Kyle Keislar

ECE524/L FPGA/ASIC Design and Optimization Using VHDL Lab



Lab 5

Averaging

11/10/2020

## 

## Introduction & Problem Statements

In this experiment a reprogrammable averaging circuit is realized to compute the running average on a stream of data. This average can handle inputs of varying bit width and produces the average of the last specified elements. A high level block diagram of the averaging circuit is shown below in Figure 5.0.A. For this experiment, a custom function was written to compute the base 2 logarithm of an accumulated value. This served as a functional division for the experiment and can be seen in the appendix as part of the *Lab\_5\_Top.vhd*.

Fig 5.0.A Averaging Circuit Block Diagram:



The theoretical behavior of the circuit is described below in table 5.1 and considers a 10 element stream of data with a five element running average. The tm, column represents the number of clock pulses since the data stream began (tm=0). Acc\_prev & Acc\_cur represent the accumulator registered values before and after the addition of the most recent data element. Finally the Avg, column represents the averaged output from the log2m function.

Table 5.1 Averaging Circuit Example:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| tm | din | din(0:m-1) | Acc\_prev | Acc\_cur | Avg (w=4, m=5, log\_2\_m=2) | Notes |
| - | ‘0' | {0,0,0,0,0} | ‘0' | ‘0' | 0000 | 0, Average Begins as 0 |
| 0 | 0000 | {0,0,0,0,0} | 0000 | 0000 | 0000 |  |
| 1 | 0001 | {1,0,0,0,0} | 0000 | 0001 | 0000 |  |
| 2 | 0002 | {2,1,0,0,0} | 0001 | 0011 | 0000 | Results truncated <4 |
| 3 | 0003 | {3,2,1,0,0} | 0011 | 0110 | 0001 |  |
| 4 | 0004 | {4,3,2,1,0} | 0110 | 1010 | 0010 |  |
| 5 | 0005 | {5,4,3,2,1} | 1010 | 1111 | 0011 | After 5 elements, Average m=5 is computed (15/5=3) |
| 6 | 0000 | {0,5,4,3,2} | 1111 | 1110 | 0011 | Delayed din |
| 7 | 0000 | {0,0,5,4,3} | 1110 | 1100 | 0011 |  |
| 8 | 0000 | {0,0,0,5,4} | 1100 | 1001 | 0010 |  |
| 9 | 0000 | {0,0,0,0,5} | 1001 | 0101 | 0001 |  |
| 10 | 0000 | {0,0,0,0,0} | 0101 | 0000 | 0000 | Running Average |

Two forms of the averaging circuit were considered in this experiment. The first utilized only Shift Register Look-Up-Tables (SRL) to synthesize the m-bit delay line. This delay was also synthesized using Block RAM resources of the FPGA. Components of the Xilnix Synthesis Guide v2019.2 were used to infer these components. Both styles of this design are explored and their simulated behavior can be seen below in the Results section.

Finally, a Linear Feedback Shift Register is designed to provide pseudo random repeating data sequence of variable length to satisfy the data requirements of the averaging circuit. This LFSR design is tested and synthesized as a separate component and its simulated behavior can be seen below in the Results section.

The code for all of the synthesized design of each of these files can be found in the Appendix below.

For this experiment, an implemented design of this circuit using the Zedboard was not realized.

## Procedure

**Task 1**: Prove mathematically why this circuit averages ‘m’ samples of input data stream.

**Task 2**: Use the proper coding style to infer SRL component in your design to implement the delay line.

The synthesized design files for Task 2 can be seen in the Appendix below as items: A.5 *Lab\_5\_Top\_SRL.vhd*, A.6 *SRL\_gen.vhd* and A.9 *Lab5\_Acc.vhd*. A test bench for this task can be seen in the Appendix as item A.1 *tb\_Lab\_5\_AVG.vhd*.

**Task 3**: Use the proper coding style to infer Block RAM in your design to implement the delay line.

The synthesized design files for Task 2 can be seen in the Appendix below as items: A.7 *Lab\_5\_Top\_BRAM.vhd*, A.8 *BRAM\_gen.vhd* and A.9 *Lab5\_Acc.vhd*. The previous test bench for Task 2, A.1 *tb\_Lab\_5\_AVG.vhd*, was utilized for this task

**Task 4**: Present a graph to show area usage when ‘m’ varies from 4 to 64 (in powers of 2) for task 2.

This graph can be seen in the Results section below as Figure 5.3 Running Average Length vs Area Graph for SRL Implementation.

**Task 5**: Design a programmable random number generator of length ‘w’ bits to supply data samples tothe averager circuit.

The LFSR design file can be seen below in the Appendix as item: A.10 *Lab\_5\_LFSR.vhd*. The testbench of this task can be seen in the Appendix as item A.2 *tb\_Lab5\_LFSR.vhd*.

**Task 6** : Provide testbench for your average, random number generator, logarithm function, and youroverall design to prove the correct functionality.

## Results (Data):

## *Waveforms graphed from generated & simulated data:*

Fig 5.1: SRL Implementation Waveforms

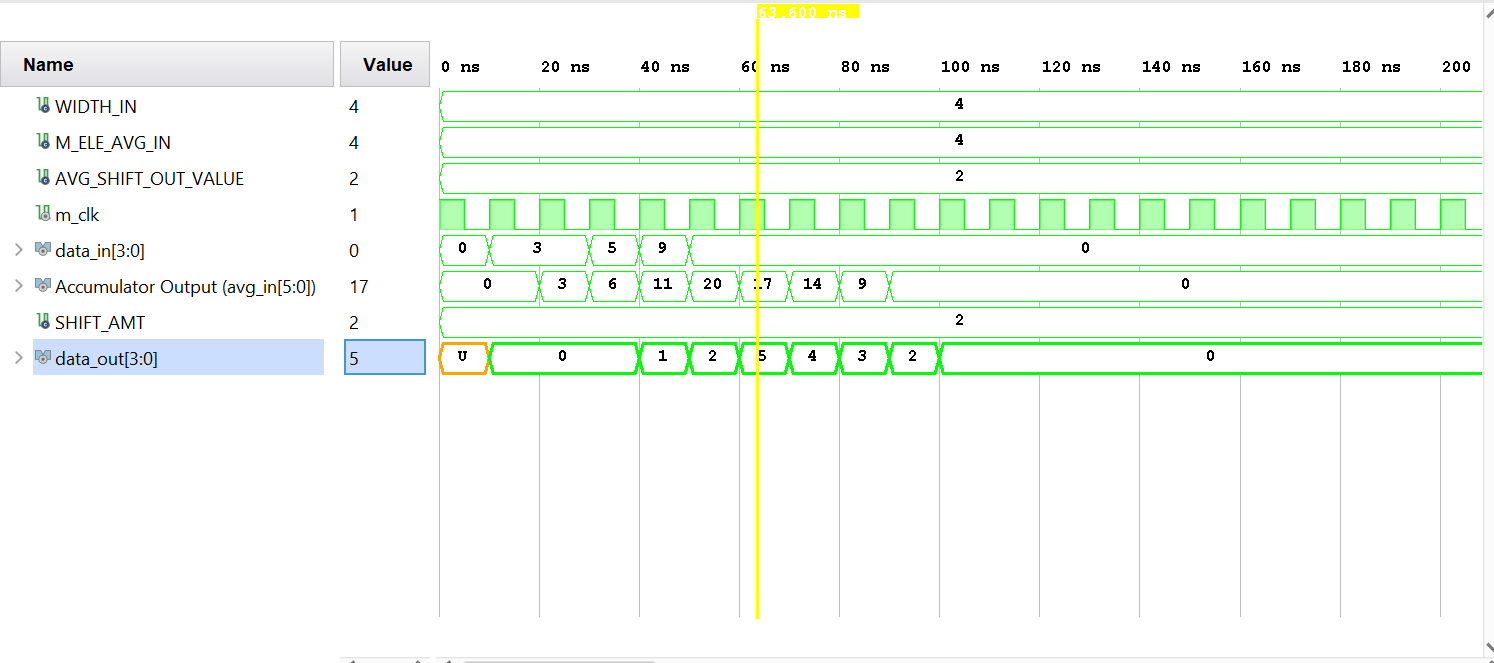


Fig 5.2: BRAM Implementation Waveforms

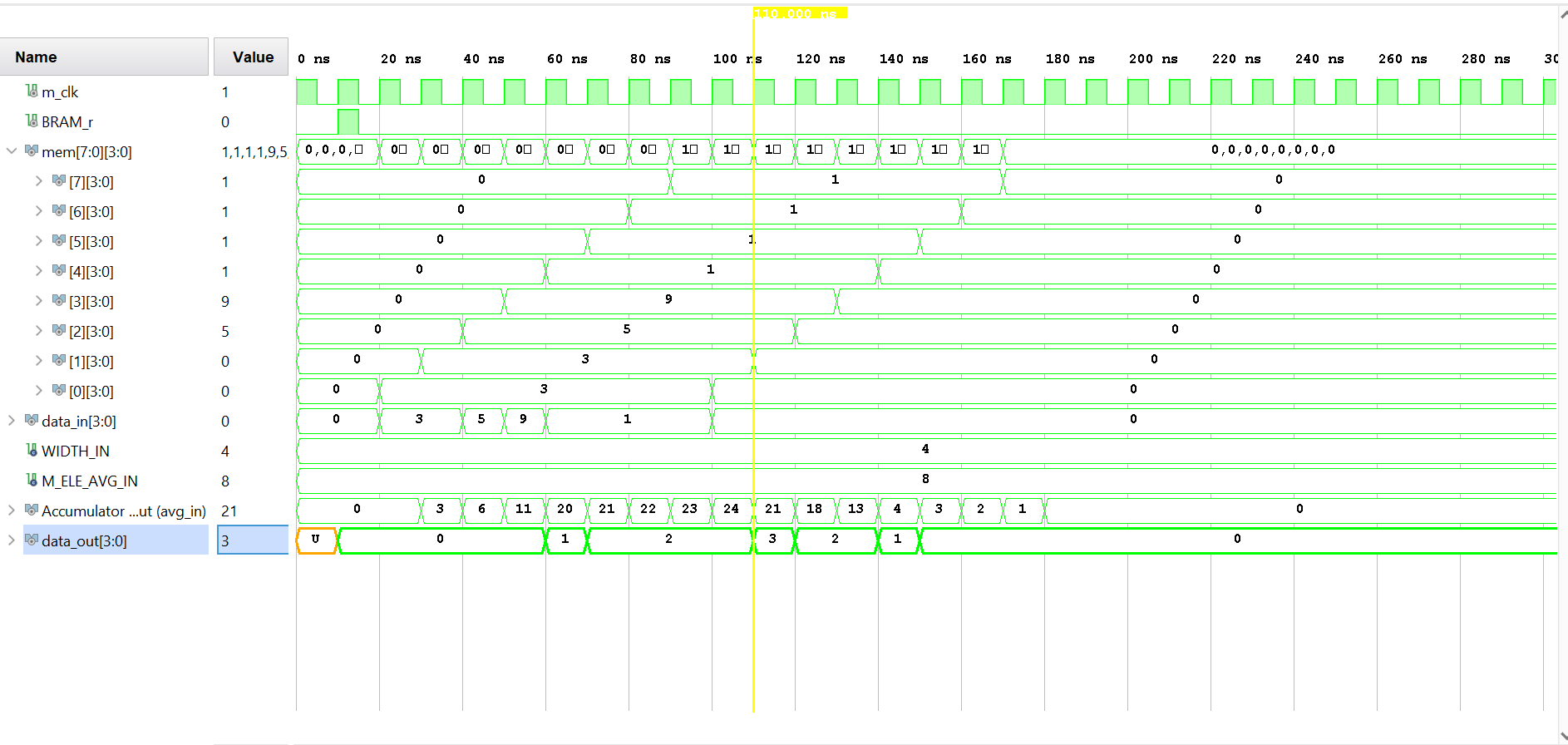


Fig 5.3: Running Average Length vs Area Graph for SRL Implementation

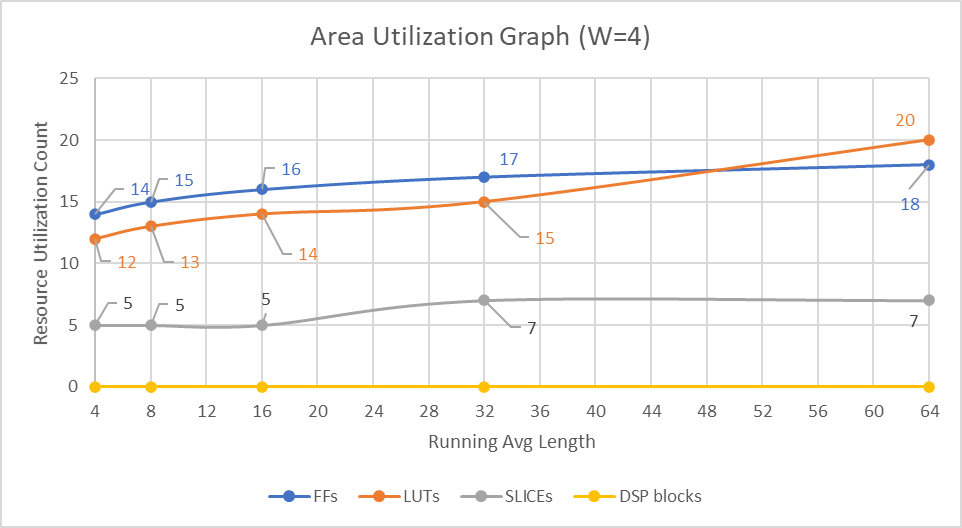


Fig 5.4: LFSR output for BRAM Implementation

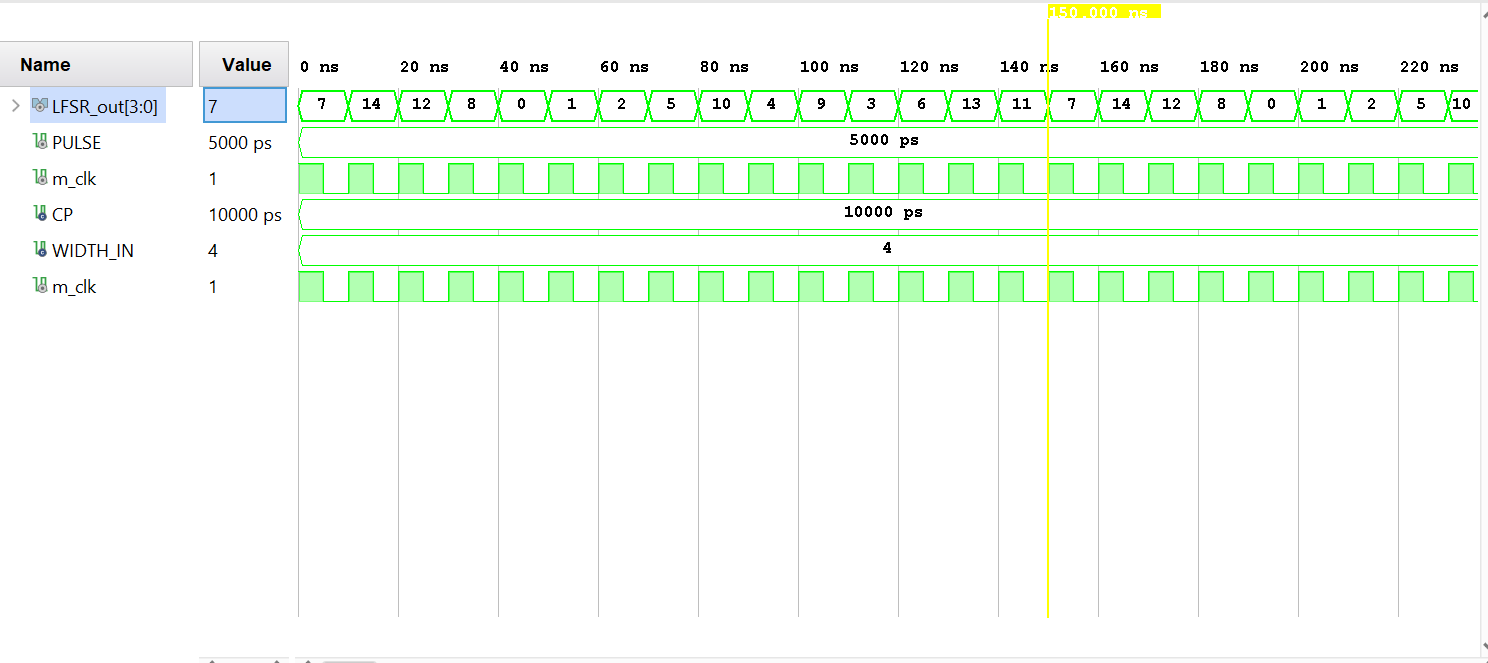
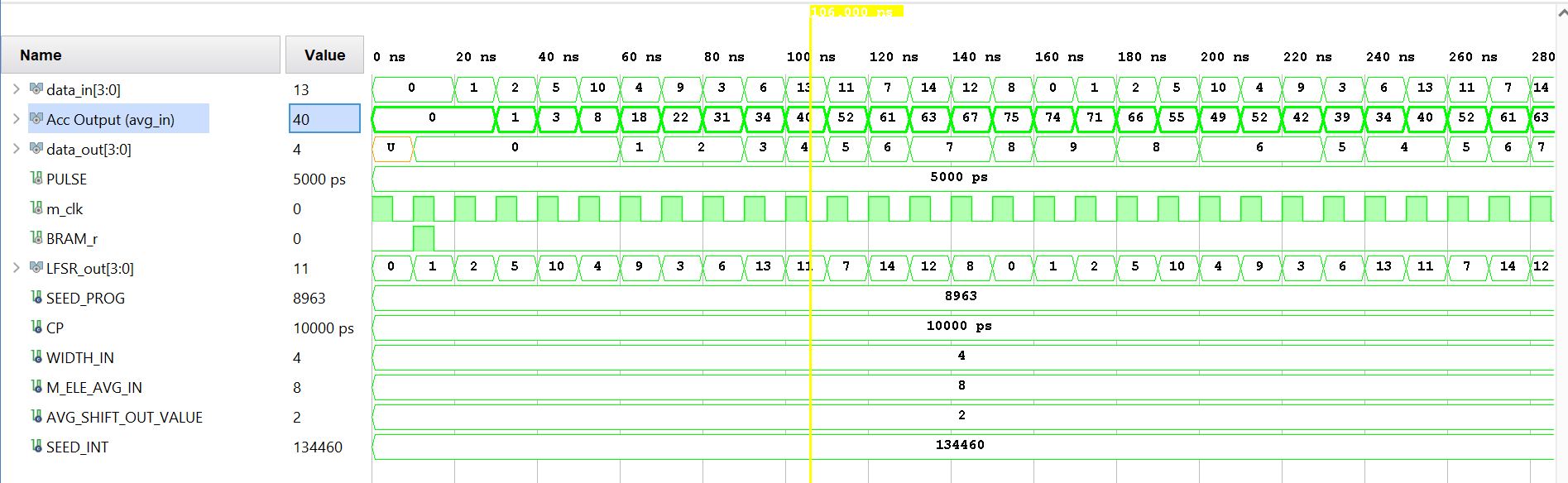


Fig 5.5: BRAM Implementation Waveforms with LFSR generated Test Vectors



## Analysis:

The simulated designs for this experiment successfully demonstrated the functionality of the running average circuit for both the SRL & BRAM styles. Their behavior is congruent with the theoretical example elaborated in Table 5.0 above with some unaccounted latency.

For the SRL synthesized design stipulated in Task 2, the behavioral waveforms shown in Fig 5.1 above confirm its functionality as a running avg circuit. The waveform shows the first 4 input data elements from the test bench as static vectors 3, 3, 5, 9. The accumulated sum of these elements is 20 as shown on the clock pulse following the last element (t=50 ns). The output average can be seen on the next clock pulse as 5. This is the same as the theoretical value (20/4=5). For the SRL synthesized design, the latency is equal to m+2 clock pulses where m represents the number of elements to be averaged.

For the BRAM synthesized design stipulated in Task 3, the behavioral waveforms shown in Fig 5.2 and 5.5 above confirm its functionality as a running avg circuit. The waveform shows the first 8 input data elements from the test bench as static vectors 3, 3, 5, 9, 1, 1, 1, 1. The accumulated sum of these elements is 24 as shown on the clock pulse following the last element (t=100 ns). Similar to the SRL design, the output average can be seen on the next clock pulse as 3. This is the same as the theoretical value for the 8 element running average (24/8=3).

A BRAM implementation was also utilized for the complete design with LFSR as shown in Fig 5.5. The waveform shows the first 8 input data elements from the LFSR as standard logic vectors 0, 1, 2, 5, 10, 4, 9, 3, 6. The accumulated sum of these elements is 40. The computed average shown on data\_out two clock pulses alter at (t=110ns) is 5. This is congruent with the theoretical value for the 8 element running average (40/8=5).

For the BRAM synthesized design, the latency is also equal to m+2 clock pulses where m represents the number of elements to be averaged. While no resource utilization for the BRAM component was included in this report, it should be noted that RAM primitive size must increase as the input width or number of elements increase. No specific primitive was instantiated for this specific design.

For the LFSR testing structure stipulated in Task 5, the behavioral waveforms shown in Fig 5.4 above confirm its functionality a serial random number generator. The waveform shows generated input data elements or a randomly repeating sequence. This sequence starts and begins with 7 at 0ns and ends with 14 at 140ns. Because the design is parameterized, the number of randomly sequenced inputs generated will always be equal to 2^w-1. For this waveform the design’s input width was set to 4 resulting in 15 data inputs. The LFSR has no latency as its output is entirely combinational. In addition, a privately generated seed value is defined to the LFSR design file to seed the initial values.

This circuit’s design utilization for the SRL style was explored by comparing the area of resource utilizations for varying running average lengths. The graph in Fig 5.3 shows the synthesis behavior as the running average length is increased for the design described in Task 2. As expected, the FPGA resources required for this design increase with the number of elements averaged. Each powered SLICE is able to process additional inputs even if all of the logic cells are not utilized. This facet of FPGA functionality explains why both 32 and 64 element averages consume the same amount of slices.

APPENDIX

1. *tb\_Lab\_5\_AVG.vhd*

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**entity** tb\_Lab5\_AVG **is**

-- Port ( );

**end** tb\_Lab5\_AVG**;**

**architecture** Behavioral **of** tb\_Lab5\_AVG **is**

--Constant Definitions

**CONSTANT** CP**:** TIME **:=** 10ns**;**--clock period

**CONSTANT** WIDTH\_IN**:** integer **:=**4**;**--data width

**CONSTANT** M\_ELE\_AVG\_IN**:** integer **:=** 4**;**--Number of Elements to Average

**CONSTANT** AVG\_SHIFT\_OUT\_VALUE**:** integer **:=** 2**;**--manually computed log base 2 of # of elements ot be average

-- == log\_2(M\_ELE\_AVG\_IN)

--Used to check result

--Signal Definitions

**signal** data\_in**,**data\_out**:** std\_logic\_vector**(**WIDTH\_IN**-**1 **downto** 0**)** **:=** **(Others** **=>**'0'**);**

**SIGNAL** PULSE**:** TIME**:=**CP**\***0.5**;**--pulse width

**signal** m\_clk**:** std\_logic **:=** '0'**;** --master clock, filter reset

**COMPONENT** Lab\_5\_top **is**

**Generic(** WIDTH**:** INTEGER **:=**4**;**

M\_ELE**:** INTEGER **:=**5**);**

**Port(** data\_in**:** **in** STD\_LOGIC\_VECTOR**(**WIDTH**-**1 **downto** 0**);**

data\_out**:** **out** STD\_LOGIC\_VECTOR**(**WIDTH**-**1 **downto** 0**);**

clk**:** **in** STD\_LOGIC**);**

**end** **COMPONENT;**

**begin**

uut**:** Lab\_5\_top

**Generic** **Map(** WIDTH**=>**WIDTH\_IN**,**

M\_ELE**=>** M\_ELE\_AVG\_IN**)**

**Port** **Map** **(** data\_in**=>**data\_in**,**

data\_out**=>**data\_out**,**

clk**=>**m\_clk**);**

--Clock Processs

m\_clock**:process** --free running clock

**begin**

m\_clk **<=** '1'**;**

**wait** **for** PULSE **;**

m\_clk **<=** '0'**;**

**wait** **for** PULSE**;**

**end** **process;**

--Test Vectors

**process**

**begin**

**wait** **for** CP**;**

data\_in**<=**"0011"**;**

**wait** **for** CP**;**

data\_in**<=**"0011"**;**

**wait** **for** CP**;**

data\_in**<=**"0101"**;**

**wait** **for** CP**;**

data\_in**<=**"1001"**;**

**wait** **for** CP**;**

data\_in**<=**"0000"**;**

**wait;**

**end** **process;**

**end** Behavioral**;**

1. *tb\_Lab5\_LFSR.vhd*

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**use** ieee**.**numeric\_std**.all;**

**use** ieee**.**math\_real**.all;**

**use** ieee**.**std\_logic\_unsigned**.all;**

**entity** tb\_Lab5\_LFSR **is**

**Generic(**SEED\_PROG**:** INTEGER **:=**12346**);** --seed programming value, chosen by user to generate LFSR seed

**end** tb\_Lab5\_LFSR**;**

**architecture** Behavioral **of** tb\_Lab5\_LFSR **is**

--Constant Definitions

**CONSTANT** CP**:** TIME **:=** 10ns**;**--clock period

**CONSTANT** WIDTH\_IN**:** integer **:=**4**;**--data width

**CONSTANT** SEED\_INT**:** integer **:=** **(**2**\*\***WIDTH\_IN**-**1**)\*(**SEED\_PROG**+**1**);**--genearte random seed form user input

--Signal Definitions

**signal** LFSR\_out**:** std\_logic\_vector**(**WIDTH\_IN**-**1 **downto** 0**)** **:=** **(Others** **=>**'0'**);**--counts indicie to wriet to LFSR

**SIGNAL** PULSE**:** TIME**:=**CP**\***0.5**;**--pulse width

**signal** m\_clk**:** std\_logic **:=** '0'**;** --master clock, LFSR reset, output bit

**COMPONENT** Lab5\_LFSR **is**

**Generic(**W**:** INTEGER**:=**4**;** --Number of bits to shift

SEED**:** INTEGER**);** --reseed output based on top level

**Port** **(** clk**:** **in** STD\_LOGIC**;**--clock enable, clock , reset

dout**:** **out** STD\_LOGIC\_VECTOR**(**W**-**1 **downto** 0**));**

**end** **COMPONENT;**

**begin**

LFSR\_uut**:** Lab5\_LFSR **Generic** **Map(** W**=>**WIDTH\_IN**,** SEED**=>**SEED\_INT**)**

**Port** **Map** **(** clk**=>**m\_clk**,**dout**=>**LFSR\_out**);** --select 1 bit form each generated register

--Clock Processs

m\_clock**:process** --free running clock

**begin**

m\_clk **<=** '1'**;**

**wait** **for** PULSE **;**

m\_clk **<=** '0'**;**

**wait** **for** PULSE**;**

**end** **process;**

**end** Behavioral**;**

1. *tb\_Lab\_5\_Log\_2.vhd*

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**use** ieee**.**numeric\_std**.all;**

**use** ieee**.**std\_logic\_unsigned**.all;**

**entity** tb\_Lab\_5\_Log\_2 **is**

-- Port ( );

**end** tb\_Lab\_5\_Log\_2**;**

**architecture** Behavioral **of** tb\_Lab\_5\_Log\_2 **is**

--signal defintions

**Signal** m\_ele**:** INTEGER **:=** 1**;**--number of elements to average(log operand)

**signal** result**,**exp\_count**:** integer**:=**0**;** --log 2 result

--CLOCK process signal definitiions

**CONSTANT** CP**:** TIME **:=** 10ns**;**--clock period

**SIGNAL** PULSE**:** TIME**:=**CP**\***0.5**;**--pulse width

**signal** m\_clk**,**hold**:** std\_logic **:=** '0'**;** --master clock, hold initial values

--logrithmic function Definition

--calculates shift value for output din\_avg

**function** Log\_2 **(**

elements**:** INTEGER --static at compile time,

--Varied to demonstrate function behavior

--M\_ELE is not a dynamic element in design

**)**

**return** Integer **is**

**variable** AVG\_WIDTH**:** INTEGER **:=** elements**;**--width of input

**variable** log\_cnt**:** std\_logic\_vector**(**AVG\_WIDTH**-**1 **downto** 0**)** **:=** std\_logic\_vector**(to\_unsigned(**AVG\_WIDTH**,**AVG\_WIDTH**));**

**variable** temp**:** std\_logic**:=**'0'**;**

**variable** shift\_val **:** INTEGER **:=** 1**;**--shfit value acumulator

**begin**

**for** i **in** 0 **to** AVG\_WIDTH**-**1 **loop**

temp**:=**log\_cnt**(**0**);**

**if(**temp**=**'1'**)** **then**

shift\_val**:=**i**;**

**end** **if;**

log\_cnt**:=**'0'**&**log\_cnt**(**AVG\_WIDTH**-**1 **downto** 1**);**

**end** **loop;**

**return** shift\_val**;**

**end** Log\_2**;**

**begin**

--Clock Processs

m\_clock**:process** --free running clock

**begin**

m\_clk **<=** '1'**;**

**wait** **for** PULSE **;**

m\_clk **<=** '0'**;**

**wait** **for** PULSE**;**

**end** **process;**

--Clock Processs

init\_hold**:process** --fholds initilized values to show 0 behavior

**begin**

hold **<=** '1'**;**

**wait** **for** CP **;**

hold **<=** '0'**;**

**wait;**

**end** **process;**

TVs**:process(**m\_clk**)**

**begin**

**if(**hold**=**'0'**)** **then**

**if(**m\_clk'**event** **AND** m\_clk**=**'1'**)** **then**

m\_ele**<=**2**\*\***exp\_count**;**

exp\_count **<=** exp\_count**+**1**;**

result**<=**Log\_2**(**m\_ele**);**

**end** **if;**

**end** **if;**

**end** **process;**

**end** Behavioral**;**

1. *tb\_Lab5\_Top.vhd*

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**use** ieee**.**numeric\_std**.all;**

**use** ieee**.**math\_real**.all;**

**use** ieee**.**std\_logic\_unsigned**.all;**

**entity** tb\_Lab5\_top **is**

**Generic(**SEED\_PROG**:** INTEGER **:=**8963**);** --seed programming value, chosen by user to generate LFSR.

--values from 1 to 2^32-1

**end** tb\_Lab5\_top**;**

**architecture** Behavioral **of** tb\_Lab5\_top **is**

--Constant Definitions

**CONSTANT** CP**:** TIME **:=** 10ns**;**--clock period

**CONSTANT** WIDTH\_IN**:** integer **:=**4**;**--data width

**CONSTANT** M\_ELE\_AVG\_IN**:** integer **:=** 8**;**--Number of Elements to Average

**CONSTANT** AVG\_SHIFT\_OUT\_VALUE**:** integer **:=** 2**;**--manually computed log base 2 of # of elements ot be average

-- == log\_2(M\_ELE\_AVG\_IN)

--Used to check result

--Signal Definitions

**signal** data\_in**,**data\_out**:** std\_logic\_vector**(**WIDTH\_IN**-**1 **downto** 0**)** **:=** **(Others** **=>**'0'**);**

**SIGNAL** PULSE**:** TIME**:=**CP**\***0.5**;**--pulse width

**signal** m\_clk**,**BRAM\_r**:** std\_logic **:=** '0'**;** --master clock, BRAM reset

--Top level Definitions

**COMPONENT** Lab\_5\_top **is**

**Generic(** WIDTH**:** INTEGER **:=**4**;**

M\_ELE**:** INTEGER **:=**5**);**

**Port(** data\_in**:** **in** STD\_LOGIC\_VECTOR**(**WIDTH**-**1 **downto** 0**);**

data\_out**:** **out** STD\_LOGIC\_VECTOR**(**WIDTH**-**1 **downto** 0**);**

clk**:** **in** STD\_LOGIC**;**

reset**:** **in** STD\_LOGIC**);**

**end** **COMPONENT;**

--LFSR Testbench Signal Definitions

**signal** LFSR\_out**:** std\_logic\_vector**(**WIDTH\_IN**-**1 **downto** 0**)** **:=** **(Others** **=>**'0'**);**--counts indicie to wriet to LFSR

**CONSTANT** SEED\_INT**:** integer **:=** **(**2**\*\***WIDTH\_IN**-**1**)\*(**SEED\_PROG**+**1**);**--genearte random seed form user input

**COMPONENT** Lab5\_LFSR **is**

**Generic(**W**:** INTEGER**:=**4**;** --Number of bits to shift

SEED**:** INTEGER**);** --reseed output based on top level

**Port** **(** clk**:** **in** STD\_LOGIC**;**--clock enable, clock , reset

dout**:** **out** STD\_LOGIC\_VECTOR**(**W**-**1 **downto** 0**));**

**end** **COMPONENT;**

**begin**

--Averging design top level instnation

uut**:** Lab\_5\_top

**Generic** **Map(** WIDTH**=>**WIDTH\_IN**,**

M\_ELE**=>** M\_ELE\_AVG\_IN**)**

**Port** **Map** **(** data\_in**=>**data\_in**,**

data\_out**=>**data\_out**,**

clk**=>**m\_clk**,**

reset**=>**BRAM\_r**);**

--LFSR instantion

LFSR\_uut**:** Lab5\_LFSR **Generic** **Map(** W**=>**WIDTH\_IN**,** SEED**=>**SEED\_INT**)**

**Port** **Map** **(** clk**=>**m\_clk**,**dout**=>**LFSR\_out**);** --select 1 bit form each generated register

--Clock Processs

m\_clock**:process** --free running clock

**begin**

m\_clk **<=** '1'**;**

**wait** **for** PULSE **;**

m\_clk **<=** '0'**;**

**wait** **for** PULSE**;**

**end** **process;**

bram\_init**:** **process**

**begin**

**wait** **for** CP**;**

BRAM\_R**<=**'1'**;**

**wait** **for** PULSE**;**

BRAM\_R**<=**'0'**;**

**wait;**

**end** **process;**

--Test Vectors

**process**

**begin**

**wait** **for** CP**;**

data\_in**<=**LFSR\_out**;**

**end** **process;**

**end** Behavioral**;**

1. *Lab\_5\_Top\_SRL.vhd*

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**use** ieee**.**numeric\_std**.all;**

**use** ieee**.**std\_logic\_unsigned**.all;**

**entity** Lab\_5\_top **is**

**Generic(** WIDTH**:** INTEGER **:=**3**;**

M\_ELE**:** INTEGER **:=**5**);**

**Port(** data\_in**:** **in** STD\_LOGIC\_VECTOR**(**WIDTH**-**1 **downto** 0**);**

data\_out**:** **out** STD\_LOGIC\_VECTOR**(**WIDTH**-**1 **downto** 0**);**

clk**:** **in** STD\_LOGIC**);**

**end** Lab\_5\_top**;**

**architecture** Behavioral **of** Lab\_5\_top **is**

--Constant definitions

**CONSTANT** ACC\_W\_DEC**:** INTEGER **:=** **(**2**\*\***WIDTH**-**1**)\***M\_ELE**;** --total decimal value accumulator must be able to hold

--signal definitiions

**signal** avg\_out**,**ele\_sub\_out **:** std\_logic\_vector**(**WIDTH**-**1 **downto** 0**)** **:=** **(Others** **=>**'0'**);** --input to Acc, subtracts previous elements to calculate running average

**signal** avg\_shift\_Val**:** integer **:=**0**;**--caluclates average shift value form log\_2() funciton

--SRL Delay Line Definition

**COMPONENT** SRL\_or\_BRAM\_gen **is**

**Generic(** W**:** INTEGER **:=**3**;**

M**:** INTEGER **:=**5**);**

**Port(** d\_in**:** **in** STD\_LOGIC\_VECTOR**(**W**-**1 **downto** 0**);**

d\_out**:** **out** STD\_LOGIC\_VECTOR**(**W**-**1 **downto** 0**);**

clk**:** **in** STD\_LOGIC**);**

**end** **COMPONENT;**

--Accumulator Definition

**COMPONENT** Lab5\_Acc **is**

**Generic(** W**:** INTEGER **:=**3**;**

ACC\_W**:** INTEGER **:=**5**);**

**Port(** d\_in**:** **in** STD\_LOGIC\_VECTOR**(**W**-**1 **downto** 0**);**

ele\_sub\_in**:** **in** STD\_LOGIC\_VECTOR**(**W**-**1 **downto** 0**);**

acc\_output**:** **out** STD\_LOGIC\_VECTOR**(**ACC\_W**-**1 **downto** 0**);**

clk**:** **in** STD\_LOGIC**);**

**end** **COMPONENT;**

--logrithmic function Definition

--calculates shift value for output din\_avg

**function** Log\_2 **(**

elements**:** INTEGER **:=** M\_ELE

**)**

**return** Integer **is**

**variable** AVG\_WIDTH**:** INTEGER **:=** elements**;**--width of input

**variable** log\_cnt**:** std\_logic\_vector**(**AVG\_WIDTH**-**1 **downto** 0**)** **:=** std\_logic\_vector**(to\_unsigned(**AVG\_WIDTH**,**AVG\_WIDTH**));**

**variable** temp**:** std\_logic**:=**'0'**;**

**variable** shift\_val **:** INTEGER **:=** 1**;**--shfit value acumulator

**begin**

**for** i **in** 0 **to** AVG\_WIDTH**-**1 **loop**

temp**:=**log\_cnt**(**0**);**

**if(**temp**=**'1'**)** **then**

shift\_val**:=**i**;**

**end** **if;**

log\_cnt**:=**'0'**&**log\_cnt**(**AVG\_WIDTH**-**1 **downto** 1**);**

**end** **loop;**

**return** shift\_val**;**

**end** Log\_2**;**

**CONSTANT** SHIFT\_AMT**:** integer **:=**Log\_2**(**M\_ELE**);**--amount ot shift acc output by to obtain truncated avg

**CONSTANT** ACC\_WIDTH**:** INTEGER **:=** Log\_2**(**ACC\_W\_DEC**)+**1**;** --width of accumulator line (must be at least 1 bit larger than input to hold up to 2 elements)

**signal** avg\_in**:** STD\_LOGIC\_VECTOR**(**ACC\_WIDTH**-**1 **downto** 0**):=** **(Others** **=>**'0'**);**--input to log function from accumulator module

**begin**

--SRL instantiation

SRL\_uut**:** SRL\_or\_BRAM\_gen

**Generic** **Map(** W**=>**WIDTH**,**

M**=>**M\_ELE**)**

**Port** **Map** **(** d\_in**=>**data\_in**,**

d\_out**=>**ele\_sub\_out**,**

clk**=>**clk**);**

--Accumulator instantation

ACC\_uut**:** Lab5\_Acc

**Generic** **Map(** W**=>**WIDTH**,**

ACC\_W**=>**ACC\_WIDTH**)**

**Port** **Map** **(** d\_in**=>**data\_in**,**

ele\_sub\_in**=>**ele\_sub\_out**,**

acc\_output**=>**avg\_in**,**

clk**=>**clk**);**

SR\_avg**:process(**clk**)**--shifts proess right, generates synchronous output

**begin**

**if(**clk'**event** **AND** clk**=**'1'**)** **then**

**for** i **in** 0 **to** WIDTH**-**1 **loop**

data\_out**(**i**)<=**avg\_in**(**i**+**SHIFT\_AMT**);**

**end** **loop;**

**end** **if;**

**end** **process;**

**end** Behavioral**;**

1. *SRL\_gen.vhd*

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**entity** SRL\_or\_BRAM\_gen **is**

**Generic(** W**:** INTEGER **:=**3**;**

M**:** INTEGER **:=**5**);**

**Port(** d\_in**:** **in** STD\_LOGIC\_VECTOR**(**W**-**1 **downto** 0**);**

d\_out**:** **out** STD\_LOGIC\_VECTOR**(**W**-**1 **downto** 0**);**

clk**:** **in** STD\_LOGIC**);**

**end** SRL\_or\_BRAM\_gen**;**

**architecture** Behavioral **of** SRL\_or\_BRAM\_gen **is**

**signal** temp\_in**,**temp\_out**:** STD\_LOGIC\_VECTOR**(**W**-**1 **downto** 0**)** **:=** **(Others** **=>**'0'**);**

--SRL definition

**type** M\_bit\_SRL **is** **array** **(**M**-**1 **downto** 0**)** **of** STD\_LOGIC\_VECTOR**(**W**-**1 **downto** 0**);**

**signal** SRL\_gen **:** M\_bit\_SRL **:=** **(others=>(others** **=>**'0'**));** --Initilize SRL\_gen to 0

--Infer SRL with appropriate attributes

**attribute** shreg\_extract **:** string**;**

**attribute** srl\_style **:** string**;**

**attribute** shreg\_extract **of** SRL\_gen**:** **signal** **is** "yes"**;** --use SRL components

**attribute** srl\_style **of** SRL\_gen**:** **signal** **is** "srl"**;** --Only Infer M-bit SRL,

--do not register inputs/outputs outside of SRL

**begin**

--clock data in, d out

**process(**clk**)**

**begin**

**if(**clk'**event** **AND** clk**=**'1'**)** **then**

SRL\_gen**(**0**)<=**d\_in**;**

d\_out**<=**SRL\_gen**(**M**-**1**);**

**for** i **in** 0 **to** M**-**2 **loop**

SRL\_gen**(**i**+**1**)<=**SRL\_gen**(**i**);**

**end** **loop;**

**end** **if;**

**end** **process;**

**end** Behavioral**;**

1. *Lab\_5\_Top\_BRAM.vhd*

ibrary IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**use** ieee**.**numeric\_std**.all;**

**use** ieee**.**std\_logic\_unsigned**.all;**

**entity** Lab\_5\_top **is**

**Generic(** WIDTH**:** INTEGER **:=**4**;**

M\_ELE**:** INTEGER **:=**4**);**

**Port(** data\_in**:** **in** STD\_LOGIC\_VECTOR**(**WIDTH**-**1 **downto** 0**);**

data\_out**:** **out** STD\_LOGIC\_VECTOR**(**WIDTH**-**1 **downto** 0**);**

clk**:** **in** STD\_LOGIC**;**

reset**:** **in** STD\_LOGIC**);**

**end** Lab\_5\_top**;**

**architecture** Behavioral **of** Lab\_5\_top **is**

--Constant definitions

**CONSTANT** ACC\_W\_DEC**:** INTEGER **:=** **(**2**\*\***WIDTH**-**1**)\***M\_ELE**;** --total decimal value accumulator must be able to hold

--signal definitiions

**signal** avg\_out**,**ele\_sub\_out **:** std\_logic\_vector**(**WIDTH**-**1 **downto** 0**)** **:=** **(Others** **=>**'0'**);** --input to Acc, subtracts previous elements to calculate running average

**signal** avg\_shift\_val**:** integer **:=**0**;**--caluclates average shift value form log\_2() funciton

--BRAM Definition

**COMPONENT** BRAM\_gen **is**

**Generic(**

W**:** INTEGER**:=**4 **;** --DATA WIDTH

M**:** INTEGER**:=**4 **;**

ADDR\_W**:** INTEGER**:=**2 **);** --Element # (Address in BRAM)

**port** **(**

clk **:** **in** std\_logic**;**

we **:** **in** std\_logic**;**

ena **:** **in** std\_logic**;**

addr **:** **in** std\_logic\_vector**(**ADDR\_W **downto** 0**);**

din **:** **in** std\_logic\_vector**(**W**-**1 **downto** 0**);**

dout **:** **out** std\_logic\_vector**(**W**-**1 **downto** 0**));**

**end** **COMPONENT;**

--Accumulator Definition

**COMPONENT** Lab5\_Acc **is**

**Generic(** W**:** INTEGER**:=**4 **;**

ACC\_W**:** INTEGER**:=**4 **);**

**Port(** d\_in**:** **in** STD\_LOGIC\_VECTOR**(**W**-**1 **downto** 0**);**

ele\_sub\_in**:** **in** STD\_LOGIC\_VECTOR**(**W**-**1 **downto** 0**);**

acc\_output**:** **out** STD\_LOGIC\_VECTOR**(**ACC\_W**-**1 **downto** 0**);**

clk**:** **in** STD\_LOGIC**);**

**end** **COMPONENT;**

--logrithmic function Definition

--calculates shift value for output din\_avg

**function** Log\_2 **(**

elements**:** INTEGER **:=** M\_ELE

**)**

**return** Integer **is**

**variable** AVG\_WIDTH**:** INTEGER **:=** elements**;**--width of input

**variable** log\_cnt**:** std\_logic\_vector**(**AVG\_WIDTH**-**1 **downto** 0**)** **:=** std\_logic\_vector**(to\_unsigned(**AVG\_WIDTH**,**AVG\_WIDTH**));**

**variable** temp**:** std\_logic**:=**'0'**;**

**variable** shift\_val **:** INTEGER **:=** 1**;**--shfit value acumulator

**begin**

**for** i **in** 0 **to** AVG\_WIDTH**-**1 **loop**

temp**:=**log\_cnt**(**0**);**

**if(**temp**=**'1'**)** **then**

shift\_val**:=**i**;**

**end** **if;**

log\_cnt**:=**'0'**&**log\_cnt**(**AVG\_WIDTH**-**1 **downto** 1**);**

**end** **loop;**

**return** shift\_val**;**

**end** Log\_2**;**

**CONSTANT** SHIFT\_AMT**:** integer **:=**Log\_2**(**M\_ELE**);**--amount ot shift acc output by to obtain truncated avg

**CONSTANT** ACC\_WIDTH**:** INTEGER **:=** Log\_2**(**ACC\_W\_DEC**)+**1**;** --width of accumulator line (must be at least 1 bit larger than input to hold up to 2 elements)

**CONSTANT** BRAM\_ADDR\_WIDTH**:** INTEGER **:=** SHIFT\_AMT**-**1**;** --width of accumulator line (must be at least 1 bit larger than input to hold up to 2 elements)

**signal** avg\_in**:** STD\_LOGIC\_VECTOR**(**ACC\_WIDTH**-**1 **downto** 0**):=** **(Others** **=>**'0'**);**--input to log function from accumulator module

**signal** w\_e**,** enable**:** STD\_LOGIC **:=** '0'**;**--BRAM write enable, enable

**signal** address **:** std\_logic\_vector**(**BRAM\_ADDR\_WIDTH **downto** 0**):=(Others=>**'0'**);**

**begin**

--BRAM instantiation

BRAM\_uut**:** BRAM\_gen

**Generic** **Map(**

W**=>**WIDTH**,** --DATA WIDTH

M**=>**M\_ELE**,**

ADDR\_W**=>**BRAM\_ADDR\_WIDTH**)**--Element # (Address in BRAM)

**Port** **Map(**

clk**=>**clk**,**

we**=>**w\_e**,**

ena**=>**enable**,**

addr**=>**address**,**

din**=>**data\_in**,**

dout**=>**ele\_sub\_out**);**

--Accumulator instantation

ACC\_uut**:** Lab5\_Acc

**Generic** **Map(** W**=>**WIDTH**,**

ACC\_W**=>**ACC\_WIDTH**)**

**Port** **Map** **(** d\_in**=>**data\_in**,**

ele\_sub\_in**=>**ele\_sub\_out**,**

acc\_output**=>**avg\_in**,**

clk**=>**clk**);**

SR\_avg**:process(**clk**)**--shifts proess right, generates synchronous output

**begin**

**if(**clk'**event** **AND** clk**=**'1'**)** **then**

**for** i **in** 0 **to** WIDTH**-**1 **loop**

data\_out**(**i**)<=**avg\_in**(**i**+**SHIFT\_AMT**);**

**end** **loop;**

**end** **if;**

**end** **process;**

bram\_addr\_en**:process(**clk**,**reset**)**--resets bram contents; initlizes all elements to 0;

**begin**

**if(**clk'**event** **and** clk**=**'1'**)** **then**

**if(**reset**=**'1'**)** **then**

enable**<=**'1'**;**

w\_e**<=**'1'**;**

address**<=(Others** **=>**'0'**);**

**else**

address**<=**address**+**1**;**

**end** **if;**

**end** **if;**

**end** **process;**

**end** Behavioral**;**

1. *BRAM\_gen.vhd*

**library** ieee**;**

**use** ieee**.**std\_logic\_1164**.all;**

**use** ieee**.**numeric\_std**.all;**

**entity** BRAM\_gen **is**

**Generic(**

W**:** INTEGER**:=**4 **;** --DATA WIDTH

M**:** INTEGER**:=**4 **;** --Element # (Address value in BRAM)

ADDR\_W**:** INTEGER**:=**2 **);** --addres vector width

**port** **(**

clk **:** **in** std\_logic**;**

we **:** **in** std\_logic**;**

ena **:** **in** std\_logic**;**

addr **:** **in** std\_logic\_vector**(**ADDR\_W **downto** 0**);**

din **:** **in** std\_logic\_vector**(**W**-**1 **downto** 0**);**

dout **:** **out** std\_logic\_vector**(**W**-**1 **downto** 0**));**

**end** BRAM\_gen**;**

**architecture** Behavioral **of** BRAM\_gen **is**

**type** mem\_t **is** **array(**integer **range<>)** **of** STD\_LOGIC\_VECTOR **(**W**-**1 **downto** 0**);**

**signal** mem **:** mem\_t**(**M**-**1 **downto** 0**):=(Others** **=>(Others** **=>**'0'**));**

**begin**

sync\_r\_w**:process(**clk**)**

**begin**

**if(**clk'**event** **and** clk**=**'1'**)** **then**

**if(rising\_edge(**ena**))** **then**

mem**<=** **(Others** **=>(Others** **=>**'0'**));**

**elsif(**ena **=** '1'**)** **then**

**if(**we **=** '1'**)** **then**

mem**(to\_integer(**unsigned**(**addr**)))** **<=** din**;**

**end** **if;**

dout **<=** mem**(to\_integer(**unsigned**(**addr**)));**

**else**

dout**<=** **(others** **=>**'0'**);**

**end** **if;**

**end** **if;**

**end** **process;**

**end** Behavioral**;**

1. *Lab5\_Acc.vhd*

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**use** ieee**.**numeric\_std**.all;**

**use** ieee**.**std\_logic\_unsigned**.all;**

**entity** Lab5\_Acc **is**

**Generic(** W**:** INTEGER**:=**4 **;**

ACC\_W**:** INTEGER**:=**4 **);**

**Port(** d\_in**:** **in** STD\_LOGIC\_VECTOR**(**W**-**1 **downto** 0**);**

ele\_sub\_in**:** **in** STD\_LOGIC\_VECTOR**(**W**-**1 **downto** 0**);**

acc\_output**:** **out** STD\_LOGIC\_VECTOR**(**ACC\_W**-**1 **downto** 0**);**

clk**:** **in** STD\_LOGIC**);**

**end** Lab5\_Acc**;**

**architecture** Behavioral **of** Lab5\_Acc **is**

**signal** Acc**,**Acc\_in**,**Acc\_out**,**Acc\_data**,**Acc\_sub**:** std\_logic\_vector**(**ACC\_W**-**1 **downto** 0**)** **:=** **(Others** **=>**'0'**);**--Accumulator:: input to log function, used to calculate din\_avg

**begin**

**process(**ele\_sub\_in**)**

**begin**

Acc\_sub**(**W**-**1 **downto** 0**)<=**ele\_sub\_in**;**

**end** **process;**

**process(**d\_in**)**

**begin**

Acc\_data**(**W**-**1 **downto** 0**)<=**d\_in**;**

**end** **process;**

**process(**clk**)**

**begin**

**if(rising\_edge(**clk**))** **then**

Acc**<=**Acc\_out**+**Acc\_in**;**

Acc\_output**<=**Acc\_out**+**Acc\_in**;**

**end** **if;**

**end** **process;**

Acc\_in**<=**Acc\_data**-**Acc\_sub**;**

Acc\_out**<=**Acc**;**

**end** Behavioral**;**

1. *Lab\_5\_LFSR.vhd*

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**use** IEEE**.**NUMERIC\_STD**.ALL;**

**use** ieee**.**std\_logic\_unsigned**.all;**

**entity** Lab5\_LFSR **is**

**Generic(**W**:** INTEGER**:=**4 **;** --Number of bits to shift

SEED**:** INTEGER**)** **;** --reseed output based on top level

**Port** **(** clk**:** **in** STD\_LOGIC**;**--clock enable, clock , reset

dout**:** **out** STD\_LOGIC**);**

**end** Lab5\_LFSR**;**

**architecture** Behavioral **of** Lab5\_LFSR **is**

--SEED definitions

**CONSTANT** SEED\_VEC\_1**:** STD\_LOGIC\_VECTOR**(**W**-**1 **downto** 0**)** **:=**std\_logic\_vector**(to\_unsigned(**SEED**,**W**));**

**CONSTANT** SEED\_VEC\_2**:** STD\_LOGIC\_VECTOR**(**W**-**1 **downto** 0**)** **:=**std\_logic\_vector**(to\_unsigned(**W**,**W**));**

**CONSTANT** SEED\_VEC**:** STD\_LOGIC\_VECTOR**(**W**-**1 **DOWNTO** 0**)** **:=**SEED\_VEC\_1 **XOR** SEED\_VEC\_2**;**

--signal defintions

**signal** LFSR **:** STD\_LOGIC\_VECTOR**(**W**-**1 **downto** 0**)** **:=** SEED\_VEC**;**--Initilize SRL\_gen to 0

--Infer SRL with appropriate attributes

**attribute** shreg\_extract **:** string**;**

**attribute** srl\_style **:** string**;**

**attribute** shreg\_extract **of** LFSR**:** **signal** **is** "yes"**;** --use SRL components

**attribute** srl\_style **of** LFSR**:** **signal** **is** "srl"**;** --Only Infer M-bit SRL,

--do not register inputs/outputs outside of SRL

**begin**

Output\_gen**:process(**clk**)**

**begin**

**if** **(** clk'**event** **and** clk **=**'1'**)** **then**

LFSR**(**W**-**1 **downto** 1**)** **<=** LFSR**(**W**-**2 **downto** 0**)** **;**

LFSR**(**0**)** **<=** **not(**LFSR**(**0**)** **XOR** LFSR**(**W**-**1**)** **);**

**end** **if;**

**end** **process;**

dout**<=**LFSR**(**0**);**

**end** Behavioral**;**