Assignment Seven
Hardware Lab: CS224

Shelke Durgesh Balkrishna: 230101093

Kavya Kumar Agrawal: 230101053

Parth Sunil Aher: 230101072

Dhruv Pansuriya: 230101071

10th April 2025

1 Problem Statement

The task is to efficiently implement long chain addition in Verilog and demonstrate it using an FPGA. The requirements are:

- Implement an iterative datapath using one adder.
- Implement an iterative datapath using four adders.
- Create a pipelined datapath with initiation interval and throughput of one per clock cycle.
- Compare area, latency, clock period, and throughput for all designs.
- Demonstrate the first design on FPGA with input and output valid signals.

2 Logic behind the circuit obtained

The circuit design is based on modular principles:

2.1 Iterative Datapath with One Adder

The circuit implements a simple sequential accumulator system that adds eight 4-bit input values using a 7-bit ripple carry adder and stores the final result. It consists of two main components: a controller and a datapath. The controller uses a finite state machine (FSM) to manage the sequence of operations, including collecting inputs, initializing the accumulator, performing sequential additions, and finalizing the result. The datapath stores the 4-bit inputs in registers, zero-extends them to 7 bits, and adds them one by one to an internal accumulator using a 7-bit ripple carry adder made from chained 1-bit full adders. The system waits for a ‘start‘ signal, collects inputs one at a time when ‘in_valid‘ is asserted, and then performs the addition over several clock cycles. Once all inputs are added, the final sum is stored in the ‘result‘ output, and the ‘done‘ signal is raised to indicate completion. The ‘ready_for_input‘ signal helps synchronize input loading with the controller state.

2.2 Iterative Datapath with Four Adders

The logic behind the circuit is centered around implementing parallel addition using four ripple carry adders controlled by a finite state machine (FSM) for sequential operations. The circuit takes eight 4-bit inputs and processes them in stages to compute the final 7-bit sum. In the first stage, four ripple carry adders operate in parallel to compute partial sums from pairs of inputs (e.g., REG0 + REG1, REG2 + REG3, REG4 + REG5, REG6 + REG7). These intermediate results are stored in 7-bit registers (REGX0-REGX3). In the second stage, two additional adders combine these partial sums into two aggregated sums stored in REGY0 and REGY1. Finally, a single adder computes the final sum by adding the values in REGY0 and REGY1, storing the result in REGZ. Multiplexers dynamically route data between stages based on control signals generated by the FSM, enabling efficient reuse of resources. The FSM transitions through states to manage input loading, intermediate calculations, and output generation, ensuring proper synchronization of operations. This design balances parallel computation for speed with sequential aggregation for resource efficiency.

2.3 Pipelined Datapath

The pipelined design achieves high throughput by overlapping computations across multiple pipeline stages, allowing efficient processing of inputs with minimal latency. The circuit is divided into three pipeline stages: the first stage uses four 4-bit adders to compute partial sums from eight input registers (REG0-REG7), storing the results in intermediate 7-bit registers (REGX0-REGX3). The second stage aggregates these partial sums using two 7-bit adders, storing results in REGY0 and REGY1. Finally, the third stage computes the overall sum using a single 7-bit adder, storing the result in REGZ. A finite state machine (FSM) coordinates the pipeline operation, ensuring proper synchronization and data flow between stages. Multiplexers dynamically route input data and intermediate results based on control signals, enabling efficient reuse of hardware resources. By overlapping computations across stages, this design achieves a throughput of one result per clock cycle after an initial latency of four cycles, balancing speed and resource utilization effectively.

Each design balances trade-offs between area, latency, and throughput.

3 Design Diagrams

3.1 ASM for One Adder

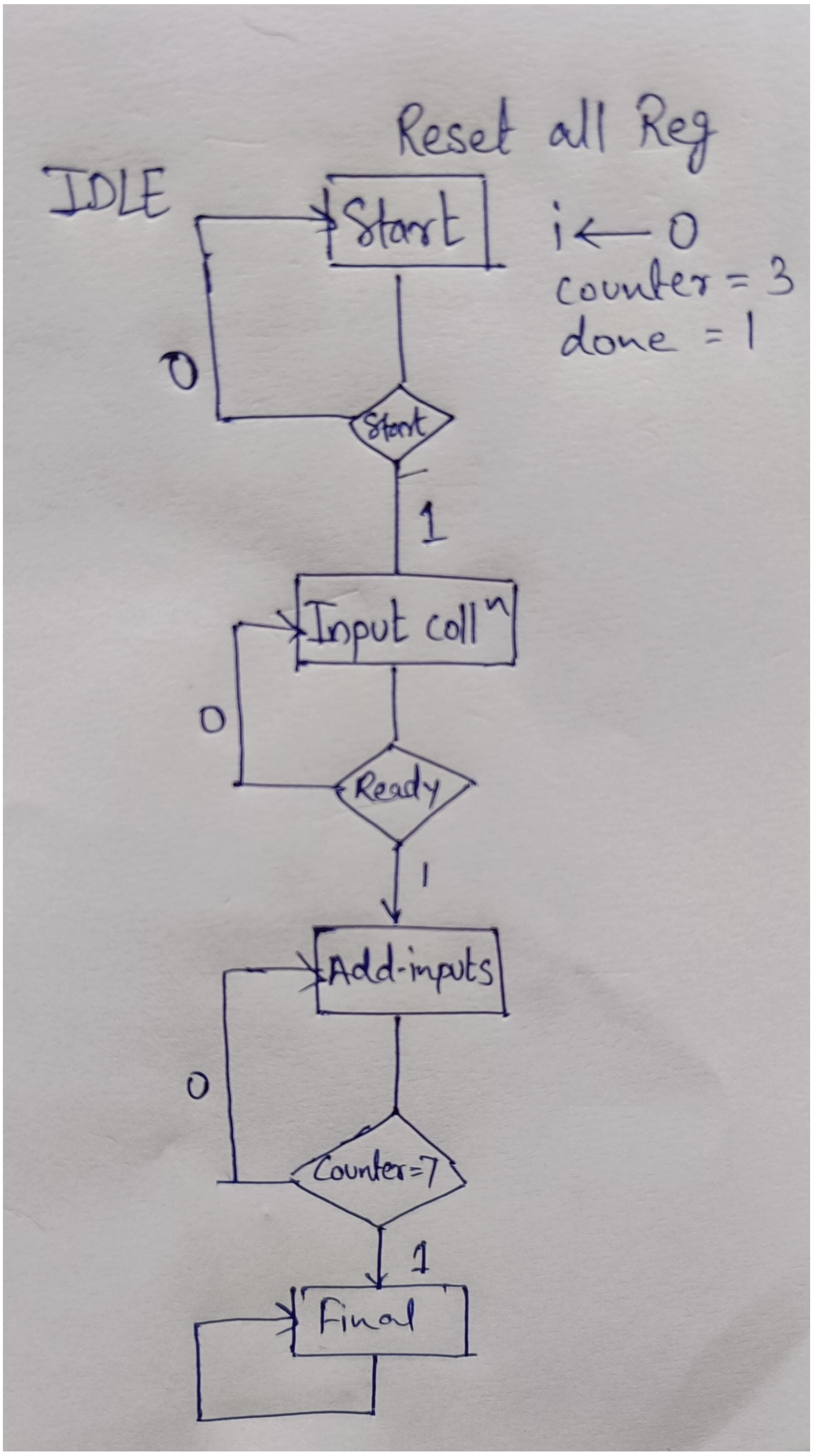


Figure 1: ASM for One Adder (Conceptual Representation)

3.2 FSM for One Adder

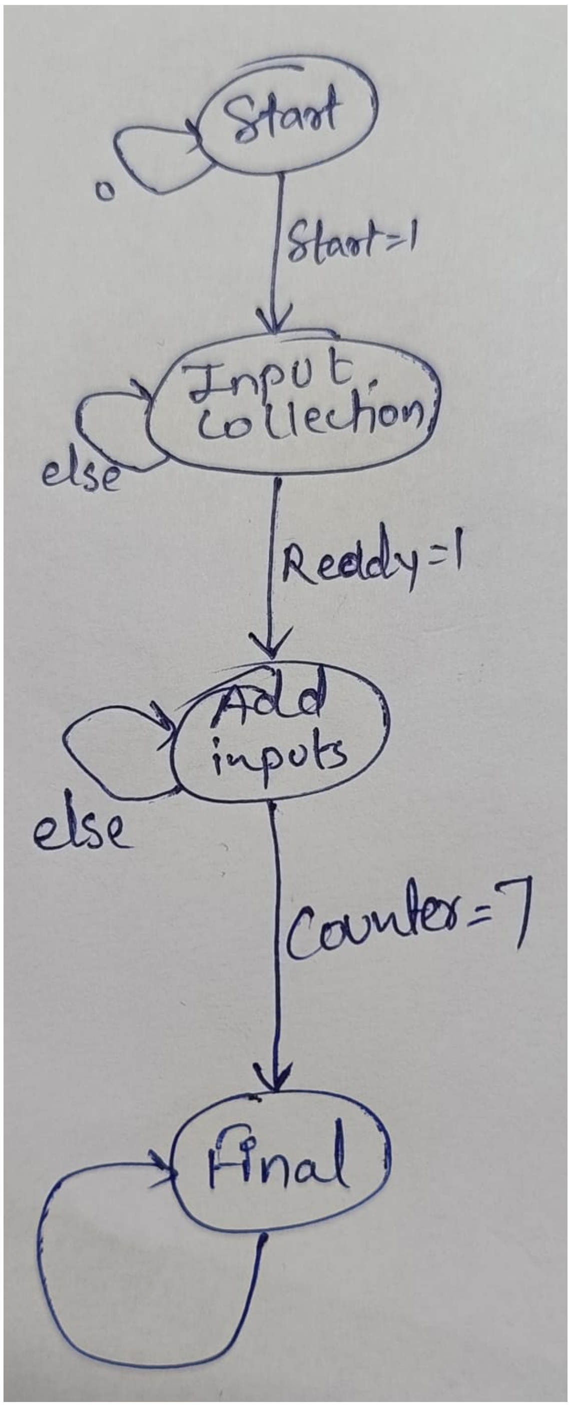


Figure 2: FSM for One Adder (Conceptual Representation)

3.3 Iterative Datapath with One Adder

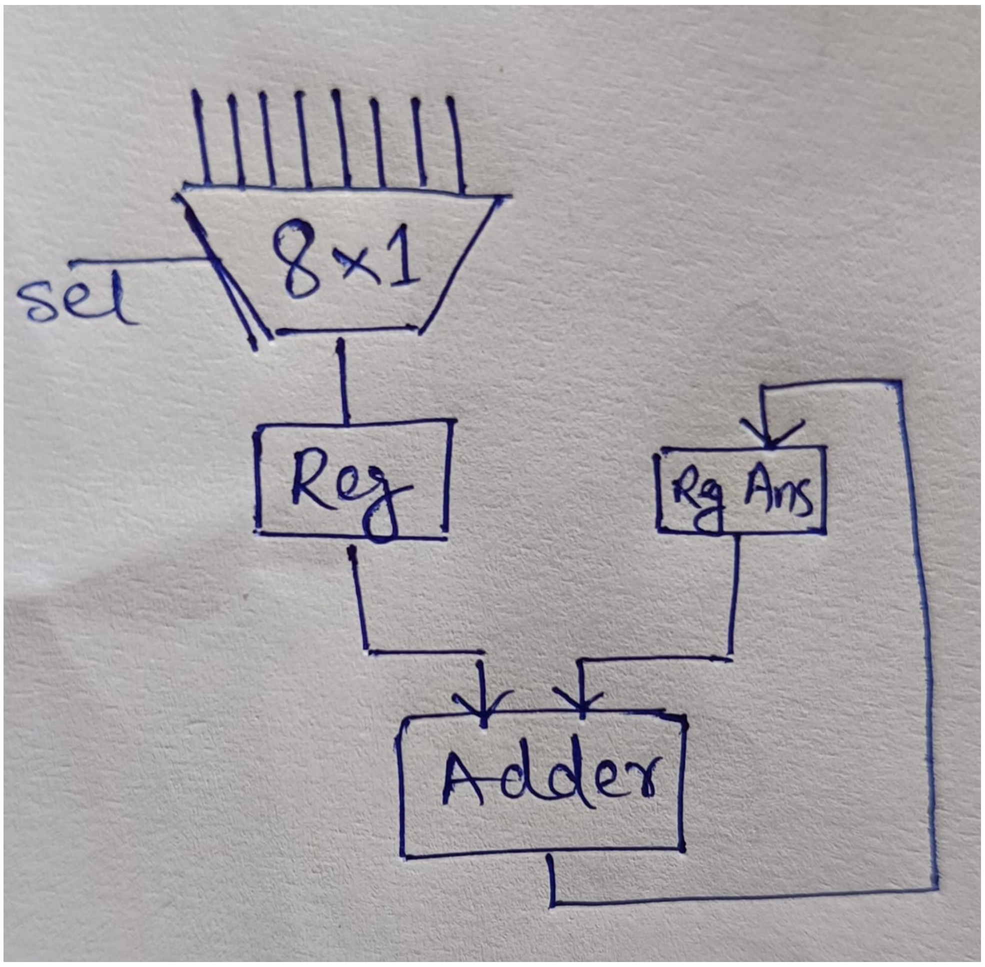


Figure 3: Iterative Datapath with One Adder (Conceptual Representation)

3.4 ASM for four Adder

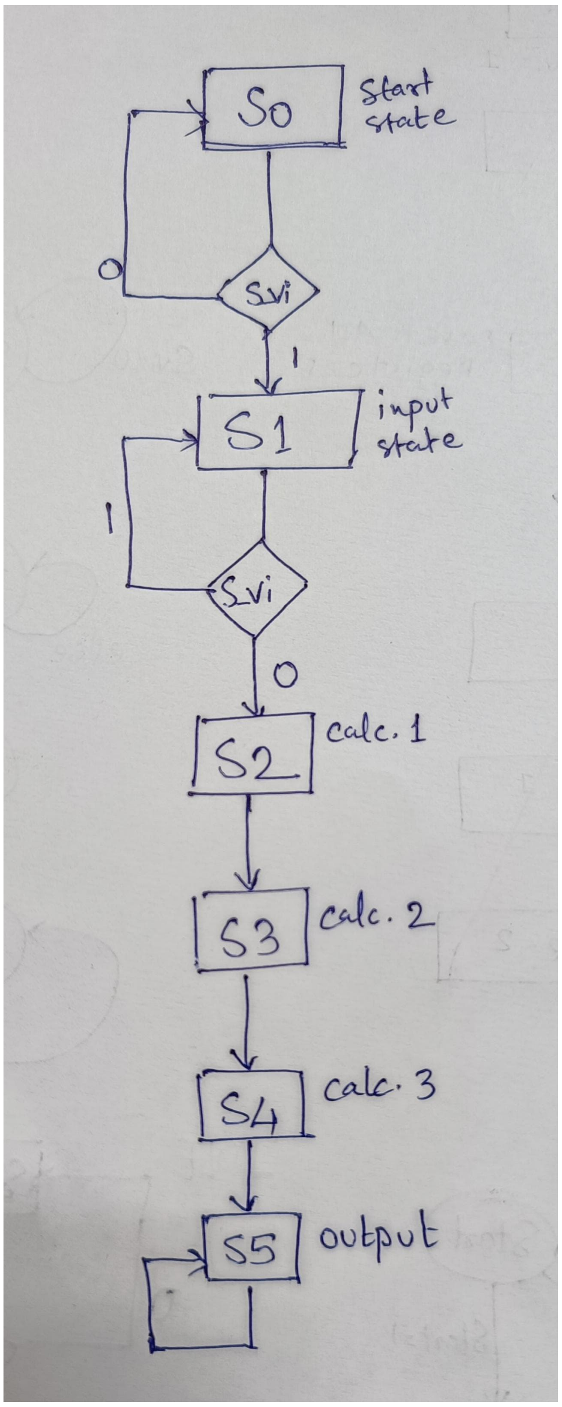


Figure 4: ASM for four Adder (Conceptual Representation)

3.5 FSM for four Adder

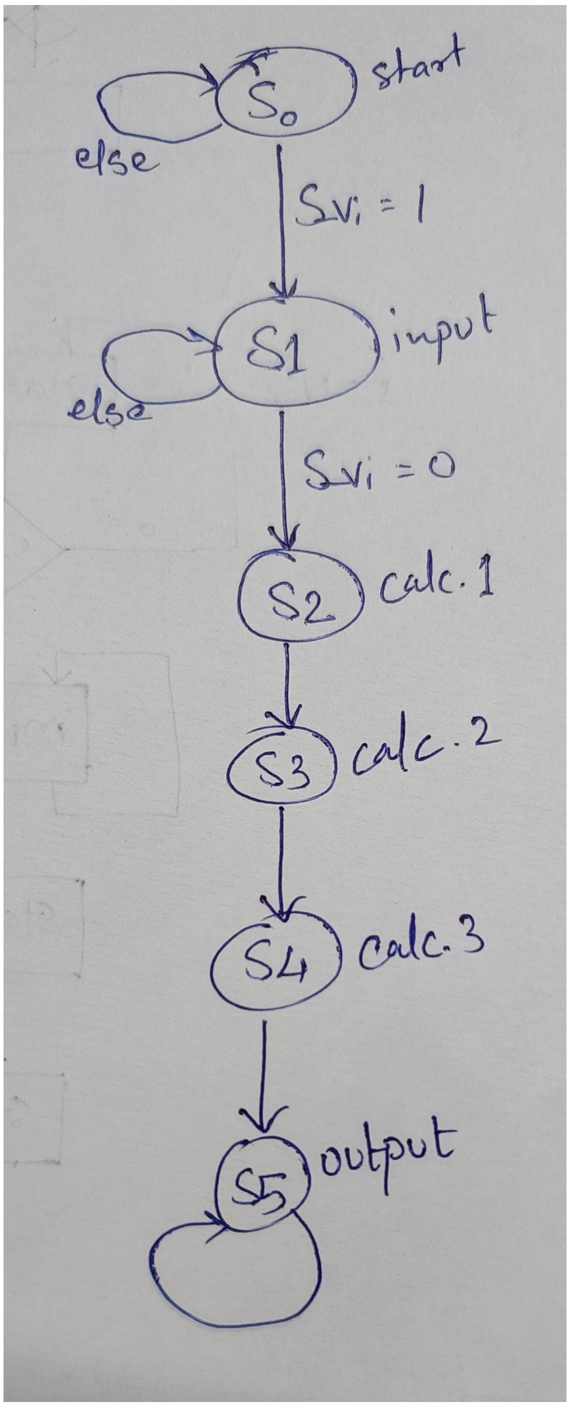


Figure 5: FSM for four Adder (Conceptual Representation)

3.6 Block diagram with One Adder

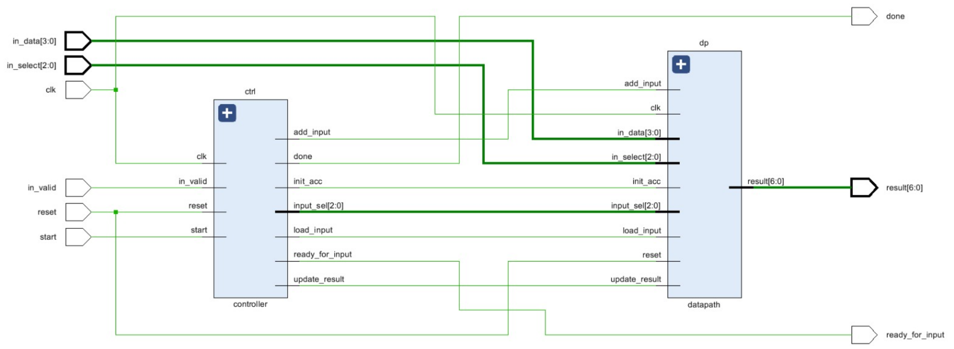


Figure 6: Iterative Datapath with One Adder (Conceptual Representation)

3.7 Block diagram1 with Four Adders

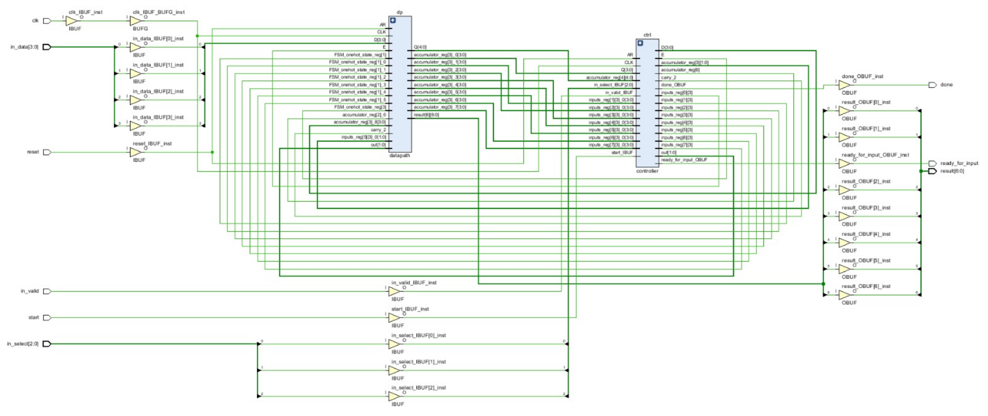


Figure 7: Block diagram1 with Four Adders

3.8 Block diagram2 with Four Adders

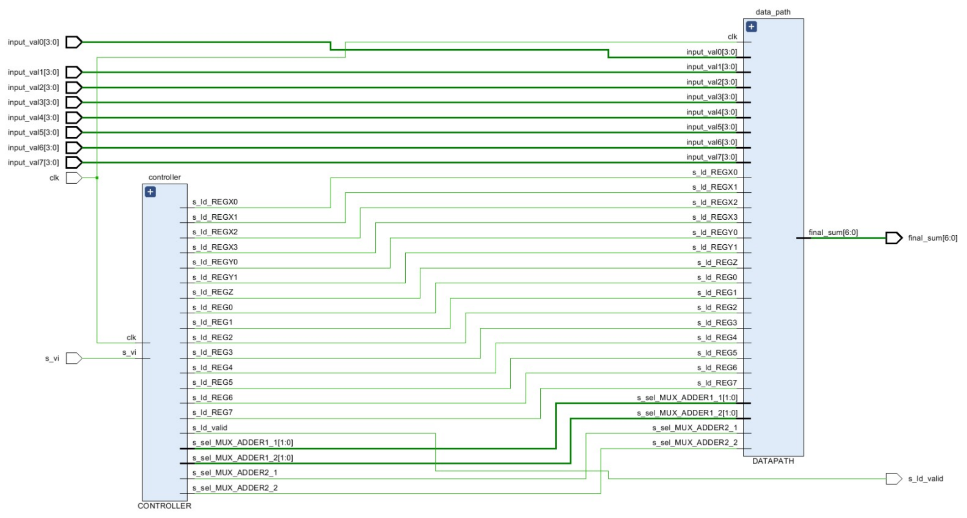


Figure 8: Block diagram2 with Four Adders

3.9 Pipelined ASM

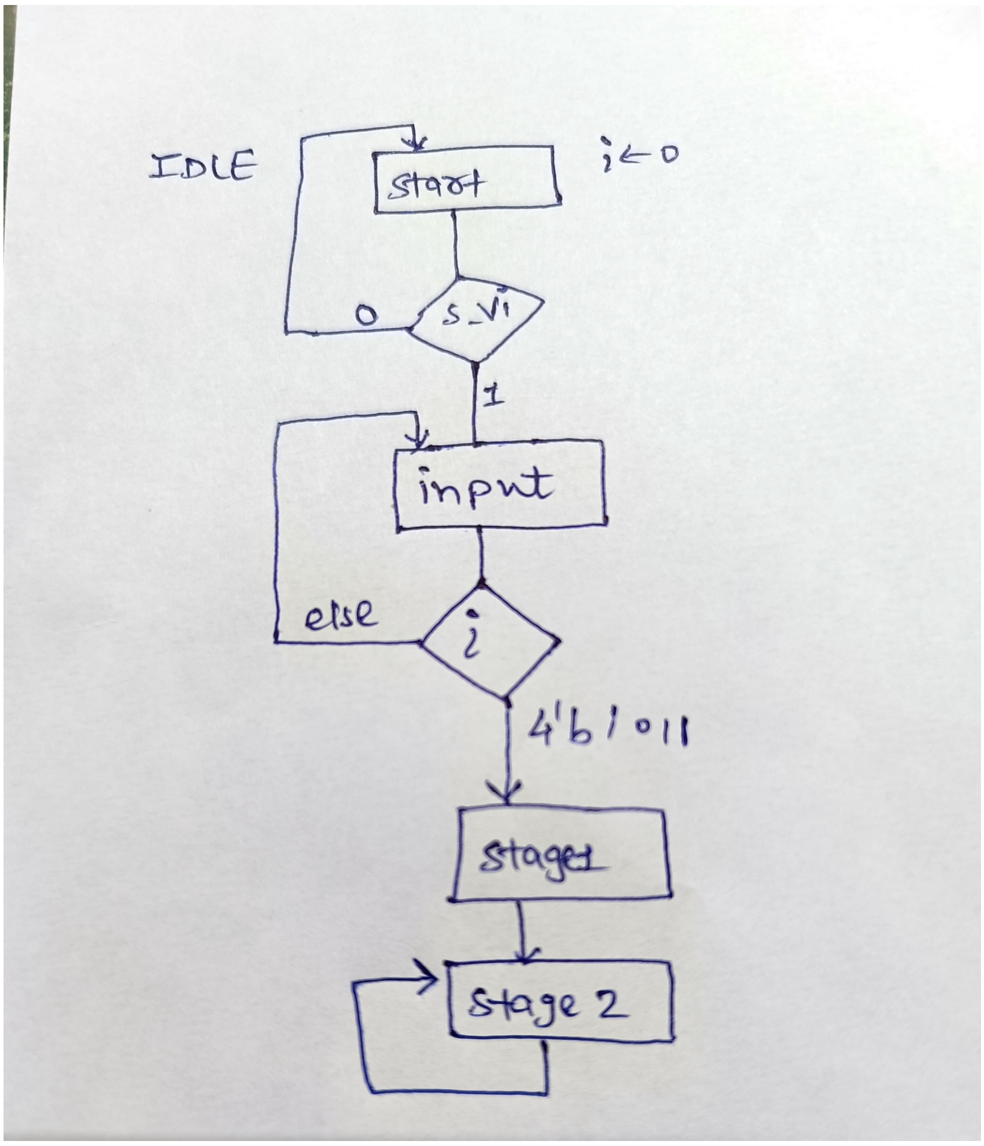


Figure 9: Pipelined ASM (Conceptual Representation)

3.10 Pipelined Datapath Design

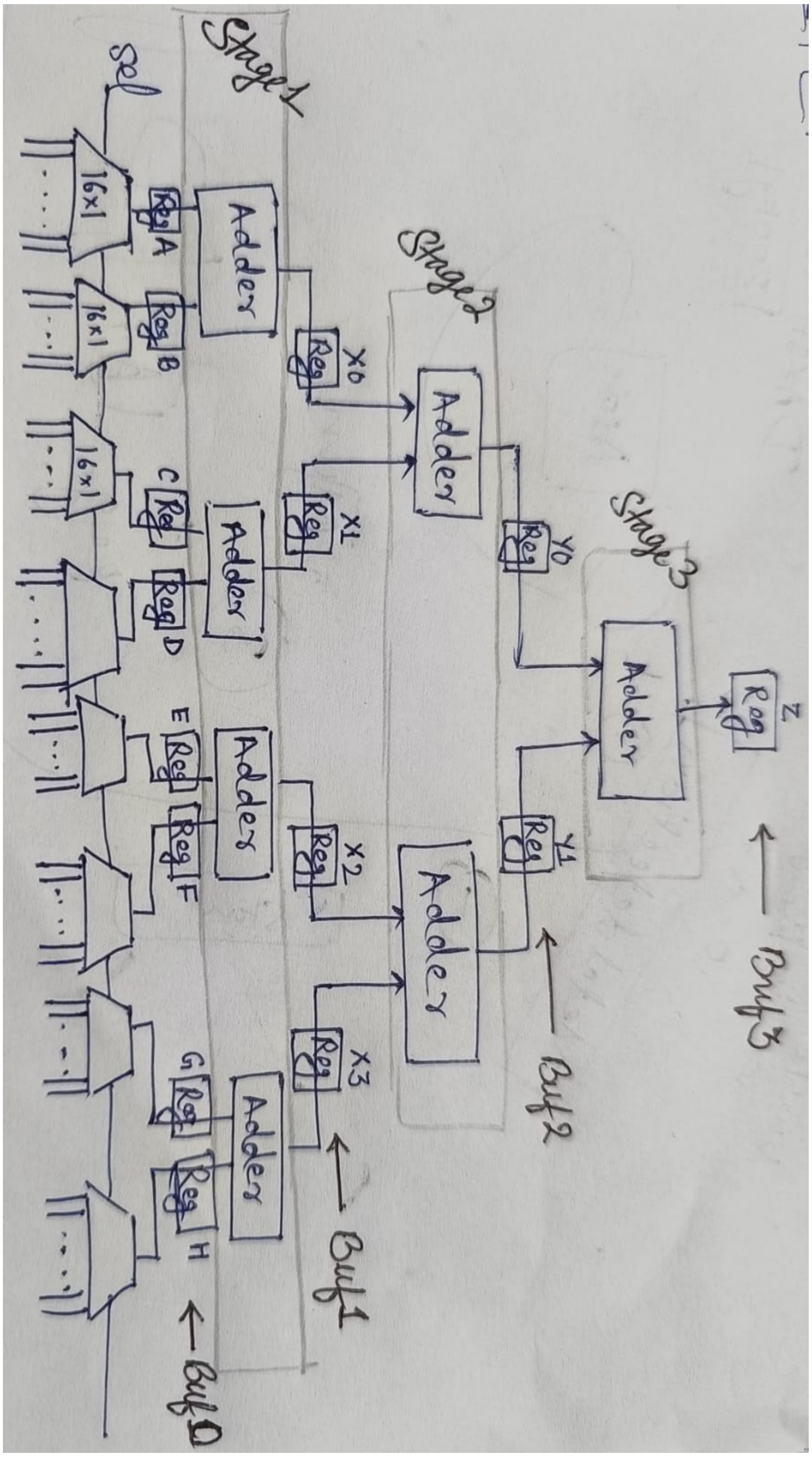


Figure 10: Pipelined Datapath (Conceptual Representation)

3.11 Pipelined block diagram

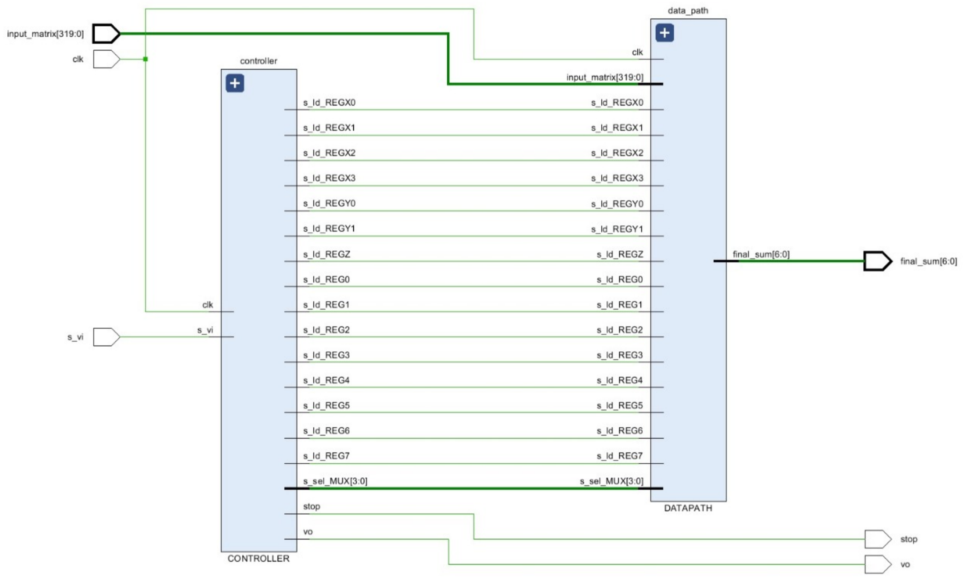


Figure 11: Pipelined block diagram

4 Simulation Results

Simulation was performed for all three designs using a stream of 10 input sets:

- Iterative Datapath with One Adder: Latency $= 8T_{clk}$, Area = Low.
- Iterative Datapath with Four Adders: Latency $= 2T_{clk}$, Area = Medium.
- Pipelined Datapath: Latency $= 4T_{clk}$, Throughput $= 1/T_{clk}$, Area = High.

Waveforms for each design are shown below:

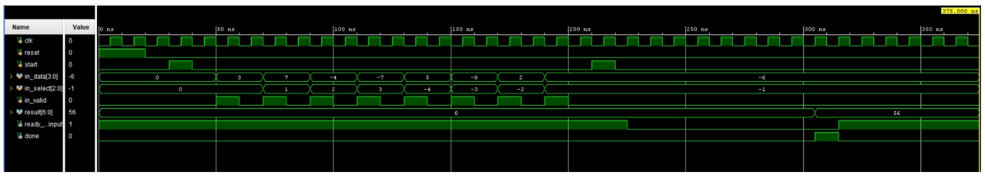
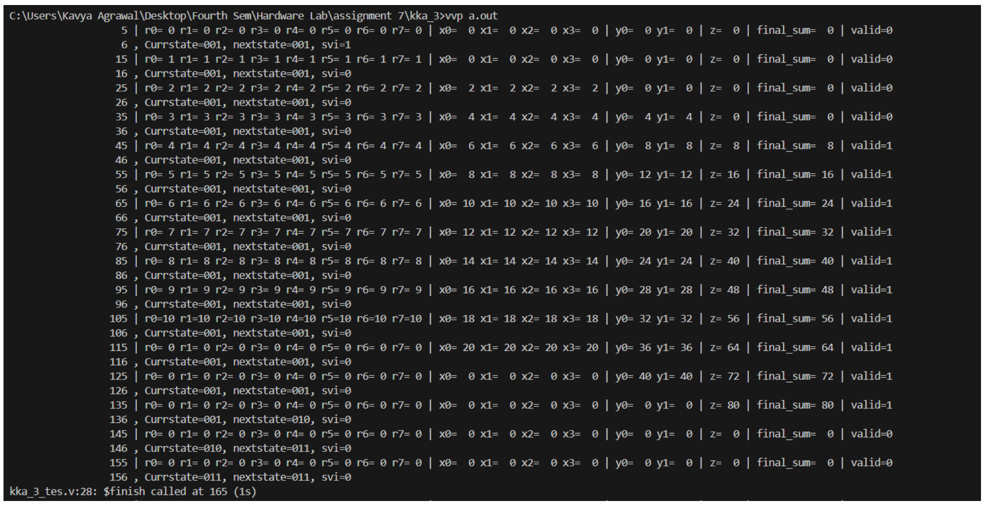


Figure 12: Simulation Waveform for Q1



```
C:\Users\Kavya Agrawal\Desktop\Fourth Sem\Hardware Lab\assignment 7\kka_3>vvp a.out
                   5 | r0= 0 r1= 0 r2= 0 r3= 0 r4= 0 r5= 0 r6= 0 r7= 0 | x0=  0 x1=  0 x2=  0 x3=  0 | y0=  0 y1=  0 | z=  0 | final_sum=  0 | valid=0
                   6 , Currstate=001, nextstate=001, svi=1
                  15 | r0= 1 r1= 1 r2= 1 r3= 1 r4= 1 r5= 1 r6= 1 r7= 1 | x0=  0 x1=  0 x2=  0 x3=  0 | y0=  0 y1=  0 | z=  0 | final_sum=  0 | valid=0
                  16 , Currstate=001, nextstate=001, svi=0
                  25 | r0= 2 r1= 2 r2= 2 r3= 2 r4= 2 r5= 2 r6= 2 r7= 2 | x0=  2 x1=  2 x2=  2 x3=  2 | y0=  0 y1=  0 | z=  0 | final_sum=  0 | valid=0
                  26 , Currstate=001, nextstate=001, svi=0
                  35 | r0= 3 r1= 3 r2= 3 r3= 3 r4= 3 r5= 3 r6= 3 r7= 3 | x0=  4 x1=  4 x2=  4 x3=  4 | y0=  4 y1=  4 | z=  0 | final_sum=  0 | valid=0
                  36 , Currstate=001, nextstate=001, svi=0
                  45 | r0= 4 r1= 4 r2= 4 r3= 4 r4= 4 r5= 4 r6= 4 r7= 4 | x0=  6 x1=  6 x2=  6 x3=  6 | y0=  8 y1=  8 | z=  8 | final_sum=  8 | valid=1
                  46 , Currstate=001, nextstate=001, svi=0
                  55 | r0= 5 r1= 5 r2= 5 r3= 5 r4= 5 r5= 5 r6= 5 r7= 5 | x0=  8 x1=  8 x2=  8 x3=  8 | y0= 12 y1= 12 | z= 16 | final_sum= 16 | valid=1
                  56 , Currstate=001, nextstate=001, svi=0
                  65 | r0= 6 r1= 6 r2= 6 r3= 6 r4= 6 r5= 6 r6= 6 r7= 6 | x0= 10 x1= 10 x2= 10 x3= 10 | y0= 16 y1= 16 | z= 24 | final_sum= 24 | valid=1
                  66 , Currstate=001, nextstate=001, svi=0
                  75 | r0= 7 r1= 7 r2= 7 r3= 7 r4= 7 r5= 7 r6= 7 r7= 7 | x0= 12 x1= 12 x2= 12 x3= 12 | y0= 20 y1= 20 | z= 32 | final_sum= 32 | valid=1
                  76 , Currstate=001, nextstate=001, svi=0
                  85 | r0= 8 r1= 8 r2= 8 r3= 8 r4= 8 r5= 8 r6= 8 r7= 8 | x0= 14 x1= 14 x2= 14 x3= 14 | y0= 24 y1= 24 | z= 40 | final_sum= 40 | valid=1
                  86 , Currstate=001, nextstate=001, svi=0
                  95 | r0= 9 r1= 9 r2= 9 r3= 9 r4= 9 r5= 9 r6= 9 r7= 9 | x0= 16 x1= 16 x2= 16 x3= 16 | y0= 28 y1= 28 | z= 48 | final_sum= 48 | valid=1
                  96 , Currstate=001, nextstate=001, svi=0
                 105 | r0=10 r1=10 r2=10 r3=10 r4=10 r5=10 r6=10 r7=10 | x0= 18 x1= 18 x2= 18 x3= 18 | y0= 32 y1= 32 | z= 56 | final_sum= 56 | valid=1
                 106 , Currstate=001, nextstate=001, svi=0
                 115 | r0= 0 r1= 0 r2= 0 r3= 0 r4= 0 r5= 0 r6= 0 r7= 0 | x0= 20 x1= 20 x2= 20 x3= 20 | y0= 36 y1= 36 | z= 64 | final_sum= 64 | valid=1
                 116 , Currstate=001, nextstate=001, svi=0
                 125 | r0= 0 r1= 0 r2= 0 r3= 0 r4= 0 r5= 0 r6= 0 r7= 0 | x0=  0 x1=  0 x2=  0 x3=  0 | y0= 40 y1= 40 | z= 72 | final_sum= 72 | valid=1
                 126 , Currstate=001, nextstate=001, svi=0
                 135 | r0= 0 r1= 0 r2= 0 r3= 0 r4= 0 r5= 0 r6= 0 r7= 0 | x0=  0 x1=  0 x2=  0 x3=  0 | y0=  0 y1=  0 | z= 80 | final_sum= 80 | valid=1
                 136 , Currstate=001, nextstate=010, svi=0
                 145 | r0= 0 r1= 0 r2= 0 r3= 0 r4= 0 r5= 0 r6= 0 r7= 0 | x0=  0 x1=  0 x2=  0 x3=  0 | y0=  0 y1=  0 | z=  0 | final_sum=  0 | valid=0
                 146 , Currstate=010, nextstate=011, svi=0
                 155 | r0= 0 r1= 0 r2= 0 r3= 0 r4= 0 r5= 0 r6= 0 r7= 0 | x0=  0 x1=  0 x2=  0 x3=  0 | y0=  0 y1=  0 | z=  0 | final_sum=  0 | valid=0
                 156 , Currstate=011, nextstate=011, svi=0
kka_3_tes.v:28: $finish called at 165 (1s)
```

Figure 13: Simulation Waveform for Q3

5 FPGA Demonstration

For FPGA implementation: - Inputs were read sequentially using a data valid signal. - Outputs were displayed using an out valid signal.

I/O pin mapping for FPGA implementation:

Table 1: I/O Pin Mapping for FPGA Implementation

Signal Name	Port Name	FPGA Pin
clk	clk	P1
reset	reset	P2
in[3:0]	in	P3-P6
out[6:0]	out	P7-P13
valid	valid	P14
done	done	P15

6 Conclusion

This assignment explored three implementations of long chain addition: 1. Iterative datapath with one adder. 2. Iterative datapath with four adders. 3. Pipelined datapath.

Through simulation and FPGA demonstration, we analyzed trade-offs between area, latency, and throughput. The pipelined design achieved the highest throughput but required more resources. The iterative design with one adder was resource-efficient but slower.

This work highlights the importance of architectural choices in digital system design and their impact on performance metrics.

7 Observations

The three different architectures for long chain addition each demonstrate unique trade-offs between performance and hardware resource usage. Key observations from simulation and synthesis are:

- **Iterative Datapath with One Adder:** Utilizes minimal area and is highly resource-efficient. However, the latency is highest among all due to its sequential nature. It is best suited for low-speed, resource-constrained applications.
- **Iterative Datapath with Four Adders:** Offers a balance between speed and resource usage. The partial parallelism reduces latency significantly compared to the single-adder design while avoiding the complexity of pipelining.
- **Pipelined Datapath:** Achieves the best throughput with one output per clock cycle after initial latency. It is the fastest in processing multiple input sets but consumes the most area due to additional pipeline registers and combinational logic.

Table 2: Comparison of All Three Architectures

Architecture	**Area Usage**	**Latency**	**Clock Period**	**Throughput**
Iterative (1 Adder)	Low	8 cycles	High	1 result / 8 cycles
Iterative (4 Adders)	Medium	2 cycles	Medium	1 result / cycle
Pipelined	High	4 cycles (initial)	Low	1 result / cycle