Lab 4: Datapath and Control Path

Written By: Joel Bailey

Date: October 16th, 2020

ECE 524/Lab

Lab 4: Datapath and Control Path

Table of Contents

I. Introduction

A finite state machine is designed to control a system that reads 32 8-bit numbers, accumulates the sum, and outputs the average of the 32 numbers. A clear data path and control path is outlined in blue and green, respectively, of the following schematic:

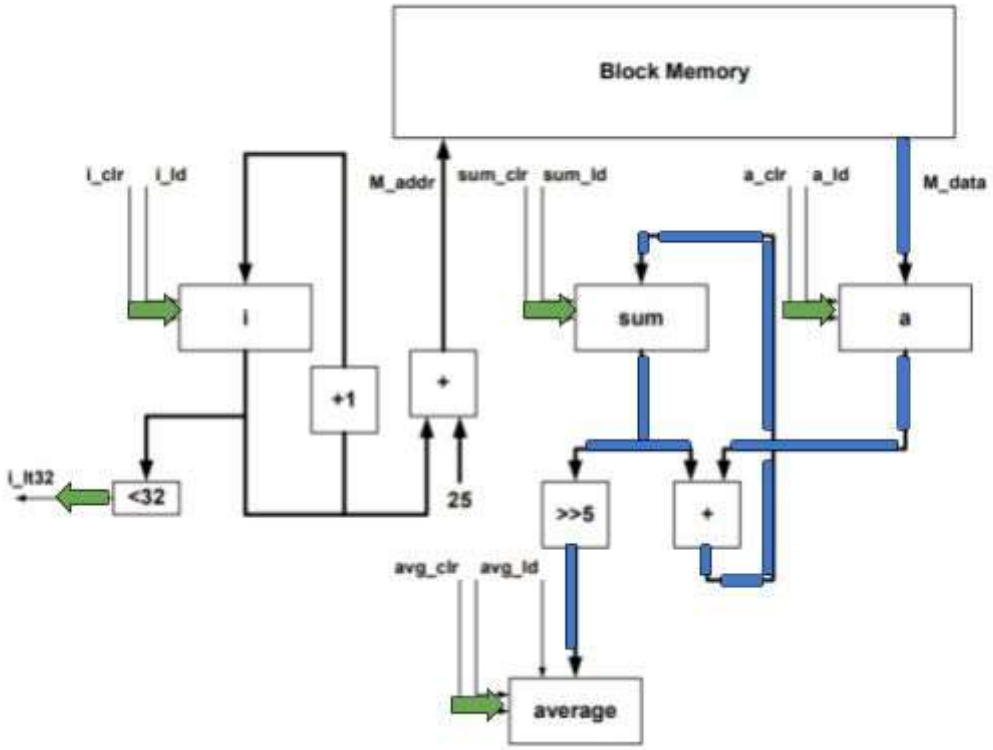

Figure 1: High level schematic with datapath highlighted in blue and the control path in green.

The values to be averaged are stored in single-port block memory of appropriate width and depth, which is set to readonly.

Explicit tasks for this lab are:

Task 1: Implement the complete design, including data path and control path. Simulate the design for correct functionality. Include simulations of the working design.

Task 2: Modify the design to indicate overflow. Prove the logic works.

Task 3: Modify the design to average 50 8-bit numbers. Prove the logic works.

II. Procedure

The design began with defining the states the machine needed to be in to accumulate and output an average. The follow diagram is a 3-state implementation:

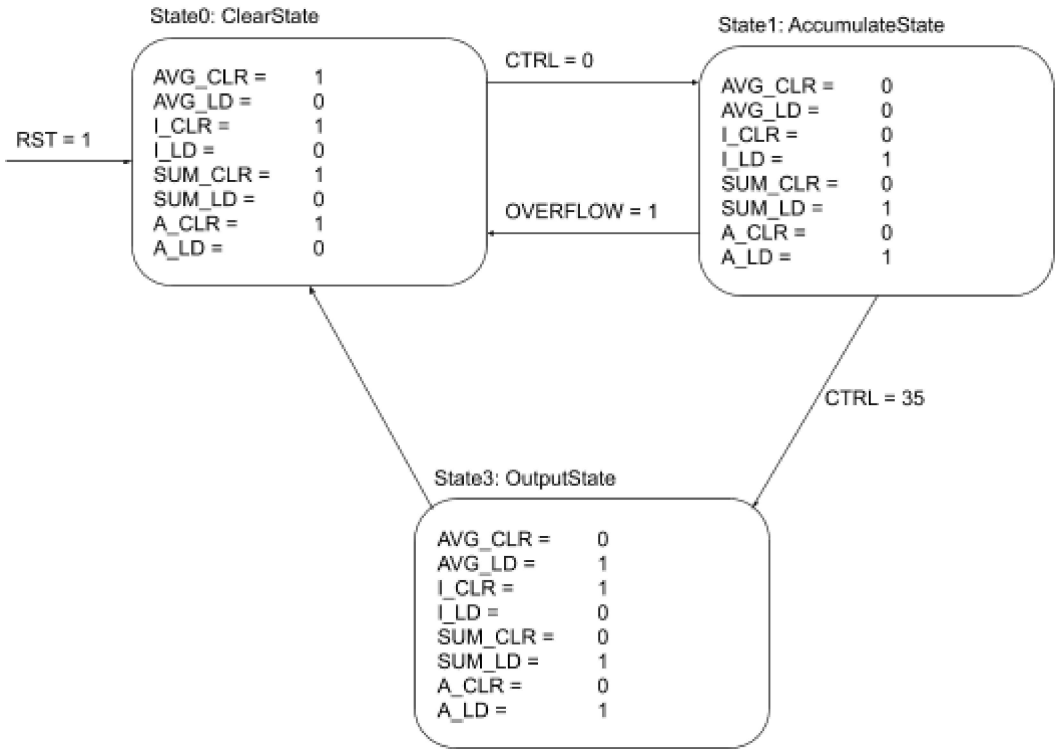

Figure 2: State diagram for controller. Note: overflow is included in FSM diagram, but not in task 1 design.

Where an asynchronous reset sets the state machine back to its clear state, which in turn, synchronously clears all registers along the datapath. Once the FSM detects the counter has been reset, it enables the registers by moving to the AccumulateState. In the AccumulateState, the control counter increments itself and the address, which is read into the BRAM. The A register then reads BRAM, and in the next clock, sends the value to be accumulated. Once all 32 numbers have been accumulated and moved through the datapath, the FSM then moves to OutputState. The only condition to move from output state to clear state is that outputstate has been enabled. This allows the average to be set to the output, then all registers reset for the next average.

Each register is edge triggered and is designed such that CLR has priority over LD, and if neither is enabled, the registers hold value.

To accomplish the overflow detection. I included 2 variables in my SUM.vhd file where the accumulated value is sign extended by a bit and then added to the incoming data. If the 2 MSB's do NOT equal, then that indicates the original value has overflowed as its MSB has flipped. I then tied the overflow detection to the FSM as a control that brings the machine back to its clear state, due to the error.

The hierarchy of the design is as such:

```
Testbench
|
|____TOP: top level connections
      |
      |____CONTROLLER: FSM control
      |____COUNTER: Address and ctrl counter
      |____A: Holds output value from RAM
      |____SUM: Accumulates the sum of BRAM values
      |____AVERAGE: Computes the average of 32 or 50 values
      |____RAM: top level RAM connections
           |____BRAM
```

The BRAM was instantiated within its own submodule opposed to the top level of the design to keep the organization of the top level design uniform. For conceptualization, the high level elaborated design can be found below:

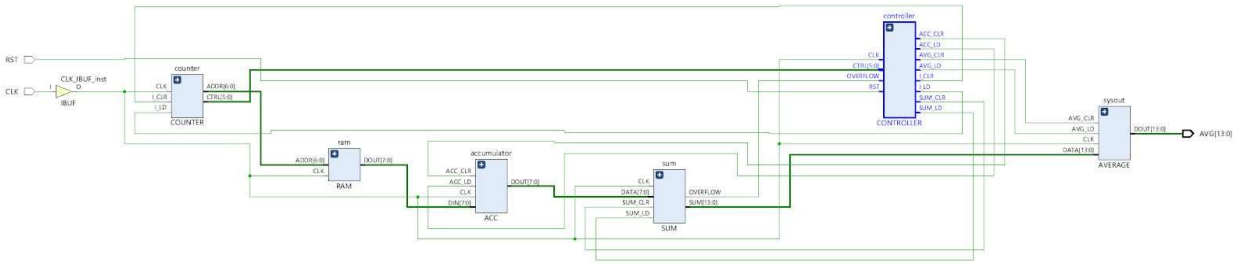

Figure 3: Elaborated design of implementation

III. Testing Strategy

The testing strategy revolved around ensuring my data path and control path were synchronized with each other. With a 1 cycle delay in the BRAM and having each stage clocked, I made a spreadsheet to track the desired functionality where the input values ranged from 1...32:

TIME	CTRL	ADDRESS	RAMOUT	AOUT	SUMOUT	SYSOUT	STATE
CP+	0	24	0	0	0	0	ClearState
CP+	0	24	0	0	0	0	AccumulateState
CP+	1	25	0	0	0	0	AccumulateState
CP+	2	26	1	0	0	0	AccumulateState
CP+	3	27	2	1	0	0	AccumulateState
CP+	4	28	3	2	1	0	AccumulateState
CP+	5	29	4	3	3	0	AccumulateState
CP+	6	30	5	4	6	0	AccumulateState
CP+	7	31	6	5	10	0	AccumulateState
CP+	8	32	7	6	15	0	AccumulateState
CP+	9	33	8	7	21	0	AccumulateState
CP+	10	34	9	8	28	0	AccumulateState
CP+	11	35	10	9	36	0	AccumulateState
CP+	12	36	11	10	45	0	AccumulateState
CP+	13	37	12	11	55	0	AccumulateState
CP+	14	38	13	12	66	0	AccumulateState
CP+	15	39	14	13	78	0	AccumulateState
CP+	16	40	15	14	91	0	AccumulateState
CP+	17	41	16	15	105	0	AccumulateState
CP+	18	42	17	16	120	0	AccumulateState
CP+	19	43	18	17	136	0	AccumulateState
CP+	20	44	19	18	153	0	AccumulateState
CP+	21	45	20	19	171	0	AccumulateState
CP+	22	46	21	20	190	0	AccumulateState
CP+	23	47	22	21	210	0	AccumulateState
CP+	24	48	23	22	231	0	AccumulateState
CP+	25	49	24	23	253	0	AccumulateState
CP+	26	50	25	24	276	0	AccumulateState
CP+	27	51	26	25	300	0	AccumulateState
CP+	28	52	27	26	325	0	AccumulateState
CP+	29	53	28	27	351	0	AccumulateState
CP+	30	54	29	28	378	0	AccumulateState
CP+	31	55	30	29	406	0	AccumulateState

CP+	32	56	31	30	435	0	AccumulateState
CP+	33	57	32	31	465	0	AccumulateState
CP+	34	58	0	32	496	0	AccumulateState
CP+	35	59	0	0	528	0	OutputState
CP+	0	24	0	0	528	16	ClearState

Figure 4: Timing / control analysis of 32 8-bit numbers

With values from 1...32, the average is 528 / 32 ~= 16 and I would be able to change my state when the counter reached 34.

For task 2, the overflow signal was added and hooked to the FSM. A test vector was run by reducing the bit number of the output, forcing an overflow. Finally task 3 was tested using the same analysis done for the timing on task 1, except the state change to OutputState needed to occur at 54.

IV. Results (Data)

Results have been split into tasks for waveform display.

Results Overview:

Task 1 -

Input: 1...32
Sum: 528
Truncated Avg: 16

Task 2 -

Input: others=>127
Sum: FAILS due to overflow
Avg: FAILS due to overflow

Task 3 -

Input: others=>127
Sum: 6350
Avg: 127

Task 1 Waveforms

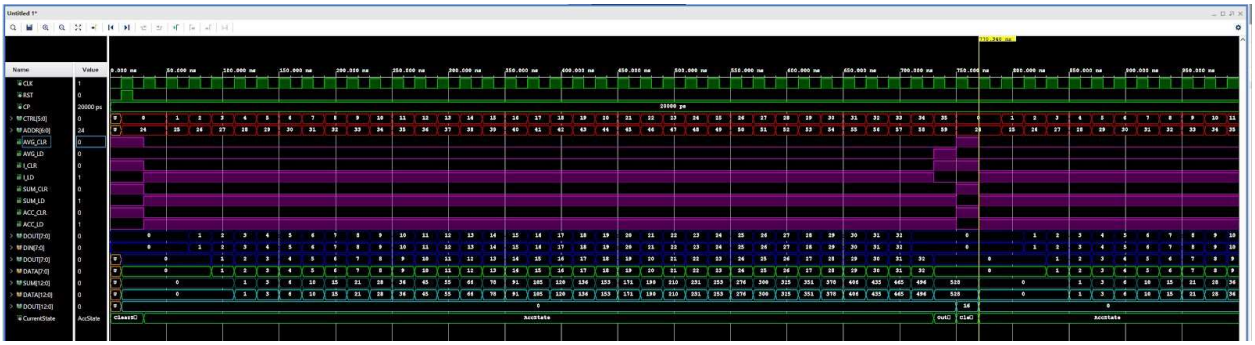

Figure 5: Task 1 full simulation

Figure 6: Task 1 final values and output of average

Task 2 Waveform

Figure 7: Overflow detection firing and resetting FSM

Task 3 Waveform

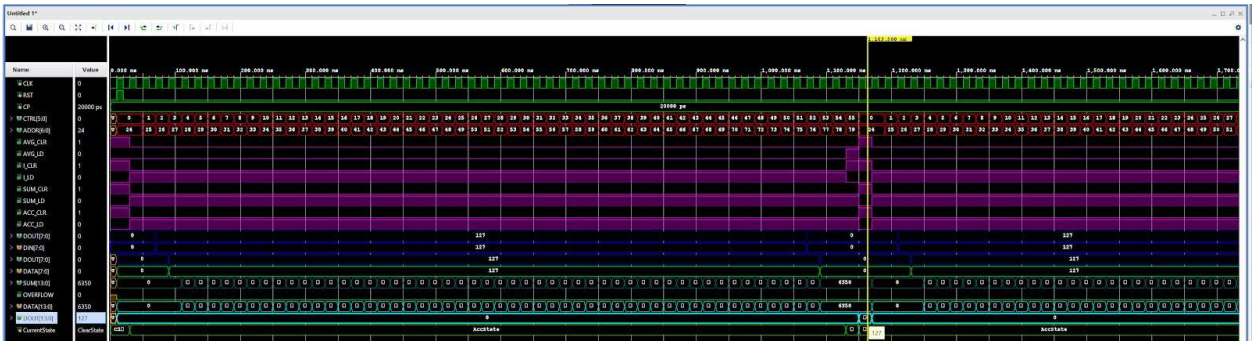

Figure 8: Task 3 full simulation view

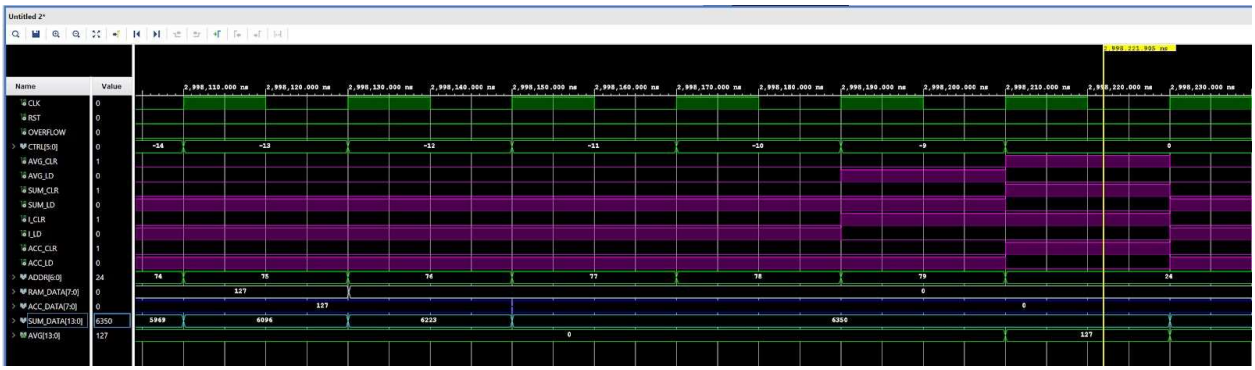

Figure 9: Task 3 final values and output of average

V. Analysis / Conclusion

In the original implementation of my design, the 5 bit control counter was found to be too short in width to allow the data to fully propagate through to the output. I was receiving an output about 3 stages too soon. To remedy the issue, I added a bit to the counter and did a timing analysis spreadsheet, which can be found in the procedure section of this report. With the timing analysis, I was able to visualize the delays caused by each stage of my design and I was better able to adjust my control signal and allow the proper output time to propagate through the design. When implementing designs with multiple stages and components with inherent delay, I will be applying the same timing analysis before the components are coded, which will help with the confusion I ran into when running my first few test benches.

Additional Questions:

1. Estimation of resources:

    a. LUTs: 9 - FSM, Counter, Address, Average, Sum, Shift, A, Overflow, Reset
    b. FFs: 51 - (2) NS/CS, (13) Counter, (8) A, (14) Sum, (14) Avg
    c. Memory: 1 - BRAM
    d. DSP: 0

2. State diagrams can be found in the procedure section of this report.

3. Actual resources:

    a. LUTs: 28
    b. FF's: 53
    c. Memory: 1
    d. DSP: 0

    The discrepancy in the LUT count is due to my exclusion of comparison expressions in my original estimations. I was short 2 FF's which were 2 latches I created for overflow.