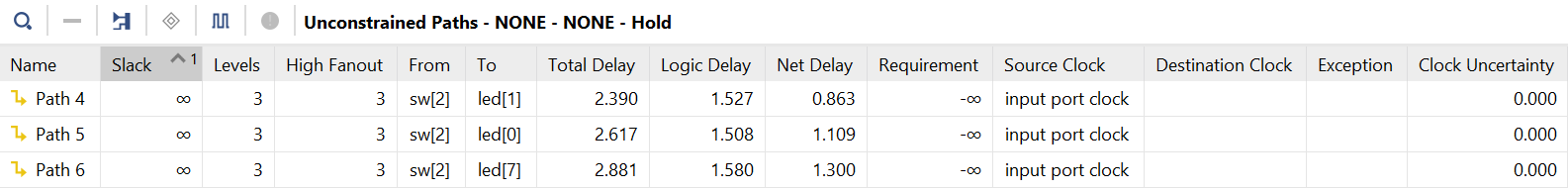


Priority Encoder Technology Schematic

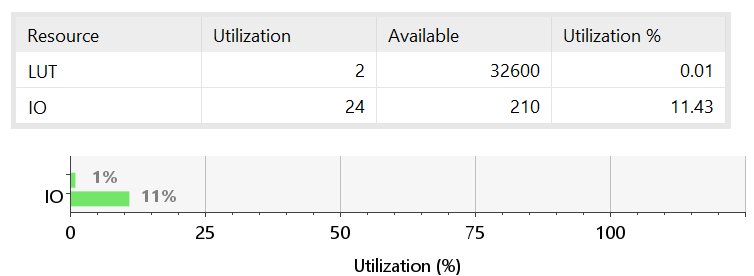
It can be seen in the schematic, after the implementation all the gates are turned into LUTs. “an”, ”cat” and “led[2:6]” outputs created LUTs that are empty. As it can be seen in the technology schematic that the 2-input LUT2 is connected to OUT[1] which makes sense due to the one OR gate it requires to assign. V is a 4-input or process which requires just one 4-input LUT and that also can be seen in the figure. The last LUT that in-use is for the 3-input required assignment OUT[0].

****

Priority Encoder Setup Timings - Primitive Gates

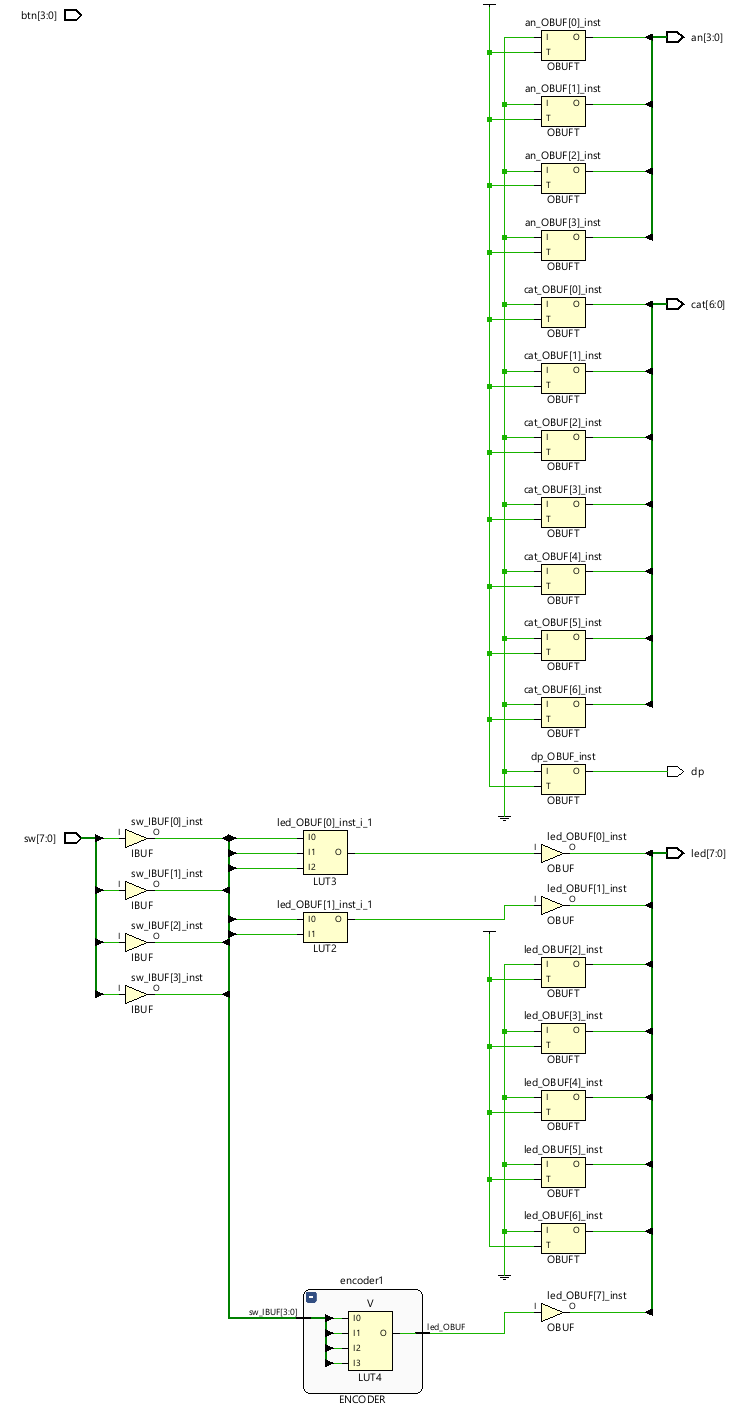
****

Priority Encoder Hold Timings - Primitive Gates

****

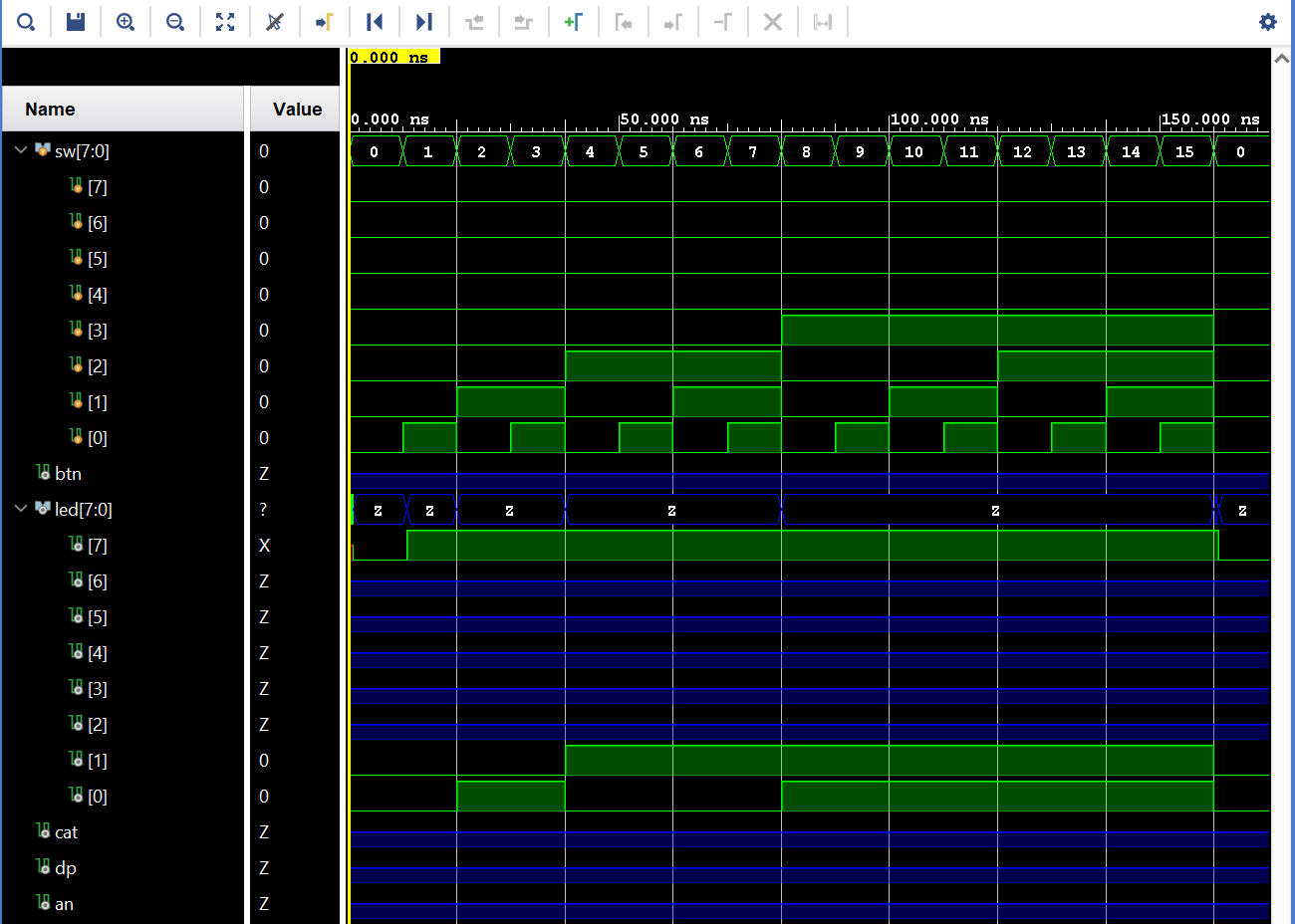
Priority Encoder Utility Summary - Primitive Gates

**Always – Case Structure:**

****

Priority Encoder Technology Schematic - always/case

The difference of technology schematics between primitive gate design and always/case design is the different paths of the LUT3 block uses. It seems that other than the path there is no practical difference.

****

Priority Encoder Post-Synthesis Simulation - always/case

**A screenshot of a computer

Description automatically generated**

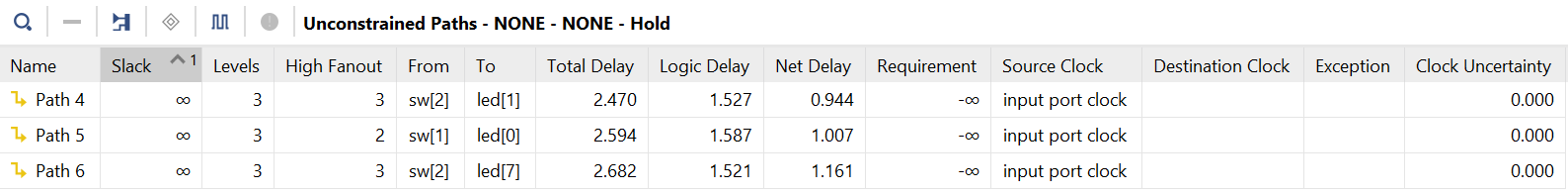
Priority Encoder Post-Synthesis Simulation - always/case (detailed)

The difference of *Post-Synthesis Simulations* between primitive gate design and always/case design is due to the change on the paths the delay on the led[0] is cancelled.

**A group of people on a white background

Description automatically generated**

Priority Encoder Setup Timings - always/case

****

Priority Encoder Hold Timings - always/case

To observe the differences in terms of timing, it is necessary to inspect both logic delay and net delay separately. In terms of total delay, besides Path 2 and Path 4 all the paths’ delays have increased. For Path 4 the net delay has decreased, and logic delay has stayed the same, that’s why the total delay has decreased. Its same for the Path 2 but Path 2’s logic delay has also decreased. For other paths it can be seen that even though net delays have decreased for most, the logic delays have increased more which results in the increase of the total delay.

**A screenshot of a graph

Description automatically generated**

Priority Encoder Utility Summary - always/case

**3. Multiplexer (MUX)**

`timescale 1ns **/** 1ps

**module** top\_module**(**

**input** **[**7**:**0**]** sw**,**

**input** **[**3**:**0**]** btn**,**

**output** **[**7**:**0**]** led**,**

**output** **[**6**:**0**]** cat**,**

**output** **[**3**:**0**]** an**,**

**output** dp

**);**

**wire** **[**3**:**0**]** input\_d**;**

**wire** **[**1**:**0**]** input\_s**;**

**wire** output\_o**;**

**assign** input\_d**[**3**:**0**]** **=** sw**[**3**:**0**];**

**assign** input\_s**[**1**:**0**]** **=** btn**[**1**:**0**];**

**assign** led**[**0**]** **=** output\_o**;**

MUX mux1**(**

**.**D**(**input\_d**),**

**.**S**(**input\_s**),**

**.**O**(**output\_o**)**

**);**

**endmodule**

`timescale 1ns **/** 1ps

**module** MUX**(**

**input** **[**3**:**0**]** D**,**

**input** **[**1**:**0**]** S**,**

**output** O

**);**

**assign** O **=** **(!**S**[**0**]** **&** **!**S**[**1**]** **&** D**[**0**])** **+** **(**S**[**0**]** **&** **!**S**[**1**]** **&** D**[**1**])** **+** **(!**S**[**0**]** **&** S**[**1**]** **&** D**[**2**])** **+** **(**S**[**0**]** **&** S**[**1**]** **&** D**[**3**]);**

**endmodule**

MUX Verilog Code

MUX Verilog Code - top\_module

A diagram of a circuit

Description automatically generated

MUX RTL Schematic

MUX Testbench Code

`timescale 1ns **/** 1ps

**module** top\_modul\_tb**();**

**reg** **[**7**:**0**]** sw**;**

**reg** **[**3**:**0**]** btn**;**

**wire** **[**7**:**0**]** led**;**

top\_module uut**(**

**.**sw**(**sw**),**

**.**btn**(**btn**),**

**.**led**(**led**),**

**.**cat**(**cat**),**

**.**an**(**an**),**

**.**dp**(**dp**)**

**);**

**initial** **begin**

sw **=** 8'b0000\_0000**;** btn **=** 2'b00**;** **#**10**;**

sw **=** 8'b0000\_0001**;** btn **=** 2'b00**;** **#**10**;**

sw **=** 8'b0000\_0001**;** btn **=** 2'b00**;** **#**10**;**

sw **=** 8'b0000\_0010**;** btn **=** 2'b00**;** **#**10**;**

sw **=** 8'b0000\_0010**;** btn **=** 2'b01**;** **#**10**;**

sw **=** 8'b0000\_0100**;** btn **=** 2'b00**;** **#**10**;**

sw **=** 8'b0000\_0100**;** btn **=** 2'b10**;** **#**10**;**

sw **=** 8'b0000\_1000**;** btn **=** 2'b00**;** **#**10**;**

sw **=** 8'b0000\_1000**;** btn **=** 2'b11**;** **#**10**;**

sw **=** 8'b0000\_0000**;** btn **=** 2'b00**;** **#**10**;**

$finish**();**

**end**

**endmodule**

A screenshot of a computer

Description automatically generated

MUX Behavioral Simulation

A diagram of a computer

Description automatically generated

MUX Technology Schematic



MUX Timing - Setup



MUX Timing - Hold

A screenshot of a graph

Description automatically generated

MUX Utilization Summary

**Always/case Structure:**

A diagram of a circuit

Description automatically generated

MUX RTL Schematic - always/case

MUX Verilog Code - always/case

`timescale 1ns **/** 1ps

**module** MUX**(**

**input** **[**3**:**0**]** D**,**

**input** **[**1**:**0**]** S**,**

**output** **reg** O

**);**

**always** **@(\*)** **begin**

**case(**S**)**

2'b00**:** O **=** D**[**0**];**

2'b01**:** O **=** D**[**1**];**

2'b10**:** O **=** D**[**2**];**

2'b11**:** O **=** D**[**3**];**

**endcase**

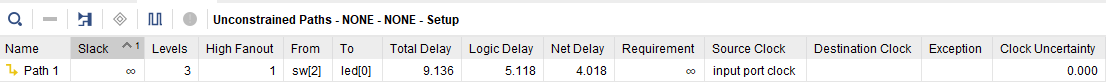
**end**

**endmodule**

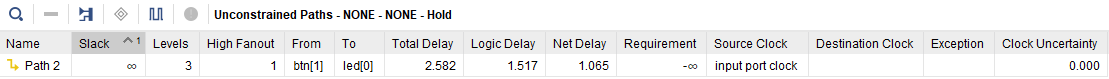
A diagram of a computer

Description automatically generated

MUX Technology Schematic - always/case



MUX Setup Timing - always/case



MUX Setup Timing - always/case

A screenshot of a graph

Description automatically generated

MUX Utilization Summary - always/case

Comparing the two different methods’ timing and utilization, there is no difference in terms of utilization and the delays are increased significantly in the always/case method. To be specific, almost all the logic delays stayed same for both *Path 1* and *Path 2* however, net delay of the Path 1 has increased 251% and for Path 2 158%. Which caused increases in total delays.

**4. Demultiplexer (DEMUX)**

DEMUX Verilog Code

`timescale 1ns **/** 1ps

**module** DEMUX**(**

**input** D**,**

**input** **[**1**:**0**]** S**,**

**output** **[**3**:**0**]** O

**);**

**wire** enable\_O0**,** enable\_O1**,** enable\_O2**,** enable\_O3**;**

// AND and NOT gates:

**assign** enable\_O0 **=** **!**S**[**0**]** **&** **!**S**[**1**];**

**assign** enable\_O1 **=** S**[**0**]** **&** **!**S**[**1**];**

**assign** enable\_O2 **=** **!**S**[**0**]** **&** S**[**1**];**

**assign** enable\_O3 **=** S**[**0**]** **&** S**[**1**];**

//TRI gates:

**assign** O**[**0**]** **=** enable\_O0 **?** D **:** 1'bz**;**

**assign** O**[**1**]** **=** enable\_O1 **?** D **:** 1'bz**;**

**assign** O**[**2**]** **=** enable\_O2 **?** D **:** 1'bz**;**

**assign** O**[**3**]** **=** enable\_O3 **?** D **:** 1'bz**;**

**endmodule**

DEMUX Verilog Code - top\_module

`timescale 1ns **/** 1ps

**module** top\_module**(**

**input** **[**7**:**0**]** sw**,**

**input** **[**3**:**0**]** btn**,**

**output** **[**7**:**0**]** led**,**

**output** **[**6**:**0**]** cat**,**

**output** **[**3**:**0**]** an**,**

**output** dp

**);**

**wire** input\_d**;**

**wire** **[**1**:**0**]** input\_s**;**

**wire** **[**3**:**0**]** output\_o**;**

**assign** input\_d **=** sw**[**0**];**

**assign** input\_s**[**1**:**0**]** **=** btn**[**1**:**0**];**

**assign** led**[**3**:**0**]** **=** output\_o**[**3**:**0**];**

DEMUX demux1**(**

**.**D**(**input\_d**),**

**.**S**(**input\_s**),**

**.**O**(**output\_o**)**

**);**

**endmodule**

A diagram of a circuit

Description automatically generated

DEMUX RTL Schematic

DEMUX Testbench Code

`timescale 1ns **/** 1ps

**module** top\_modul\_tb**();**

**reg** **[**7**:**0**]** sw**;**

**reg** **[**3**:**0**]** btn**;**

**wire** **[**7**:**0**]** led**;**

top\_module uut**(**

**.**sw**(**sw**),**

**.**btn**(**btn**),**

**.**led**(**led**),**

**.**cat**(**cat**),**

**.**an**(**an**),**

**.**dp**(**dp**)**

**);**

**initial** **begin**

sw **=** 8'b0000\_0000**;** btn **=** 2'b00**;** **#**10**;**

sw **=** 8'b0000\_0001**;** btn **=** 2'b00**;** **#**10**;**

sw **=** 8'b0000\_0000**;** btn **=** 2'b01**;** **#**10**;**

sw **=** 8'b0000\_0001**;** btn **=** 2'b01**;** **#**10**;**

sw **=** 8'b0000\_0000**;** btn **=** 2'b10**;** **#**10**;**

sw **=** 8'b0000\_0001**;** btn **=** 2'b10**;** **#**10**;**

sw **=** 8'b0000\_0000**;** btn **=** 2'b11**;** **#**10**;**

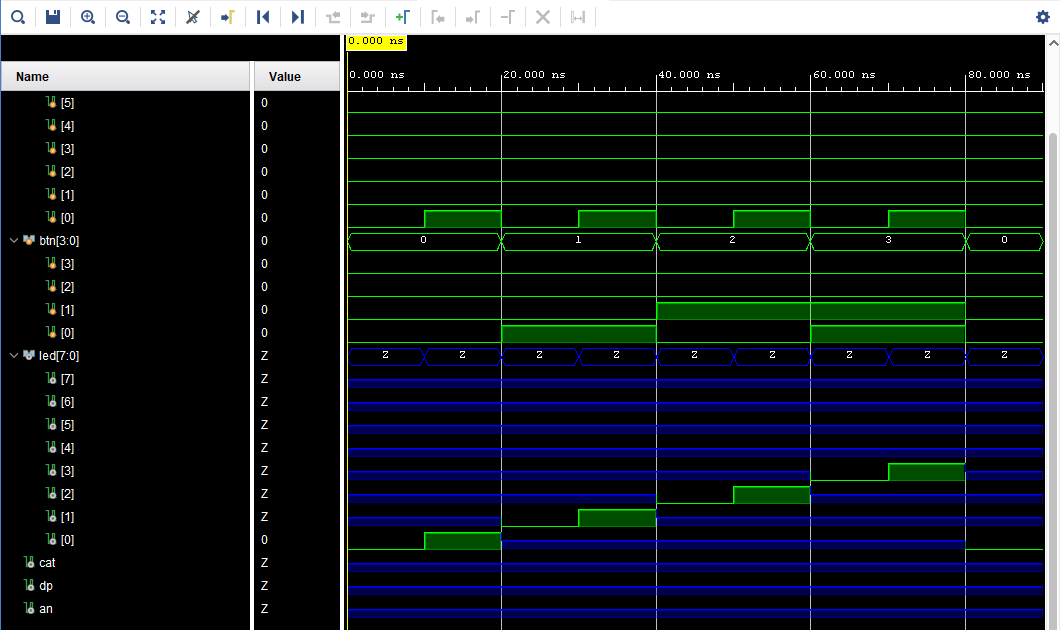
sw **=** 8'b0000\_0001**;** btn **=** 2'b11**;** **#**10**;**

sw **=** 8'b0000\_0000**;** btn **=** 2'b00**;** **#**10**;**

$finish**();**

**end**

**endmodule**



DEMUX Behavioral Simulation

A diagram of a computer

Description automatically generated

DEMUX Technology Schematic - Active Outputs

**5. Research**

**Latches:** Latches are memory elements that stores a single bit of information. D-Latches are the most common latches in the FPGA but there are also SR, JK and T latch types. In HDL design the latches are almost always refers to an error in the design due to lack of field of operation it required. A latch does not need a clock to operate which means a latch might work out of time. This creates a possibility of circuit break-down in the process so, the designers should avoid implementing a design with latches in it.1

**Leading Zero Counters:** It is a circuit that commonly used for floating-point arithmetic, data compressing and digital signal processing. The circuit counts the zeroes of a binary code until the first countered one.2

To make such a circuit, it is common to use a priority encoder which is a digital circuit that takes multiple inputs to output the highest-priority active input (1 in binary number). Let say there is a binary number with the total bit length of **N**. After finding the highest-priority one with a position of **K** with a priority encoder, leading zero count can be found by **Z = N – K – 1** expression where the **Z** is the number of zeroes counted.

**Fan-in and fan-out:** Fan-in refers to the number of inputs a gate can accept and fan-out refers to the number of outputs a gate can drive. In practice a fan-in limitation is use for keeping the delays manageable and avoid excessive power consumption. Fan-out in practical use aims to ensure the reliability and avoid unnecessary power consumption.3

**References**

1. Vivado Design Suite User Guide: Synthesis (UG901), Chapter: “HDL Coding Techniques, Flip-Flops, Registers and Latches”, Chapter: “Latches”
2. <https://digitalsystemdesign.in/leading-zero-counter/?srsltid=AfmBOorDdQog8Ro7shAFHPwplmMKFvdikEHBwGfNL6NohjjQh6sXHzsU#google_vignette>
3. <https://vhdlwhiz.com/terminology/fan-out/>