All In One Counter

Guna Sai Kiran Nekkanti
BT20ECE075 IIIT NAGPUR

Abstract
The project is an All-in-One Counter implemented using structural Verilog coding, incorporating decoders and AND gates for counter access and reset functionalities. Developed as part of an HDL course project at IIIT Nagpur under the guidance of Mr. Vipin Kamble, the counter module integrates various counter modes, including up counters, down counters, Mod10 up counters, Mod5 down counters, ring counters, Johnson counters, odd counters, and even counters. The project demonstrates the versatility and flexibility of the counter module, allowing users to select and reset specific counters based on their requirements. By utilizing structural Verilog coding techniques, the design achieves a modular and organized representation of the counter components, making it readable and maintainable. The implementation of the project utilizes Xilinx FPGA for its rapid prototyping, hardware synthesis, and scalability capabilities, enabling real-time testing and validation of the counter functionalities.

Contents

1 Introduction

Counters are fundamental components in digital circuit design, enabling the counting and tracking of events or signals. The All-in-One Counter project aims to provide a comprehensive solution by integrating various counter modes into a single module. This project was undertaken as part of an HDL course project at IIIT Nagpur, guided by Mr. Vipin Kamble.

The All-in-One Counter module incorporates different counter modes such as up counters, down counters, Mod10 up counters, Mod5 down counters, ring counters, Johnson counters, odd counters, and even counters. These counter modes offer versatile functionality for counting applications, accommodating different counting directions, modulo operations, and sequence patterns.

To facilitate counter access and reset functionalities, the project utilizes decoders and AND gates. The decoders act as combinational logic circuits that translate input signals into unique outputs, allowing the selection of specific counters. The AND gates generate control signals to enable counter reset operations.

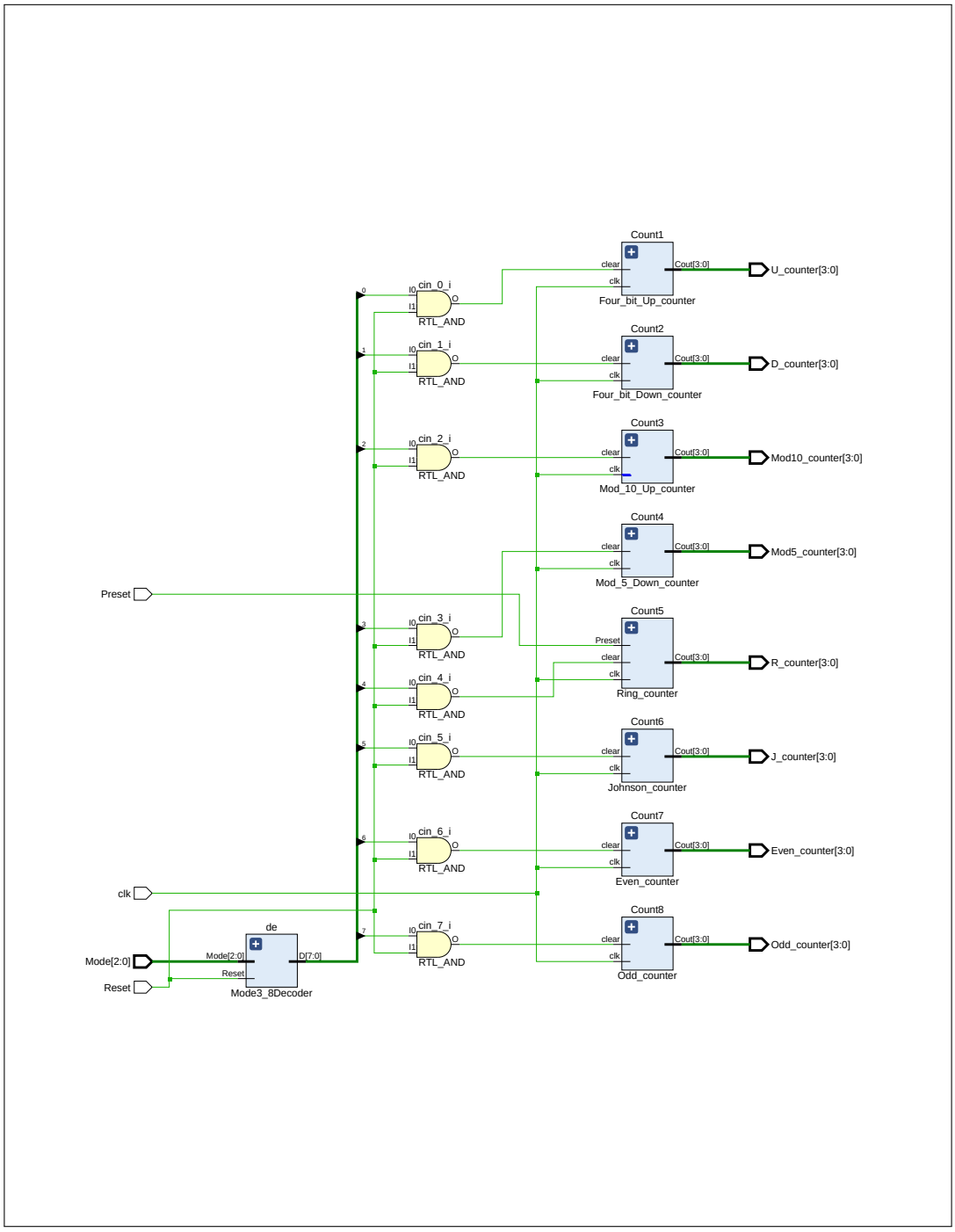
The project adopts structural Verilog coding, which enables the hierarchical composition of modules using gate-level primitives. This coding style results in a modular and organized representation of the counter components, enhancing readability and maintainability. Structural Verilog coding facilitates the seamless integration of decoders, AND gates, and counter modules, forming a coherent and functional All-in-One Counter design.

For implementation, the project utilizes the Xilinx FPGA platform. The choice of Xilinx FPGA offers advantages such as rapid prototyping, hardware synthesis, and scalability. This platform enables efficient synthesis of the All-in-One Counter design onto the FPGA, facilitating real-time testing and validation of the counter functionalities.

By developing the All-in-One Counter, the project provides practical hands-on experience in structural Verilog coding, digital circuit design, and FPGA implementation. The project's structured approach enhances understanding of design principles and promotes good coding practices.

In summary, the All-in-One Counter project serves as an educational tool, demonstrating the integration of multiple counter modes using structural Verilog coding. By incorporating decoders and AND gates, the project enables counter access and reset operations, enhancing the versatility and functionality of the counter module. The project's contributions lie in its practical application of digital circuit design concepts and its potential for further exploration and enhancements in the field of counters and related applications.

2 Circuit Diagram:

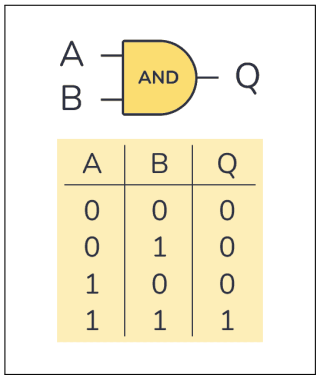


3 Digital Circuits:

In this Section I will explain about the digital circuits which are used in my project

3.1 And Gate

A 2-input AND gate is a basic digital logic gate that takes two binary inputs (A and B) and produces an output (Y) based on the logical AND operation between the inputs. The output of the AND gate is true (logic high) only when both input A and input B are true (logic high).



A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

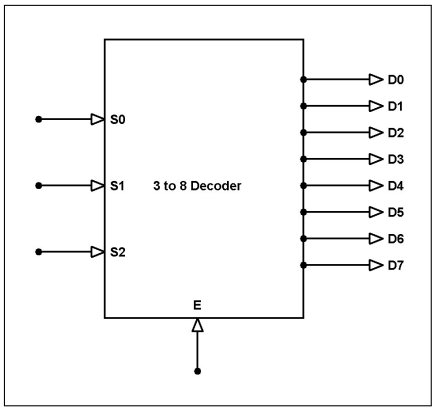
The logical expression for a 2-input AND gate can be written as: Y = A AND B

2-input AND gates are fundamental building blocks in digital logic circuits. They are used extensively in various applications, including arithmetic operations, data manipulation, signal processing, and control systems. They play a crucial role in performing logical operations and combining multiple signals in digital systems.

3.2 3:8 Decoder

A 3:8 decoder is a combinational logic circuit that takes in a 3-bit input and activates one of the eight output lines based on the input value. It essentially decodes a binary input into one of the eight possible output combinations.

The 3:8 decoder consists of three input lines (A, B, C) and eight output lines (Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7). Each output line corresponds to a specific input combination, representing a unique binary value.



The input lines (A, B, C) can be set to either 0 or 1, representing the binary values 0 and 1, respectively. The combination of these three input lines determines which output line will be activated or turned on.

For example, if the input combination is A=0, B=1, C=0, the decoder will activate output line Y2. Each input combination corresponds to a specific output line, following a one-hot encoding scheme.

3:8 decoders are commonly used in digital systems to select one of multiple outputs based on a binary input. They are also used as building blocks for more complex circuits, such as multiplexers and address decoders.

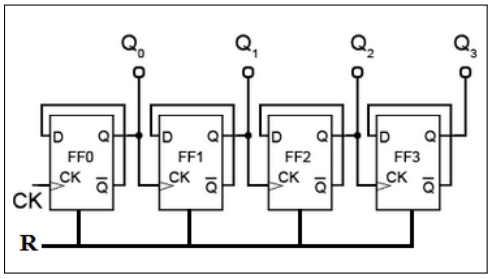
3.3 4-Bit Up Counter

A four-bit up counter is a digital circuit that counts sequentially from 0 to 15 (in binary) in an upward direction. It consists of four flip-flops, each representing a bit position, and combinational logic that generates the input signals to the flip-flops.

The counter starts at 0000 (0 in decimal) and increments by 1 for each clock cycle. As the clock signal pulses, the flip-flops store and propagate the current count value, resulting in a binary sequence from 0000 to 1111 (15 in decimal). Once the counter reaches its maximum value, it rolls over back to 0000 and continues counting.

The combinational logic used in a four-bit up counter is responsible for generating the input signals to the flip-flops. These signals are derived from the clock signal and the outputs of the flip-flops, ensuring the correct sequencing and incrementation of the counter.

To implement a four-bit up counter, you would typically use D flip-flops as the storage elements for each bit. The clock signal serves as the input to the flip-flops, causing them to latch the current value on the rising edge of the clock. The output of each flip-flop is connected to the next flip-flop's input, creating a ripple effect that propagates the count value.



The four-bit up counter is a fundamental building block in digital circuit design and finds applications in various fields, including digital clocks, timers, and control systems. It offers a simple and efficient solution for counting and tracking events in a sequential manner.

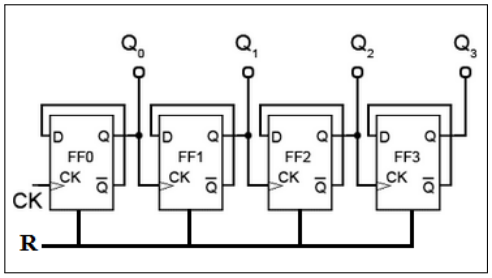
3.4 4-Bit Down Counter

A four-bit down counter is a digital circuit that counts sequentially from 15 to 0 (in binary) in a downward direction. Similar to the four-bit up counter, it consists of four flip-flops and combinational logic that generates the input signals to the flip-flops.

The counter starts at 1111 (15 in decimal) and decrements by 1 for each

clock cycle. As the clock signal pulses, the flip-flops store and propagate the current count value, resulting in a binary sequence from 1111 to 0000 (0 in decimal). Once the counter reaches its minimum value, it rolls over back to 1111 and continues counting in a downward direction.

The combinational logic used in a four-bit down counter is responsible for generating the input signals to the flip-flops, similar to the up counter. These signals are derived from the clock signal and the outputs of the flip-flops, ensuring the correct sequencing and decrementation of the counter.



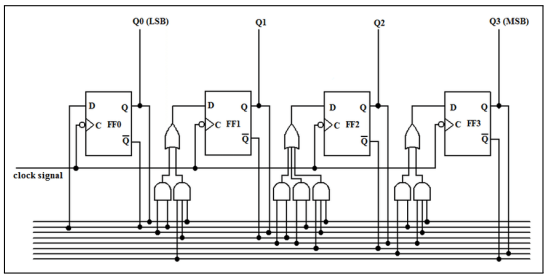
To implement a four-bit down counter, you would typically use D flip-flops as the storage elements for each bit, similar to the up counter. The clock signal serves as the input to the flip-flops, causing them to latch the current value on the rising edge of the clock. The output of each flip-flop is connected to the next flip-flop's input, creating a ripple effect that propagates the count value in the downward direction.

The four-bit down counter is also a fundamental component in digital circuit design and finds applications in various scenarios where counting or decrementing is required, such as countdown timers, data processing, and control systems. It provides a simple and efficient solution for sequential counting in a descending order.

3.5 MOD-10 Counter

A MOD 10 counter is a digital circuit that counts sequentially from 0 to 9 (in decimal) and then repeats the counting cycle. It is also known as a decade counter, as it counts through the ten decimal digits (0-9) before restarting.

A MOD 10 counter typically consists of four flip-flops and combinational logic that generates the input signals to the flip-flops. The flip-flops store the current count value, while the combinational logic determines the next count value based on the current count and input signals.



At the start, the MOD 10 counter is set to 0000 (0 in decimal). On each clock cycle, the counter increments by one, moving through the sequence 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Once the count reaches 9, it wraps back to 0, completing the cycle. This behavior gives the MOD 10 counter its name, as it counts

modulo 10.

The combinational logic in a MOD 10 counter generates the input signals to the flip-flops. These signals are derived from the current count value and the clock signal. The logic determines the next count value based on the current count and the clock signal, ensuring the correct sequencing and reset behavior.

Implementing a MOD 10 counter typically involves using D flip-flops as the storage elements for each bit. The clock signal serves as the input to the flip-flops, causing them to latch the current count value on the rising edge of the clock. The output of each flip-flop is connected to the next flip-flop's input, creating a ripple effect that propagates the count value.

MOD 10 counters are widely used in applications that require a cycle of ten states, such as decimal counters, digital clocks, frequency dividers, and various control systems. They provide a straightforward and efficient solution for counting in a modulo 10 sequence, allowing for repeated and controlled operations.

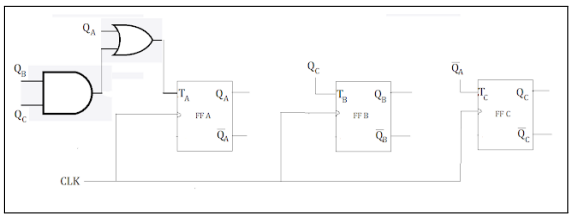
3.6 MOD-5 Counter

A MOD 5 counter is a digital circuit that counts sequentially from 0 to 4 (in decimal) and then repeats the counting cycle. It is a type of counter that follows a modulo-5 counting sequence.

A MOD 5 counter typically consists of three flip-flops and combinational logic that generates the input signals to the flip-flops. The flip-flops store the current count value, while the combinational logic determines the next count value based on the current count and input signals.

At the start, the MOD 5 counter is set to 000 (0 in decimal). On each clock cycle, the counter increments by one, moving through the sequence 0, 1, 2, 3, 4. Once the count reaches 4, it wraps back to 0, completing the cycle. This behavior gives the MOD 5 counter its name, as it counts modulo 5.

The combinational logic in a MOD 5 counter generates the input signals to the flip-flops. These signals are derived from the current count value and the clock signal. The logic determines the next count value based on the current count and the clock signal, ensuring the correct sequencing and reset behavior.



To implement a MOD 5 counter, D flip-flops are commonly used as the storage elements for each bit. The clock signal serves as the input to the flip-flops, causing them to latch the current count value on the rising edge of the clock. The output of each flip-flop is connected to the next flip-flop's input, creating a ripple effect that propagates

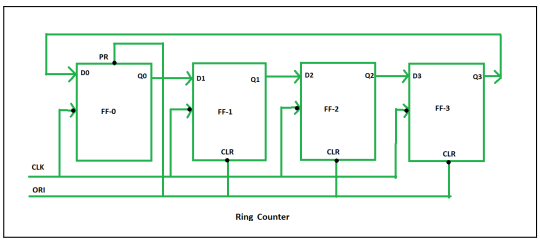
the count value.

MOD 5 counters find applications in various fields, such as frequency division, digital clocks, and control systems, where a cycle of five states is needed. They provide a simple and efficient solution for counting in a modulo 5 sequence, allowing for repeated and controlled operations.

3.7 Ring Counter

A ring counter is a type of digital counter that circulates a single active bit through a series of flip-flops in a cyclic pattern. It forms a closed loop of flip-flops, where the output of each flip-flop is connected to the input of the next flip-flop in the ring.

In a ring counter, only one flip-flop is active at any given time, representing the current position of the active bit. As the clock signal pulses, the active bit moves from one flip-flop to the next in a circular fashion, creating a rotating pattern.



The number of flip-flops in a ring counter determines the number of unique states or positions in the sequence. For example, a three-bit ring counter consists of three flip-flops and has a sequence of eight states (000, 001, 010, 100, 101, 110, 011, 111), as the active bit circulates through the three flip-flops.

The advantage of a ring counter is its simplicity and ability to create a cyclic pattern without requiring complex control logic. It is commonly used in applications that require a rotating sequence or synchronization, such as shift registers, timing circuits, and digital communication systems.

One limitation of a ring counter is the requirement for an external signal or input to initiate the sequence. This input is used to set the initial state of the ring counter and determine the starting position of the active bit.

Overall, the ring counter provides a straightforward and efficient solution for creating a circular or rotating pattern of bits. Its simplicity and cyclic nature make it a valuable component in various digital systems and applications.

3.8 Johnson Counter

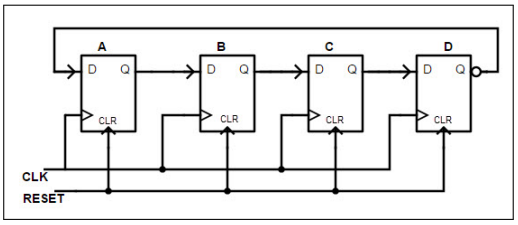
A Johnson counter is a type of digital counter that follows a specific sequence known as the Johnson sequence or the "twisted ring" sequence. It is a shift register-based counter that produces a sequence of all possible bit patterns, with each pattern differing from its adjacent patterns by only one bit.

A Johnson counter typically consists of flip-flops connected in a feed-

back loop, forming a shift register. The output of each flip-flop is connected to the input of the next flip-flop, and the inverted output of the last flip-flop is fed back to the input of the first flip-flop.

The counter starts with an initial state, and with each clock pulse, the bits shift through the flip-flops. The Johnson counter generates a sequence of bit patterns in a cyclic manner, where each pattern is obtained by shifting the previous pattern one bit to the right or left and wrapping around.

For example, a four-bit Johnson counter will produce the following sequence of eight patterns: 0000, 0001, 0011, 0111, 1111, 1110, 1100, 1000. The transition from one pattern to the next occurs by flipping only one bit, resulting in a smooth and systematic progression.



Johnson counters find applications in various fields, including digital communications, coding theory, and digital signal processing. They are particularly useful in applications that require error detection, code generation, and frequency division.

One advantage of Johnson counters is their ability to generate all possible bit patterns, which can be valuable in testing and verification scenarios. However, one limitation is that Johnson counters require additional logic to control the start and stop points of the sequence, as they do not inherently reset to a specific initial state.

In summary, a Johnson counter is a shift register-based counter that produces a cyclic sequence of bit patterns known as the Johnson sequence. It offers systematic and predictable behavior, making it useful in various digital applications that require encoding, decoding, and frequency division.

3.9 4-Bit Even Counter

A 4-bit even counter is a digital circuit that counts sequentially in even numbers from 0 to 14 (in decimal) using a four-bit binary representation. It increments the count value by two on each clock cycle, generating a sequence of even numbers.

The 4-bit even counter consists of four flip-flops, each representing a bit position, and combinational logic that generates the input signals to the flip-flops. The flip-flops store and propagate the current count value, while the combinational logic determines the next count value based on the current count and the clock signal.

At the start, the even counter is set to 0000 (0 in decimal). On each clock cycle, the counter increments by two, moving through the sequence 0, 2, 4,

6, 8, 10, 12, 14. The counter skips odd numbers since it only increments by two on each cycle.

The combinational logic in a 4-bit even counter generates the input signals to the flip-flops. These signals are derived from the current count value and the clock signal. The logic determines the next count value based on the current count and the clock signal, ensuring the correct sequencing and incrementation by two.

To implement a 4-bit even counter, D flip-flops are typically used as the storage elements for each bit. The clock signal serves as the input to the flip-flops, causing them to latch the current count value on the rising edge of the clock. The output of each flip-flop is connected to the next flip-flop's input, creating a ripple effect that propagates the count value.

4-bit even counters are commonly used in applications where counting in even numbers is required, such as in timers, data processing, and control systems.

3.10 4-Bit Odd Counter

A 4-bit odd counter is a digital circuit that counts sequentially in odd numbers from 1 to 15 (in decimal) using a four-bit binary representation. It increments the count value by two on each clock cycle, generating a sequence of odd numbers.

Similar to a 4-bit even counter, the 4-bit odd counter consists of four flip-flops and combinational logic that generates the input signals to the flip-flops. The flip-flops store and propagate the current count value, while the combinational logic determines the next count value based on the current count and the clock signal.

At the start, the odd counter is set to 0001 (1 in decimal). On each clock cycle, the counter increments by two, moving through the sequence 1, 3, 5, 7, 9, 11, 13, 15. The counter skips even numbers since it only increments by two on each cycle.

The combinational logic in a 4-bit odd counter generates the input signals to the flip-flops. These signals are derived from the current count value and the clock signal. The logic determines the next count value based on the current count and the clock signal, ensuring the correct sequencing and incrementation by two.

To implement a 4-bit odd counter, D flip-flops are typically used as the storage elements for each bit. The clock signal serves as the input to the flip-flops, causing them to latch the current count value on the rising edge of the clock. The output of each flip-flop is connected to the next flip-flop's input, creating a ripple effect that propagates the count value.

4 Working

In this Section I will explain Working of my project

I have designed an all-in-one counter using RTL coding in Verilog, utilizing a 3:8 decoder, AND gates, and multiple counters.

Verilog HDL is a hardware description language used for designing digital systems. It allows hierarchical and behavioral modeling, enabling the description of circuit functionality and interconnections. Verilog supports simulation, synthesis, testbenches, and verification techniques, making it essential for digital design.

Here's a step-by-step explanation of how my project works:

- **Input Signals:**We have three input signals (A, B, and C) that act as control signals for the counters. These signals are used to select which counter(s) to operate and reset.
- **3:8 Decoder:** The three input signals (A, B, and C) are connected to a 3:8 decoder. The decoder decodes the input signals into eight output lines, with each line corresponding to a specific counter.
- **AND Gates:** Each output line of the 3:8 decoder is connected to an AND gate, along with a clear input signal.
- **Counters:**I have multiple counters in My project, and their outputs are connected to the desired output display or further processing logic. Each counter keeps track of a specific value or counts according to the given sequence.
- **Operation:** When you want to increment a particular counter, you set the corresponding input signals (A, B, and C) to the appropriate values that activate the corresponding output line of the 3:8 decoder.
- **Reset Functionality:**To reset a specific counter, you control the clear input signal using the AND gate. The AND gate receives the output from the corresponding output line of the 3:8 decoder and a clear input signal. If the output of the AND gate is low (0), it indicates that the conditions for resetting the counter have been met. In this case, the clear input signal activates, resetting the counter to its initial value (e.g., 0).

By selecting the appropriate input combination, I activate the corresponding counter, and its output is displayed on the connected display device. The counters increment or decrement according to their defined behavior, allowing you to visualize and track various counting sequences on the displays.

5 Verilog Codes

In this Section I will Provide My project Verilog Codes

5.1 Main Code:

```
`timescale 1ns / 1ps
module All_in_one(
    input [2:0] Mode,
    input Reset,
    input Preset,
    input clk,
    output [3:0] U_counter,
    output [3:0] D_counter,
    output [3:0] Mod10_counter,
    output [3:0] Mod5_counter,
    output [3:0] R_counter,
    output [3:0] J_counter,
    output [3:0] Even_counter,
    output [3:0] Odd_counter
    );
    wire[7:0]D;
    wire[7:0]cin;
    Mode3_8Decoder de(Mode[2:0],Reset,D[7:0]);
    and(cin[0],D[0],Reset);
    and(cin[2],D[2],Reset);
    and(cin[3],D[3],Reset);
    and(cin[4],D[4],Reset);
    and(cin[5],D[5],Reset);
    and(cin[6],D[6],Reset);
    and(cin[7],D[7],Reset);
    and(cin[1],D[1],Reset);
    Four_bit_Up_counter Count1(cin[0],clk,U_counter[3:0]);
    Four_bit_Down_counter Count2(cin[1],clk,D_counter[3:0])
   ;
    Mod_10_Up_counter Count3(cin[2],clk,Mod10_counter[3:0])
   ;
```

```
    Mod_5_Down_counter Count4(cin[3],clk,Mod5_counter[3:0])
   ;
    Ring_counter Count5(cin[4],clk,Preset,R_counter[3:0]);
    Johnson_counter Count6(cin[5],clk,J_counter[3:0]);
    Even_counter Count7(cin[6],clk,Even_counter[3:0]);
    Odd_counter Count8(cin[7],clk,Odd_counter[3:0]);
endmodule
```

5.2 3:8 Decoder

```
`timescale 1ns / 1ps
module Mode3_8Decoder(
    input [2:0] Mode,
    input Reset,
    output reg [7:0] D
    );
    always @(*)
    begin
      if(Reset==1)
      begin
      D=8'd0;
       case (Mode)
       3'b000: D=8'd1;
       3'b001: D=8'd2;
       3'b010: D=8'd4;
       3'b011: D=8'd8;
       3'b100: D=8'd16;
       3'b101: D=8'd32;
       3'b110: D=8'd64;
       3'b111: D=8'd128;
       default:D=8'd0;
       endcase
      end
      else
      D=8'd0;
    end
endmodule
```

5.3 4-Bit-Up Counter

This counter increments its value by one on each clock cycle, starting from 0000 and counting up to 1111. The output of this counter is connected to a display to show the current count value.

```
`timescale 1ns / 1ps
module Four_bit_Up_counter(
    input clear,
    input clk,
    output reg [3:0] Cout
    );
     initial
      Cout=0;
     always @ (posedge clk)
     begin
    if (clear==0)
      Cout <= 0;
    else
      Cout <= Cout + 1;
  end
endmodule
```

5.4 4-Bit-Down Counter

This counter decrements its value by one on each clock cycle, starting from 1111 and counting down to 0000. The output of this counter is connected to a display to show the current count value.

```
`timescale 1ns / 1ps
module Four_bit_Down_counter(
    input clear,
    input clk,
    output reg [3:0] Cout
    );
     initial
     Cout=4'b1111;

     always @ (posedge clk)
     begin
    if (clear==0)
      Cout <= 0;
```

```
    else
      Cout <= Cout - 1;
  end
endmodule
```

5.5 Mod 10 Counter

This counter counts from 0 to 9 in a cyclic manner. It increments its value by one on each clock cycle, and when the count reaches 9, it wraps around to 0. The output of this counter is connected to a display to show the current count value.

```
`timescale 1ns / 1ps
module Mod_10_Up_counter(
    input clear,
    input clk,
    output reg [3:0] Cout
    );
    initial
     Cout=4'b0000;
    always@ (posedge clk)
begin
if( clear==0 | Cout==4'b1001)
Cout <= 4'b0000;
else
Cout <= Cout+ 1;
end
endmodule
```

5.6 Mod 5 Counter

This counter counts from 0 to 4 in a cyclic manner. It increments its value by one on each clock cycle, and when the count reaches 4, it wraps around to 0. The output of this counter is connected to a display to show the current count value.

```
`timescale 1ns / 1ps
module Mod_5_Down_counter(
    input clear,
    input clk,
    output reg [3:0] Cout
    );
    initial
```

```
    Cout=4'b0101;
    always@ (posedge clk)
    begin
      if( clear==0)
        Cout <= 4'b0000;
      else
        begin
        if(Cout<=5&Cout>0)
            Cout<=Cout-1;
        else
            Cout<=4'b0101;
        end
  end
endmodule
```

5.7 Ring Counter

A ring counter is a shift register arrangement where the output of the last flip-flop is connected to the input of the first flip-flop. It generates a cyclic sequence of states, with only one flip-flop active at a time. The output of this counter is connected to a display to indicate the active flip-flop.

```
`timescale 1ns / 1ps
module Ring_counter(
    input clear,
    input clk,
    input Preset,
    output reg [3:0] Cout
    );
    integer i;
      always @ (posedge clk) begin
      if(clear==0)
          Cout <=0;
      else
      begin
      if (Preset)
         Cout <= 1;
      else begin
        Cout[3] <= Cout[0];
        for (i = 0; i < 3; i=i+1)
         begin
```

```
            Cout[i] <= Cout[i+1];
          end
        end
        end
  end
endmodule
```

5.8 Johnson Counter

A Johnson counter is a shift register arrangement that circulates a single active bit through its stages. It generates a sequence of states where only one bit is active at any given time, and it circulates through the stages. The output of this counter is connected to a display to indicate the active bit position.

```
`timescale 1ns / 1ps
module Johnson_counter(
    input clear,
    input clk,
    output reg [3:0] Cout
    );
    integer i;
    initial
     Cout=4'b0000;
    always @ (posedge clk)
    begin
      if (clear==0)
         Cout <= 0;
      else begin
        Cout[3] <= ~Cout[0];
        for (i = 0; i < 3; i=i+1) begin
          Cout[i] <= Cout[i+1];
        end
      end
  end
endmodule
```

5.9 4-Bit Odd Counter

This counter counts in odd numbers, incrementing by two on each clock cycle. It starts from 0001 and counts up to 1111. The output of this counter is connected to a display to show the current count value.

```
`timescale 1ns / 1ps
module Odd_counter(
    input clear,
    input clk,
    output reg [3:0] Cout
    );
    initial
     Cout=4'b0001;
    always@(posedge clk)

    begin
         if(clear==0)

            Cout<=1; // 1 because this is an odd counter
         else
            begin
              if(Cout % 2 == 0) // checking if the counter
   started at an even state
                     Cout<=1;
              else
                     begin
                     if(Cout == 15) // max count check
                         Cout<=1;
                     else
                         Cout<=Cout+2; // odd counting
   1,3,5,7,9
                     end
              end
     end
endmodule
```

5.10 4-Bit Even Counter

This counter counts in even numbers, incrementing by two on each clock cycle. It starts from 0000 and counts up to 1110. The output of this counter is connected to a display to show the current count value.

```
`timescale 1ns / 1ps
module Even_counter(
    input clear,
    input clk,
    output reg [3:0] Cout
    );
    initial
     Cout=4'b0000;
    always@(posedge clk)

    begin
         if(clear==0)

            Cout<=0; // 1 because this is an odd counter
         else
            begin
              if(Cout % 2 != 0) // checking if the counter
   started at an even state
                    Cout<=0;
              else
                    begin
                    if(Cout == 14) // max count check
                        Cout<=0;
                    else
                        Cout<=Cout+2; // odd counting
   1,3,5,7,9
                    end
              end
     end
endmodule
```

5.11 Test bench

Verilog HDL allows us to create testbenches, which are modules specifically designed to provide stimulus and monitor the behavior of the design under test (DUT). Testbenches simulate the environment in which the design will operate, allowing you to validate its functionality and performance.

```
`timescale 1ns / 1ps
module testbench;
    reg[2:0] Mode;
    reg Reset;
    reg Preset;
    reg clk;
    wire [3:0] U_counter; wire [3:0] D_counter;
    wire [3:0] Mod10_counter; wire [3:0] Mod5_counter;
    wire [3:0] R_counter; wire [3:0] J_counter;
    wire [3:0] Even_counter; wire [3:0] Odd_counter;
    All_in_one Dut(Mode,Reset,Preset,clk,U_counter,D_counter
    ,Mod10_counter,Mod5_counter,R_counter,J_counter,
    Even_counter,Odd_counter);
    integer i,j;
    parameter WAIT=290;
    initial
    begin
    clk=0;
    Reset=1;
    Preset=1;
    for(j=0;j<3;j=j+1)
    begin
          for(i=0;i<8;i=i+1)
              begin
                  Mode=i;
                  #WAIT;
              end
    end
    $finish;
    end
    always #10 clk=~clk;
    always #1175 Preset=~Preset;
    always #2320 Reset=~Reset;
endmodule
```

Test Bench Explanation:

1. Module Declaration: - The code begins with the declaration of a module named "testbench."

2. Signal Declarations:

- `reg [2:0] Mode;` declares a 3-bit register `Mode` used to select different counter modes.
- `reg Reset;` declares a register `Reset` used to reset the counters.
- `reg Preset;` declares a register `Preset` used for presetting the counters.
- `reg clk;` declares a register `clk` representing the clock signal.
- `wire [3:0] U_counter;` declares a 4-bit wire `U_counter` for the up counter output.
- `wire [3:0] D_counter;` declares a 4-bit wire `D_counter` for the down counter output.
- `wire [3:0] Mod10_counter;` declares a 4-bit wire `Mod10_counter` for the mod 10 counter output.
- `wire [3:0] Mod5_counter;` declares a 4-bit wire `Mod5_counter` for the mod 5 counter output.
- `wire [3:0] R_counter;` declares a 4-bit wire `R_counter` for the ring counter output.
- `wire [3:0] J_counter;` declares a 4-bit wire `J_counter` for the Johnson counter output.
- `wire [3:0] Even_counter;` declares a 4-bit wire `Even_counter` for the even counter output.
- `wire [3:0] Odd_counter;` declares a 4-bit wire `Odd_counter` for the odd counter output.

3. Module Instantiation:

- `All_in_one Dut(...);` instantiates the "All_in_one" module and connects its inputs and outputs to the corresponding signals declared above.

4. Integer Variables and Parameters:

- `integer i, j;` declares two integer variables used for looping in the initial block.
- `parameter WAIT=290;` defines a parameter `WAIT` with a value of 290. This parameter is used for controlling the delay in the simulation.

5. Initial Block:

```
– The initial block begins with ‘initial begin‘ and
contains the initial simulation code.
– ‘clk=0;‘ initializes the ‘clk‘ signal to 0.
– ‘Reset=1;‘ initializes the ‘Reset‘ signal to 1,
indicating a reset state.
– ‘Preset=1;‘ initializes the ‘Preset‘ signal to 1,
indicating a preset state.
– The nested for-loops iterate through different modes (
i) and multiple cycles (j) to test various counter modes
.
– ‘Mode=i;‘ assigns the current value of ‘i‘ to the ‘
Mode‘ signal, representing the selected counter mode.
– ‘#WAIT;‘ delays the simulation for the specified time,
 defined by the ‘WAIT‘ parameter.
– ‘$finish;‘ terminates the simulation after all modes
have been tested.
```

6. Always Blocks:

```
– ‘always #10 clk=~clk;‘ toggles the ‘clk‘ signal every
10 time units, creating a clock signal with a period of
20 time units.
– ‘always #1175 Preset=~Preset;‘ toggles the ‘Preset‘
signal every 1175 time units, simulating a preset
operation for the ring counter..
– ‘always #2320 Reset=~Reset;‘ toggles the ‘Reset‘
signal every 2320 time units, simulating a reset
operation for all counters.
```

Explanation: By utilizing the iterator variables (i and j) and parameters (WAIT), My testbench iterates through various counter modes, waits for a specified time, and controls the clock signal, preset signal for the ring counter, and reset signal for all counters. This allows me to observe the behavior of the counters and verify their functionality in the simulation.

6 Output Waveforms:

```
Mode 0 –4–Bit–Up Counter
Mode 1 –4–Bit–Down Counter
Mode 2 –Mod 10 Counter
Mode 3 –Mod 5 Counter
Mode 4 –Ring Counter
Mode 5 –Johnson Counter
Mode 6 –4–Bit–Even Counter
Mode 7 –4–Bit–Odd Counter
```

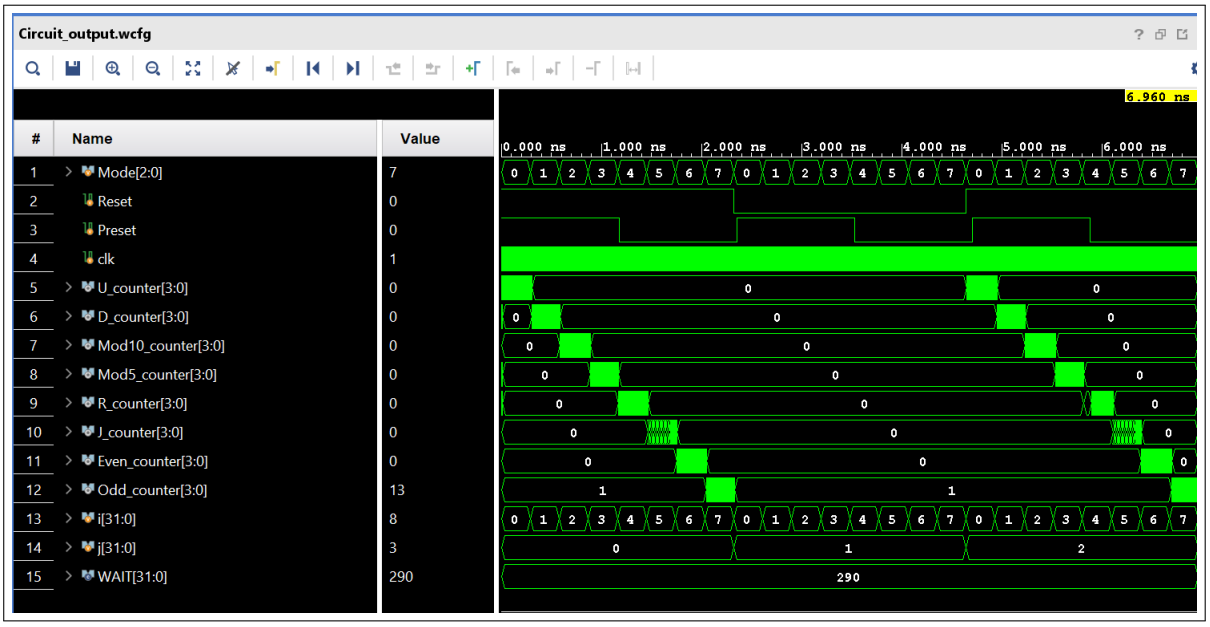


Figure 1: Output

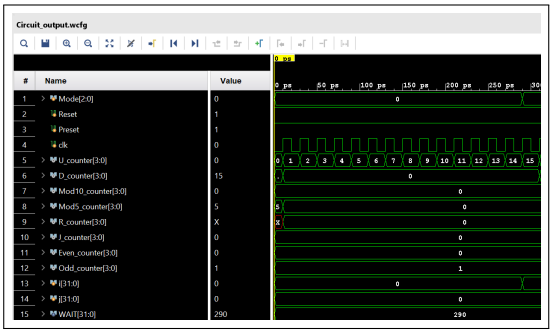


Figure 2: 4-Bit-Up Counter

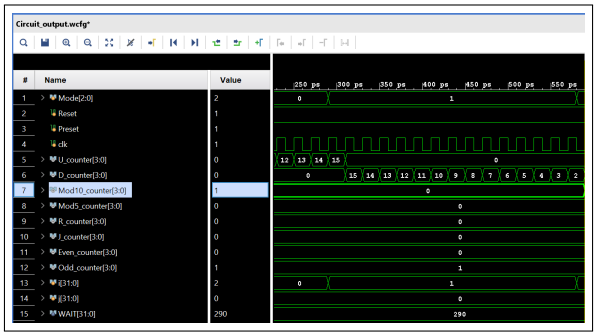


Figure 3: 4-Bit-Down Counter

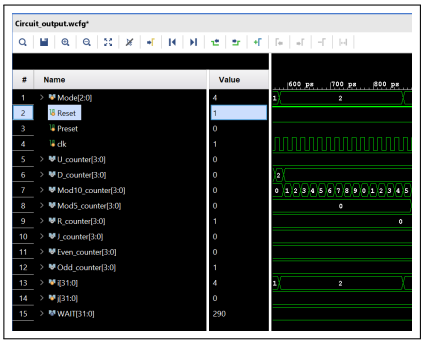


Figure 4: Mod 10 Counter

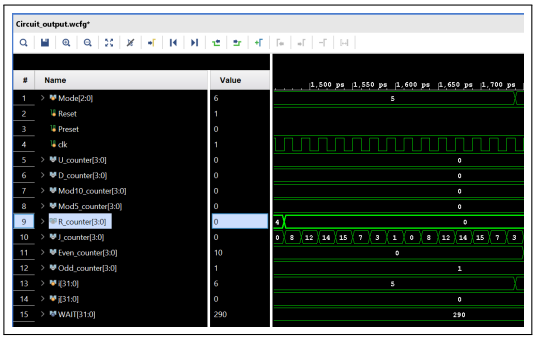


Figure 7: Johnson Counter

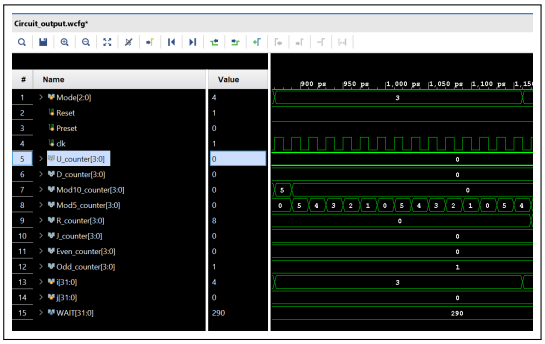


Figure 5: Mod 5 Counter

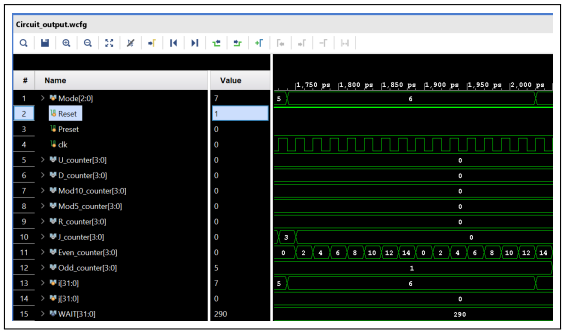


Figure 8: 4-Bit-Even Counter

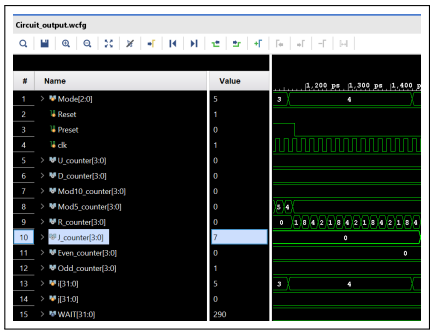


Figure 6: Ring Counter

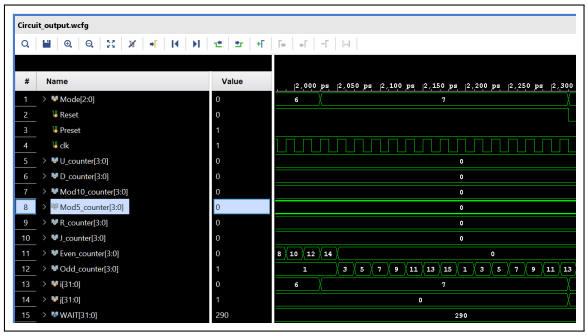


Figure 9: 4-Bit-Odd Counter

7 Conclusions

In conclusion, the project aimed to implement an "All-in-One" counter using Verilog HDL. The project involved designing and testing various counters, including a 4-bit up counter, 4-bit down counter, mod 10 counter, mod 5 counter, ring counter, Johnson counter, 4-bit even counter, and 4-bit odd counter.

The implemented design utilized a 3:8 decoder to access the different counters based on the input mode. An AND gate with a clear input was used to reset the counters when the output of the AND gate was 0.

The provided testbench code simulated the behavior of the counters by generating clock signals, setting/resetting the preset signal for the ring counter, and controlling the mode selection. The simulation verified the functionality of the counters and their expected behavior in different modes.

Overall, the project successfully demonstrated the implementation of various counters using structural Verilog coding and validated their operation through simulation.

8 Refrences

1. "Digital Design and Computer Architecture" by David Harris and Sarah Harris: This book provides a comprehensive introduction to digital design principles and includes a section on Verilog HDL.
2. "Verilog HDL: A Guide to Digital Design and Synthesis" by Samir Palnitkar: This book offers a detailed explanation of Verilog HDL and covers both the basics and advanced topics of digital design using Verilog.
3. "RTL Hardware Design Using VHDL: Coding for Efficiency, Portability, and Scalability" by Pong P. Chu: Although this book focuses on VHDL, it provides valuable insights into hardware design concepts and coding techniques that are applicable to Verilog HDL as well.
4. Xilinx Documentation: Xilinx, the provider of the FPGA platform you mentioned, offers extensive documentation and application notes on Verilog HDL and their design tools. You can refer to their official website for resources specific to your Xilinx FPGA.
5. Online Tutorials and Websites: There are various online tutorials and websites that provide Verilog HDL tutorials, examples, and reference materials. Some popular websites include FPGA4fun (www.fpga4fun.com), ASIC World (www.asic-world.com), and VerilogPro (www.verilogpro.com).