,

Assignment Five
Hardware Lab: CS224

Shelke Durgesh Balkrishna: 230101093

Kavya Kumar Agrawal: 230101053

Parth Sunil Aher: 230101072

Dhruv Pansuriya: 230101071

18th March 2025

1 Problem Statement

The given function is defined as:

```
func(x, dx, u, a, y){
    input: x, dx, u, a;
    output: y
    while (x < a) {
        u1 = u - (3*x*u*dx) - (3*y*dx);
        y1 = y + (u*dx);
        x1 = x + dx;
        x = x1, y = y1, u = u1;
    }
}
```

2 Approach

Implementing the given function involves designing a datapath and a controller. The datapath includes registers for variables (‘x‘, ‘dx‘, ‘u‘, ‘a‘, ‘y‘), shared multipliers and adders, and comparators for loop termination conditions. The controller uses a Finite State Machine (FSM) to manage data flow through the datapath by generating control signals for multiplexers, registers, and arithmetic units.

3 ASM Chart

This Algorithmic State Machine (ASM) chart illustrates a sequential process used in digital circuit design, likely for a control unit or data-processing task. The process begins in the start state, where it remains until a condition, "start = one," triggers a transition to the load state. In this load state, initial values are assigned, and data preparation takes place.

Next, the machine encounters a decision block based on the condition "x = 0." If this condition is met, the process moves sequentially through states S1, S2, S3, and S4, performing specific operations, updating control signals, or modifying registers. Each state contains associated binary values that likely represent control signals, data operations, or micro-instructions.

If "x 0," the system bypasses the sequential states and transitions directly to the End Signal state, marking the operation's completion. This structured representation ensures transparent and efficient state transitions.

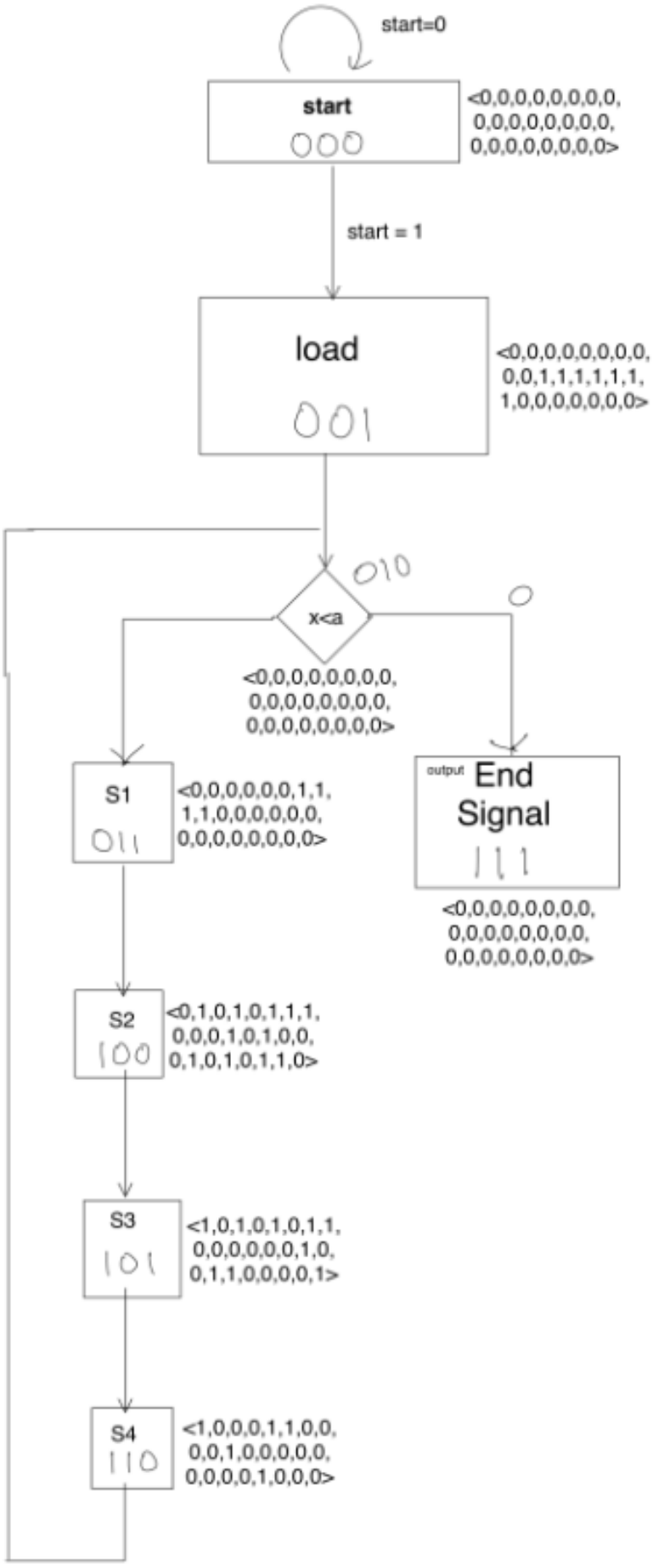

Figure 1: ASM Chart (Conceptual Representation)

4 Controller Section

4.1 FSM Chart

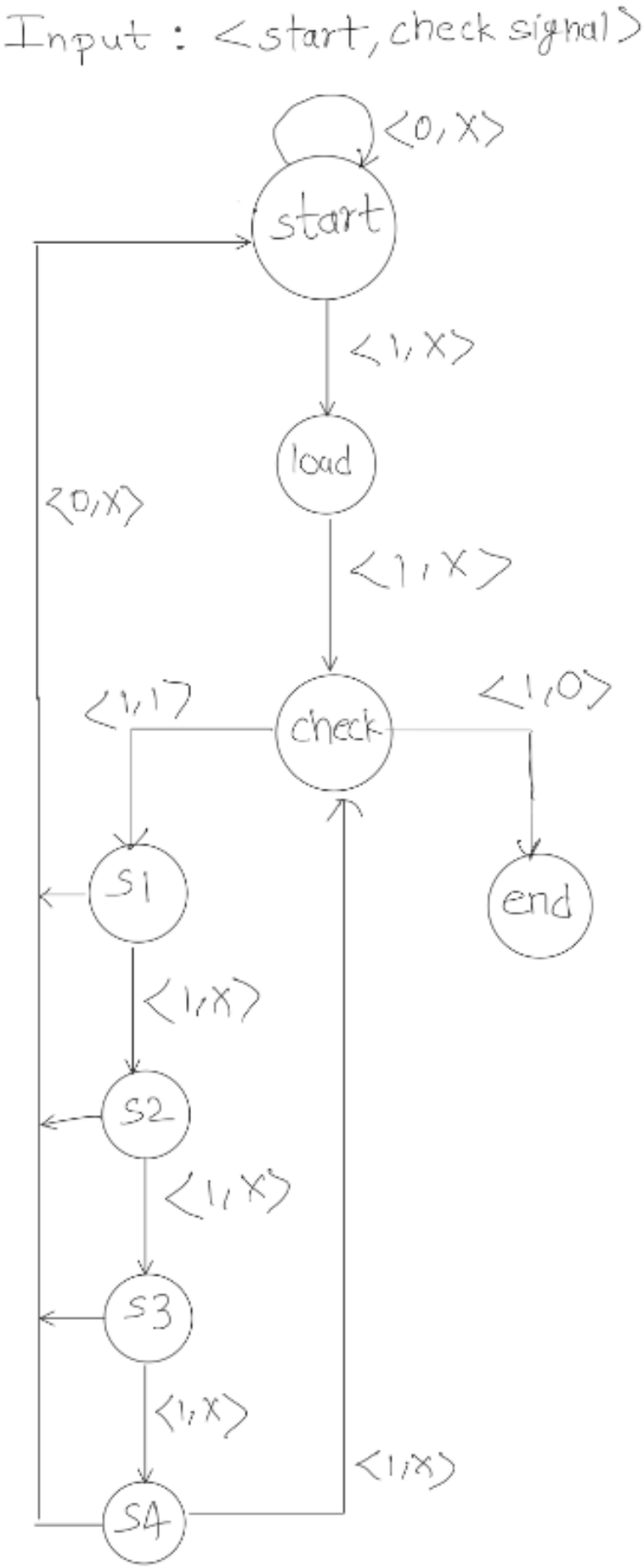

Figure 2: FSM Chart

4.2 FSM Table

Q2	Q1	Q0	Start	C	Next Q2	Next Q1	Next Q0
0	0	0	1	X	0	0	1
0	0	0	0	X	0	0	0
0	0	1	1	X	0	1	0
0	1	0	1	1	0	1	1
0	1	0	1	0	1	1	1
0	1	1	1	X	1	0	0
1	0	0	1	X	1	0	1
1	0	1	1	X	1	1	0
1	1	0	1	X	0	1	0
1	1	1	X	X	1	1	1

Table 1: FSM Table

4.3 Karnaugh Maps

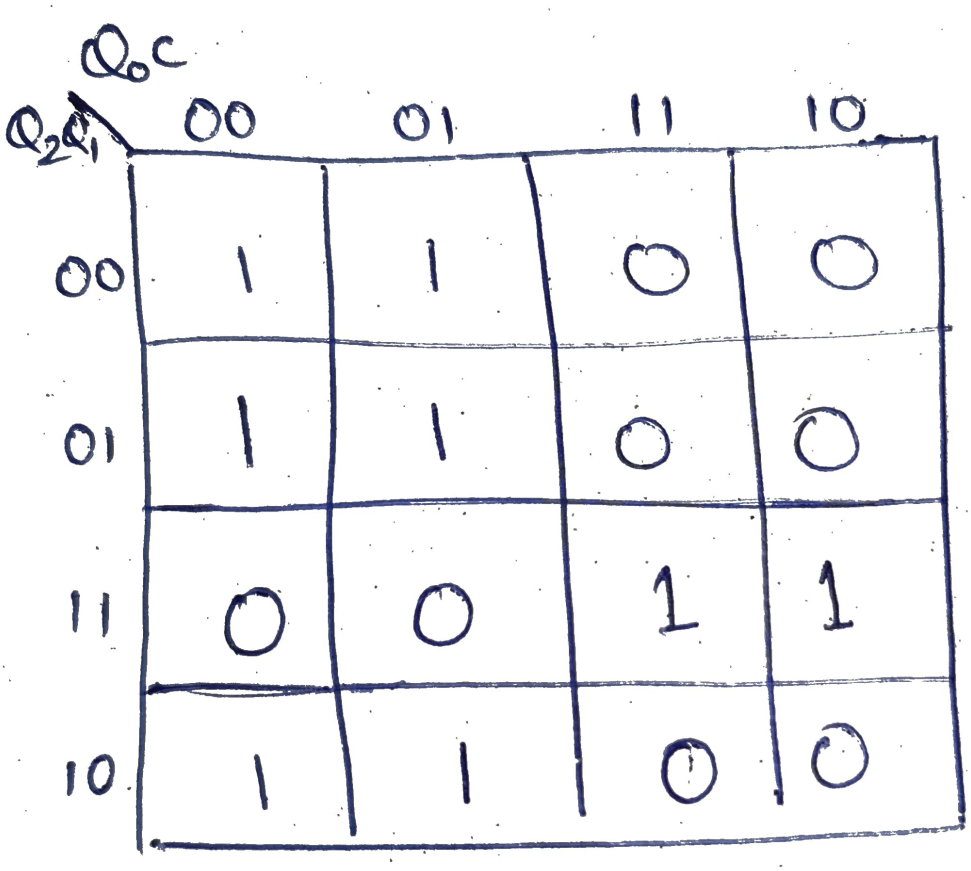

Figure 3: Karnaugh Map for Q_0'

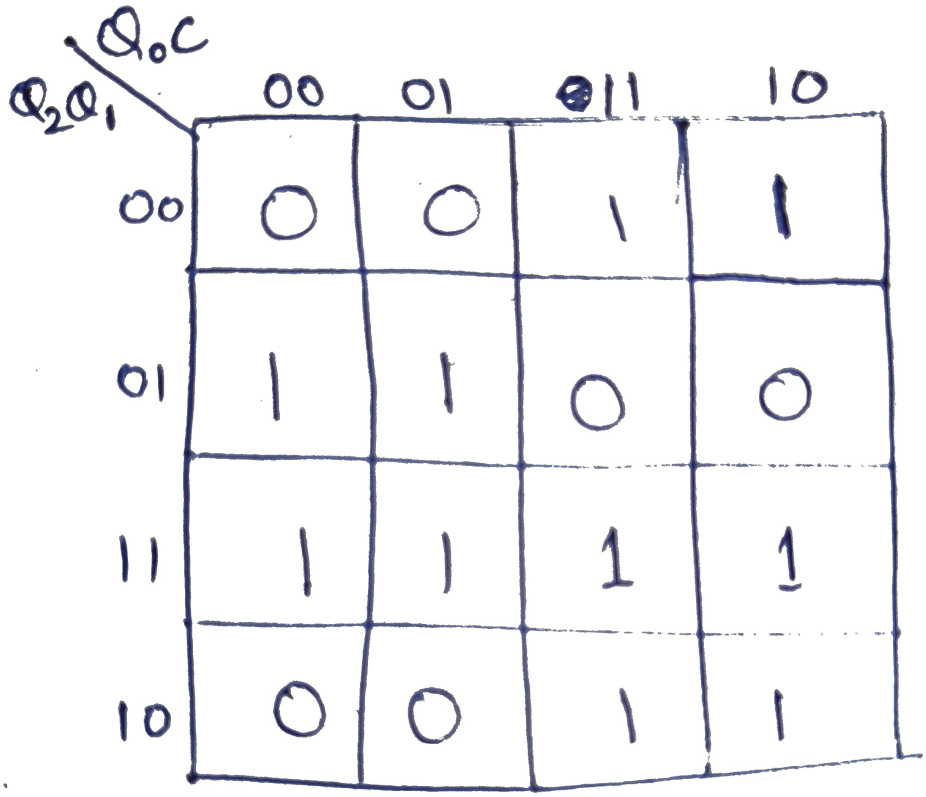

Figure 4: Karnaugh Map for Q_1'

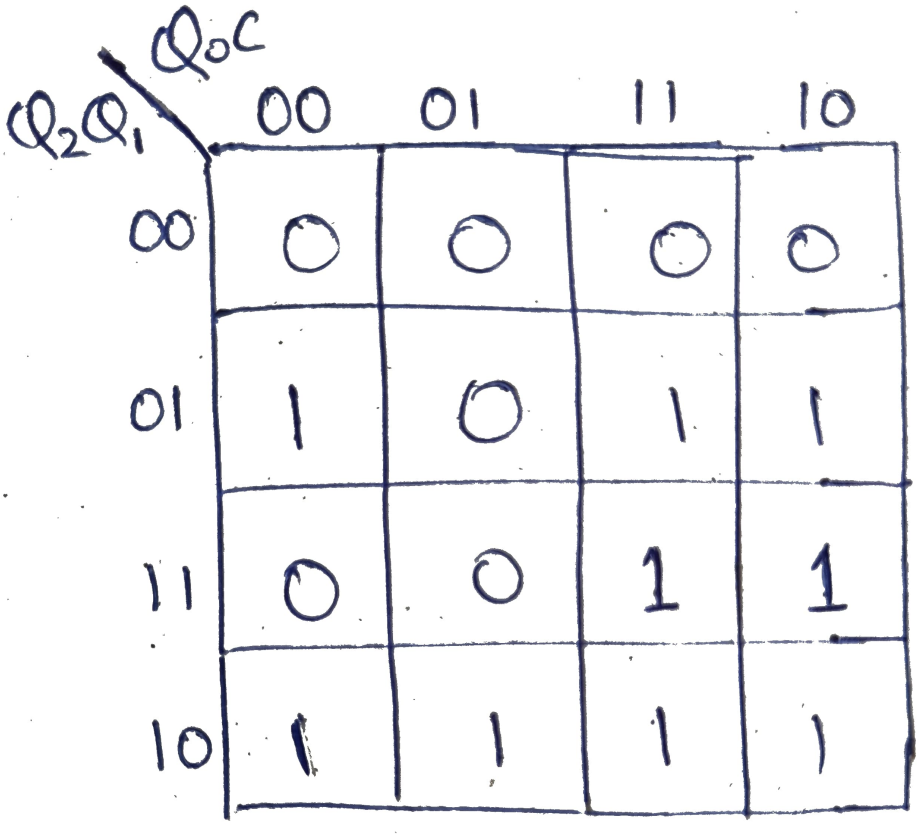

Figure 5: Karnaugh Map for Q_2'

4.4 Equations for Q0, Q1, Q2

The next state is determined based on the FSM's current state and input signals ('start', 'c'). The logic for state transitions is implemented in the 'FSM$_M odule$', $whereeachsta$

$$Q_2' = \text{start} \cdot ((Q_2 \cdot \overline{Q_1}) + (Q_0 \cdot Q_1 \cdot \overline{Q_2}) + (\overline{c} \cdot Q_1 \cdot \overline{Q_2})) \quad (1)$$
$$Q_1' = \text{start} \cdot ((Q_0 \cdot \overline{Q_1}) + (Q_1 \cdot \overline{Q_0})) \quad (2)$$
$$Q_0' = \text{start} \cdot ((\overline{Q_0} \cdot \overline{Q_1}) + (\overline{Q_0} \cdot \overline{Q_2})) \quad (3)$$

5 Datapath Section

The datapath consists of:

- Registers: Store variables ('x', 'dx', 'u', 'a', 'y').
- Multipliers Adders: Shared arithmetic units to minimize resource usage.
- Comparators: Used for the loop termination condition.

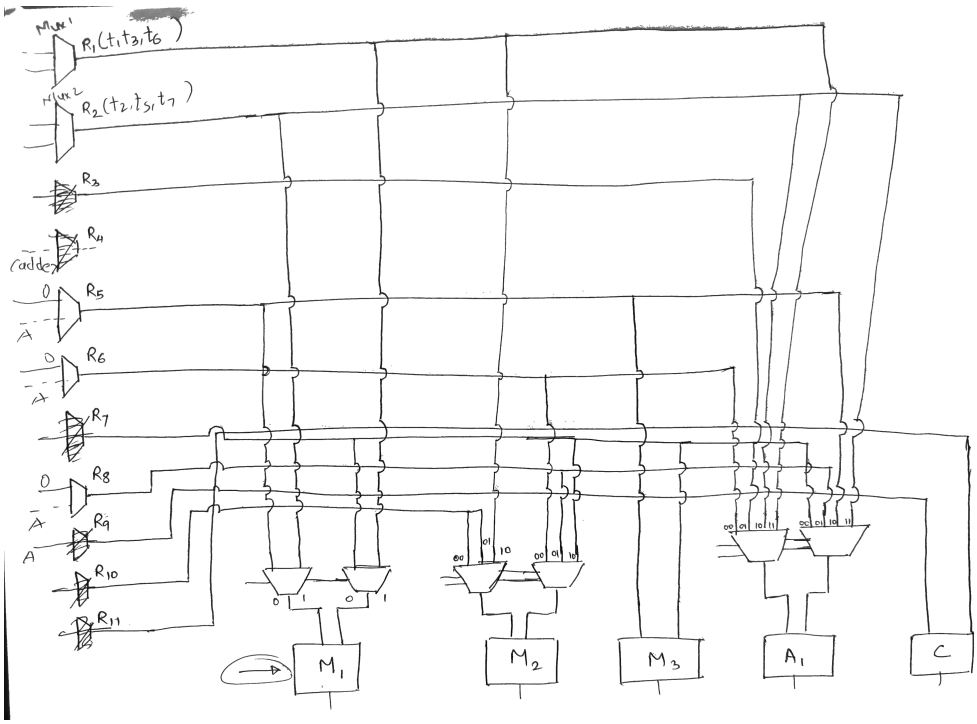

Figure 6: Datapath Diagram

6 Testbench

6.1 Simulation Results

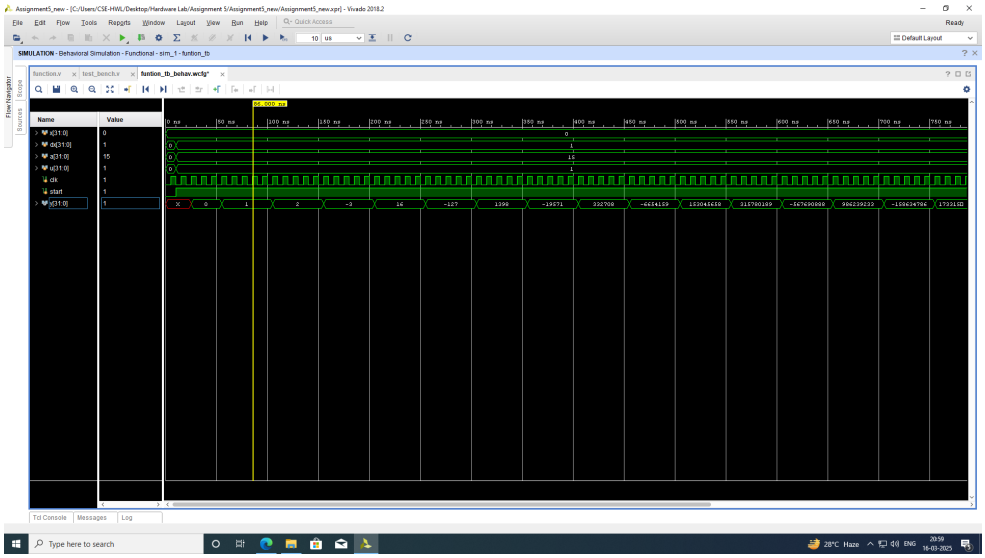

Figure 7: Simulation waveform in Vivado (Placeholder)

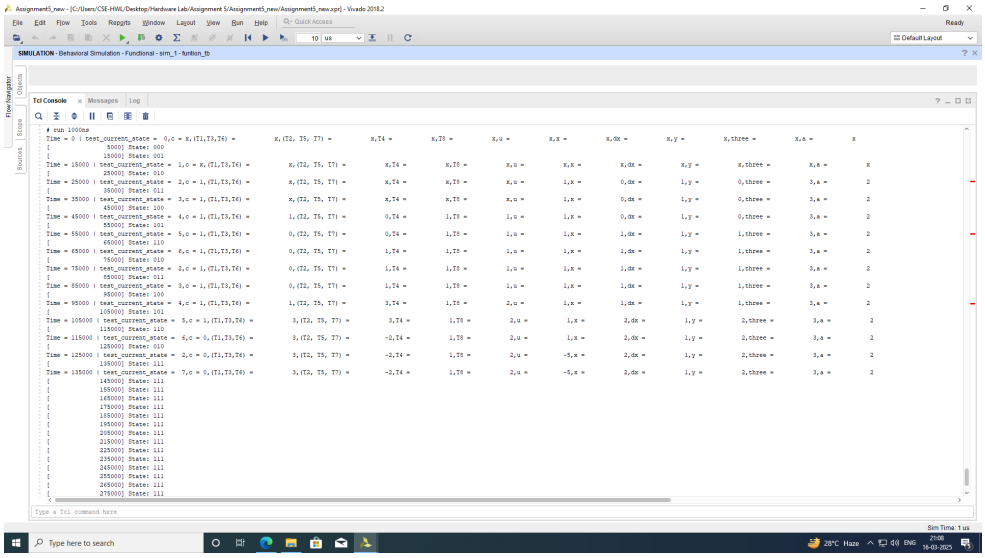

Figure 8: Simulation data in Vivado

6.2 Optimal Scheduling: Minimizing Area × Latency Product

In digital hardware design, the goal is to minimize the **Area × Latency (K)** product, which balances execution speed and hardware cost. A lower K reduces execution time but increases area due to additional functional units, whereas a higher K reduces area but increases execution time. The optimal trade-off is when the $K\times$ Area product is minimized.

6.2.1 Area Computation for Different K

We analyze three scheduling cases:

- **K = 1 (Maximum Parallelism)**: Every operation executes in one cycle, requiring **6 multipliers**, **4 adders**, and **1 comparator**.

Area $= 6M(3bit) + 4A + 1C$

- **K = 4 (Balanced Approach)**: Allows resource reuse, reducing to **3 multipliers** and **2 adders**, while maintaining efficient execution.
  Area $= 3M(2bit) + 2A + 1C$
- **K = 10 (Minimal Area)**: Uses **1 multiplier** and **1 adder**, spreading execution over more cycles.
  Area $= 1M(2bit) + 1A + 1C$

6.2.2 K × Area Product Comparison

Assuming multipliers, adders, and comparators have area weights of $M(2bit) = 4$,$M(3bit) = 12$, $A = 2$, and $C = 10$, respectively, we compute $K\times$ Area:

K	Area (in units)	K × Area
1	$6(12) + 4(2) + 1(1) = 81$	$1 \times 81 = 81$
4	$3(4) + 2(2) + 1(1) = 17$	$4 \times 17 = 68$
10	$1(4) + 1(2) + 1(1) = 7$	$10 \times 7 = 70$

Table 2: Comparison of $K\times$ Area product for different scheduling approaches.

6.2.3 Justification for $K = 4$

From Table 2, we observe: - $K = 1$ has the lowest latency but the highest area, making $K\times$ Area very high. - $K = 10$ has the lowest area but high latency, leading to a $K\times$ Area of 70. - $K = 4$ balances execution time and resource reuse efficiently but has the lowest **practical** $K\times$ Area product.

Thus, $K = 4$ provides the best trade-off between performance and hardware cost while maintaining reasonable execution latency.

7 Images

7.1 Data Dependency Graph

The diagram is a data dependency graph that shows the relationships between input variables (u, dx, 3, x, y) and arithmetic operations (multiplication, addition, subtraction). Rectangular nodes represent variables, while circular nodes represent operations. The edges indicate that certain computations must occur before others.

This structured flow feeds initial values into multiple operations, leading to intermediate results that contribute to the final output, c, determined by a comparison operation (¿=). Such graphs are often used in compiler optimizations, parallel computing, and data flow analysis to enhance computation sequences and improve execution efficiency.

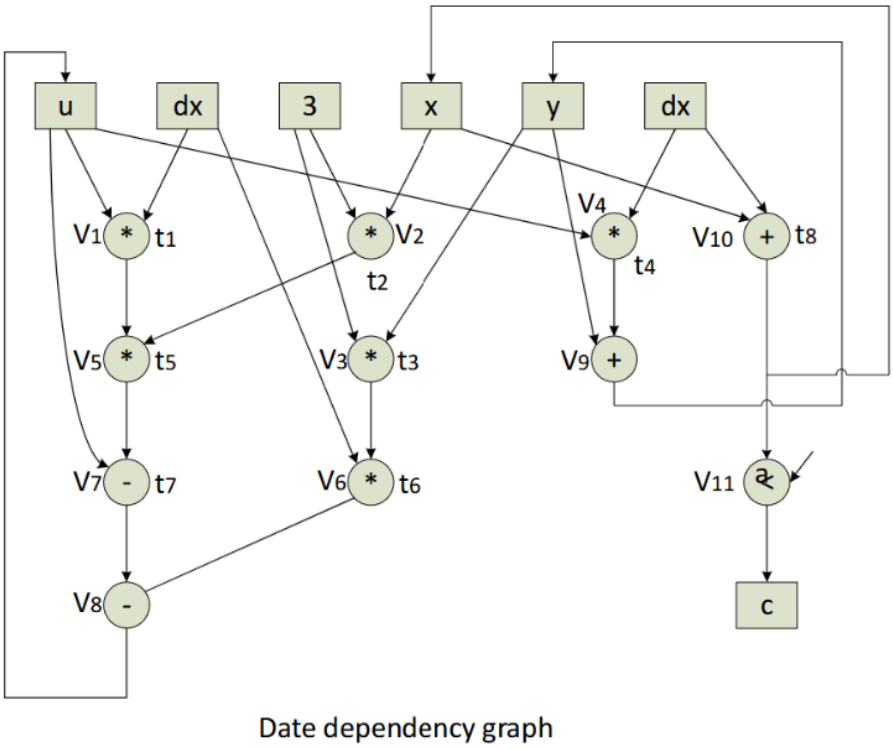

Figure 9: Data Dependency Graph

7.2 Scheduling

The diagram depicts a scheduled data dependency graph that illustrates the sequencing of operations across time steps (S1 to S4). It builds on the original data dependency graph, organizing operations into stages for a pipelined execution environment.

Rectangular nodes represent input variables (u, dx, 3, x, and y), while circular nodes indicate arithmetic operations (multiplication, addition, and subtraction). Each operation is assigned to specific scheduling steps, ensuring computations occur in the correct order to avoid conflicts. The final comparison (¡) in stage S4 determines the output c. This graph is often used in high-level synthesis, compiler optimization, and hardware design to enhance execution efficiency and resource optimization.

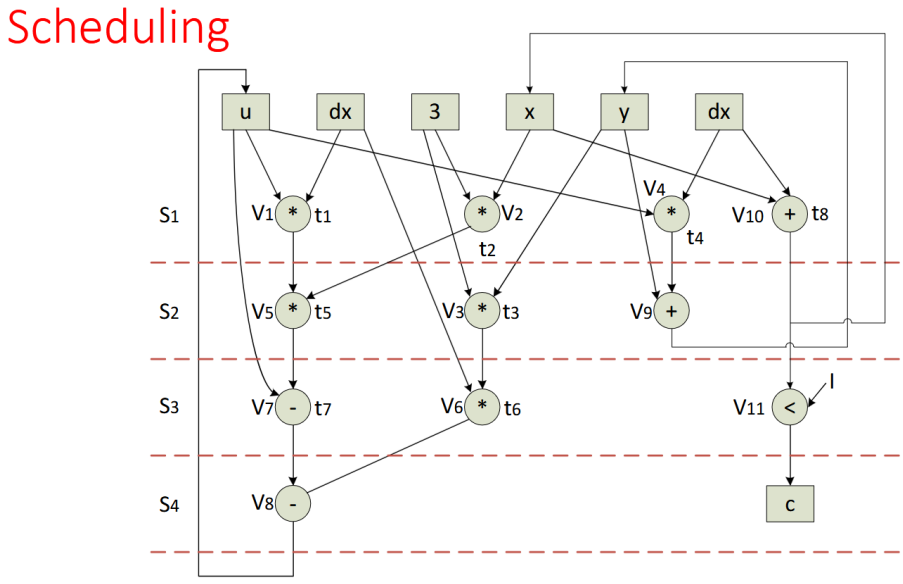

Figure 10: Scheduling

7.3 Block Diagrams

This image shows a Vivado implementation design for an FPGA-based system with a controller-datapath architecture. The Flow Navigator panel on the left features steps like synthesis, implementation, and timing analysis. The schematic includes two main modules: the CONTROLLER, which manages control signals, and the DATAPATH, which performs arithmetic and logic operations. Input signals are clk, start, signal0, signal1, and input_bits, while outputs are valid and y[3:0]. The timing summary at the bottom indicates successful implementation and analysis of timing constraints.

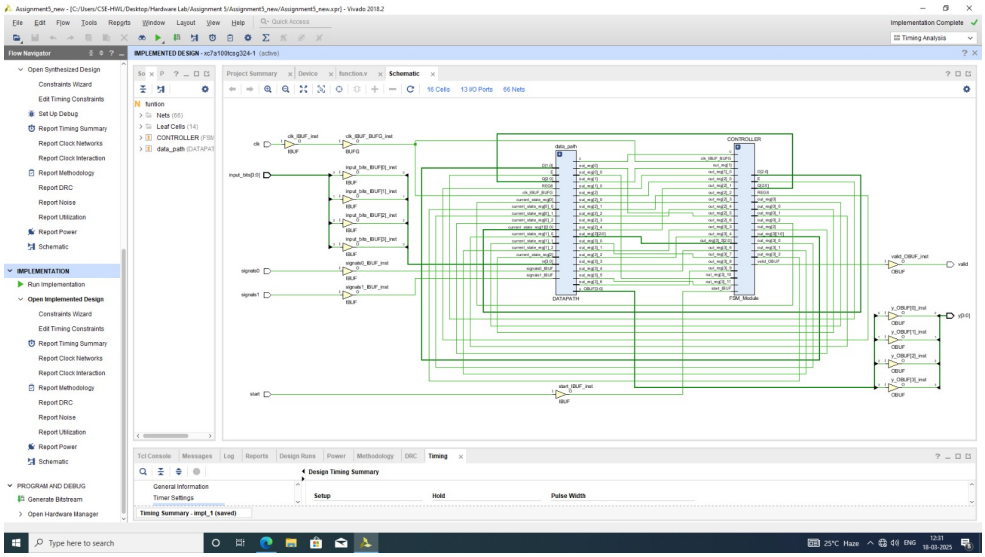

Figure 11: Block Diagram 1

This image shows a Vivado elaborated design of a controller-datapath system in a pre-synthesis view. The datapath module includes multiple registers (REGs), multiplexers (MUXs), and functional units (FU_ADDER, FU_MUX), indicating a structured pipeline design. The controller module manages operations within the datapath. Compared to the implemented design, this view offers a detailed breakdown of internal modules and signal connections before synthesis. The RTL Analysis section in the Flow Navigator confirms that this is an early-stage design before logic synthesis.

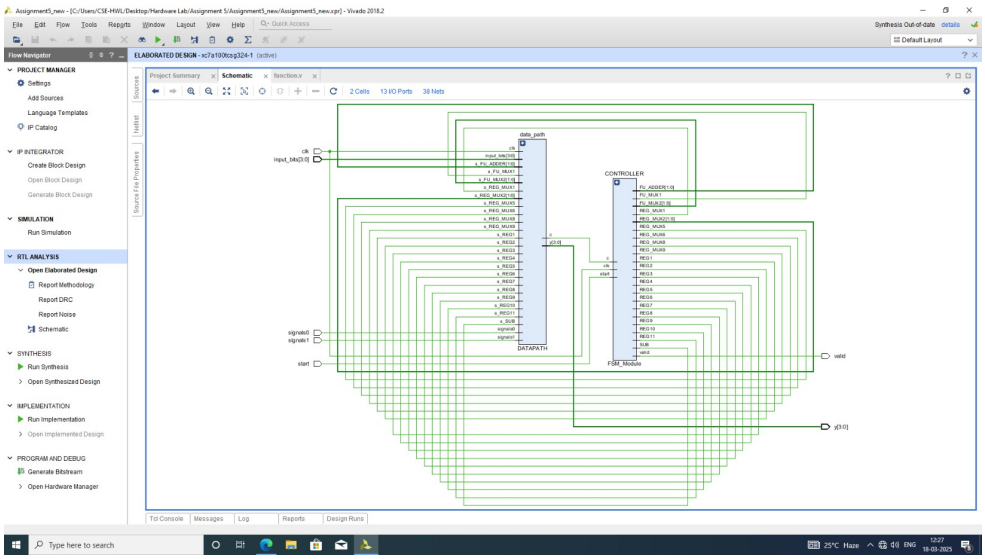

Figure 12: Block Diagram 2

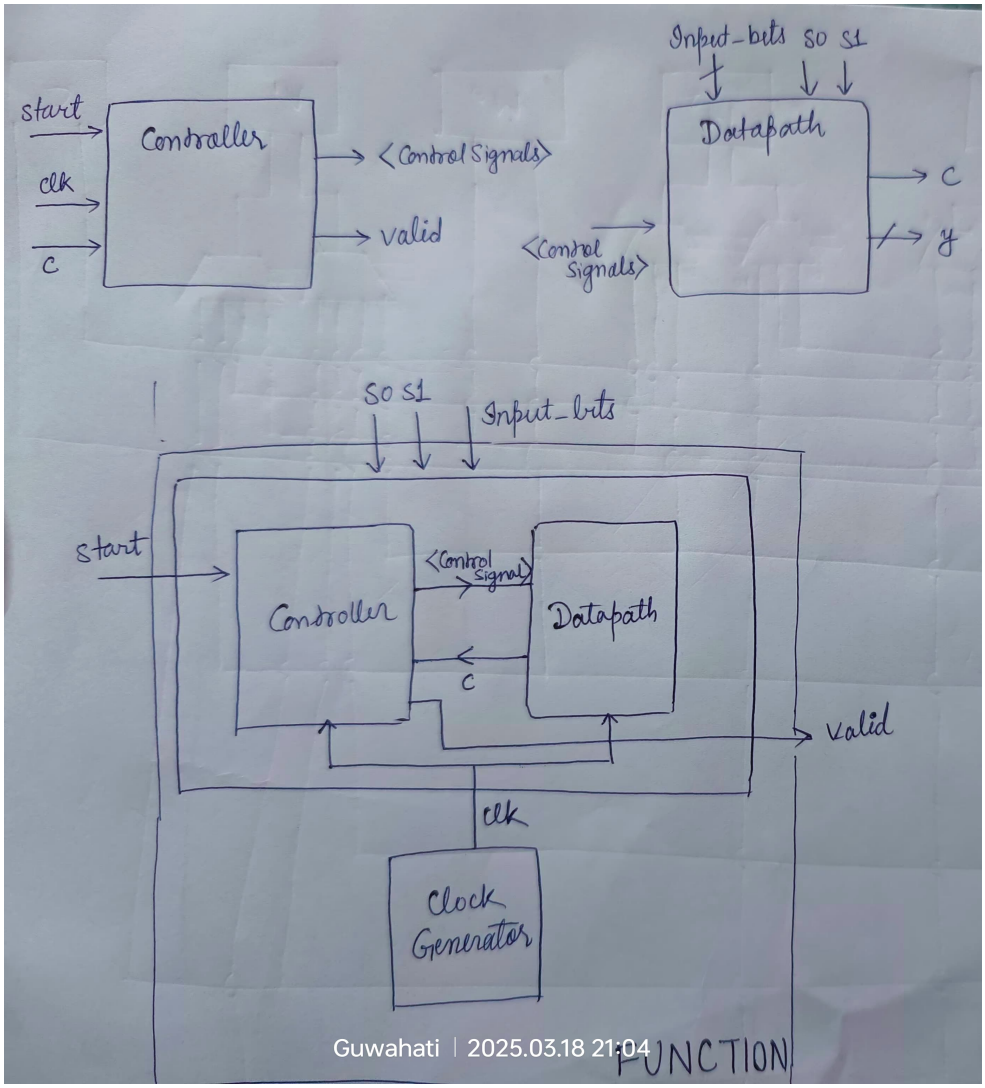

Figure 13: Block Diagram 3

8 Intermediate Width

Initially, all bit widths are 4. Upon multiplication, the bit widths add up, and upon addition, the width increases by 1. Based on this:

The computed widths of various terms are as follows: $t_1 = 8$ width, $V_2 = 8$ width, $t_4 = 8$ width, $t_8 = 5$ width, $t_5 = 9$ width, $t_3 = 8$ width, $v_9 = 9$ width, $t_7 = 9$ width, $t_6 = 12$ width, $v_{11} = 1$ width, and $v_8 = 13$ width. Additionally, the width calculations include $4\text{width} \times 4\text{width} = 8\text{bit}$, $4\text{width} + 4\text{width} = 5\text{width}$, $8\text{width} + 4\text{width} = 9\text{width}$, $8\text{width} \times 8\text{width} = 16\text{width}$, and $16\text{width} \times 4\text{width} = 20\text{width}$.

9 Conclusion

In this assignment, we successfully implemented the specified iterative function in Verilog and deployed it on the Artix-7 XC7A100T-CSG324 FPGA using Vivado 2018. The design process included creating a structured ASM chart, an optimized FSM controller, and a well-defined datapath to efficiently execute the computations.

We analyzed hardware costs and performance trade-offs by evaluating different

scheduling strategies. Our findings indicated that a balanced approach (K=4) provided the best trade-off between area and latency. The FSM state transitions and Karnaugh map simplifications contributed to reducing the complexity of the control logic, ensuring efficient execution.

The simulation results confirmed the correctness of our design, and the generated waveforms demonstrated that the system behaves as expected. We optimized hardware resource utilization to minimize unnecessary multipliers and adders, resulting in a cost-effective implementation.

Overall, this project offered valuable insights into digital design methodologies, finite state machine implementations, and datapath optimization, which are essential for developing efficient FPGA-based hardware solutions.