# Lab 5: Averager

Written By: Joel Bailey
Date: October 27th, 2020
ECE 524/Lab

# Lab 5: Averager

## Table of Contents

## I. Introduction

In this lab, a circuit is designed that will take an input stream of data and outputs the average of the last m-number of inputs. A high level schematic of the design can be seen below:

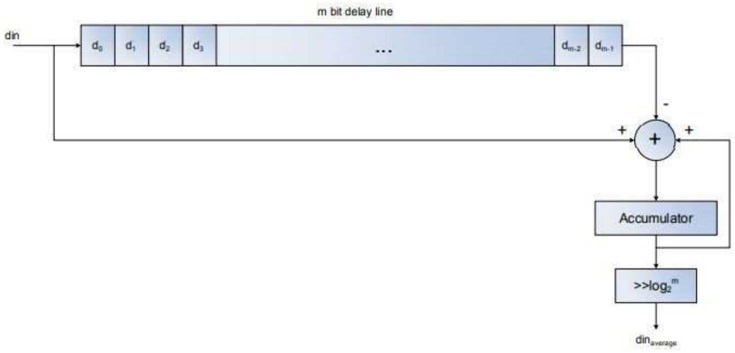


Figure 1: Data stream averager

As the data is read into the circuit, it is accumulated, and upon a m-number of clock cycles later, the original value is removed from the accumulation and a new value is added. The running average is kept throughout. As the final action of the circuit is to shift the accumulated value to the right, m is restricted to $m = 2^n, \text{ where } n >= 0$.

Task 1: A proof of why the circuit averages m samples of data, is given in the following example:

$$Avg = \frac{1}{m}\sum_{i=0}^{m} d_i \text{ , where division by m is realized by srl } log2(m)$$

Let: $m = 8$ and $d = [8, 8, 8, 8, 8, 8, 8, 8, 16]$

$$srl\ log2(m) == srl\ log2(8) == srl\ 3 === \frac{1}{8}$$

Anticipated averages from calculations:

$$d_{0->7} : (8 * 8) / 8 = 8$$

$$d_{1->8} : ((8 * 7) + 16) / 8 = 9$$

Average based upon output of circuit:

| $n$ | $d_n$ | $acc = acc + d_n - d_{n-m}$ | $\frac{1}{m}(acc)$ |
|---|---|---|---|
| 0 | 8 | $8 = 0 + 8 - 0$ | 1 |
| 1 | 8 | $16 = 8 + 8 - 0$ | 2 |
| 2 | 8 | $24 = 16 + 8 - 0$ | 3 |

| | | | |
|---|---|---|---|
| 3 | 8 | $32 = 24 + 8 - 0$ | 4 |
| 4 | 8 | $40 = 32 + 8 - 0$ | 5 |
| 5 | 8 | $48 = 40 + 8 - 0$ | 6 |
| 6 | 8 | $56 = 48 + 8 - 0$ | 7 |
| 7 | 8 | $64 = 56 + 8 - 0$ | 8 |
| 8 | 16 | $72 = 64 + 16 - 8$ | 9 |

When n = 7, the accumulator has accumulated 8 values, therefore the output is the average of the last 8 values. Once n = 8, the value at n = 0 is removed from the accumulator, while the value at n = 8 is added, creating a running average.

## II. Procedure

This design began with the analysis seen in the introduction of this report. Once a working algorithm was developed, the first component created was the shift register. The shift register component is modeled from the example given on page 84 of Vivado Design Suite User Guide: Synthesis. Some modifications were made to the model to allow a variable width of the input data. This was accomplished by creating an array of STD_LOGIC_VECTORS in place of the single-bit registers used in the example. The following code snippet is the declaration of the array:

```
type SRL_ARRAY is array (DEPTH - 1 downto 0) of STD_LOGIC_VECTOR (WIDTH - 1 downto 0 );
signal SHREG : SRL_ARRAY;
```

Next, the accumulator was designed as a simple memory component that adds the input of the circuit to itself and removes the output of the shift in a single expression: $Acc = Acc + Data_{in} - Shift_{out}$, where acc is an integer signal that is cast to an STD_LOGIC_VECTOR on output.

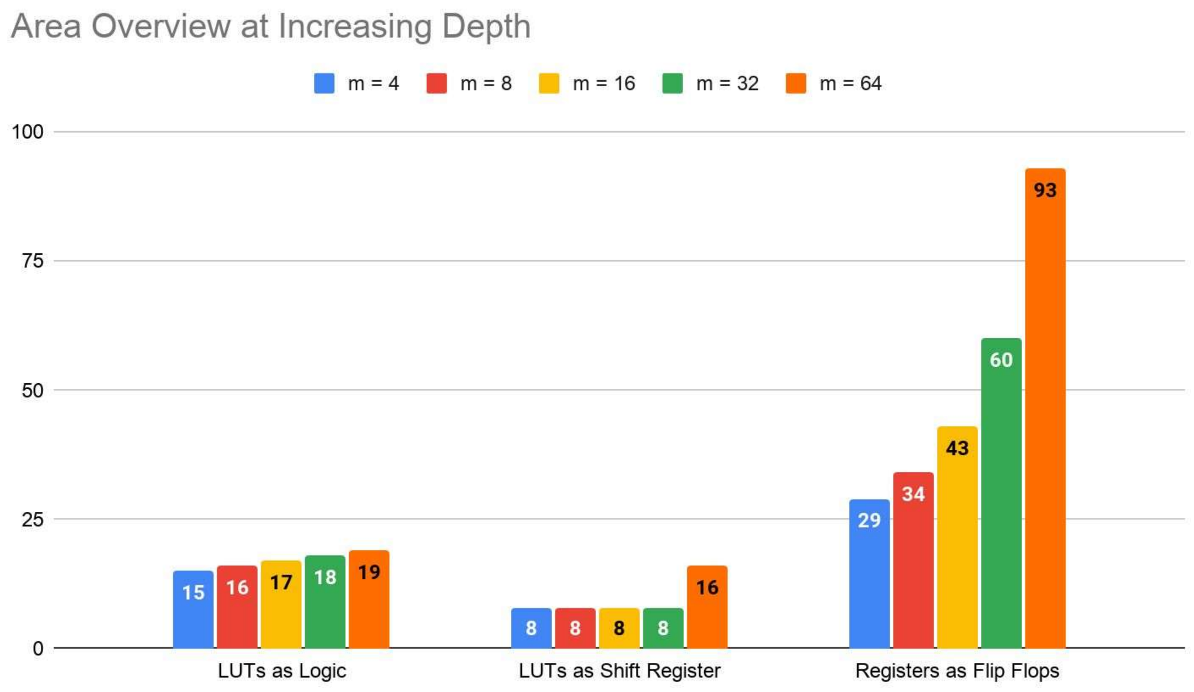
The accumulator was then tied to the average out component. This component was combinational with an input tied to the accumulator and the output as the only ports. Within this component 2 variables are declared in the process to accomplish the logarithmic and shift functions. For the log2 function, a function was developed that inputs an integer and calls the IEEE.MATH_REAL.ALL functions ceil, log2, and real, then outputs the truncated, rounded up, logarithm of the input. When the variable is created in the process, it is automatically assigned the value returned from the function. The second variable is used to hold the shift value and was created to make the type casting easier to read.

The random number generator was then developed to test the circuit, and is covered in more detail in the testing strategy section of this report.

A block ram was then inferred using the example on page 134 of the Vivado Synthesis guide, to act as the delay in place of the SRL. This example was modified to remove extra control signals that were not needed for the design. An automatic addresser was also added to the BRAM that cycled through each address of the memory on each clock cycle.

These components were instantiated and interface with each other through a top level that serves no other function.

Task 4: Area usage for my design of the SRL implementation can be found in the following graph where each bar is an increase in depth by 2^n from values 4 to 64.



## III. Testing Strategy

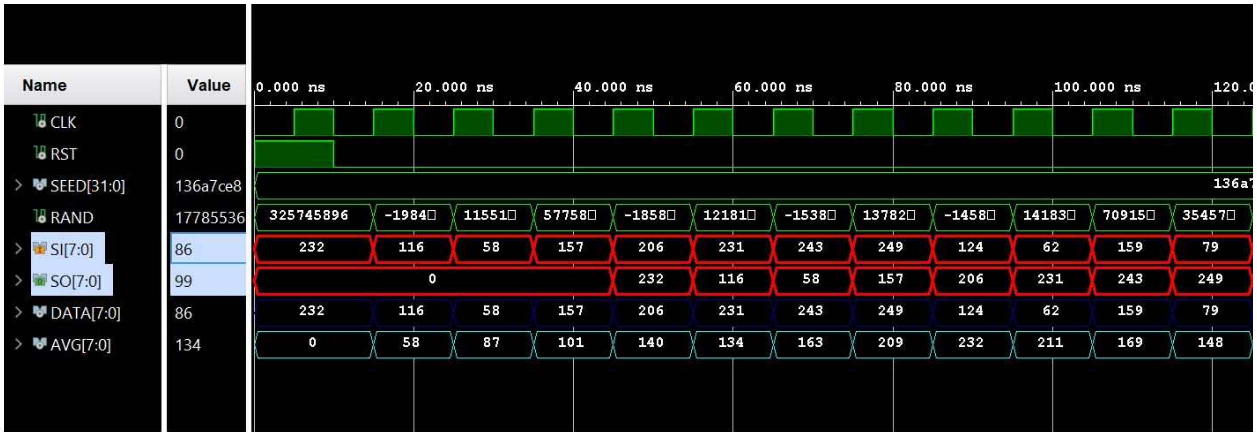
To test both the SRL and BRAM versions of the design, a pseudo random number generator (RNG) was designed to provide the inputs. The RNG component was designed as a 32-bit shift register where bits 3, 7 , 10, and 31 are fed to an exclusive-or gate, which outputs to the MSB of the shift register.

The RNG element is instantiated in the test bench, along with the unit under test. A seed is given to the RNG component during the reset cycle of the test bench. The seed used for this design was 325,745,896. The original seed selected was too small for a 32-bit number, and resulted in a string of 0’s at the beginning of the RNG cycles, therefore a number was selected that would have 1’s at higher bit positions. As the RNG component was 32-bits and the remaining components were of variable width and depth, the RNG output was modulus 2^WIDTH at the input of the circuit to ensure only valid unsigned numbers were used in the test.

## IV. Results (Data)

For the results, a width of 8-bits and a depth (m) of 4 was used for ease of analysis. Colors are as follows: Red indicates the input and output of the delay component, dark blue is the input of the Dn of the accumulator, and light blue is the output, which is the average of the last m values.

SRL Implementation:

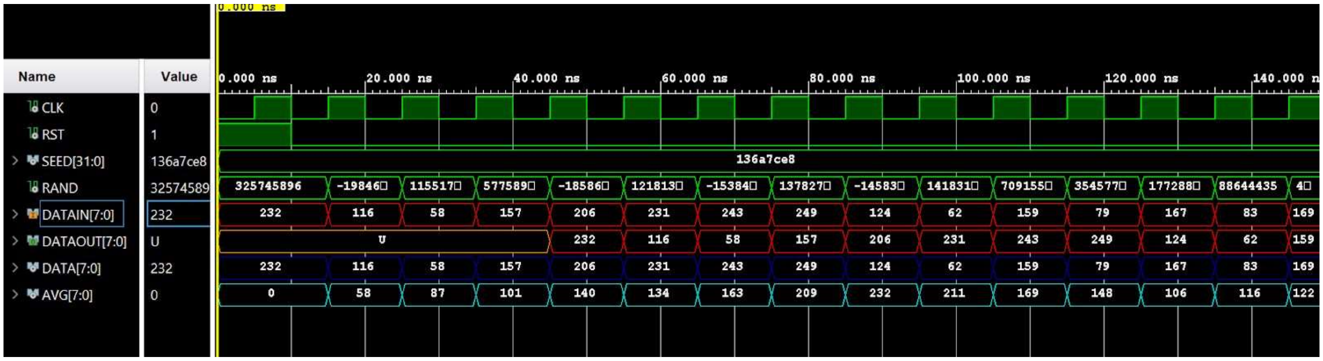


| Name | Value | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CLK | 0 | | | | | | | | | | | | |
| RST | 0 | | | | | | | | | | | | |
| SEED[31:0] | 136a7ce8 | 136a7 | | | | | | | | | | | |
| RAND | 17785536 | 325745896 | -1984□ | 11551□ | 57758□ | -1858□ | 12181□ | -1538□ | 13782□ | -1458□ | 14183□ | 70915□ | 35457□ |
| SI[7:0] | 86 | 232 | 116 | 58 | 157 | 206 | 231 | 243 | 249 | 124 | 62 | 159 | 79 |
| SO[7:0] | 99 | 0 | | | | 232 | 116 | 58 | 157 | 206 | 231 | 243 | 249 |
| DATA[7:0] | 86 | 232 | 116 | 58 | 157 | 206 | 231 | 243 | 249 | 124 | 62 | 159 | 79 |
| AVG[7:0] | 134 | 0 | 58 | 87 | 101 | 140 | 134 | 163 | 209 | 232 | 211 | 169 | 148 |

| $n$ | $d_n$ | $acc = acc + d_n - d_{n-m}$ | $\frac{1}{m}(acc)$ |
|---|---|---|---|
| 0 | 232 | $232 = 0 + 232 - 0$ | 58 |
| 1 | 116 | $348 = 232 + 116 - 0$ | 87 |
| 2 | 58 | $406 = 348 + 58 - 0$ | 101.5 |
| 3 | 157 | $563 = 406 + 157 - 0$ | 140.75 |

| | | | |
|---|---|---|---|
| 4 | 206 | $537 = 563 + 206 - 232$ | 134.25 |

BRAM Implementation:



Note, the output table for the SRL implementation is valid also for the BRAM as the same seed was used in both.

## V. Analysis / Conclusion

When analyzing the usage reports during this lab, I realized that my BRAM inference wasn't actually causing BRAM usage in synthesis. I thought it may have to do with the automatic adresser in the component, so I removed it and added back the control signals I removed when designing it, and it still did not infer the BRAM in the synthesis report. My next test was to increase the sizes of the width and depth to see if the larger size forced a BRAM usage. The new depth was set to 1,000 and the utilization reported a BRAM usage as seen below:

```
+-------------------+------+-------+-----------+-------+
|     Site Type     | Used | Fixed | Available | Util% |
+-------------------+------+-------+-----------+-------+
| Block RAM Tile    |  0.5 |     0 |       140 |  0.36 |
|   RAMB36/FIFO*    |    0 |     0 |       140 |  0.00 |
|   RAMB18          |    1 |     0 |       280 |  0.36 |
|     RAMB18E1 only |    1 |       |           |       |
+-------------------+------+-------+-----------+-------+
```

Another item of note is the delay between the input and the average is a clock pulse that would need to be accounted for in a full implementation of the design.