Project Report Outline: Hardware Design Contest

Introduction

This project is part of the "Hardware Design Contest," where the objective is to design a hardware system that will execute on a Cyclone V FPGA. Following the given requirements, an initial C program was converted into Verilog to achieve a design that runs on the FPGA.

Project Requirements:

- Write a C Program:
  - array of 256 integers.
  - At least two loops.
  - At least two if-else conditionals.
  - Arithmetic expressions.

- Hardware Conversion:
  - Block Memory Usage: Use block memory for the array and results.
  - Verilog Implementation:
    Control Unit: 3 always blocks (for state register, next state, and output logic).
    Datapath: Optimized for both delay and area.

- Simulation:
  Perform simulation using a Verilog testbench.

- FPGA Upload:
  Use the FPGA board's features (at least 8 switches, 4 7-segment displays, 10 LEDs, and UART) to demonstrate the design.

C Code Explanation

The following C code was used in the project:

```
#define SIZE 256

int main() {
  int array[SIZE];
  int sum_even = 0;
  int sum_odd = 0;
  int count_odd = 0;
  // Initialize array with random values
  srand(time(NULL));
  for (int i = 0; i < SIZE; i++) {
    array[i] = i + 1;
  }
  // Compute sum of even numbers and average of odd numbers
  for (int i = 0; i < SIZE; i++) {
    if (array[i] % 2 == 0) {
      sum_even += array[i];
    } else {
      sum_odd += array[i];
      count_odd++;
    }
  }
  // Second if-else to ensure two separate conditional structures
  double avg_odd;
  if (count_odd > 0) {
    avg_odd = (double)sum_odd / count_odd;
  } else {
    avg_odd = 0;
  }
  printf("Sum of even numbers: %d\n", sum_even);
  printf("Average of odd numbers: %.2f\n", avg_odd);
  return 0;
}
```

Explanation:

- Array Initialization: A 256-element array is initialized with values.
- Arithmetic Operations: Calculates the sum of even numbers (sum_even) and the average of odd numbers (avg_odd).

Verilog Hardware Design

This C program was converted into the following Verilog modules:

1. Control Unit:
   - State Machine: Contains three states: IDLE, COMPUTE, and DONE.
   - Output Logic: Generates output signals based on the current state.
   - Next State Logic: Controls the state transitions.

2. Datapath:
   - Array Initialization: The array is initialized with simple values for simulation purposes.
   - Arithmetic Operations: Calculates the sum of even numbers and the average of odd numbers.

3. Top Module:
   - Hierarchical Structure: Combines the control unit and datapath modules.

Project Requirements Compliance

1. Block Memory Usage:
   - array in the datapath module is implemented as a 256-element memory.

```
reg [7:0] array [0:255];
```

2. Datapath:
   - The datapath module is optimized for both delay and area.

```
always @(posedge clk or posedge reset) begin
    if (compute) begin
        if (array[data_in] % 2 == 0) begin
            sum_even <= sum_even + array[data_in];
        end else begin
            sum_odd <= sum_odd + array[data_in];
            count_odd <= count_odd + 1;
        end
    end
end
```

3. Control Unit:
   - The control_unit module contains 3 always blocks for state register, next state logic, and output logic.

```verilog
always @(posedge clk or posedge reset) begin
    if (reset)
        state <= IDLE;
    else
        state <= next_state;
end
```

Next State Logic:

```verilog
always @(*) begin
    case (state)
        IDLE: next_state = COMPUTE;
        COMPUTE: if (index == 8'd255) next_state = DONE; else next_state = COMPUTE;
        DONE: next_state = IDLE;
        default: next_state = IDLE;
    endcase
end
```

Output Logic:

```verilog
always @(posedge clk or posedge reset) begin
    if (reset) begin
        data_in <= 0;
        index <= 0;
        compute <= 0;
        done <= 0;
    end else begin
        case (state)
            IDLE: begin
                index <= 0;
                data_in <= 0;
                compute <= 1;
                done <= 0;
            end
            COMPUTE: begin
                data_in <= index;
                index <= index + 1;
                compute <= 1;
                done <= 0;
            end
            DONE: begin
                compute <= 0;
                done <= 1;
            end
            default: begin
                compute <= 0;
                done <= 0;
            end
        endcase
    end
end
```

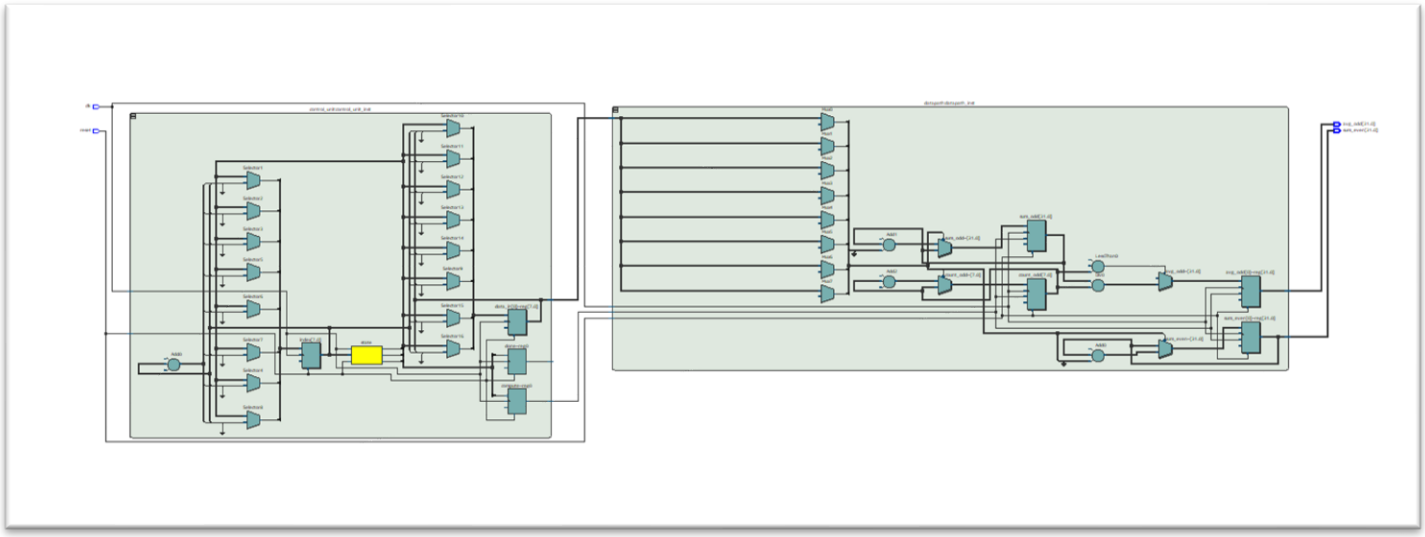
4. Modularity and Hierarchical Design:
   - The top_module includes both the control_unit and datapath modules.

```
control_unit control_unit_inst(
    .clk(clk),
    .reset(reset),
    .data_in(data_in),
    .compute(compute),
    .done(done)
);

datapath datapath_inst(
    .clk(clk),
    .reset(reset),
    .data_in(data_in),
    .compute(compute),
    .sum_even(sum_even),
    .avg_odd(avg_odd)
);
```

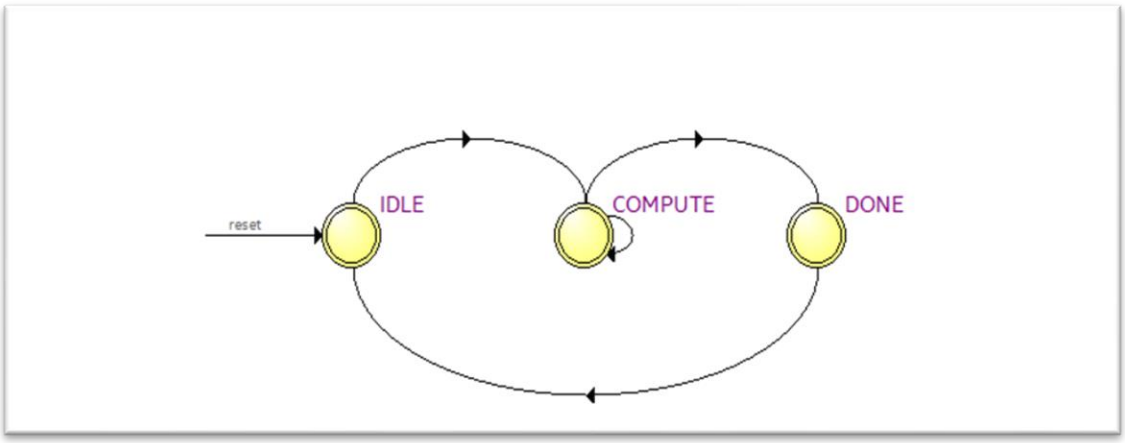
Adding RTL Viewer and State Machine Viewer

1. RTL Viewer
   - The RTL Viewer provides a graphical representation of the design's Register Transfer Level (RTL). It shows how different modules, components, and signals are interconnected.



2. State Machine Viewer
   - The State Machine Viewer visualizes the state transitions of the finite state machine (FSM) in the control unit.

     Control Unit FSM:



THANKS FOR READING

ÇAĞRI YILDIZ 1901042630