Digital Systems Lab
Seven-segment display

Konstantinos Seraskeris

October 2024

1 Summary

Report on the development and the functional testing of a **7-Segment Display Driver**.

This report consists of four parts:

- **Part A**: Implementation of a **combinational LED decoder** that receives a character and converts it into output signals which activate the corresponding segments.
- **Part B**: **Sequential activation** of the eight LED digits with a delay of 0.20µs, achieved using a **Finite State Machine (FSM)** and a clock manager circuit.
- **Part C**: **Message rotation** using a **button**.
- **Part D**: **Automatic message rotation** with a delay of 1.67 seconds.

At each stage, functionality checks and testbench simulations are conducted to ensure correctness.

2 Introduction

The objective of this project is to implement a **driver** for the eight 7-segment LED displays on the Nexys A7-100T board to achieve the **rotational display of a 16-character message**. The message can be any sequence, as long as it consists of the 16 hexadecimal characters: *0123456789abcdeF*.

- **Part A** focuses on implementing a decoder that receives a **character** and converts it into **a bit sequence** representing the segment states (on/off) to correctly display the corresponding character.
- **Part B** involves displaying the **eight digits** by **sequentially activating** each one for 0.20µs. This is achieved by multiplying the **FPGA clock** period (10ns) by **20x** and then controlling the signals based on the new, slower clock.

- **Part C** implements the **message rotation** with the press of a **button**. The 16 characters are stored in memory in a continuous register array and a counter serves as an index "pointing" to the active memory address. After the last character is displayed, the sequence loops back to the beginning, enabling **continuous rotation** of the message.
- **Part D replaces the button** functionality from Part C with a fixed delay of 1.6777214 seconds. This is achieved using a 23-bit counter that counts 8,388,607 cycles before incrementing the index by 1.

3 Part A - Implementation of a 7-Segment Decoder

3.1 Implementation:

A combinational decoder circuit converts the 4-bit input 'char' into a 7-bit output 'LED'. The 7-bit sequence represents the seven segments of a single digit, with order from MSB → LSB as 'GFEDCBA'. The decimal point (DP) remains off for all characters and is not included in the bit sequence. Additionally, bit 1 corresponds to the "off" state, while bit 0 represents the "on" state.

The implemented characters are '123456789abcdeF', and their seven-segment representations are shown in Figure 1.

In Verilog, the implementation uses an always block with the input 'char' in the sensitivity list. Inside the block, a switch/case statement is used to map all possible bit sequences for the characters, defining which segments should be on or off.

3.2 Verification:

The testbench for this circuit exhausts all possible input variations of 'char', prints the corresponding segment states, and allows for manual verification of correctness.

3.3 Final Implementation

This part of the project does not require programming the FPGA.

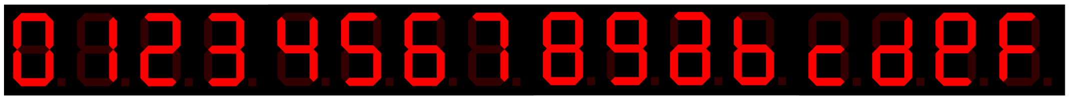


Figure 1: Representation of characters '123456789abcdeF' in this implementation.

4 Part B - Eight Digits Driver

4.1 Implementation:

This section invloves with the design of a sequential circuit with a clock and asynchronous reset. The circuit **drives the anodes and cathodes** of the segments using a **Finite State Machine (FSM)**, as shown in Table 1. In the table, the in-between rows of 1's serve the purpose of preventing overlaps between anode transitions. The dataflow of the circuit components is shown in Figure 2.

The main components and their functionality:

- MMCME2_BASE: Takes the FPGA's high-frequency clock as input and generates a slower clock ($20\times$ slower), which is used throughout the circuit.
- 2FF-synchronizer: A sequence of two flip-flops in series that prevents metastability in the RESET signal.
- debouncer: Stabilizes the input signal before passing it to the output. It waits 32,768 clock cycles (6.5ms) before passing the new signal to the output. Used for the RESET signal. Inputs:
  - The output of the 2FF-Synchronizer,
  - The XOR expression of the first and second flip-flop outputs from the 2FF-Synchronizer.

  Inside an always block with the sensitivity list consisting of the positive edge of the clock:

  - If the XOR expression is 1, the signal has changed, and the counter is reset..
  - Otherwise, if the signal remains stable for 32,768 cycles, the counter reaches zero, and the new value is passed to the output.
  - Otherwise, the circuit is in an intermediate state, and the counter decrements by 1.
- counter: A 5-bit counter used by the FSM.
- AN-FSM (Anode FSM): Controls the anodes based on specific counter values according to the state diagram Table 1.

  Inside an always block with the sensitivity list consisting of the positive edge of both the clock and the reset signal:

  - If reset $= 1$, all anodes are set high (1).
  - Otherwise, in a switch/case, each anode is set low (0) one cycle before its corresponding counter value appears in the state diagram.
  - Otherwise, in the default case, all anodes are set high (1).
- CA-FSM (Cathode FSM): Loads the seven-segment values two cycles before the anode is pulled low (0) for a given digit and holds them for one cycle after.

  Inside an always block with the sensitivity list consisting of the positive edge of both the clock and the reset signal:

  - If reset $= 1$, all segment displays are off.

  - Otherwise, in a switch/case, the segments are activated **one cycle before** the cathodes are lowered, according to the LED decoder output.

- feedDecoder: Loads the character into the LED decoder **one cycle before** the cathodes are pulled low (0) in CA-FSM. The feedDecoder acts **two cycles before** the counter reaches the point where the anode is lowered. This ensures that the new character appears in the next cycle rather than the previous one.

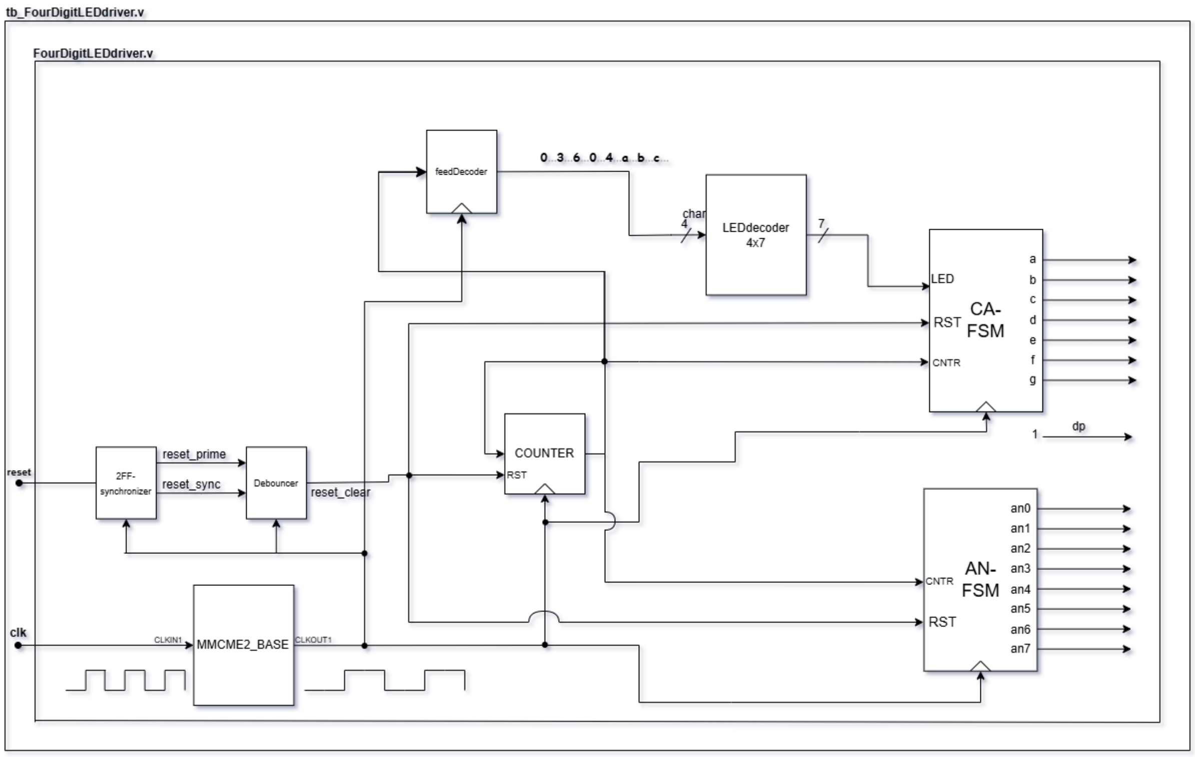


Figure 2: Circuit Dataflow in Part B

Counter Value	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0
11111	1	1	1	1	1	1	1	1
11110	0	1	1	1	1	1	1	1
11101	1	1	1	1	1	1	1	1
11100	1	1	1	1	1	1	1	1
11011	1	1	1	1	1	1	1	1
11010	1	0	1	1	1	1	1	1
11001	1	1	1	1	1	1	1	1
11000	1	1	1	1	1	1	1	1
10111	1	1	1	1	1	1	1	1
10110	1	1	0	1	1	1	1	1
10101	1	1	1	1	1	1	1	1
10100	1	1	1	1	1	1	1	1
10011	1	1	1	1	1	1	1	1
10010	1	1	1	0	1	1	1	1
10001	1	1	1	1	1	1	1	1
10000	1	1	1	1	1	1	1	1
01111	1	1	1	1	1	1	1	1
01110	1	1	1	1	0	1	1	1
01101	1	1	1	1	1	1	1	1
01100	1	1	1	1	1	1	1	1
01011	1	1	1	1	1	1	1	1
01010	1	1	1	1	1	0	1	1
01001	1	1	1	1	1	1	1	1
01000	1	1	1	1	1	1	1	1
00111	1	1	1	1	1	1	1	1
00110	1	1	1	1	1	1	0	1
00101	1	1	1	1	1	1	1	1
00100	1	1	1	1	1	1	1	1
00011	1	1	1	1	1	1	1	1
00010	1	1	1	1	1	1	1	0
00001	1	1	1	1	1	1	1	1
00000	1	1	1	1	1	1	1	1

Table 1: FSM for Anode Activation

4.2 Verification:

The displayed message is: '03604abc', which has been preloaded into the feedDecoder module, mapping each character to a specific anode. For the counter value that drives the cathodes of anode 7, the LEDdecoder loads the character '0'. For anode 6, it loads '3'. For anode 5, it loads '6', and so on. In the testbench, we first apply the RESET signal for a short duration and then for a longer one to verify the debouncer functionality.

After releasing the reset, we allow the simulation to run for a sufficient time and then examine the waveforms to ensure that the displayed characters appear correctly and at the expected time, as illustrated in Figure 3 and Figure 4.

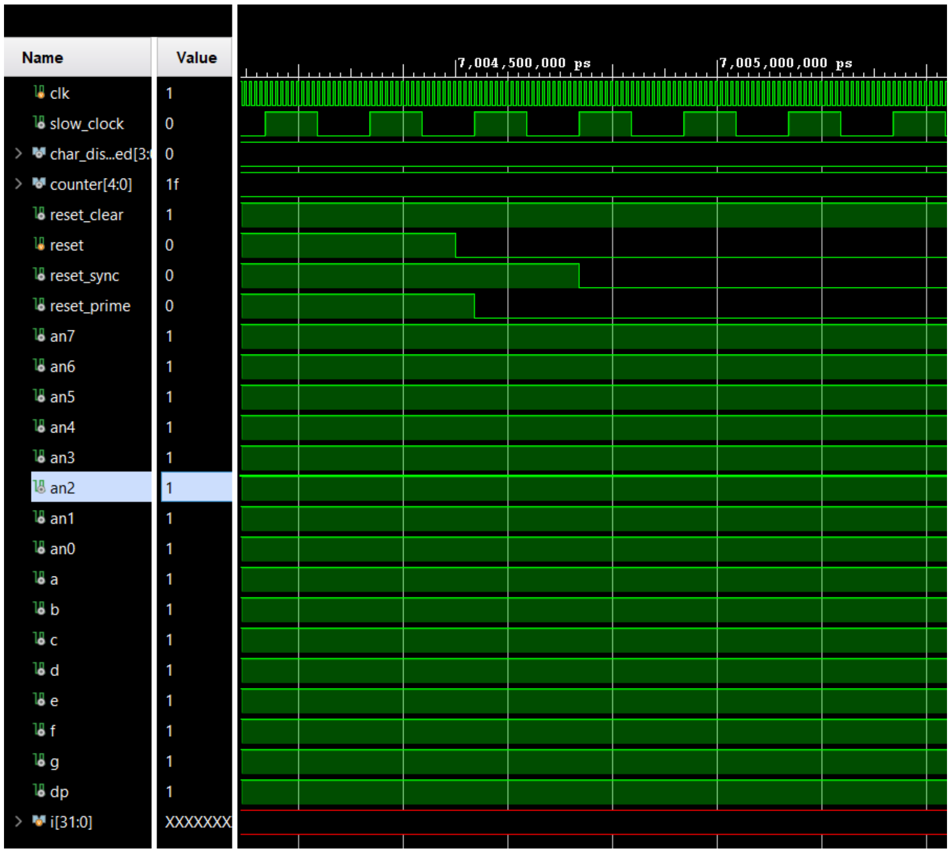


Figure 3: Operation of the synchronizer (reset_prime is the output of the first and reset_sync is the output of the second flip-flop)

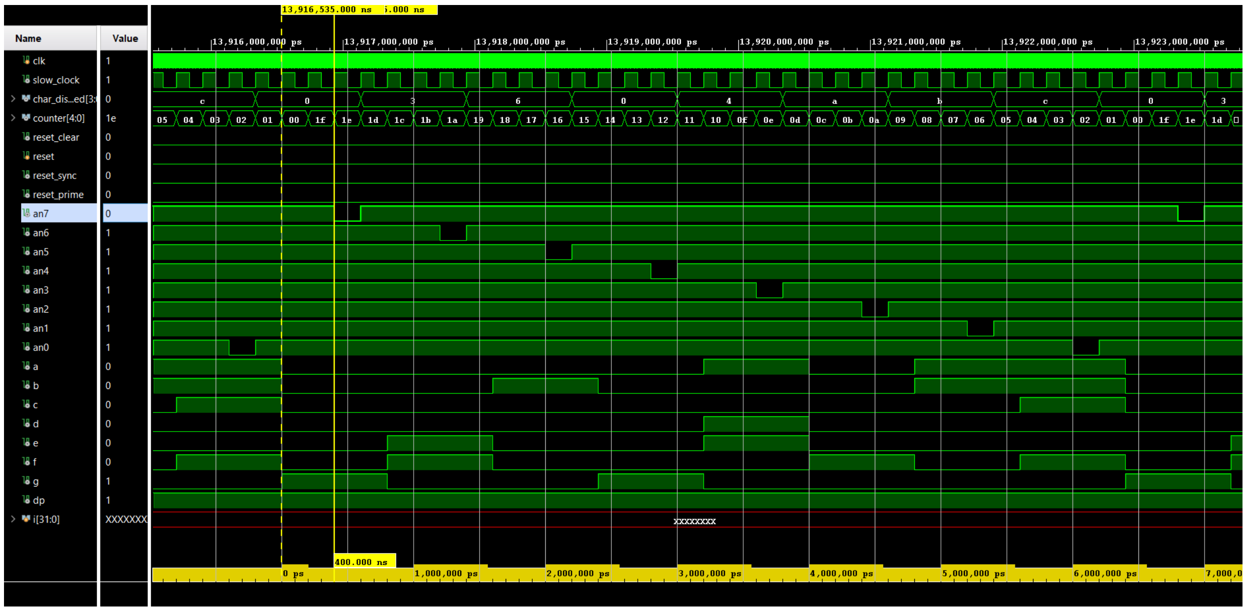


Figure 4: Circuit Operation (Note: The cathodes are activated 2 cycles before the anode, while the char displayed changes 1 cycle before the cathode's activation)

4.3 Final Implementation:

The final result is shown in Figure 5.

5 Part C - Stepwise Rotation of the Message Using a Button

5.1 Implementation:

In this part, the message consists of 16 characters and rotates each time a button is pressed. A counter/index determines the starting location from which the 8 displayed characters are selected. The circuit builds upon the design from Part B, with **additional functionality** for **handling the button signal** and using it as a trigger to **change the memory address**. The dataflow of the circuit components is shown in Figure 6.

The new modules and their functions:

- 2FF-synchronizer, debouncer: Now also used for the button input.
- SinglePulse: Converts the long button press into a single pulse signal: The circuit in Verilog:

```
    always @(posedge clk) begin
        q1 <= signal;
        q2 <= signal & q1;
    end

    assign out = q1 & ~q2;
```

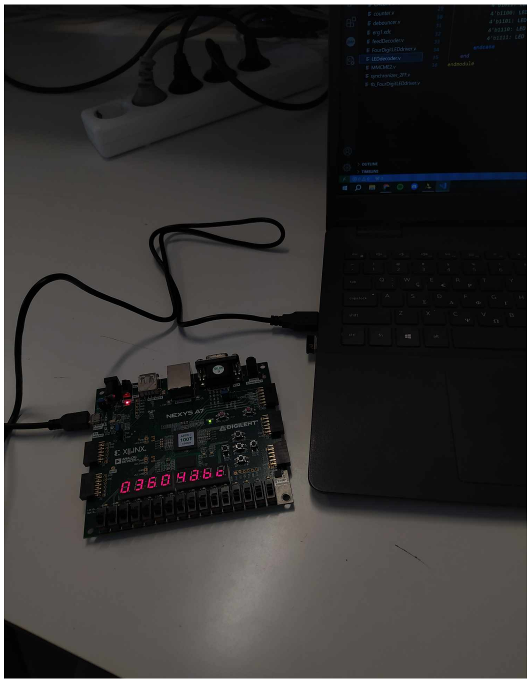


Figure 5: Part B displayed on the FPGA board

- ADDRESSchange:A counter that increments only when the button signal is triggered (i.e., on the SinglePulse output). Inside an always block with the sensitivity list consisting of the positive edge of both the clock and the reset signal:
  - If reset is active, the index resets to zero.
  - If the button is pressed, the index:
    - Resets to zero if it reaches the maximum limit.
    - Otherwise, increments by 1.

- feedDecoder (new version): Uses the address index from ADDRESSchange to determine which 8 characters are displayed. Since the message wraps around, we ensure that indices beyond 15 wrap back to 0 manually:

```
char_displayed_i <= (pointer+i > 15) ?
                    message[pointer + i - 16]:
                    message[pointer + i];
```

Example for index 14:

- First digit → 14 + 0 = 14 (valid, no wraparound)
- Third digit → 14 + 2 = 16 (exceeds 15, wraps around to 0)

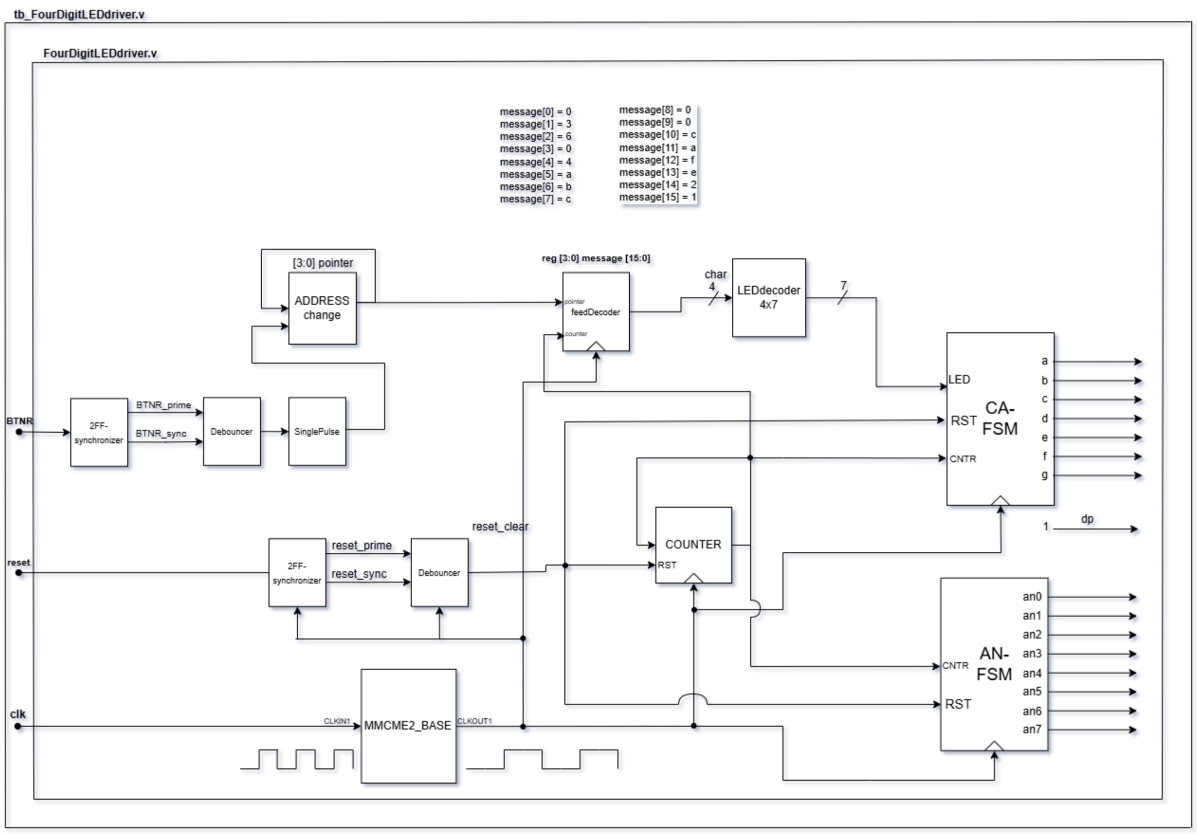


Figure 6: Circuit Dataflow in Part C

5.2 Verification:

The complete message displayed after a full rotation: '03604abc00cafe21' which has been initially stored in the feedDecoder module as reg [3:0] message [0:15] and is reinitialized each time the reset is activated.

In the testbench, we first activate the RESET signal for a shorter duration and then for a longer duration to verify the functionality of the debouncer. Then, upon the falling edge of the reset, we activate the BUTTON for a sufficient amount of time before deactivating it.

This process is repeated at least 16 times to ensure a full rotation of the message, verify the pointer's operation, and check the correct display of all characters. Finally, we examine the waveforms to confirm the correct indications and timings, as shown in Figure 7, Figure 8 and Figure 9.

5.3 Final Implementation:

The final result is shown in Figure 10, Figure 11 and Figure 12.

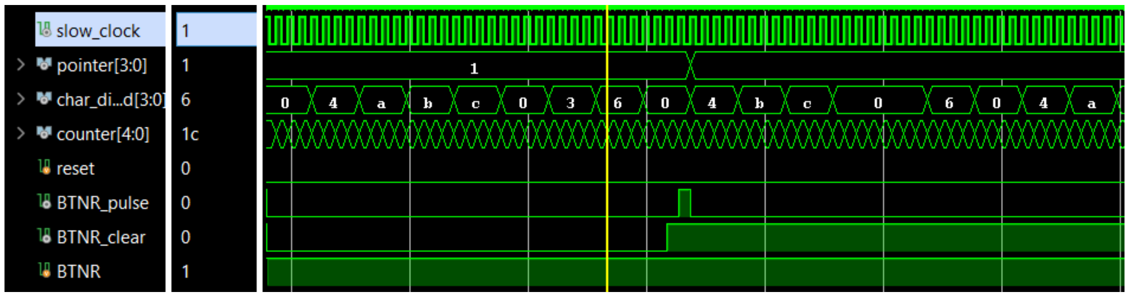


Figure 7: Conversion of BUTTON signal into a single pulse.

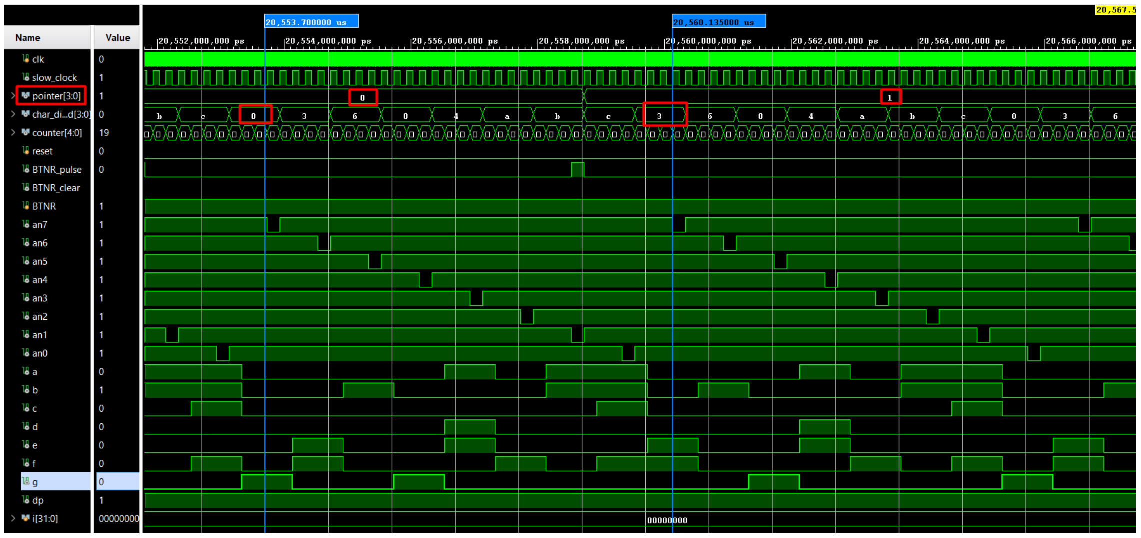


Figure 8: Address index changes with correct message shifting.

6 Part D - Stepwise Rotation with Fixed Delay

6.1 Implementation:

This part extends Part C but removes the button input. Instead, the message rotates automatically based on a clock-driven delay. Specifically, the address index updates when an 8,388,607-cycle counter (1.6777214s) expires. The other modules remain unchanged. The updated design is shown in Figure 13.

The new modules and their functions:

- ADDRESSchange (new version): A 23-bit counter increments the address index automatically.

  Inside an always block with the sensitivity list consisting of the positive edge of both the clock and the reset signal:

  - If reset is active, the index resets to zero.
  - If the counter expires, the index:
    - Resets to zero if at the maximum.
    - Otherwise, increments by 1.

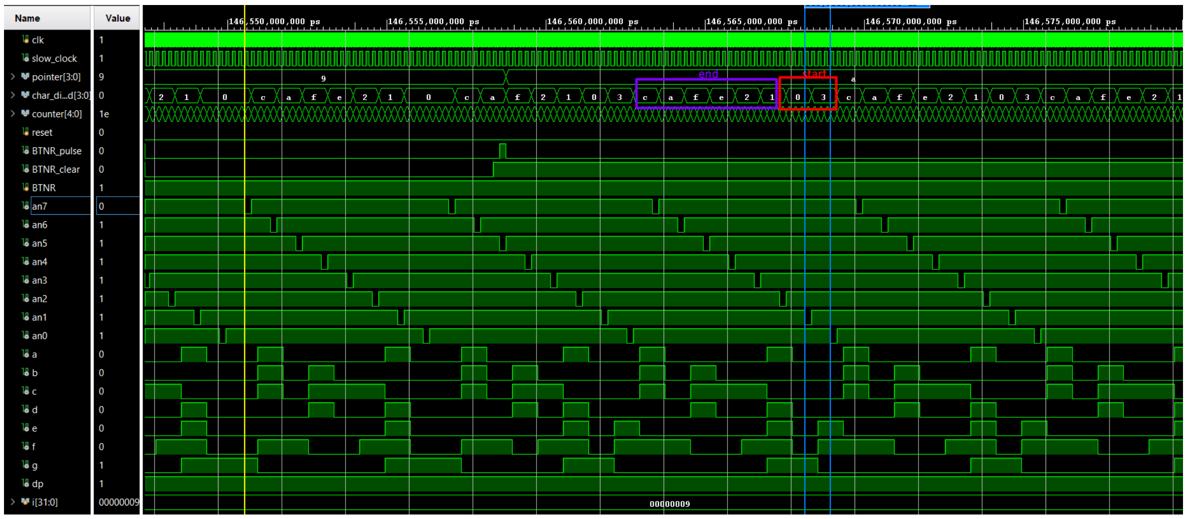
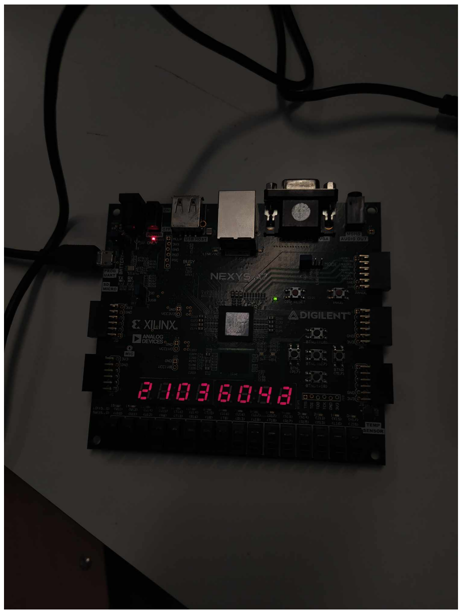
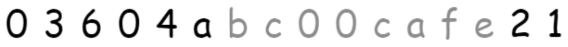


Figure 9: Full message rotation, ensuring proper looping.



(a) The FPGA board

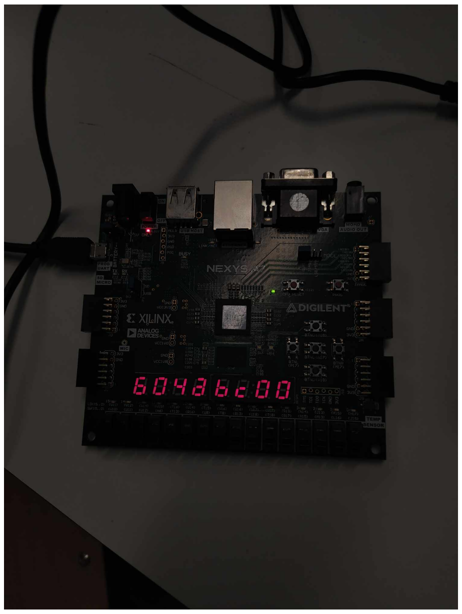


(b) The part of the message being displayed

Figure 10: Part C displayed on the FPGA board

6.2 Verification:

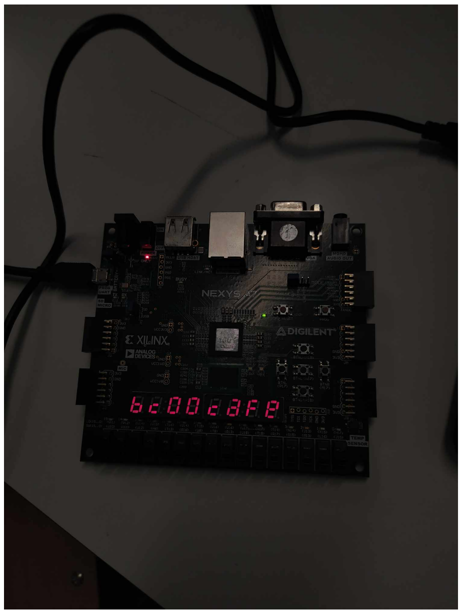
The testbench is left running for several seconds to observe multiple counter expirations and message shifts. The expected waveforms are shown in Figure 14 and Figure 15.



0 3 6 0 4 a b c 0 0 c a f e 2 1

(a) The FPGA board (b) The part of the message being displayed

Figure 11: Part C displayed on the FPGA board



0 3 6 0 4 a b c 0 0 c a f e 2 1

(a) The FPGA board (b) The part of the message being displayed

Figure 12: Part C displayed on the FPGA board

6.3 Final Implementation:

The final results are shown in Figure 10, Figure 11 and Figure 12.

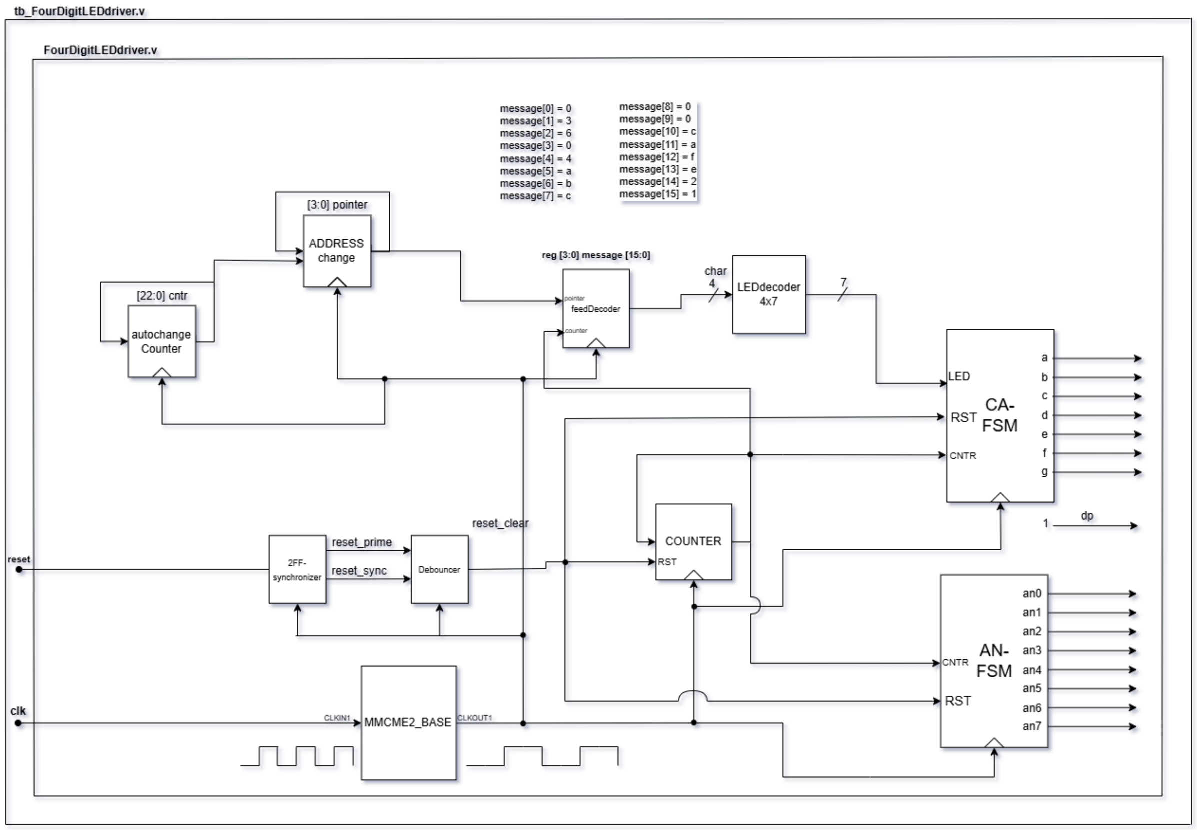


Figure 13: Dataflow κυκλώματος στο Μέρος Δ

7 Conclusions

The different parts of the project were first designed in Dataflow, and after verifying the correctness of their interconnections, they were implemented in Verilog. The verification was primarily conducted using the testbench and signal waveforms.

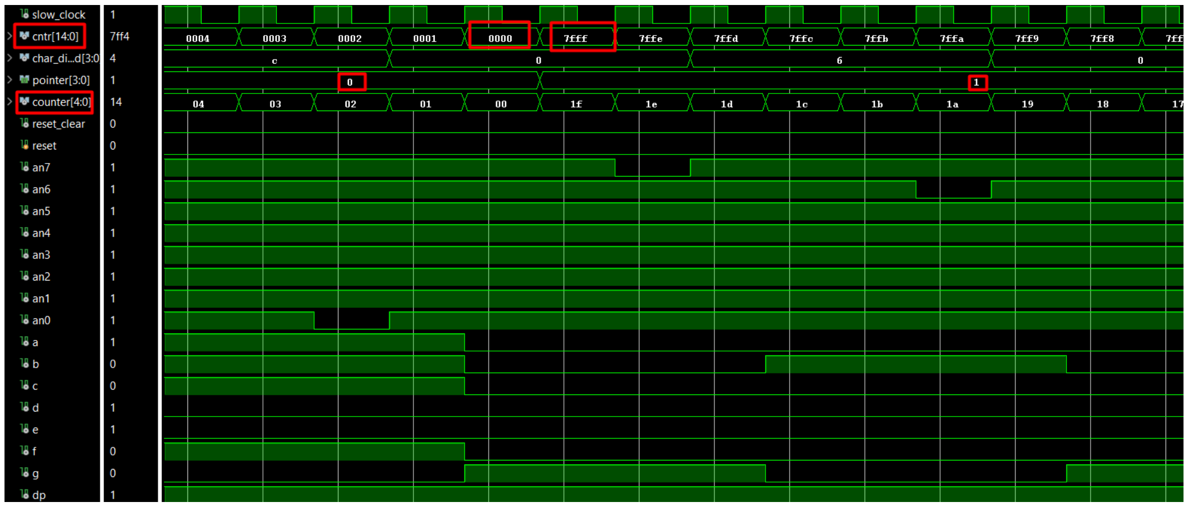


Figure 14: Reset of the counter and increase of the pointer value

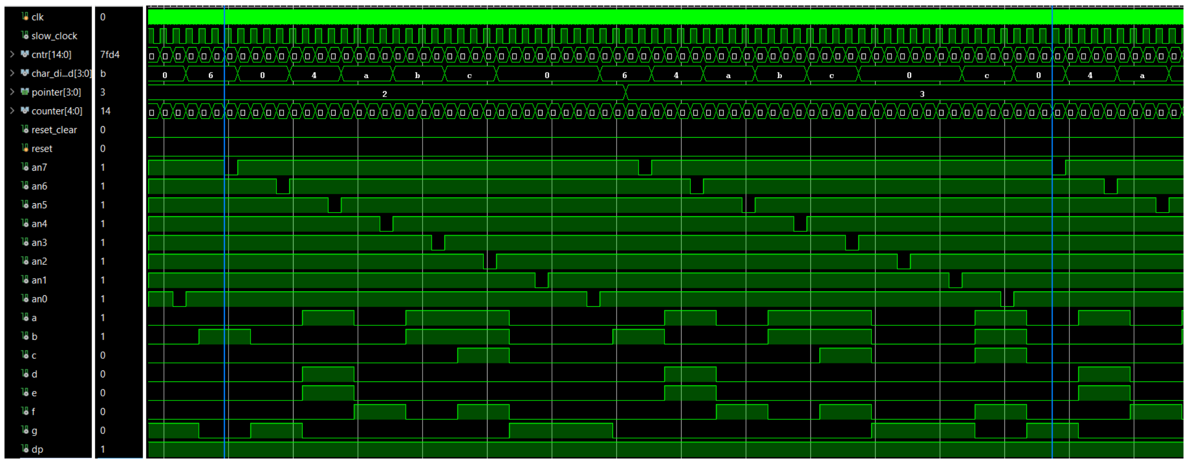


Figure 15: Verification of message shifting after counter reset