CPE 324-03: Advanced Digital Logic Design Laboratory

Lab 4

Capacitance Tester

Submitted by: Hannah Kelley

Date of Experiment: January 27 and 29 and February 2, and 5, 2026

Report Deadline: February 10, 2026

Demonstration Deadline: February 10, 2026

1 Introduction

The purpose of this laboratory project is to design and implement a combined digital–analog instrument capable of measuring the capacitance of a capacitor. This system integrates analog timing circuitry with digital logic to demonstrate the interaction between hardware components and hardware description language (HDL) design. The digital portion of the instrument is developed using Verilog HDL and deployed on the Terasic DE2-115 (lab FPGA) educational development board. The board interfaces with an external 555 timer integrated circuit, which provides a time-dependent signal related to the unknown capacitance value.

A key objective of this lab is to explore different digital design methodologies by creating two distinct Verilog implementations of the same measurement system. These versions differ significantly in their levels of abstraction, allowing comparison between lower-level structural design techniques and higher-level behavioral modeling approaches. Through this process, the lab reinforces concepts in digital logic design, hardware description languages, and mixed-signal system integration, while also providing practical experience in FPGA-based system development and real-world signal measurement.

2 Experiment Description

2.1 Theory

2.1.1 Binary-Coded Decimal (BCD)

Binary-Coded Decimal (BCD) is a method of representing decimal numbers where each digit (0–9) is encoded separately using four binary bits. For example, the decimal number 45 is written in BCD as 0100 0101, with each 4-bit group representing one decimal digit. This differs from pure binary, where the entire number is converted as a single value. BCD is widely used in digital systems that display numerical results, such as seven-segment displays, because each digit can be decoded independently into a decimal output. In digital design, BCD counters count from 0000 (0) to 1001 (9) before resetting and carrying to the next digit. When performing BCD addition, a correction of 0110 (6) must be added if a digit exceeds 9 to maintain a valid decimal representation. Overall, BCD simplifies the connection between binary hardware and human-readable decimal displays, making it useful in measurement and timing applications.

2.2 Part 1 Phase 1 - Astable Multivibrator Design Implementation on the Solderless Breadboard

The first step in this experiment was to construct the 555 IC timer in astable mode on a solderless breadboard and connect it to the laboratory FPGA system to verify proper operation. The internal configuration and pinout of the 555 IC timer are shown in Figure 1, and the completed circuit connections are illustrated in Figure 2.

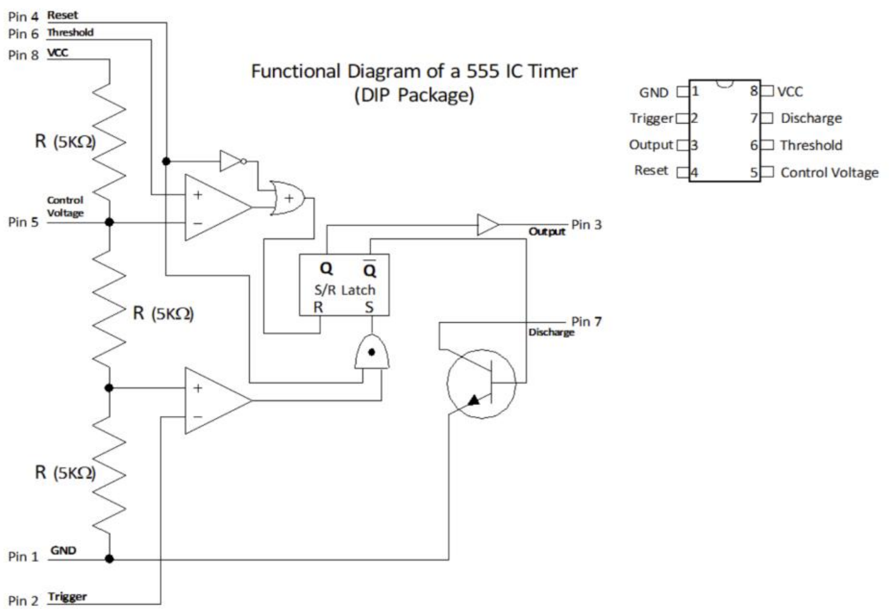


Figure 1: 555 IC Timer Internal Configuration and Pinout

Signal Name	Cyclone IV E Signal Number	DE2-115 EE2-115 UAH ECE Breakout Board	555 IC Pin Number(s)	555 IC Pin Name
VCC		5V	8, 4	V_{CC}, Reset
GND		GND	1	GND
SIG	PIN_AH23	D34	3	Output

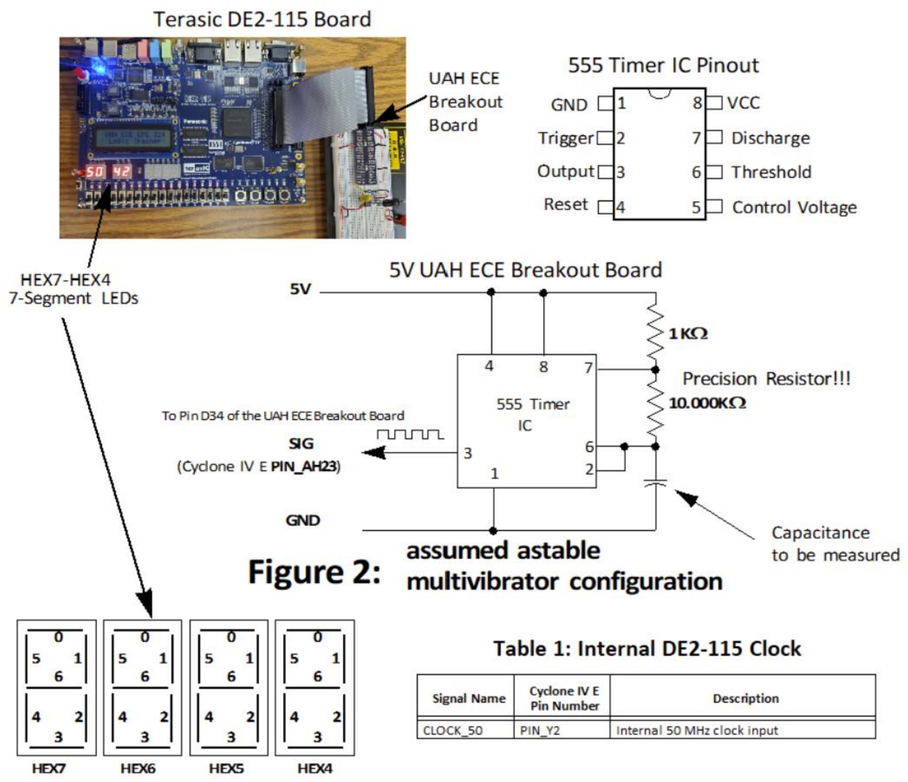


Table 1: Internal DE2-115 Clock

Signal Name	Cyclone IV E Pin Number	Description
CLOCK_50	PIN_Y2	Internal 50 MHz clock input

Figure 2: 555 IC Timer and DE2-115 Circuit Diagram

Before powering the circuit, the actual values of the 1 μF capacitor and the 10.00 kΩ resistor were measured using a digital multimeter. These measured values were recorded and later used to calculate the percent error relative to their nominal ratings.

After assembling the circuit, the 555 timer output waveform was observed using an oscilloscope to confirm oscillation. Using the measured component values and standard astable 555 timing equations, the expected discharge time (the interval during which the output is at a logic low and the capacitor discharges through the 10.00 kΩ resistor) was calculated using the derivation in the Lab 4 Overview PowerPoint (further detail in Appendix A).

$$t \approx 6931C$$
$$t \approx 6931(1\ \mu F)$$
$$t \approx 0.006931\ s = 6.931\ ms$$

The oscilloscope was then used to measure the actual discharge time directly from the waveform. A screenshot of one to two periods of the output signal was captured for inclusion in the report. Finally, the measured discharge time was compared with the theoretical value to evaluate accuracy and discuss any discrepancies.

2.3 Part 1 Phase 2 - count_enable Timing Modifications

In this phase, the digital timing of the system was configured to generate correctly spaced enable pulses for capacitance measurement. This was accomplished by modifying the Verilog parameters associated with the count_enable module when it was instantiated in the top-level design. The required timing values were determined by calculating how many 50 MHz system clock cycles occur during the discharge interval of the 555 timer when using a 0.01 μF capacitor, which defines the measurement resolution of the instrument.

The count_enable module was designed to output a single–clock-cycle enable pulse at intervals equal to the discharge time of a 0.01 μF capacitor. As a result, for larger capacitors, the module produces multiple enable pulses—one for each 0.01 μF equivalent discharge interval. To find the timing values necessary for the design to function as intended, the derivation for capacitor discharge time provided in the Lab 4 Overview PowerPoint (further detail in Appendix A) was used to obtain the discharge time for a 0.01 μF capacitor:

$$t \approx 6931C$$
$$t \approx 6931(0.01\ \mu F)$$
$$t \approx 0.00006931\ s$$

Once the discharge time was found, it could be used to find the equivalent number of clock cycles represented by the parameter full_count by multiplying the time spent discharging by the clock frequency. This value was updated in the Verilog implementation:

$$full\_count = t \cdot 50MHz$$
$$full\_count = 0.00006931\ s \cdot 50MHz$$
$$full\_count = 3465.5$$

$$full\_count \approx 3466$$

The final parameter bus_width was calculated using the following formula given in the lab documentation:

$$bus\_width = \lceil \log_2 full\_count \rceil$$
$$bus\_width = \lceil \log_2 3466 \rceil$$
$$bus\_width = \lceil 11.750 \rceil$$
$$bus\_width = 12$$

After updating the timing parameters, the design was synthesized and programmed onto the FPGA. The measured capacitance value was observed on the HEX displays; at this stage, the output appeared in hexadecimal because the counter operated in binary rather than BCD.

2.4 Part 1 Phase 3 - bcd_counter Module Modification

Next, the provided template code was modified to allow the 7-segment LED displays to show the BCD representation of the number. This was achieved by replacing the given bcd_counter module, which contained no BCD implementation, to the code found in Table 1.

```
module bcd_counter ( input clear, input en, input clk, output reg [15:0] o);

reg [15:0] count;

always @(posedge clk or posedge clear) begin
   if (clear) begin
      count <= 16'd0;
   end
   else if (en) begin
      // Ones digit
      if (count[3:0] == 4'd9) begin
         count[3:0] <= 4'd0;

         // Tens digit
         if (count[7:4] == 4'd9) begin
            count[7:4] <= 4'd0;

            // Hundreds digit
            if (count[11:8] == 4'd9) begin
               count[11:8] <= 4'd0;

               // Thousands digit
               if (count[15:12] == 4'd9)
                  count[15:12] <= 4'd0;
               else
                  count[15:12] <= count[15:12] + 4'd1;
            end else begin
               count[11:8] <= count[11:8] + 4'd1;
            end

         end else begin
            count[7:4] <= count[7:4] + 4'd1;
         end

      end else begin
         count[3:0] <= count[3:0] + 4'd1;
      end
   end
end

always @(*) begin
   o = count;
end

endmodule
```

Table 1: bcd_counter Module Code

The design was then synthesized and programmed onto the lab FPGA. The measured capacitance value was observed on the HEX displays in BCD. The display was still unstable as debouncing was not yet implemented.

The design was synthesized and programmed onto the FPGA. The measured capacitance value was observed on the HEX displays; at this stage, the output appeared in BCD and not hexadecimal.

2.5 Part 1 Phase 4 - Creation of the Debounce/Metastability Filter Musing Schematic Capture Design Entry Techniques

In this phase of the experiment, a basic debouncing module was developed to allow the 7-segment LED displays to show a more stable output. To implement the filter, a structural design was created using eight D flip-flops connected in series to form an 8-bit shift register. The asynchronous input signal was applied to the first flip-flop and shifted through the register on each clock cycle. The outputs of all eight stages were then monitored using both an 8-input AND gate and an 8-input NOR gate. The code for this module is shown in Table 2.

```
module debounce(output sig_out,input sig_in,clk);

  wire [7:0] q;
         wire ones;
         wire zeroes;

         // 8 bit shift reg
```

```
        d_ff d0(sig_in, clk, q[0]);
        d_ff d1(q[0], clk, q[1]);
        d_ff d2(q[1], clk, q[2]);
        d_ff d3(q[2], clk, q[3]);
        d_ff d4(q[3], clk, q[4]);
        d_ff d5(q[4], clk, q[5]);
        d_ff d6(q[5], clk, q[6]);
        d_ff d07(q[6], clk, q[7]);

        assign ones = &q;       // are all dffs 1?
        assign zeroes = ~|q;    // are all dffs 0?

   jk_ff out_ff(ones, zeroes, clk, sig_out);

endmodule
```

Table 2: debounce Module Code

The AND gate produces a logic HIGH only when all eight flip-flop outputs are HIGH, while the NOR gate produces a logic HIGH only when all eight outputs are LOW. These two signals were used to drive the inputs of a JK flip-flop configured to change state only when either condition is met. As a result, the output of the JK flip-flop transitions only after the input signal has remained consistently HIGH or LOW for eight consecutive clock cycles.

This design ensures that any brief glitches or bounce-induced transitions shorter than eight clock periods do not propagate to the output. The circuit therefore acts as a digital debounce filter, allowing output transitions only after the input signal has stabilized. The expected timing behavior for both LOW-to-HIGH and HIGH-to-LOW transitions is illustrated in Figure 3.

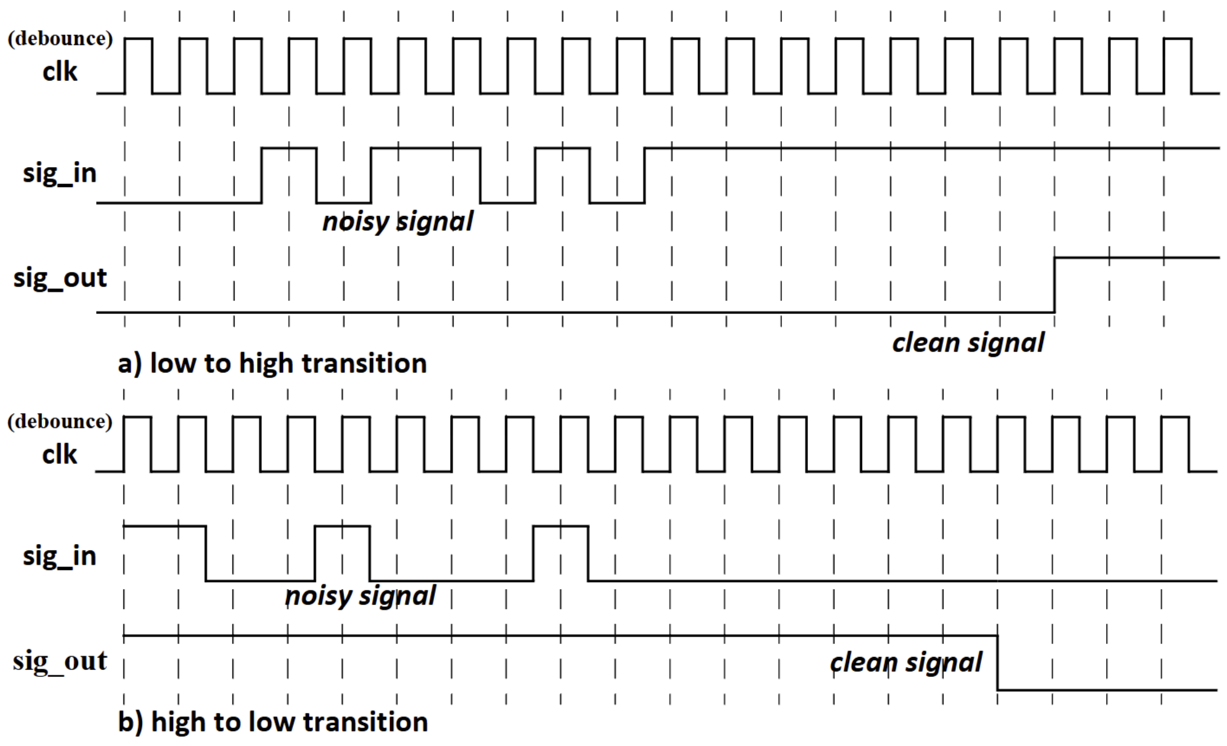


Figure 3: Debounce Expected Output

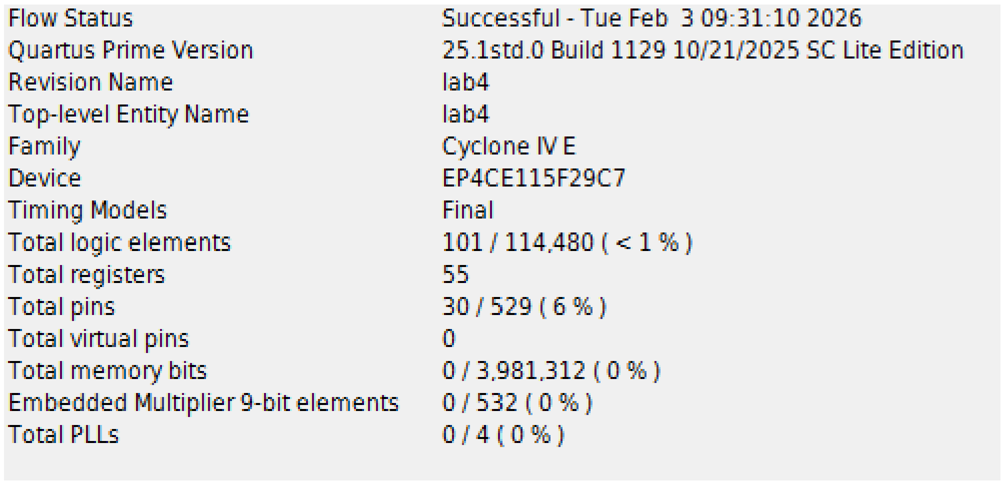
The design was synthesized and programmed onto the FPGA. The measured capacitance value was observed on the HEX displays with much less fluctuation than the previous two partial designs. The final Verilog code for Part 1 can be found in Appendix B.

2.6 Part 2 - Single Module Behavioral Design

After completing the structural design, a behavioral design was implemented to produce the same output as the structural design. Three templates were given as a starting point. I chose template 1 to develop into the final behavioral design. The full behavioral design code can be found in Appendix C.

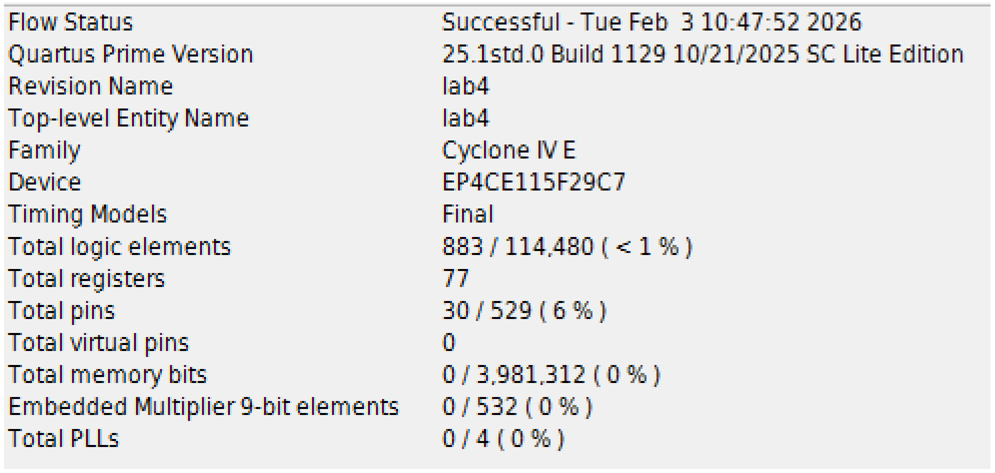
3 Results

The structural design in Part 1 and the beahivoral design in Part 2 were overall very similar, though they differed slightly in FPGA resource allocation. The total FPGA resource allocation for Parts 1 and are shown in Figures 4 and 5, respectively.



Flow Status	Successful - Tue Feb 3 09:31:10 2026
Quartus Prime Version	25.1std.0 Build 1129 10/21/2025 SC Lite Edition
Revision Name	lab4
Top-level Entity Name	lab4
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	101 / 114,480 (< 1 %)
Total registers	55
Total pins	30 / 529 (6 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)

Figure 4: Part 1 FPGA Resource Allocation



Flow Status	Successful - Tue Feb 3 10:47:52 2026
Quartus Prime Version	25.1std.0 Build 1129 10/21/2025 SC Lite Edition
Revision Name	lab4
Top-level Entity Name	lab4
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	883 / 114,480 (< 1 %)
Total registers	77
Total pins	30 / 529 (6 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)

Figure 5: Part 2 FPGA Resource Allocation

The difference in resource usage is primarily due to how synthesis tools interpret structural versus behavioral descriptions. In the structural design, the circuit was built explicitly from interconnected hardware components such as flip-flops, counters, and logic gates. Because the hardware organization was directly specified, the synthesis tool had less ambiguity and could map the design efficiently onto FPGA primitives with minimal additional logic.

In contrast, the behavioral design described the intended functionality using higher-level constructs such as arithmetic operations, division, and conditional statements. While this approach is often easier to write and understand, the synthesis tool must generate the underlying hardware automatically. Operations such as division and binary-to-BCD conversion can require additional combinational logic, registers, and routing resources, which increases overall FPGA utilization.

Therefore, although both designs achieve the same functional behavior, the structural approach resulted in a more resource-efficient implementation, while the behavioral approach offered greater abstraction and design simplicity at the cost of higher hardware usage.

In addition to differences in resource allocation, structural and behavioral design approaches also differ in the ease with which circuits are developed. Structural design is typically more detail-oriented, requiring the designer to explicitly specify individual components and their interconnections. This demands a deeper understanding of the underlying hardware elements and how they operate together within the system.

Behavioral design, in contrast, focuses on describing the overall function of the system rather than its exact hardware structure. Because the synthesis tools automatically determine how to implement the described behavior, this approach generally requires less code and allows designs to be created more quickly. As a result, behavioral modeling is often easier to implement, while structural modeling offers greater control over hardware usage and optimization.

3.1 Part 1 Results

In Phase 1, The precision resistor and capacitor values were measured against the expected values. After the values were measures, the 555 IC timer was implemented as an astable multivibrator on the solderless breadboard. Table 3 shows the precision resistor and capacitor values. Figure 6 shows the 555 IC timer output on an oscilloscope. Table 4 shows the important values from the oscilloscope.

Component	**Expected Value**	**Recorded Value**	**Percent Error**
Precision Resistor	10 kΩ	9999.2 kΩ	0.008%
Capacitor	1 µF	0.996 µF	0.4%

Table 3: Precision Resistor and Capacitor Values

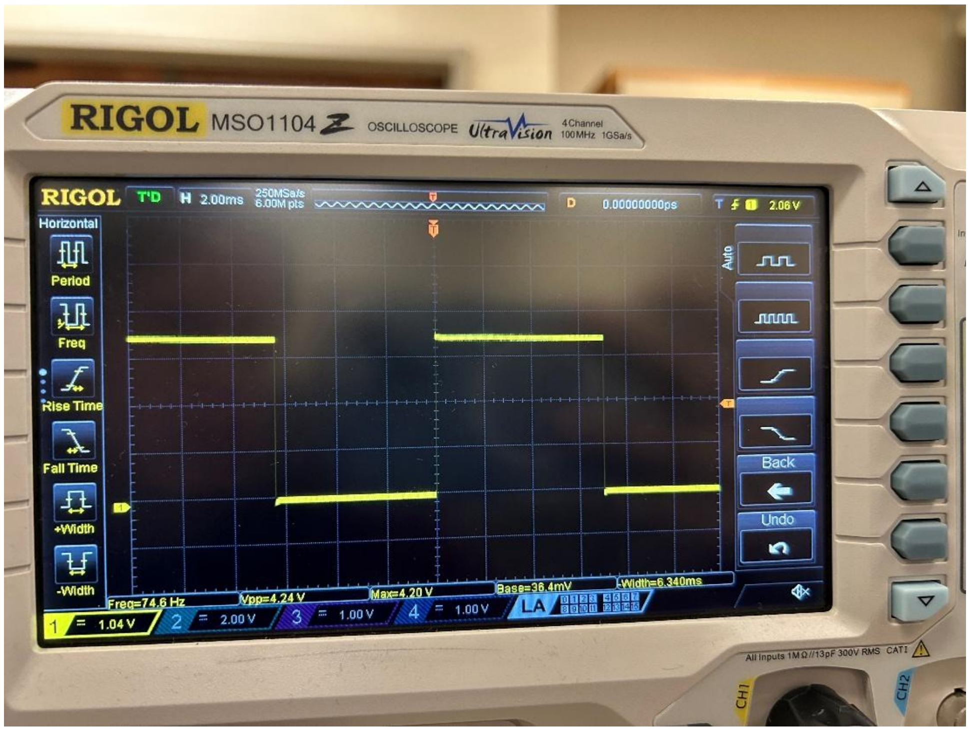


Figure 6: 555 IC Timer Oscilloscope Output

Parameter	Value
Frequency	74.6 Hz
-Width	6.340 ms

Table 4: Important Oscilloscope Values

The actual precision resistor and capacitor values were consistent with the measured values, with percent errors remaining below 0.5%. This agreement indicates that the components used in the circuit were highly accurate and closely matched their nominal specifications, which minimizes discrepancies in theoretical and experimental results. Despite this, the oscilloscope output showed a slight deviation from the predicted behavior. Specifically, the signal spent 6.340 ms at the lowest voltage level, slightly shorter than the expected 6.931. This discrepancy of approximately 0.591 ms, though small, may be attributed to practical factors such as parasitic resistances or capacitances in the circuit, minor tolerances in the measurement equipment, or the finite response time of the oscilloscope. Overall, the close agreement between measured and theoretical values demonstrates the reliability of the components and the accuracy of the theoretical model, while also highlighting real-world imperfections in experimental measurements.

3.2 Part 1 Phase 2 Results

In Phase 2, the FPGA was configured to show the measured capacitance values in hexadecimal on the 7-segment LED display. The output was unstable, however, as the debouncing code was not yet implemented, and appeared to be fluctuating between values rapidly.

3.3 Part 1 Phase 3 Results

In Phase 3, the FPGA was configured to show the measured capacitance values in BCD on the 7-segment LED display. The output was again unstable as the debouncing code was not yet implemented. The output again appeared to be fluctuating rapidly between values.

3.4 Part 1 Phase 4 Results

In Phase 4, the FPGA was configured to show the measured capacitance values in BCD on the 7-segment LED display. The output was stable as the debouncing code was added in this phase. Figures 9, 10, and 11 show the structural design’s output with 1 µF, 2.2 µF, and 33 µF capacitors, respectively.

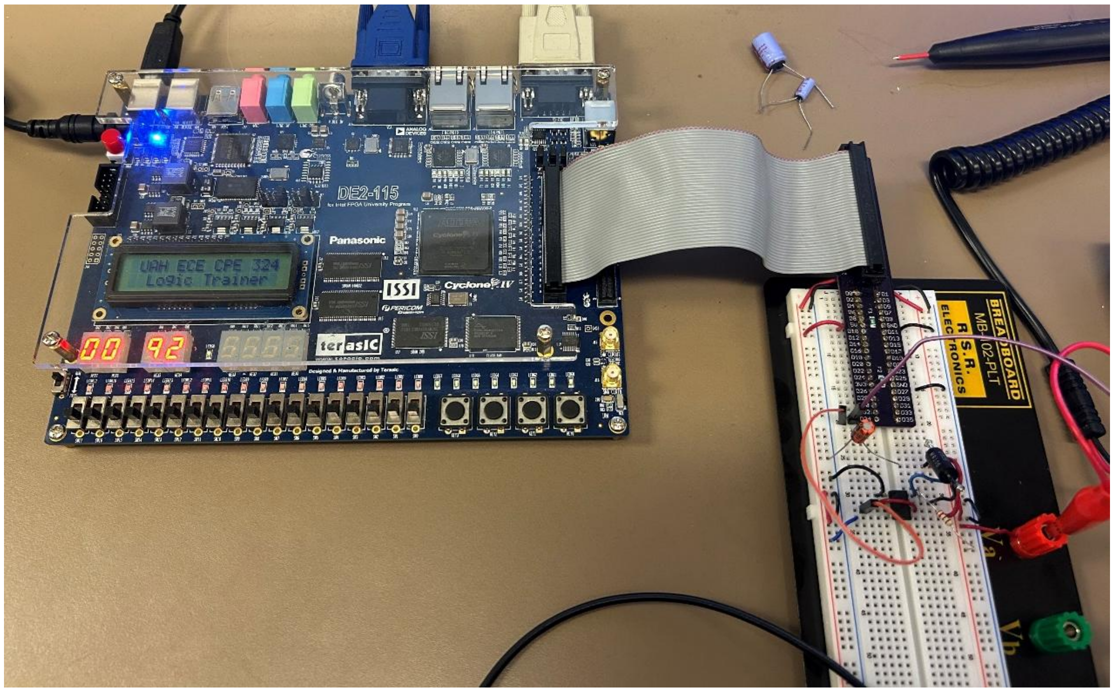


Figure 9: Part 1 Phase 4 Output with 1 μF Capacitor

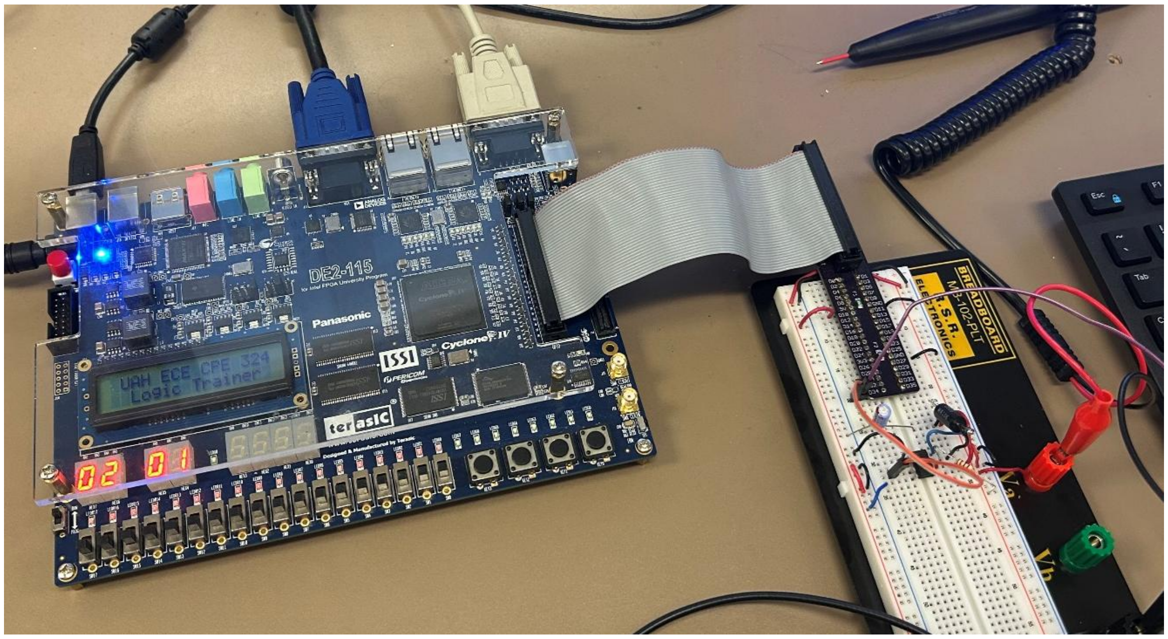


Figure 10: Part 1 Phase 4 Output with 2.2 μF Capacitor

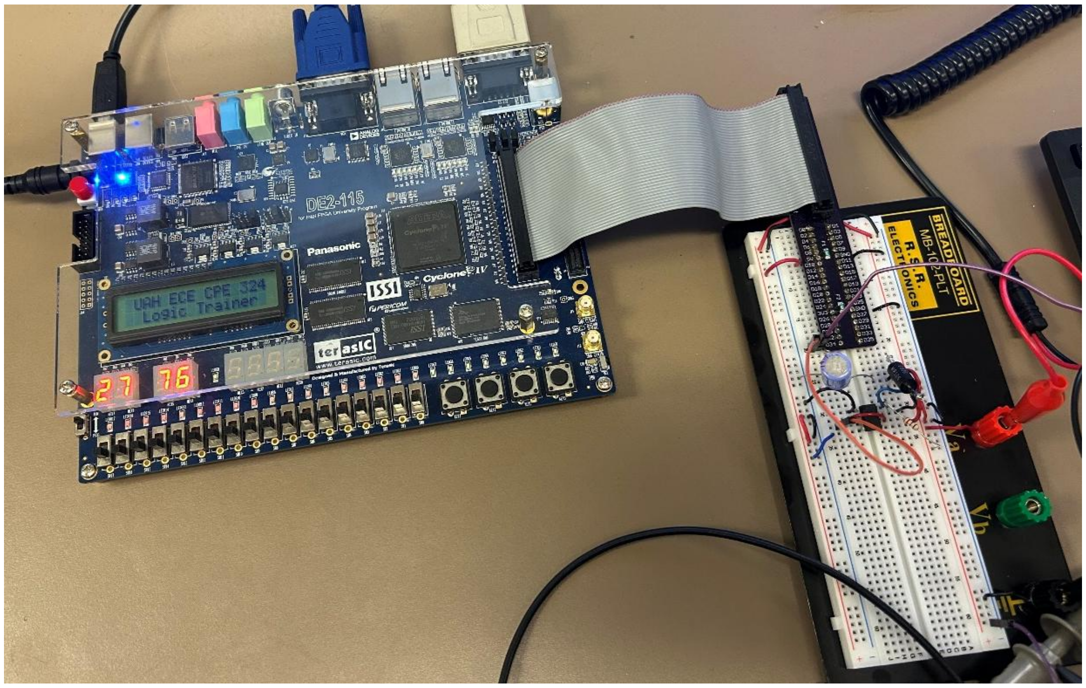


Figure 11: Part 1 Phase 4 Output with 30 µF Capacitor

3.5 Part 2 Results

In part 2, the FPGA was configured to perform the same operations as the design in part 1 using a behavioral model instead of a structural model. BCD encoding and debouncing were incorporated in the design, as well. Figures 12, 13, and 14 show the output of the behavioral design with 1 µF, 2.2 µF, and 33 µF capacitors, respectively.

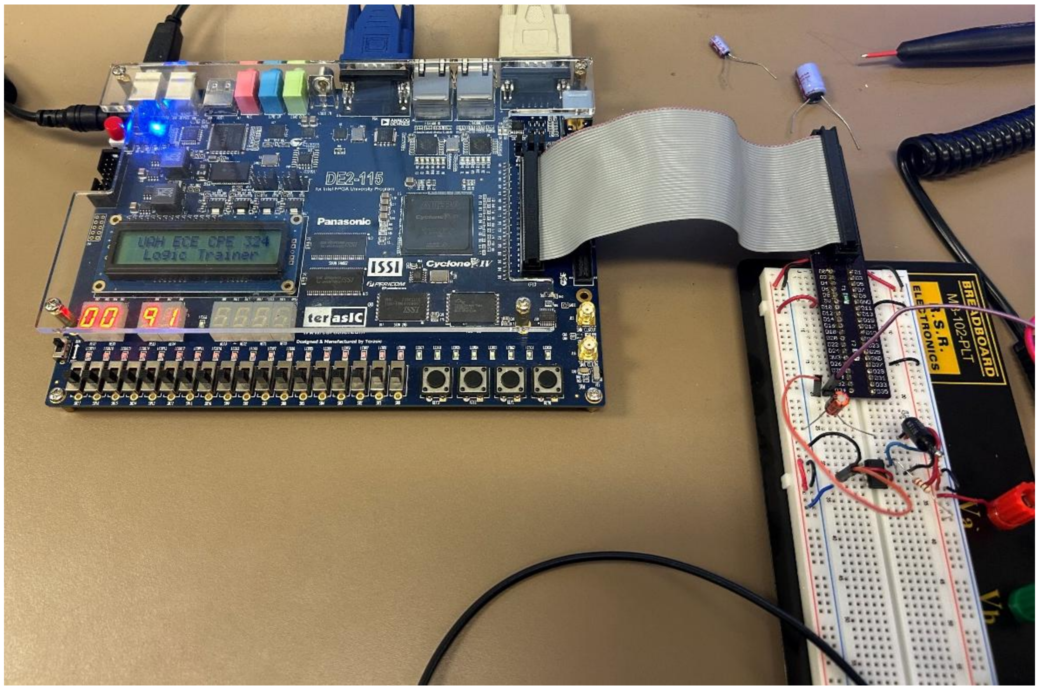


Figure 12: Part 2 Output with 1 µF Capacitor

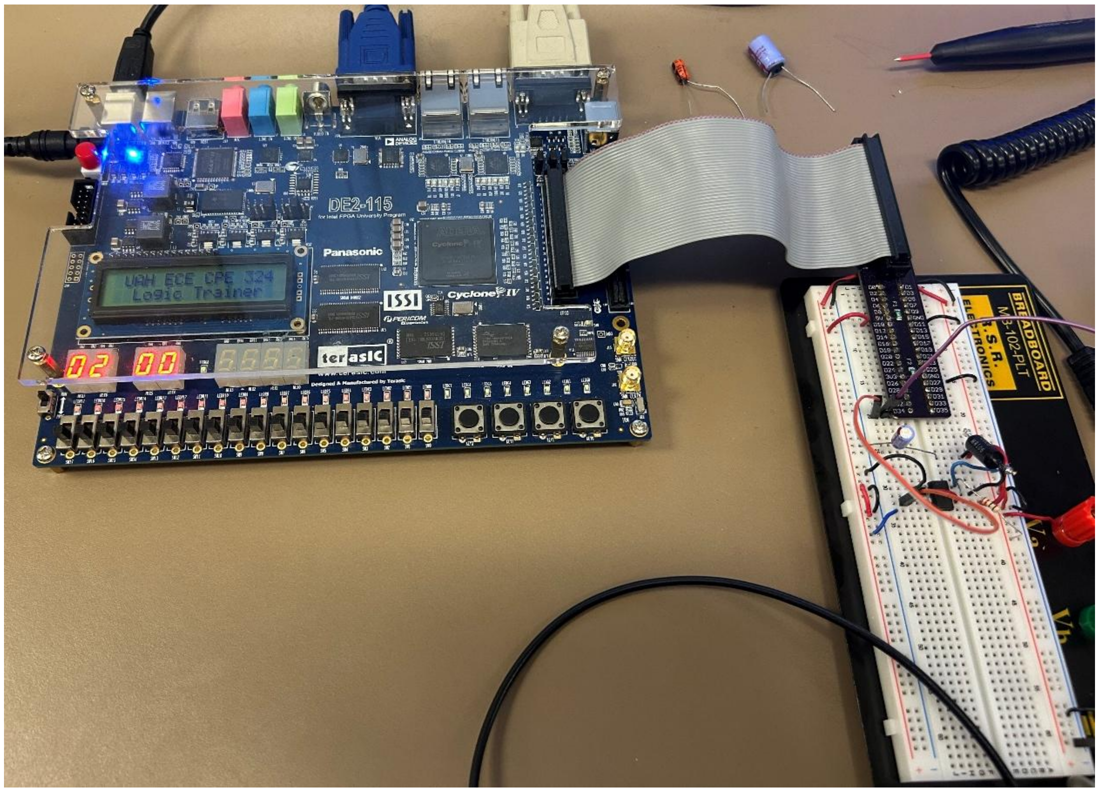


Figure 13: Part 2 Output with 2.2 µF Capacitor

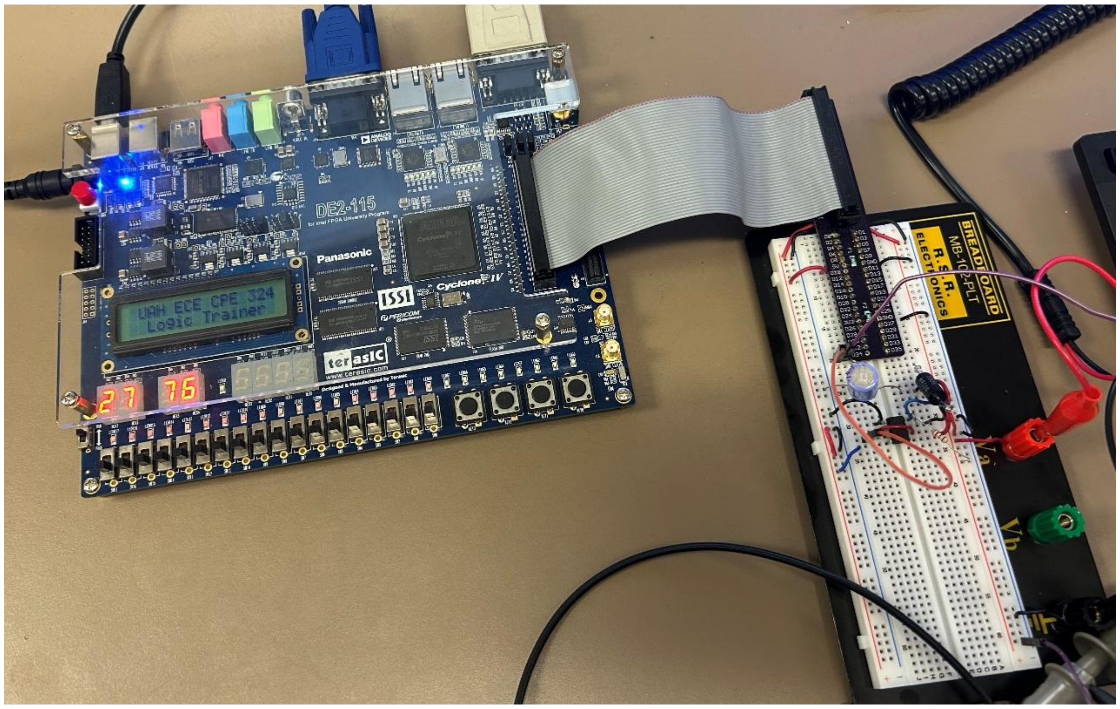


Figure 14: Part 2 Output with 30 µF Capacitor

3.6 Post-Lab Questions

After the lab concluded, the following questions were considered:

Explain the basic operation of the 555 timer when it is placed in multivibrator mode.
In stable multivibrator mode, the 555 IC timer continuously oscillates between high and low output states without an external trigger. The timing of the output waveform in determined by the external resistor and capacitor network connected to the IC. The external capacitor charges through a series combination of resistors until it reaches $\frac{2}{3}V_{CC}$, at which point the timer switches the output to low, and the capacitor discharges through a resistor until it reaches $\frac{1}{3}V_{CC}$, at which point the output switches high again. This cycle repeats indefinitely, generating a square wave with a predictable frequency and duty cycle based on the resistor and capacitor values.

In terms of accuracy why is it not as important that the 10K resistor have a tight resistance tolerance?
The 10K resistor primarily influence the discharge path of the capacitor and has little effect on the timing of the timer IC. Small variations in its resistance cause minor changes in the capacitor's time constant which do not significantly impact the overall clock frequency or output waveform. Because of this, a loose tolerance is acceptable and will not introduce meaningful errors in the circuit or waveform.

How did you modify the parameters to the count_enabler module of the mostly structural design? How did you chose the values to use for these parameters that would generate the desired enable rate?
The count_enabler module typically uses a counter to generate a slower enable pulse from a faster system clock. To modify it, we adjusted the parameter that sets the maximum count value, which determines how many input clock cycles occur before producing a single enable pulse. The value was chosen by calculating the desired enable period and dividing it by the clock period. For example, if the input clock runs at 50 MHz and we wanted an enable pulse every 1 ms, we set the counter to 50,000 cycles. This ensures the output enable signal toggles at the correct rate for the application.

In the structural design shown in Figure 3 (of the lab documentation), what is the purpose of the 1-bit D flip-flop? What happens if it is removed?
The 1-bit D flip-flop serves to synchronize signals and eliminate potential glitches caused by asynchronous transitions. It ensures that the output changes only on the clock edge, providing a stable and predictable timing relationship. If the flip-flop is removed, the circuit may experience erratic pulses or timing errors because combinational logic could respond to changes at unpredictable times, leading to unreliable or inconsistent operation.

What is the function of a one shot in Figure 3 (of the lab documentation)? Why is this important in this design?

The one-shot circuit generates a fixed-duration pulse in response to a triggering event. Its importance in the design is that it produces a clean, single pulse regardless of the input signal duration, preventing multiple triggers from a single event. This ensures that the enable or control signals are precise and predictable, which is crucial in timing-sensitive digital circuits such as counters or sequential logic.

Describe the thought process for your behavioral design. How did this differ from the mostly structural one?
In the behavioral design, the focus was on describing the functionality of the circuit using high-level constructs, such as if-else statements, case statements, and arithmetic operations, rather than explicitly connecting gates and flip-flops. This approach allows the synthesizer to infer the necessary hardware from the description, simplifying design and debugging. In contrast, the mostly structural design required specifying individual modules and their interconnections, closely resembling the actual gate-level hardware. While structural design gives more control over the exact implementation, behavioral design is more concise and flexible.

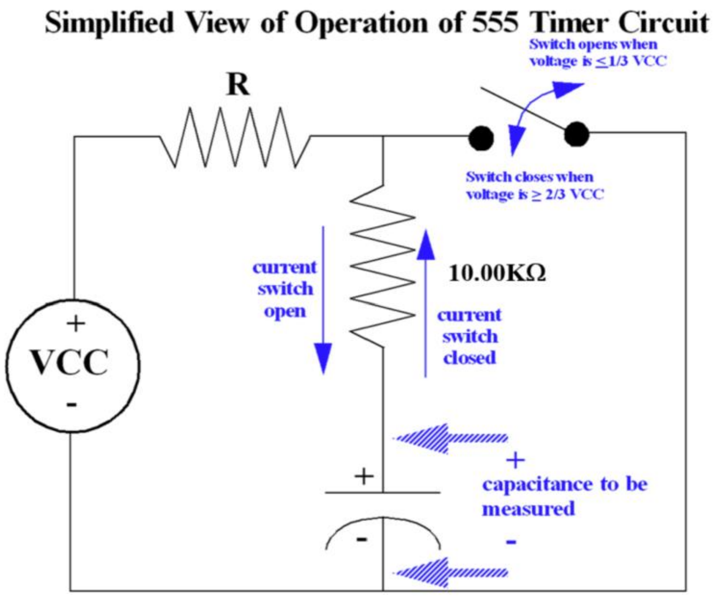
Compare the reported FPGA resource usage of both designs. Which is more efficient in terms of internal logic elements that are utilized? Which was easier to implement? Explain.
The difference in FPGA resource usage arises from how synthesis tools handle structural versus behavioral designs. The structural design explicitly specifies hardware components and their interconnections, allowing the tool to map the circuit efficiently with minimal extra logic, making it more resource efficient. The behavioral design describes functionality at a higher level, requiring the tool to infer underlying hardware, which can increase logic, registers, and routing usage. In terms of implementation, behavioral design is generally easier and faster to write, while structural design provides more control over hardware optimization.

4 Conclusion

In this lab, a combined digital–analog instrument was successfully designed and implemented to measure capacitor values, demonstrating the integration of analog timing circuits with FPGA-based digital logic. Both structural and behavioral Verilog designs were developed and deployed on the DE2-115 board, allowing for a comparison of abstraction levels, implementation ease, and resource utilization. The experimental results closely matched theoretical predictions, with component tolerances and timing deviations remaining minimal, validating the accuracy of the system. The structural design proved more resource efficient, while the behavioral design offered faster development and easier implementation. Overall, the lab reinforced the interplay between analog and digital systems, highlighted the trade-offs between design methodologies, and provided hands-on experience in FPGA development and practical signal measurement techniques.

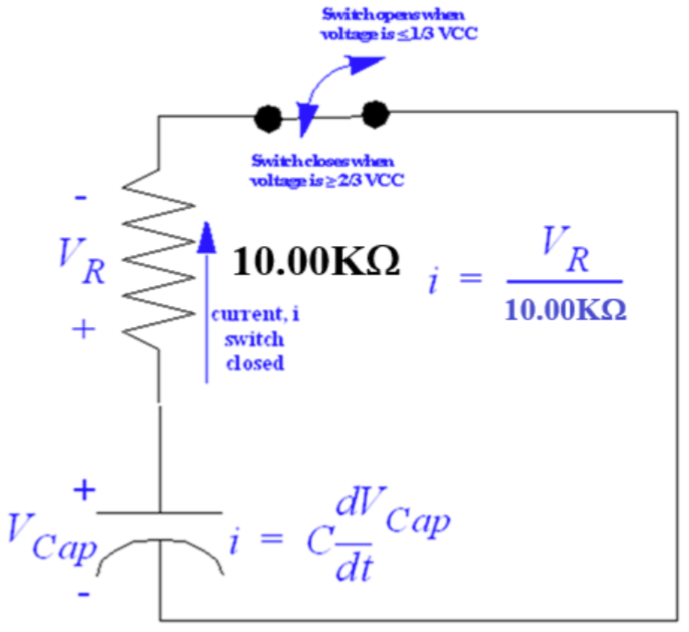
Appendix A – Capacitor Discharge Time Derivation

Consider the circuit shown in Figure 15 and the simplified circuit in Figure 16



Simplified View of Operation of 555 Timer Circuit
Switch opens when voltage is ≤1/3 VCC
R
Switch closes when voltage is ≥ 2/3 VCC
current switch open
10.00KΩ
current switch closed
+
VCC
-
+
capacitance to be measured
-

Figure 15: Simplified View of Operation of 555 Timer Circuit



Switch opens when voltage is ≤1/3 VCC
Switch closes when voltage is ≥2/3 VCC
-
V_R
+
10.00KΩ
current, i switch closed
$i = \frac{V_R}{10.00K\Omega}$
+
V_{Cap}
-
$i = C\frac{dV_{Cap}}{dt}$

Figure 15: Simplified View of Operation of 555 Timer Circuit (Discharge Cycle)

From applying Kirchhoff’s Current Law to the circuit in Figure 15 we get the following:

$$\frac{V_R}{10\ k\Omega} - C\frac{dV_{Cap}}{dt} = 0$$

After applying Kirchhoff's Voltage Law, we know that $V_R = V_{Cap}$. Then the two equations derived from Kirchhoff's Laws can be combined to produce the following:

$$\frac{V_{Cap}}{10\ k\Omega} - C\frac{dV_{Cap}}{dt} = 0$$

Solving for V$_{\text{Cap}}$ yields:

$$V_{Cap} - (10k\Omega)C\frac{dV_{Cap}}{dt} = 0$$

$$V_{Cap} = V_{Cap\ inital} \cdot e^{\frac{-t}{10k\Omega \cdot \mathrm{C}}}$$

From Figure 14 we know that $V_{Cap\ initial} = \frac{2}{3}V_{CC}$ and $V_{Cap\ final} = \frac{1}{3}V_{CC}$. Using these values in the above equation gives:

$$\frac{1}{3}V_{CC} = \frac{2}{3}V_{CC} \cdot e^{\frac{-t}{10k\Omega \cdot \mathrm{C}}}$$

Solving for t gives:

$$\frac{1}{2} = e^{\frac{-t}{10k\Omega \cdot \mathrm{C}}}$$

$$\ln\left(\frac{1}{2}\right) = \left(\ln e^{\frac{-t}{10k\Omega \cdot \mathrm{C}}}\right)$$

$$\ln\left(\frac{1}{2}\right) = e^{\frac{-t}{10k\Omega \cdot \mathrm{C}}}$$

$$t = -\ln\left(\frac{1}{2}\right) \cdot 10k\Omega \cdot \mathrm{C}$$

$$t \approx 0.693 \cdot 10k\Omega \cdot \mathrm{C}$$

$$t \approx 6931 \cdot C$$

This final equation was used in Part 1 phases 1 and 2 to calculate the capacitor discharge time.

Appendix B – Full Verilog Code for Experiment Part 1

```
// CPE 324 Template File for Part 1 of Lab 4
// Mostly Structural Design of Capacitance Tester
// B. Earl Wells
// Note: all modules are in this one file. This file has
//      been named lab4.v which matches the name of the
//      top level module, lab4. The module bcd_counter
//      should be completed as part this lab by the student.
//      The design itself is structural at the top level.
//      The component modules themselves are implemented
//      in various ways (i.e. behavioral, dataflow or structural).
// lab4 (Part 1) top-level module
module lab4(input SIG, CLOCK_50, output [6:0] HEX7, HEX6, HEX5, HEX4);

  wire en_pulse, cnt_clr, ff_en;
  wire SIG_deb;
  wire [15:0] cnt_out, ff_out; // 16-bit counter out and D FF out

  // need to set the two parameters for this module in order
  // for it to generate the correct pulse rate
  count_enabler #(3466,12) C0 (.reset(SIG_deb), .clk(CLOCK_50), .en_out(en_pulse));

  bcd_counter     C1 (.clear(cnt_clr), .en(en_pulse), .clk(CLOCK_50), .o(cnt_out));
  d_ff_16_en      C2 (.d(cnt_out), .en(ff_en), .clk(CLOCK_50), .q(ff_out));
  bcdtohex_driver C3 (.bcd_in(ff_out), .hex4(HEX4), .hex5(HEX5), .hex6(HEX6),
.hex7(HEX7));
  debounce        C4 (.sig_out(SIG_deb), .sig_in(SIG), .clk(CLOCK_50));
  d_ff            C5 (.d(SIG_deb), .clk(CLOCK_50), .q(cnt_clr));
  one_shot        C6 (.tr(SIG_deb), .clk(CLOCK_50), .o(ff_en));

endmodule

// positive pulse active high one-shot element
module one_shot (input tr, clk, output o);
  reg state;
  assign o = ~state & tr;

  always @(posedge clk)
    state = tr;
endmodule

// counter enabling module
module count_enabler(input reset, clk, output reg en_out);

  parameter full_count = 3466, buswidth = 12;
  reg [buswidth-1:0] count;

  always @(posedge clk or posedge reset)
  begin
    if (reset)
    begin
      count  = full_count/2;
      en_out = 1;
    end
    else if (count)
    begin
```

```
      count  = count - 1;
      en_out = 0;
    end
    else
    begin
      count  = full_count;
      en_out = 1;
    end
  end
endmodule

// Advances the 4-digit (16 bit) BCD Counter
module bcd_counter(input clear, en, clk, output reg [15:0] o);

  reg [15:0] count;

  always @(posedge clk or posedge clear)
  begin
    if (clear)
      count = 0;
    else if (en)
    begin
      if (count[3:0] == 4'd9)
      begin
        count[3:0] = 0;
        if (count[7:4] == 4'd9)
        begin
          count[7:4] = 0;
          if (count[11:8] == 4'd9)
          begin
            count[11:8] = 0;
            if (count[15:12] == 4'd9)
              count[15:12] = 0;
            else
              count[15:12] = count[15:12] + 1;
          end
          else
            count[11:8] = count[11:8] + 1;
        end
        else
          count[7:4] = count[7:4] + 1;
      end
      else
        count[3:0] = count[3:0] + 1;
    end
    o = count;
  end
endmodule

// A continuously enabled D Flip-flop
module d_ff(input d, clk, output reg q);
  always @(posedge clk)
    q = d;
endmodule

// A continuously enabled JK Flip-flop
module jk_ff(input j, k, clk, output reg q);
  always @(posedge clk)
    q <= (j & ~q) | (~k & q);
endmodule

// 16-bit D-FF with synchronous enable
module d_ff_16_en(input [15:0] d, input en, clk, output reg [15:0] q);
  always @(posedge clk)
    if (en)
      q = d;
endmodule

// 8-stage debounce/metastability filter
module debounce(output sig_out, input sig_in, clk);

  wire [7:0] q;
  wire ones, zeroes;

  d_ff d0(sig_in, clk, q[0]);
  d_ff d1(q[0],   clk, q[1]);
  d_ff d2(q[1],   clk, q[2]);
  d_ff d3(q[2],   clk, q[3]);
  d_ff d4(q[3],   clk, q[4]);
  d_ff d5(q[4],   clk, q[5]);
  d_ff d6(q[5],   clk, q[6]);
  d_ff d7(q[6],   clk, q[7]);

  assign ones   = &q;
  assign zeroes = ~|q;

  jk_ff out_ff(ones, zeroes, clk, sig_out);

endmodule

// 16-bit BCD to 7-segment driver
module bcdtohex_driver(input [15:0] bcd_in, output [6:0] hex4, hex5, hex6, hex7);
```

```
  bintohex C0(hex4, bcd_in[3:0]);
  bintohex C1(hex5, bcd_in[7:4]);
  bintohex C2(hex6, bcd_in[11:8]);
  bintohex C3(hex7, bcd_in[15:12]);
endmodule

// 4-bit binary to 7-segment hex display
module bintohex (output reg [6:0] hex_out, input [3:0] bin_in);
  always @(*)
  begin
    case (bin_in)
      0  : hex_out = 7'b1000000;
      1  : hex_out = 7'b1111001;
      2  : hex_out = 7'b0100100;
      3  : hex_out = 7'b0110000;
      4  : hex_out = 7'b0011001;
      5  : hex_out = 7'b0010010;
      6  : hex_out = 7'b0000010;
      7  : hex_out = 7'b1111000;
      8  : hex_out = 7'b0000000;
      9  : hex_out = 7'b0011000;
      10 : hex_out = 7'b0001000;
      11 : hex_out = 7'b0000011;
      12 : hex_out = 7'b1000110;
      13 : hex_out = 7'b0100001;
      14 : hex_out = 7'b0000110;
      15 : hex_out = 7'b0001110;
      default: hex_out = 7'bxxxxxxx;
    endcase
  end
endmodule
```

Appendix C – Full Verilog Code for Experiment Part 2

```
// CPE 324 Template File for Part 2 of Lab 4
// Behavioral Model of Capacitance Tester
// B. Earl Wells
// January 29, 2026

module lab4(input SIG, CLOCK_50, output reg [6:0] HEX7, HEX6, HEX5, HEX4);

  reg l_flg;               // latch flag
  reg [23:0] count;        // discharge time count
  reg [15:0] bcdout;       // BCD representation

  reg sig = 1'b1;          // debounced SIG
  reg [7:0] sh_reg = 8'hFF;  // shift register for debounce
```

```
initial begin
  l_flg = 0;
  count = 0;
end

// Debounce filter (requires 8 stable samples)
always @(posedge CLOCK_50) begin
  sh_reg <= {sh_reg[6:0], SIG};

  if (sig == 0)
    sig <= &sh_reg;   // go high only if all 8 samples are 1
  else
    sig <= |sh_reg;   // go low only if all 8 samples are 0
end

// Main behavioral model
always @(posedge CLOCK_50) begin

  // Count 50 MHz clocks while capacitor is discharging (sig = 0)
  if (!sig) begin
    if (count < 24'd34656534)
      count <= count + 1;

    l_flg <= 1;
  end

  // On first clock after sig goes high, latch and convert result
  else if (l_flg) begin
    reg [23:0] cap_units;

    cap_units = count / 3466;   // Convert to 0.01uF units

    if (cap_units > 24'd9999)
      cap_units = 24'd9999;

    bcdout <= bintobcd(cap_units[15:0]);

    bintohex(bcdout[15:12], HEX7);
    bintohex(bcdout[11:8],  HEX6);
    bintohex(bcdout[7:4],   HEX5);
    bintohex(bcdout[3:0],   HEX4);

    count <= 0;
    l_flg <= 0;
  end
```

```
end

// Binary to 7-seg HEX display task
task bintohex(input [3:0] bin_num, output reg [6:0] hex_num);
  begin
    case (bin_num)
      0  : hex_num = 7'b1000000;
      1  : hex_num = 7'b1111001;
      2  : hex_num = 7'b0100100;
      3  : hex_num = 7'b0110000;
      4  : hex_num = 7'b0011001;
      5  : hex_num = 7'b0010010;
      6  : hex_num = 7'b0000010;
      7  : hex_num = 7'b1111000;
      8  : hex_num = 7'b0000000;
      9  : hex_num = 7'b0011000;
      10 : hex_num = 7'b0001000;
      11 : hex_num = 7'b0000011;
      12 : hex_num = 7'b1000110;
      13 : hex_num = 7'b0100001;
      14 : hex_num = 7'b0000110;
      15 : hex_num = 7'b0001110;
      default: hex_num = 7'b1111111;
    endcase
  end
endtask

// Binary to BCD conversion function
function [15:0] bintobcd;
  input [15:0] bin_num;
  reg [15:0] bcd_num;
  reg [15:0] residual;
  integer i;

  begin
    bcd_num = 0;
    residual = bin_num;

    for (i = 0; i < 4; i = i + 1) begin
      bcd_num = bcd_num | ((residual % 10) << (i*4));
      residual = residual / 10;
    end

    bintobcd = bcd_num;
  end
endfunction
```

```
endmodule
```