COL215- ASSIGNMENT 7

Linked List

COL- 215

HARSH VERMA	PARTH RATHI ARVIND
2024CS10499	2024CS50875

Table of Contents

1. Aim/Introduction 1
2. Design and Implementation 1
3. Testbench and Simulation 3
4. Constraints and Hardware Testing 5
5. Resource Utilization 6
6. Schematics 7
7. Conclusion 7

1. Aim / Introduction

The aim of this experiment is to design and implement a singly linked list in Verilog HDL on an FPGA. The system supports four key operations — insertion at head, insertion at tail, deletion of a node, and traversal of the list, with results shown on a 7-segment display. The design uses arrays and pointer registers to manage node data and linking, controlled through a finite state machine (FSM). A reset condition initializes the list and displays a reset pattern on the 7-segment display. Overflow (when the list is full) and underflow (when the list is empty during delete) are indicated through LEDs, ensuring safe and error-free operation.

2. Design and Implementation

Top Module: linked_list_top

The top module acts as the integration layer for the entire design. It connects the FPGA input switches and button to the linked list core and routes the output to the LEDs and the seven-segment display.

Core Logic Module: linked_list_core

This module performs all **linked list operations** using arrays and pointer-based logic.

- Node Storage: Two arrays store node data and next pointers. A free list manages available nodes.
- Operations Supported: Insert at head, insert at tail, delete a node matching input data, traverse and display list elements.
- Control States: Finite state machine handles the sequence: idle, insertions, deletion (search and execute), traversal, reset initialization, and display states.
- Overflow and Underflow Flags: Flags signal when operations are invalid—like insertion when list is full or deletion from empty list.
- Display Output: During traversal, node data is displayed using the seven-segment display.

Node Structure:

- Data (8 bits) – Stored in node_data[]
- Next Pointer (Index) – Stored in node_next[]

Finite State Machine Operations:

- **S_IDLE** – Waits for input operations.
- **S_INSERT_HEAD / S_INSERT_TAIL** – Inserts node at head or tail.
- **S_DELETE_SEARCH / S_DELETE_EXEC** – Searches and deletes a node.
- **S_TRAVERSE_START / S_TRAVERSE_DISPLAY** – Displays list contents sequentially.
- **S_RESET_PULSE / S_RESET_DISPLAY / S_RESET_INIT** – Handles reset and "-rSt" display.

Sub Module: Seven_segment_driver

This module multiplexes four digits of a seven-segment display. It takes digits as inputs and drives the anode and cathode pins to show hexadecimal values. It also displays "-rst" for 5 seconds when reset button is used.

While traversing each element is displayed for 2 seconds on an[0] and an[1] in hexadecimal representation.

Sub Module: Debouncer

The reset push button is debounced with a counter-based filter to avoid spurious triggers. Signal changes in very short interval will be discarded.

Some decisions with respect to the design

- Switches sw15 : sw0 are used to control operation codes and input data concisely. Out of this sw7 : sw0 is used for 8-bit input data and sw15 :sw13 for the operation to be perfomed.
- Overflow and underflow conditions are flagged on separate LEDs for clear feedback.
- Overflow occurs when element is added after max sized list is reached.
- Underflow occurs when we are trying to delete in empty list.
- Debounced reset with "-rSt" display for clear system initialization.
- Parameterizable SIZE allows scalable design for educational or test workloads
- FSM carries out all the operations
- If element to be deleted is not found, then nothing is done.

3. Testbench and Simulation

The testbench verifies the following scenarios:

-Insert Head – Inserts multiple elements and confirms the updated head pointer.

-Insert Tail – Appends elements at the end and checks proper tail linkage.

-Delete Node – Deletes node with given data; ensures proper linkage update. –

-Traverse – Sequentially displays all node data.

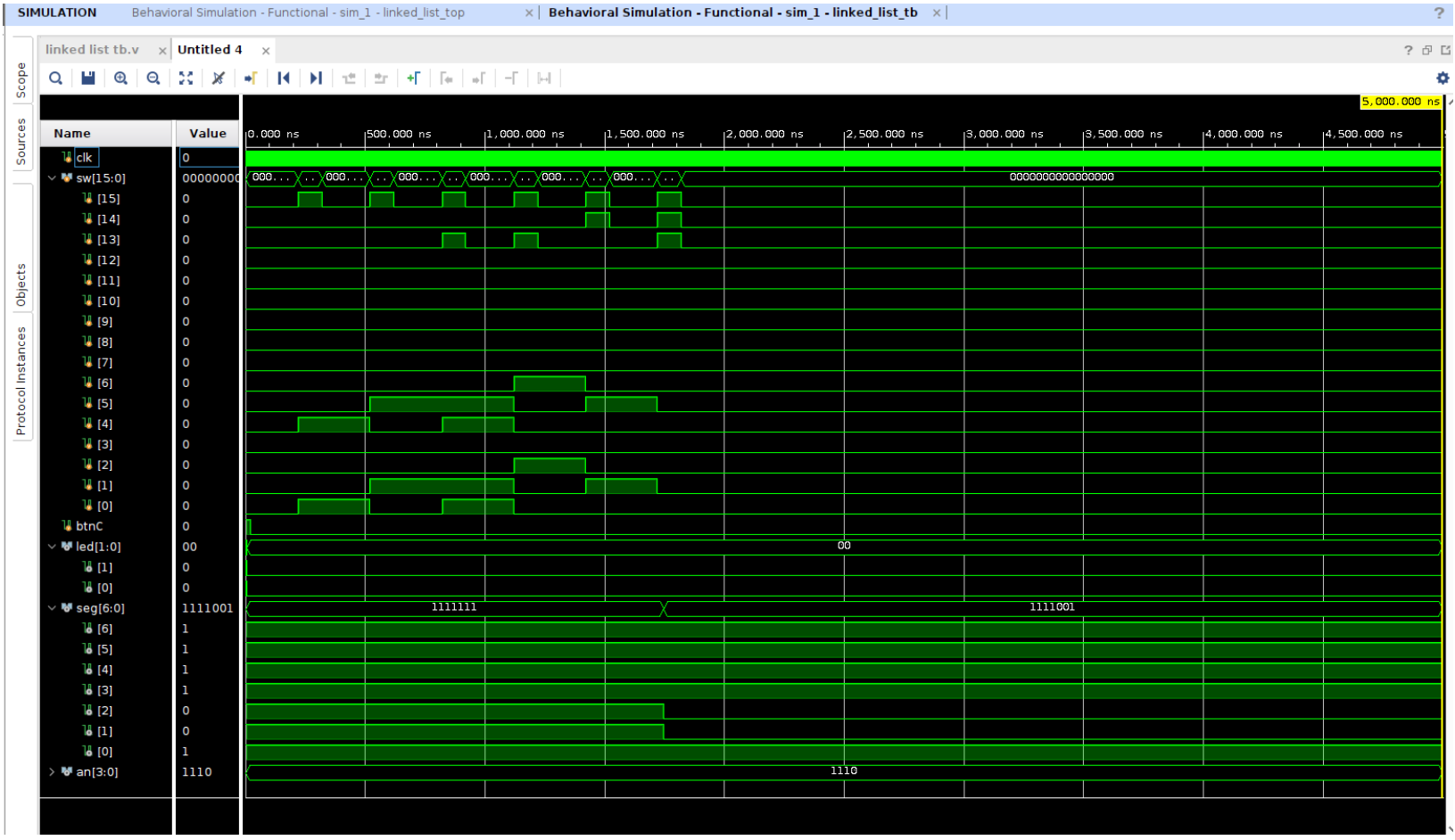
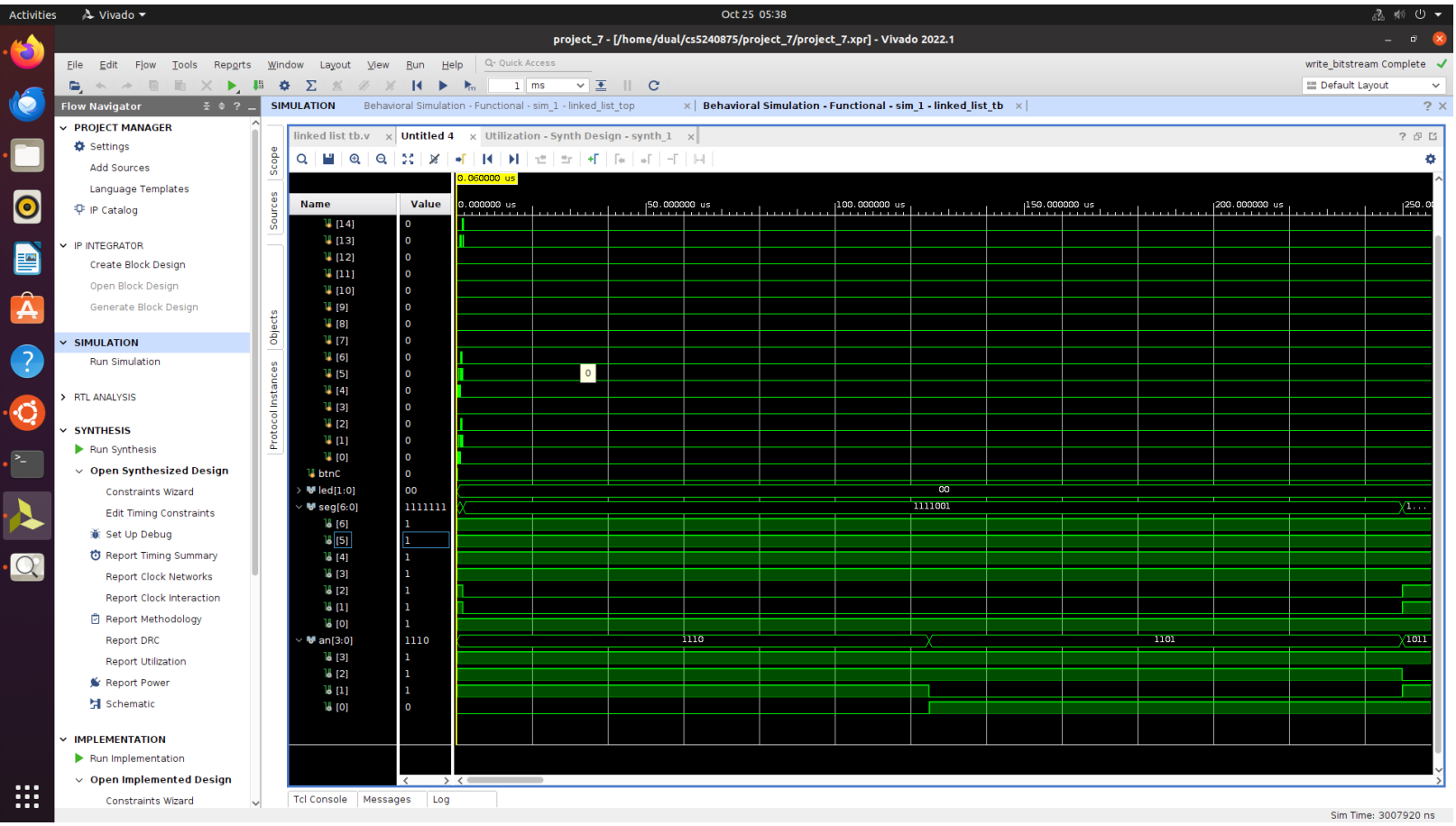


Figure 1 This shows the insert head, insert tail, delete element and traversal operation



This shows the values displayed on the anodes

Note that the simulation had to be run for longer time to display all the results of the anode.

4. Constraints and Hardware Testing

The design was synthesized on the **Basys3 FPGA** board. Switches SW7–SW0 and SW15-13 were mapped according to the assignment specifications.

During hardware testing, the following were verified:
- Proper reset indication on the 7-segment display.
- Correct insertion of data at the head and tail
- Successful deletion of the element

- Traversal of the list displayed all the element

- Underflow and Overflow conditions were observed through leds.

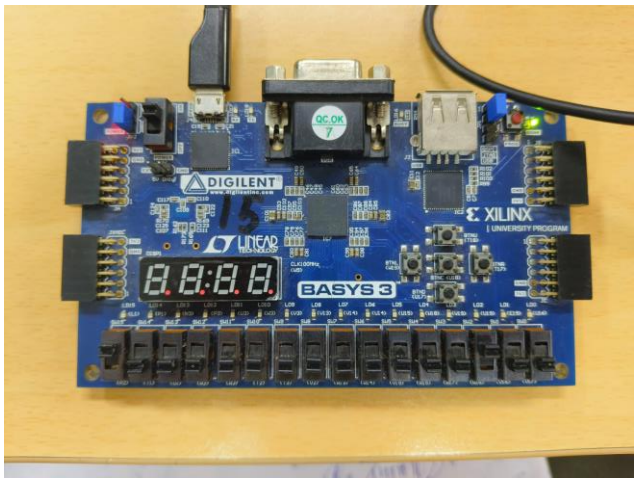


Figure 2 Inserting Head element = 4

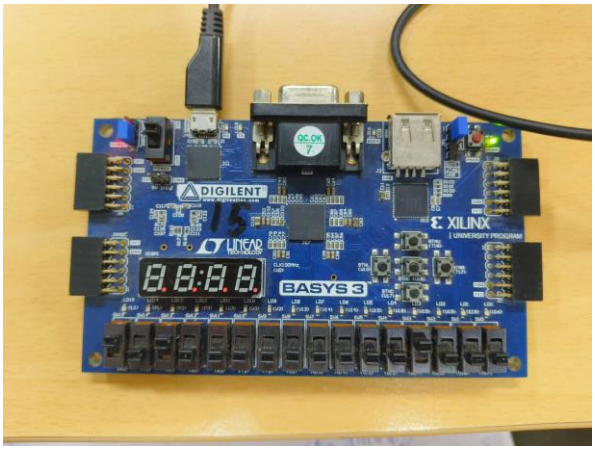


Figure 3 Inserting tail element = 8

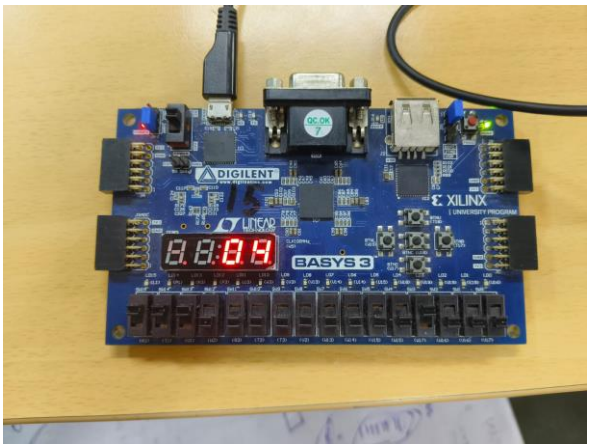


Figure 5 Traversing. element shown=4

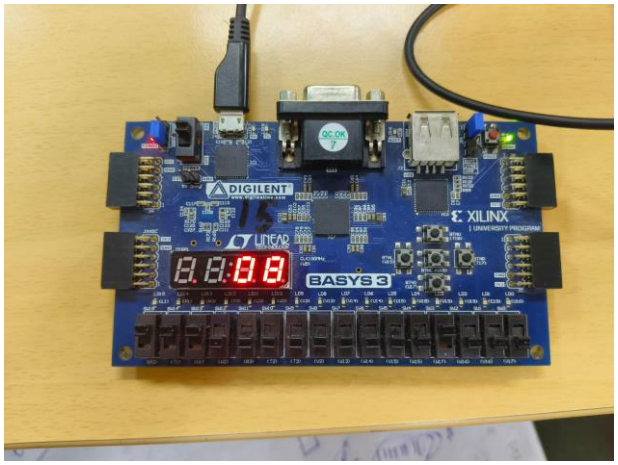


Figure 4 Traversing. Element shown =8

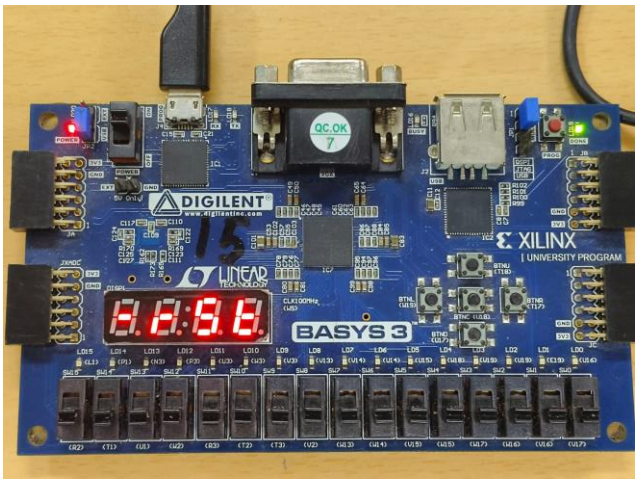


Figure 6 Reset displayed

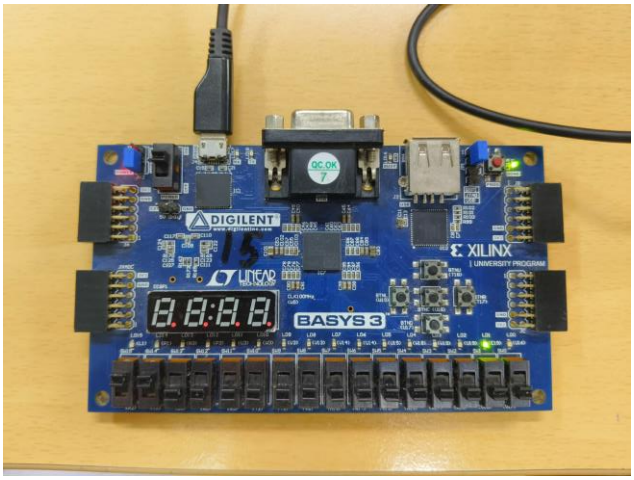


Figure 7 Underflow

5. Resource Utilization

Resource utilization after synthesis and implementation.

- No Block RAMs or DSP slices are used, as the linked list data structure relies entirely on register-based storage (distributed memory).
- The modest Flip-Flop usage highlights the sequential nature of the FSM and linked list operations.
- LUT usage reflects the combinational logic used for control flow and pointer manipulation.

Resource Type	Used	Available	Utilization (%)
LUTs	757	20800	3.64
Block RAMs	0	50	0.00
DSPs	0	90	0.00
Flip Flops	303	41600	0.728

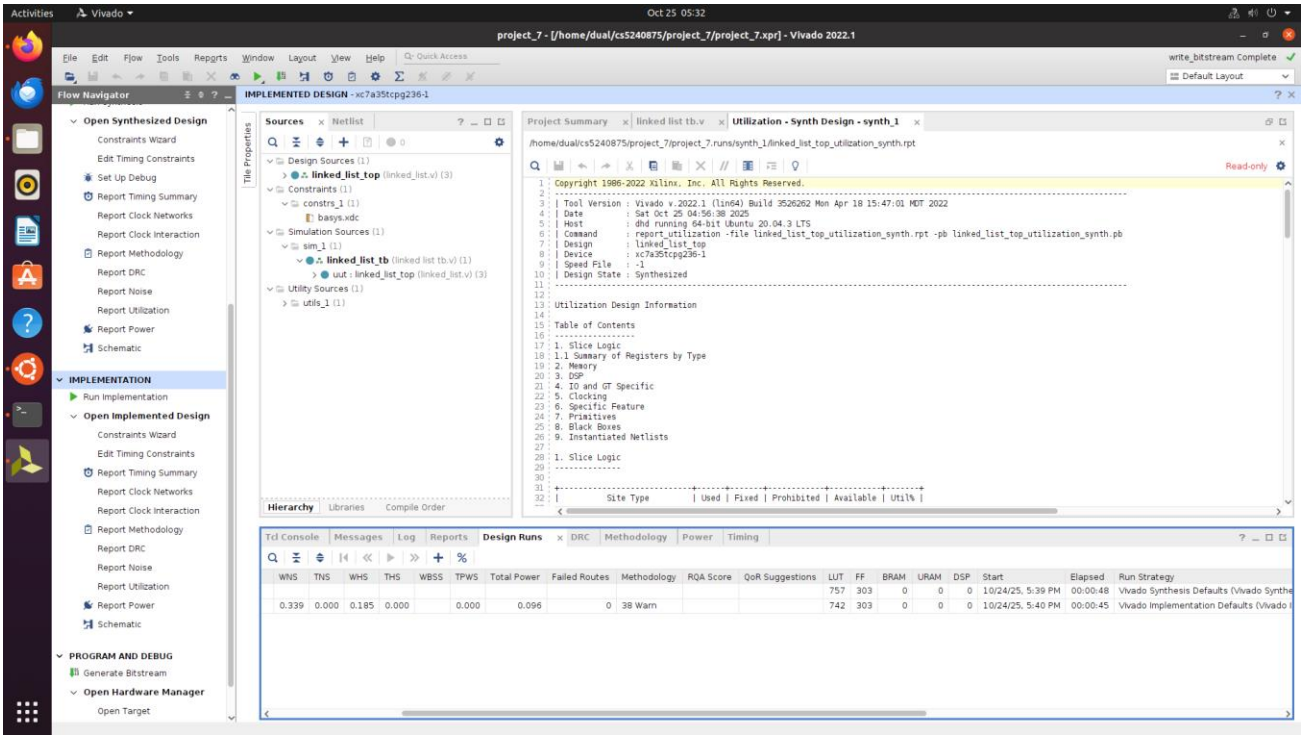


Figure 8 Luts and flip flops

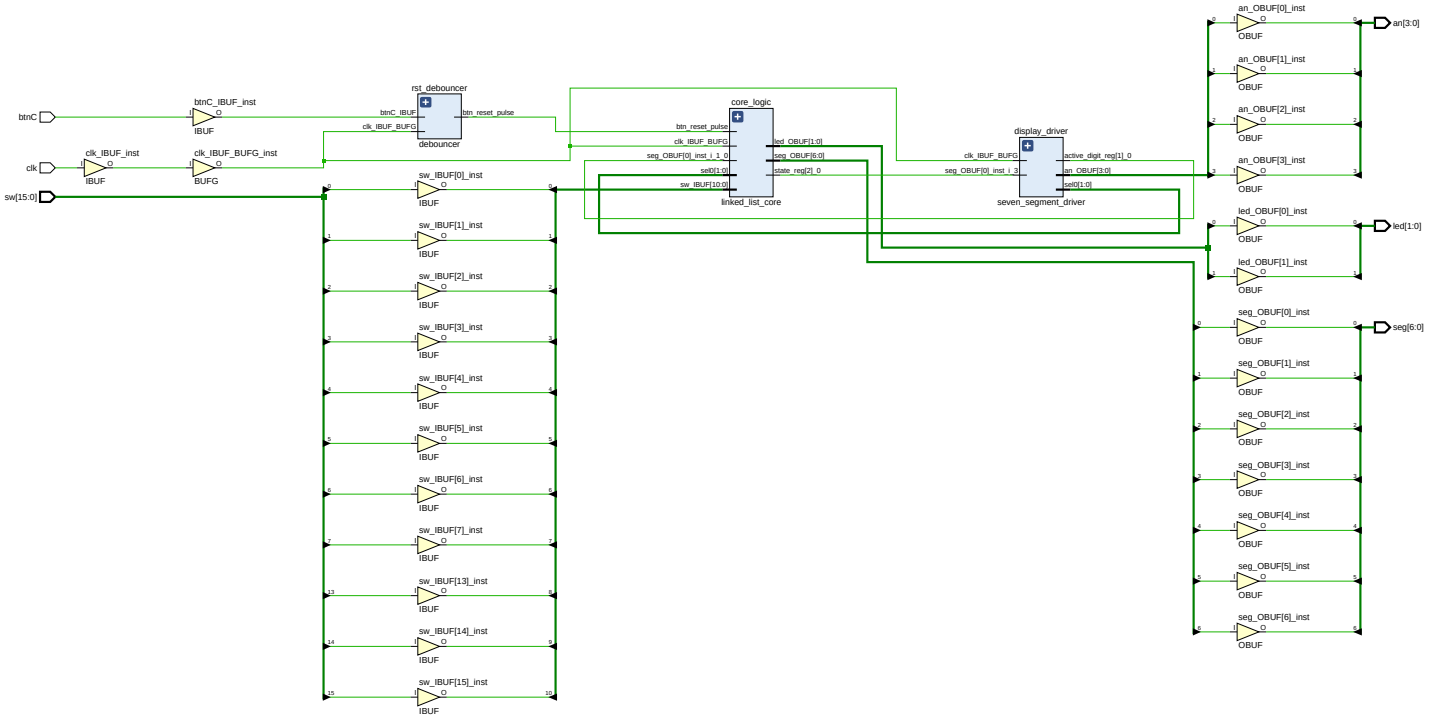
6. Schematics

The synthesized schematic consists of: A control FSM managing state transitions; Node storage arrays representing the linked list; Display driver block connected to seven-segment outputs and Debouncer module ensuring stable reset input.

Note: Schematics is shown on next page for better clarity.

7. Conclusion

The linked list hardware system was successfully implemented and verified through simulation and FPGA testing. It correctly performs all core operations—insert, delete, traverse, and reset—using efficient control logic and static memory management. The design demonstrated reliable functionality, clear visual feedback, and low hardware overhead, effectively showcasing how dynamic data structures can be realized through sequential and modular Verilog design on FPGA.


btnC
btnC_IBUF_inst
IBUF
clk
clk_IBUF_inst
IBUF
clk_IBUF_BUFG_inst
BUFG
sw[15:0]
rst_debouncer
btnC_IBUF
clk_IBUF_BUFG
btn_reset_pulse
debouncer
sw_IBUF[0]_inst
sw_IBUF[1]_inst
sw_IBUF[2]_inst
sw_IBUF[3]_inst
sw_IBUF[4]_inst
sw_IBUF[5]_inst
sw_IBUF[6]_inst
sw_IBUF[7]_inst
sw_IBUF[13]_inst
sw_IBUF[14]_inst
sw_IBUF[15]_inst
core_logic
btn_reset_pulse
clk_IBUF_BUFG
seg_OBUF[0]_inst_i_1_0
sel0[1:0]
sw_IBUF[10:0]
led_OBUF[1:0]
seg_OBUF[6:0]
state_reg[2]_0
linked_list_core
display_driver
clk_IBUF_BUFG
seg_OBUF[0]_inst_i_3
active_digit_reg[1]_0
an_OBUF[3:0]
sel0[1:0]
seven_segment_driver
an_OBUF[0]_inst
an_OBUF[1]_inst
an_OBUF[2]_inst
an_OBUF[3]_inst
an[3:0]
led_OBUF[0]_inst
led_OBUF[1]_inst
led[1:0]
seg_OBUF[0]_inst
seg_OBUF[1]_inst
seg_OBUF[2]_inst
seg_OBUF[3]_inst
seg_OBUF[4]_inst
seg_OBUF[5]_inst
seg_OBUF[6]_inst
seg[6:0]
OBUF