Kyle Keislar

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



Lab 4

Datapath & Control Path

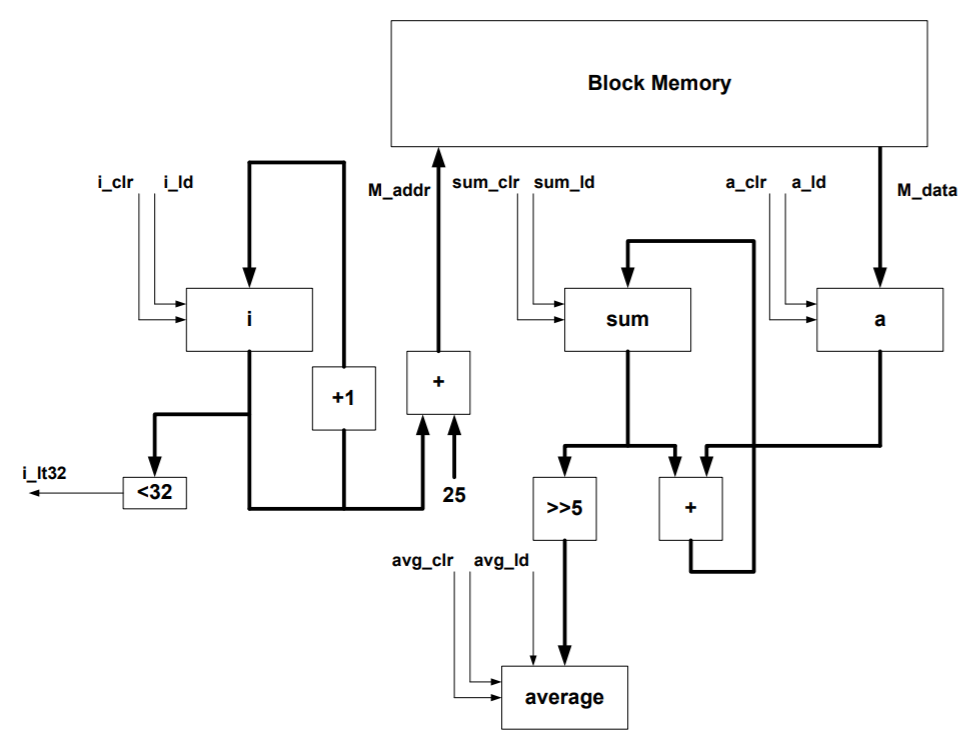
10/16/2020

## 

## Introduction & Problem Statements

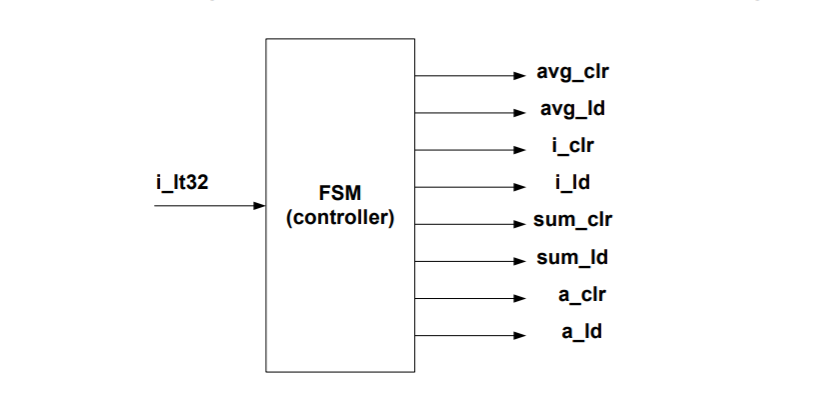
In this experiment an averaging circuit was designed to compute the average of a specific number of integer elements. The averaging circuit would receive a serial data stream from the Block Memory, compute the running average of all the passed elements, proceeding from a pre-assigned memory address until all of the elements have been read. In Fig 4.0.A, the high level Datapath diagram for the design is shown. This design assumed 32, 8-bit inputs were provided from the block ram. All elements were assumed to be in sequential memory addresses within the BRAM starting at address 25.

Fig 4.0.A: Datapath capable of computing the average of 32 elements of an array stored in memory



To regulate the averaging circuit operation, a Finite State Machine (FSM) was required. The design in Fig 4.0.A inferred 4 registered signals, each with two control signals in addition to one control signal output that would signify all elements have been read. From this information, a High Level diagram of the Averaging Circuit FSM was synthesized. This diagram can be seen below in Fig 4.0.B.

Fig 4.0.B: Averaging Finite State Machine High-Level Diagram



Using each of the signals above as a control signal, additional signal logic would be required to regulate which state the averaging circuit was operating in. These states would be regulated by i\_It32, a control signal which would be active high when 32 elements had been read. Each registered signal of the computed average (avg), the memory index to be read from the BRAM (i), the sum to be averaged (sum) and the next element to be summated (a) were supplied with an appropriate asynchronous clear and synchronous load.

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

An implemented design of this filter was not realized using the Zedboard for this experiment.

## Procedure:

***Part I***

Task 1: Implement the complete design including datapath and control path. Simulate your design forcorrect functionality. Include simulation of your design in your final report. Your simulation should clearly show your work and the average calculated by your datapath.

The design files for Task 1 can be seen in the Appendix as items A.2, A.3, A.4 and A.5. The test bench for this task can be seen as item A.1.

Task 2: Modify the design to indicate overflow. Prove that your logic works by including the propertestbench.

The testbench for this task can be seen in the appendix as item A.6.

Task 3: Modify your design if there are 50 numbers. How will the block diagram change? You need to adda logic to calculate the average of 50 numbers. Prove that your logic works by including the proper testbench.

The altered design files for the 50 element implementation can be seen in the appendix as items A.7 and A.8.

## Results (Data):

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

Fig 4.1: Task 1 Initialization

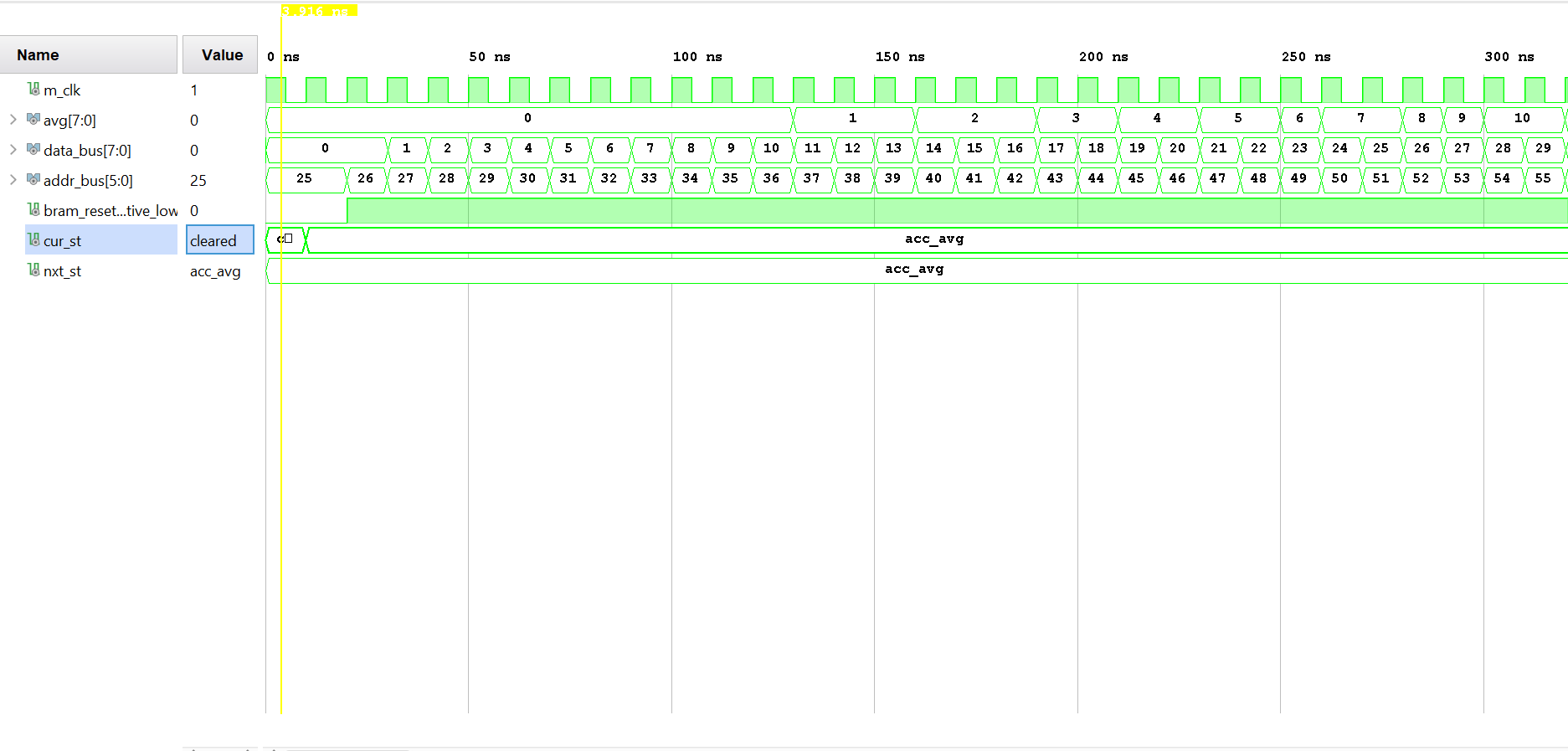


Fig 4.2: Task 1 Output

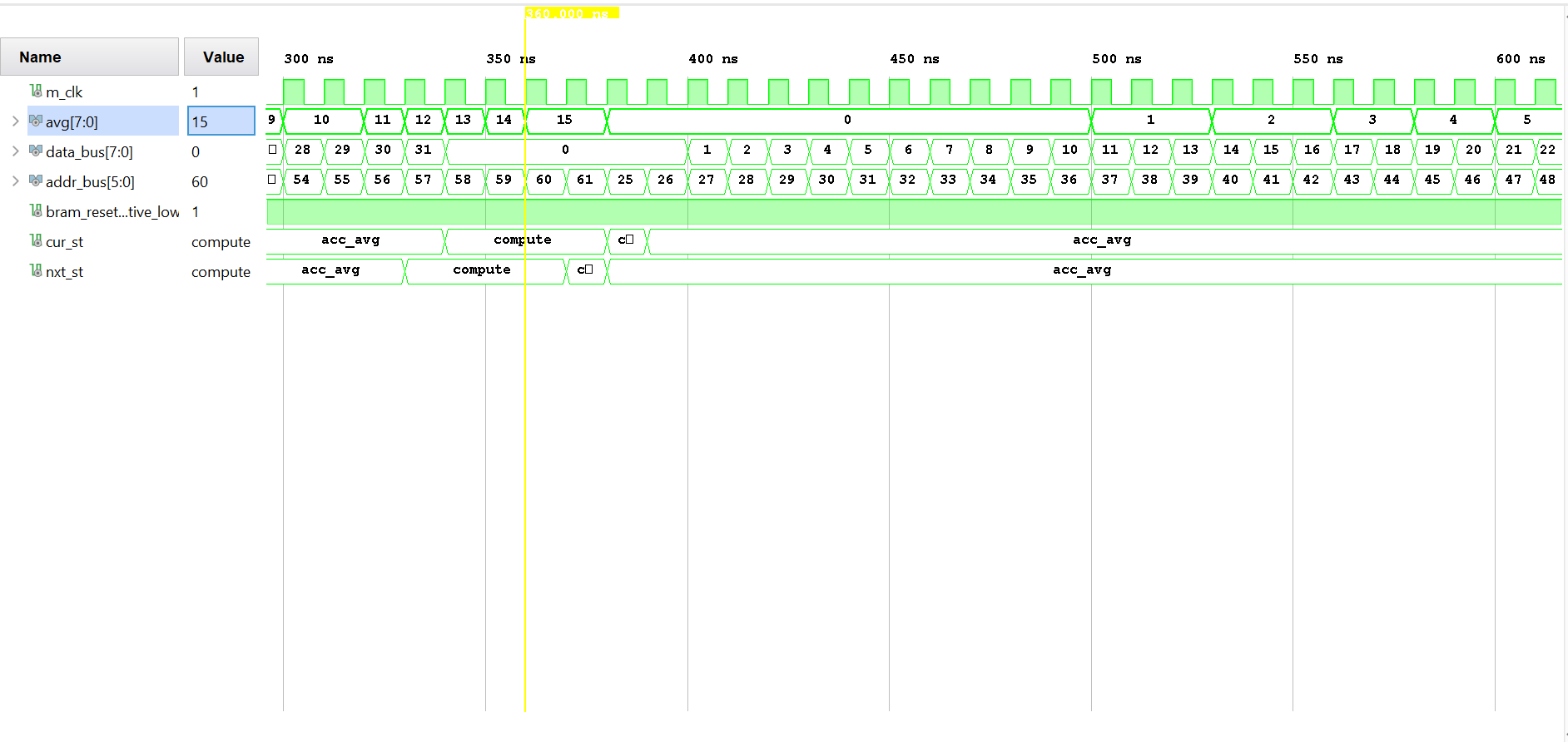


Fig 4.3: Task II Initialization

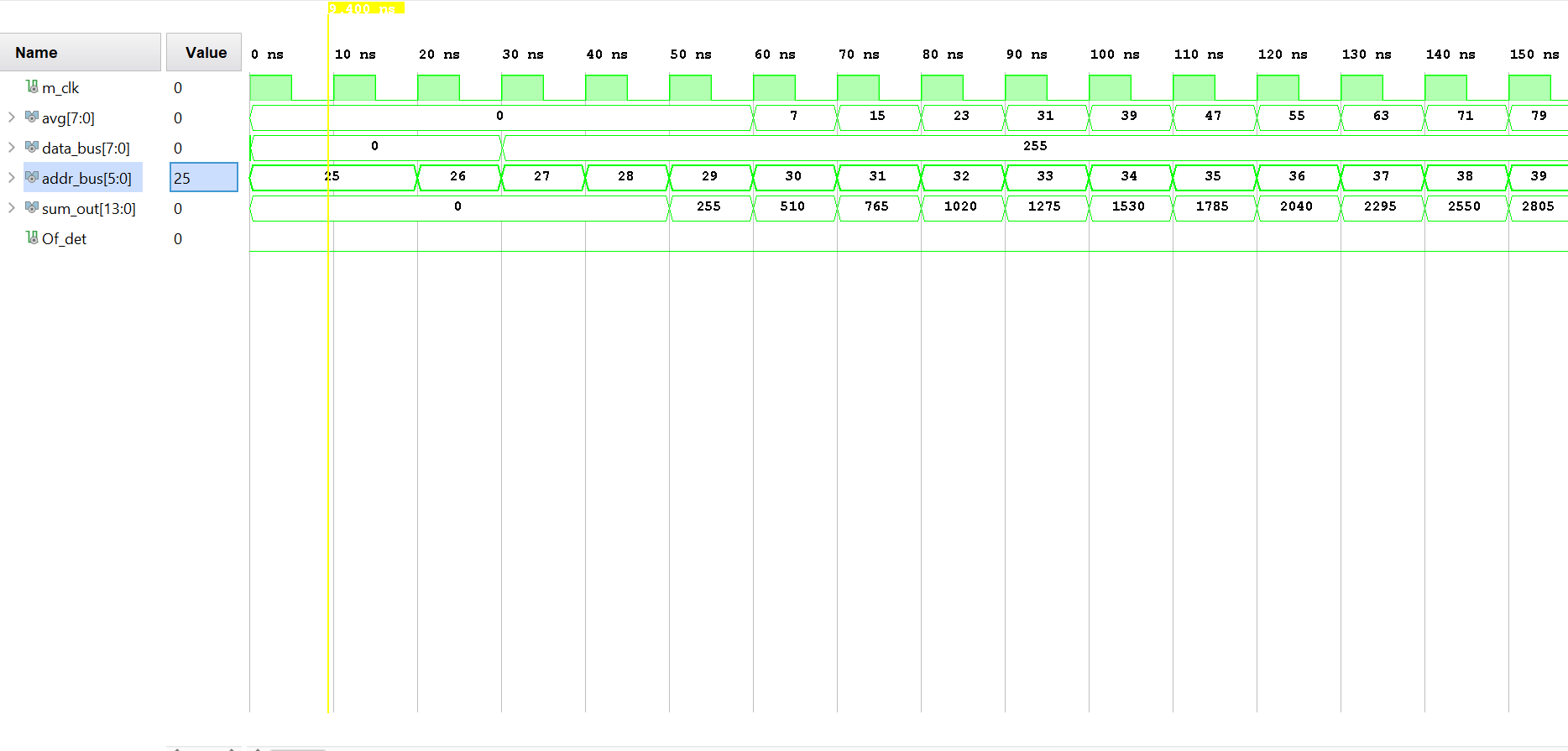


Fig 4.4: Task II Overflow Detected

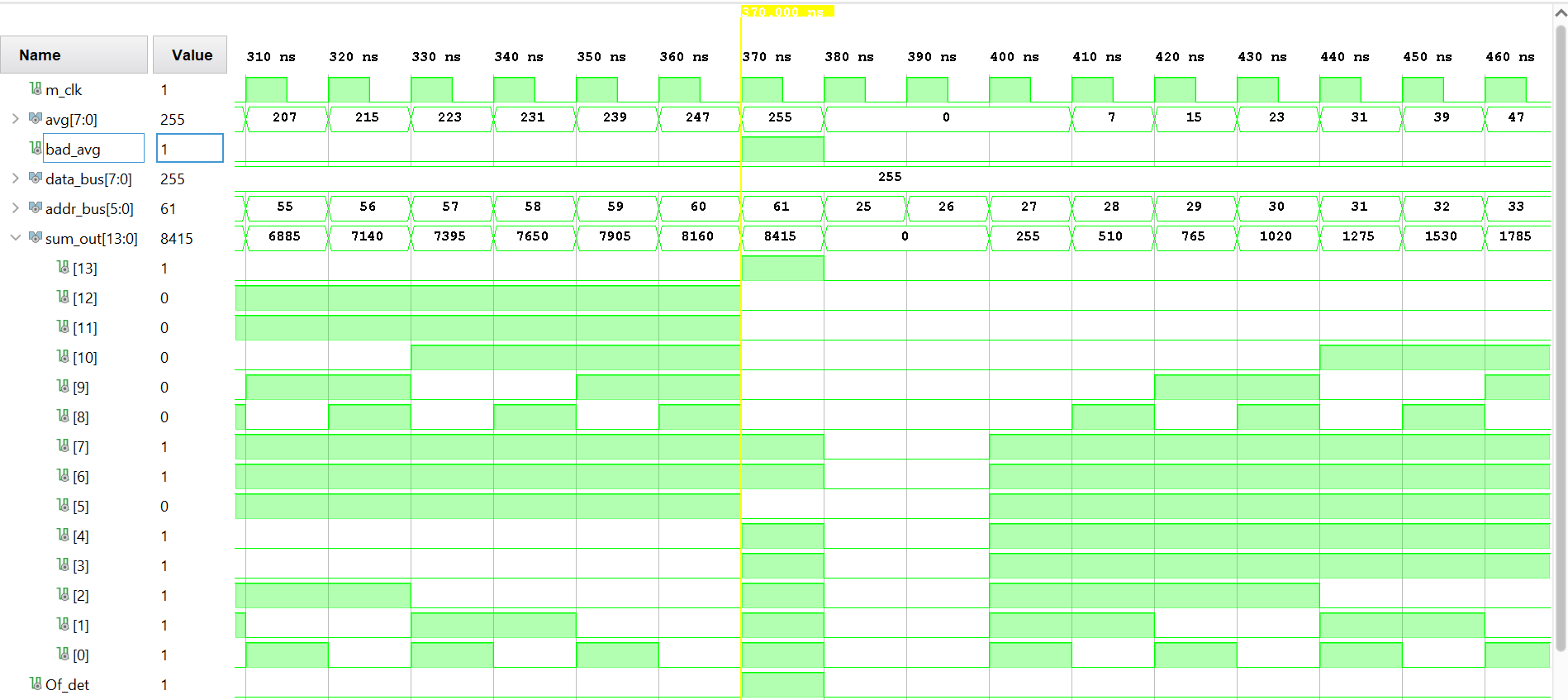


Fig 4.5: Task II Average Computation Demonstrating Overflow

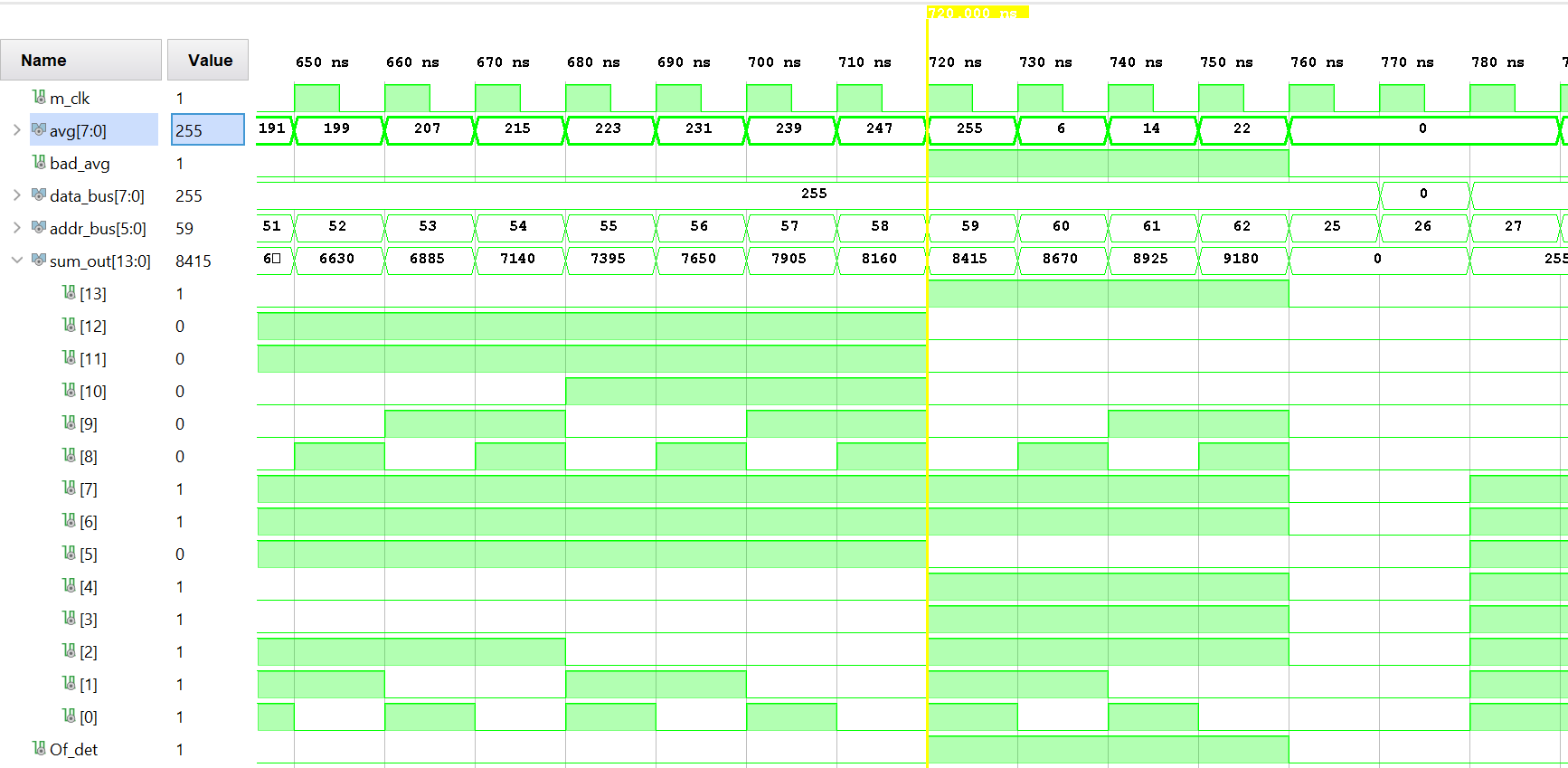


Fig 4.6: Task III Initialization

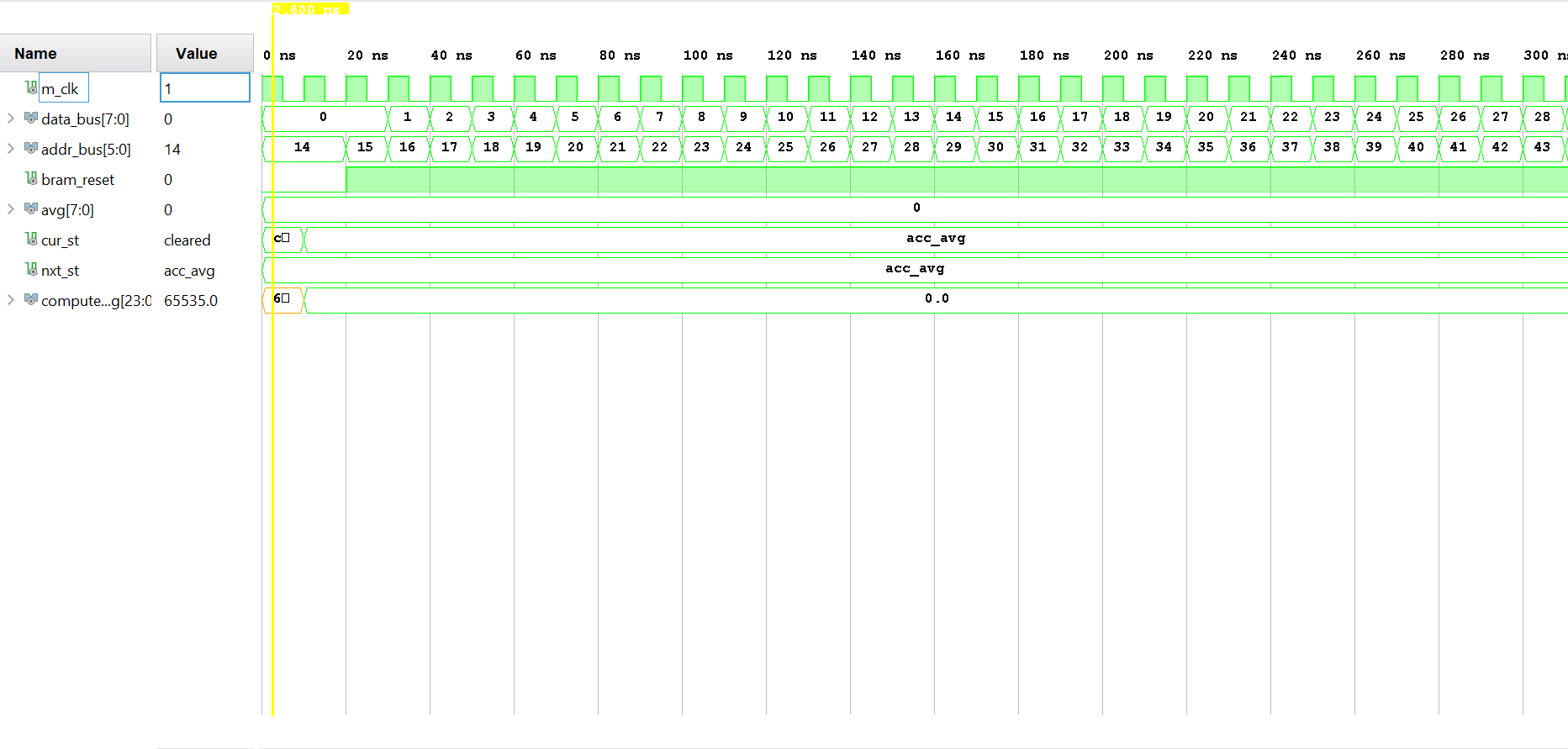


Fig 4.7: Task III Average Computed

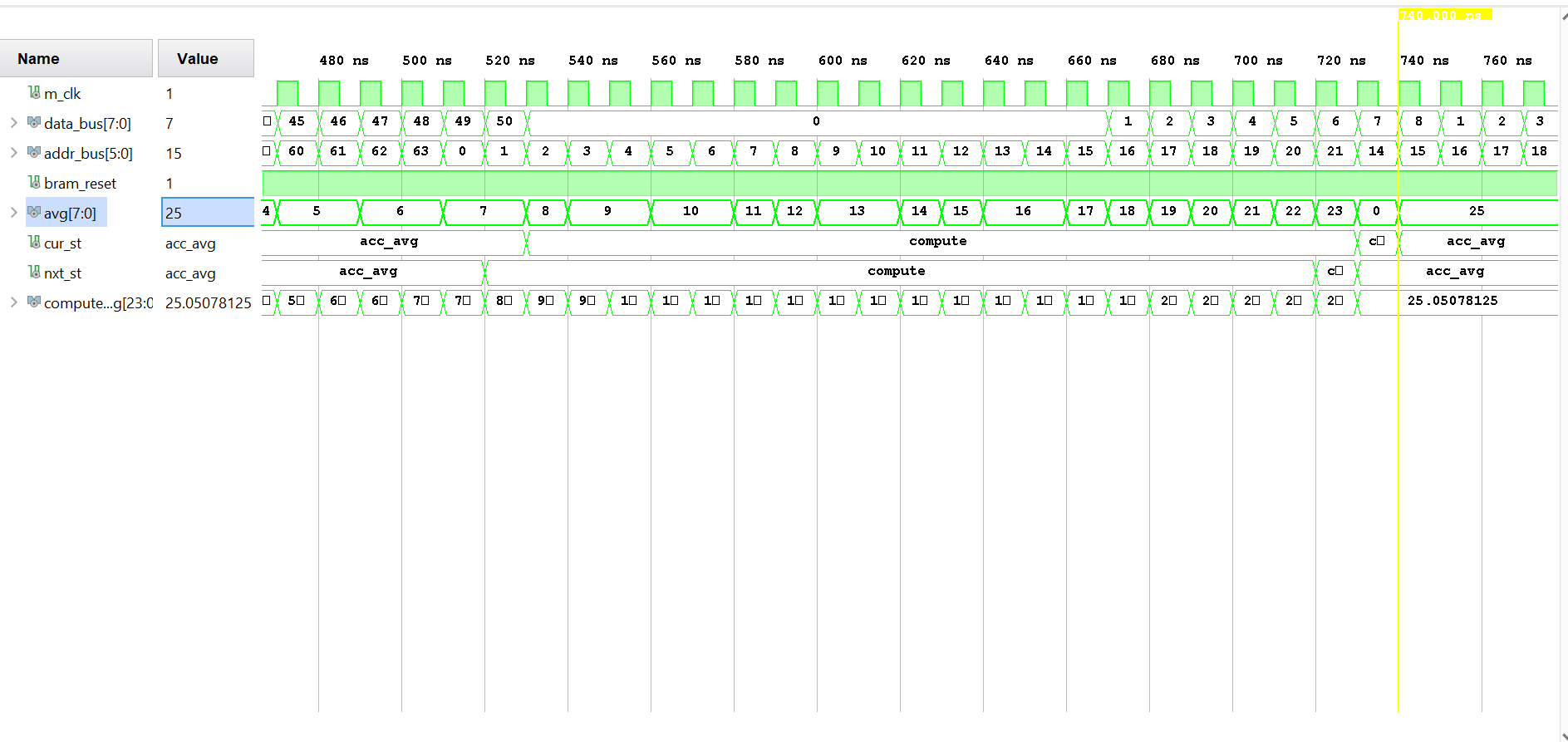


Fig 4.8: Task III Average Reset

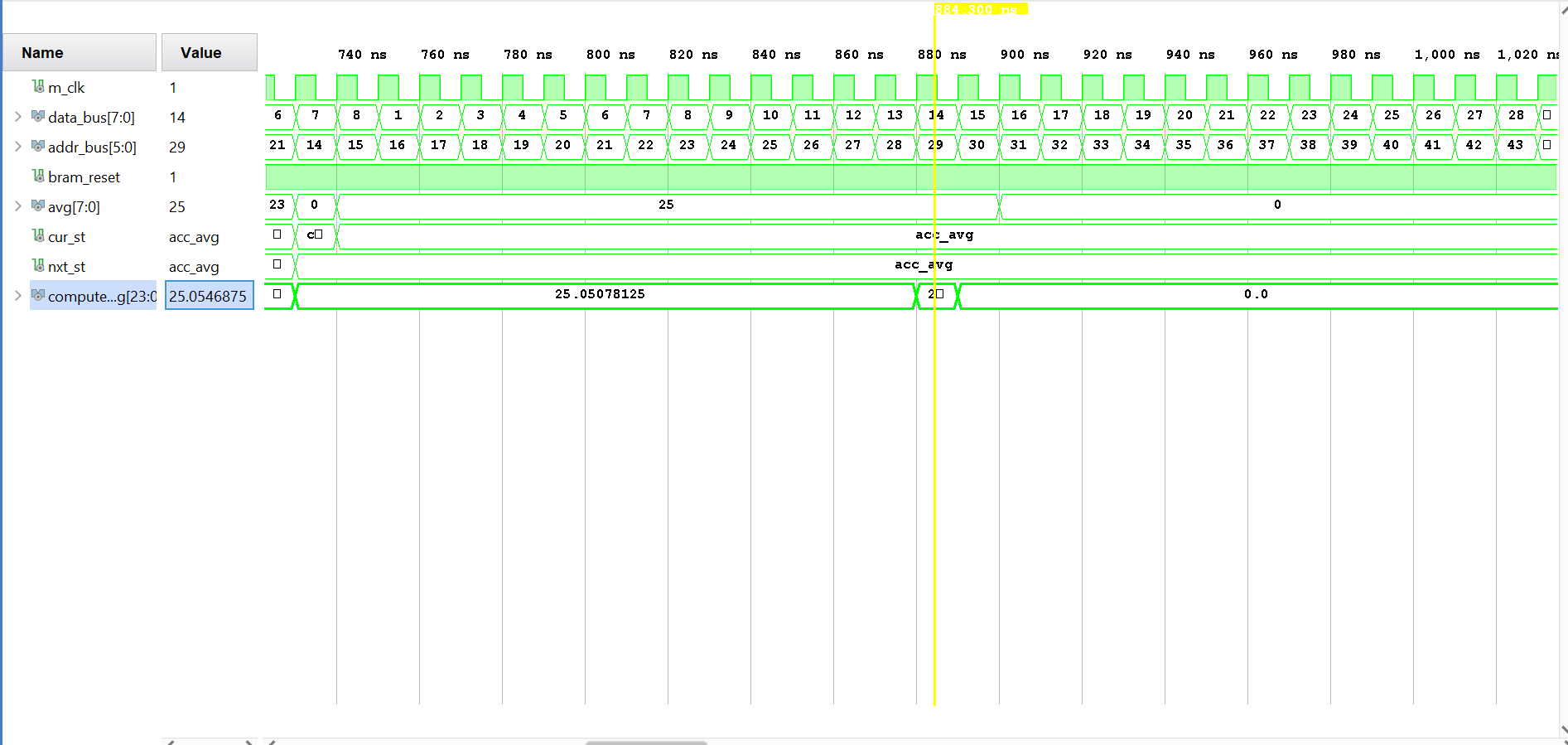
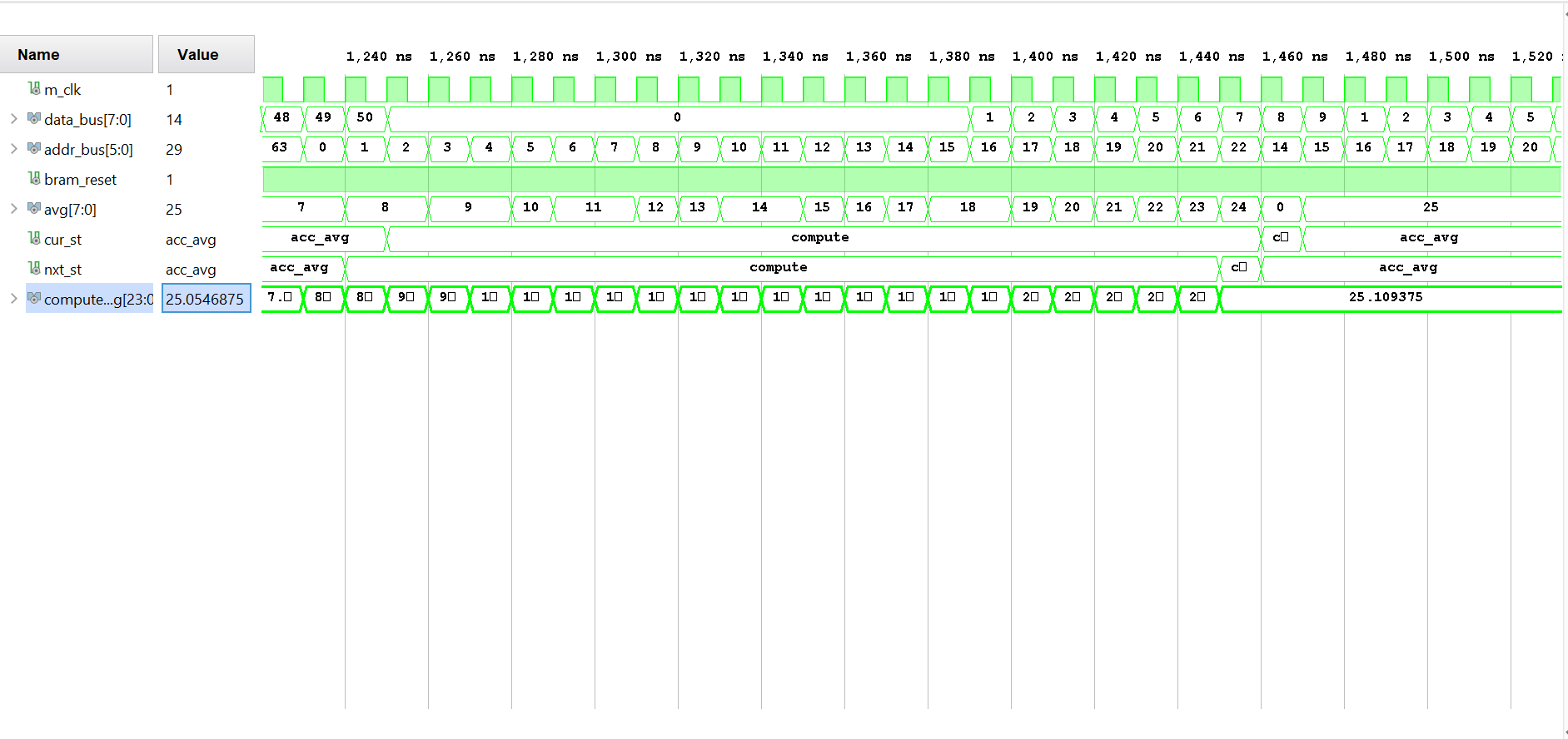


Fig 4.9: Task III Erroneous Average



## Analysis:

The simulated design for this experiment successfully demonstrates the functionality of the averaging circuit. However, this functionality is compromised by severe design latency, and erroneous persistent values which require a complete reinitialization of all design components, including BRAM, in order to demonstrate correct functionality.

Figures 4.1-4.2 demonstrate the successful functionality of the averaging circuit under default conditions. For this task the average of the first 31 decimal numbers (0-31) were computed. The result was 15, which is the integer truncation of the actual average 15.5. No dividing logic was utilized during this implementation and the precision of the input was preserved. This implementation has a delay of 4 clock cycles after 32 elements have been read from BRAM. The total latency of the design is 36 clock cycles.

Figures 4.3-4.6 demonstrate the successful functionality of the averaging circuit with overflow detection. The overflow detection will require an additional output logic from the sum register to detect when an overflow is occurred. To facilitate this logic, the register width must be increased by 1 bit to a total of 14 data bits. The extra bit serves as an overflow detection bit. The functionality of this detection module and the behavior of the circuit under overflow conditions are shown.

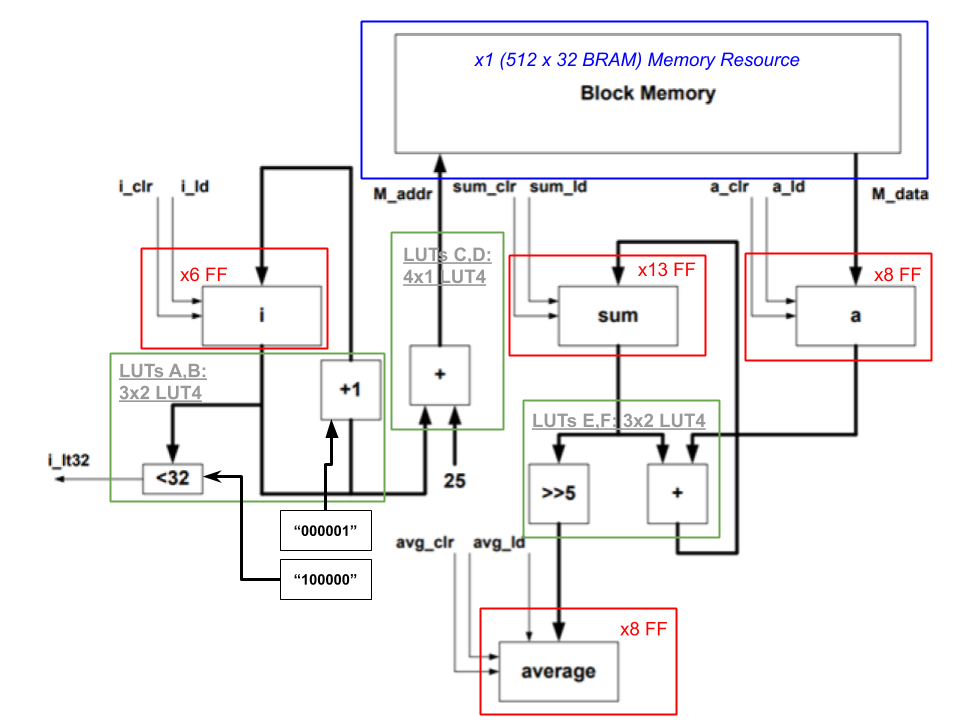
Finally Figures 4.7-4.9 show a partial functionality of the 50 element averaging circuit. This circuit has will compute erroneous values on each set of inputs after the first set. This is due to persistent values on the top level of the design. The block diagram located in Figure 4.0.A must be modified by comparing the element index signal I to a constant of 50 rather than 32. In addition, the shift left operation must be replaced by a divide module whose inputs are 50 and the sum.

APPENDIX

Questions

* 1) Estimate the FPGA area in terms of LUTs, FFs, memory, etc. for the design shown in Figure4.1 and elaborate your answer.

**Resource Usage Estimation for 32 Element Averager**



For each registered signal, one flip flop/ bit is required. Given the 8-bit data elements, signal a will also be 8-bits wide and require 8 FF. The averaged output, signal avg, will also be 8 bits and will also require 8 additional FF. Due to the design constraints of the 32 Element Averaging circuit, division implementations were constrained to a shift-left operation. This implied that the pre-divided sum (the dividend of the operation) would possesses a minimum of 13 data bits during accumulation. Therefore 13 Flip Flops were allocated to these registers. Finally, to address all memory indices required in parts I, II and III 6 bits of data were required to address the BRAM used in this experiment, resulting in an additional 6 FF.

6 total LUT4s are proposed to realize this implementation and are allocated as LUTs A, B, C, D, E and F.

LUTs A and B are used to compute the next relative element indicie with inputs from register i, and construct the control signal i\_It32 with inputs from register i. Each LUT has are 3 inputs provided for 4 bits from signals of the i register, the maximum indice value (32) and the address increment value (1). 2 LUTs are required to represent all 6 bits of these signals.

LUTs C and D are used to compute the next memory address to be read with inputs from register i. Each LUT has are 4 inputs provided for 4 bits from signals of the i register and the starting memory index value (25). 2 LUTs are required to represent all 6 bits of these signals.

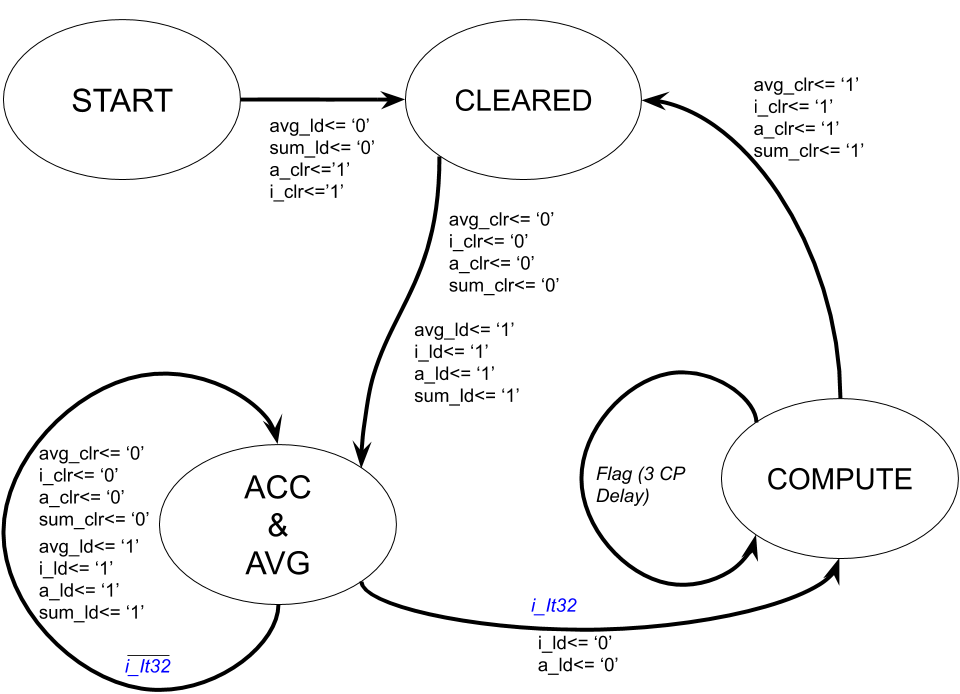
LUTs E and F are used to compute the next summated value form registers sum and a, as well as the next running average value derived from the shifted output of register sum. Each LUT has are 3 inputs provided for 4 bits from signals of the sum register, the a register and the shifting value. 2 LUTs are required to represent all 8 bits of these signals.

Finally, 1 512x32 single port BRAM primitive was instantiated by the Vivado 2019.2 synthesis tool to satisfy the memory requirements of this experiment. These dimensions were chosen for the minimum area which would hold all of the elements.

A summation of the estimated resources is provided in the table below

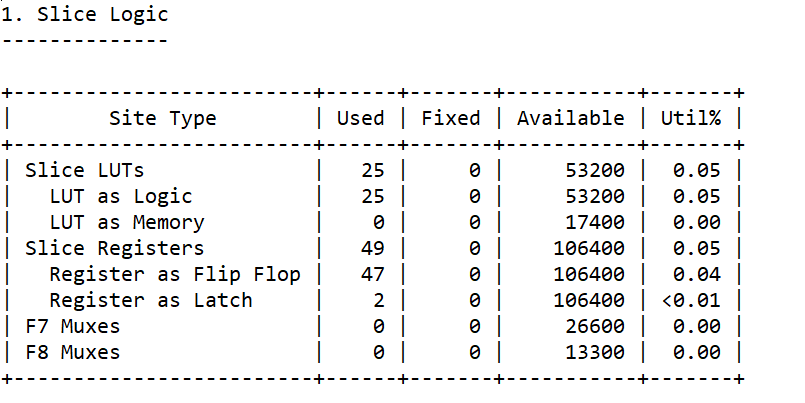
|  |  |  |
| --- | --- | --- |
| **Estimated Resources** | | |
| FF | LUT4 | BRAM primitives |
| 35 | 6 | 1 |

* 2) Draw the state diagram for FSM. The state machine block diagram is shown in Figure 4.0.B



* 3) What is the FPGA area utilization for your design? How does this compare with yourestimation in question 1? Explain your answer.

**Utilization report for 32 Element Averager**



The implemented design uses many more resources than my estimate. There are a multitude of reasons for this discrepancy outside of the estimated and actualized constraints during the design’s implementation. First the number of Look Up Tables estimated was far below what was actually requisitioned by the synthesis tool. This can be explained by the number of MAC operations used by the circuit during the computing of the average. As each SLICE only provides one adder one slice must be utilized per addition or subtraction done in a single clock pulse. This was not accounted for in my estimate. In the utilization table for the implemented design, many more additions must take place in a single clock cycle than I had originally accounted for. Each of the operations above were not constrained to be solely combinational in this design’s implementation, and many were registered. These included the control signals for the FSM and BRAM which were not estimated from the Datapath diagram located in Fig 4.0.A.

1. *Tb\_Lab\_4\_Top.vhd*

**library** IEEE**;**

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

**entity** tb\_Lab\_4\_Top **is**

-- Port ( );

**end** tb\_Lab\_4\_Top**;**

**architecture** Behavioral **of** tb\_Lab\_4\_Top **is**

--Constant Definitions

**CONSTANT** CP**:** TIME **:=** 10ns**;**--100 MHz clock period

**CONSTANT** DATA\_WIDTH**:** integer **:=** 8**;**

**CONSTANT** ADDRESS\_WIDTH**:** integer **:=** 6**;**

**CONSTANT** SUM\_WIDTH**:** integer **:=** 13**;**

--Signal Definitions

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

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

**signal** bram\_reset**:** std\_logic **:=** '0'**;** -- active low reset

**signal** avg**:** std\_logic\_vector **(**DATA\_WIDTH**-**1 **downto** 0**)** **:=** **(Others** **=>**'0'**);**

--Design Components

**COMPONENT** Lab\_4\_Top **is**

**Generic(**

DATA\_W**:** INTEGER **:=** 8**;**

ADDR\_W**:** INTEGER **:=** 6**;**

SUM\_W**:** INTEGER **:=** 13**);**

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

bram\_on**:** **in** STD\_LOGIC**;**

average\_top**:** **out** STD\_LOGIC\_VECTOR**(**DATA\_W**-**1 **downto** 0**));**

**end** **COMPONENT;**

**begin**

uut**:** Lab\_4\_Top --Averaging circuit

**Generic** **Map(**DATA\_W **=>** DATA\_WIDTH**,**

ADDR\_W**=>** ADDRESS\_WIDTH**,**

SUM\_W**=>**SUM\_WIDTH**)**

**Port** **Map(** clk **=>** m\_clk**,**

bram\_on**=>**bram\_reset**,**

average\_top **=>**avg**);**

--Clock Processs

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

**begin**

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

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

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

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

**end** **process;**

bram\_r**:process** --Asynch reset

**begin**

**wait** **for** CP**\***2**;**--clear bram for 2 clock pulses to initalize a, sum and avg registers

bram\_reset**<=**'1'**;**

**wait;**

**end** **process;**

**end** Behavioral**;**

1. *Lab\_4\_Top.vhd*

**library** IEEE**;**

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

**entity** Lab\_4\_Top **is**

**Generic(**

DATA\_W**:** INTEGER **:=** 8**;**

ADDR\_W**:** INTEGER **:=** 6**;**

SUM\_W**:** INTEGER **:=** 13**);**

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

bram\_on**:** **in** STD\_LOGIC**;**

average\_top**:** **out** STD\_LOGIC\_VECTOR**(**DATA\_W**-**1 **downto** 0**));**

**end** Lab\_4\_Top**;**

**architecture** Behavioral **of** Lab\_4\_Top **is**

**signal** data\_bus**:** STD\_LOGIC\_VECTOR**(**DATA\_W**-**1 **downto** 0**)** **:=** **(Others** **=>**'0'**);**

**signal** addr\_bus**:** STD\_LOGIC\_VECTOR**(**ADDR\_W**-**1 **downto** 0**)** **:=** **(Others** **=>**'0'**);**

**signal** i\_It32\_l**,** avg\_clr\_l**,** avg\_ld\_l**,** i\_clr\_l**,** i\_ld\_l**,** sum\_clr\_l**,** sum\_ld\_l**,** a\_clr\_l**,** a\_ld\_l**:** STD\_LOGIC **:=** '0'**;**

--BRAM

**COMPONENT** Lab\_4\_BRAM **is**

**Generic(**D\_WIDTH**:** INTEGER **:=** 8**;**

ADDR\_WIDTH**:** INTEGER **:=** 6**);**

**Port(**

M\_data**:** **out** STD\_LOGIC\_VECTOR**(**D\_WIDTH**-**1 **downto** 0**);** --avg data input, sent from BRAM

M\_addr**:** **in** STD\_LOGIC\_VECTOR**(**ADDR\_WIDTH**-**1 **downto** 0**);**--next element address; sent to BRAM

clk**:IN** STD\_LOGIC**;**

BRAM\_en**:** **IN** STD\_LOGIC**);**

**end** **COMPONENT;**

--AVG

**COMPONENT** Lab\_4\_Avg **is**

**Generic(**

D\_WIDTH**:** INTEGER **:=** 8**;** --Data Width

ADDR\_WIDTH**:** INTEGER **:=** 6**;** --Address Width

SUM\_WIDTH**:** INTEGER **:=** 13**);** --Sum bit width; set by user

**Port(**

--CLOCK SIGNALS

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

--DATA I/0

M\_data**:** **in** STD\_LOGIC\_VECTOR**(**D\_WIDTH**-**1 **downto** 0**);** --avg data input, sent from BRAM

M\_addr**:** **out** STD\_LOGIC\_VECTOR**(**ADDR\_WIDTH**-**1 **downto** 0**);**--next element address; sent to BRAM

average**:** **out** STD\_LOGIC\_VECTOR**(**D\_WIDTH**-**1 **downto** 0**);** --computed average of N elements

--CONTROL SIGNALS

i\_It32**:** **out** STD\_LOGIC**;** --input to FSM; high if all 31 memory addresses have been addressed

avg\_clr**:** **in** STD\_LOGIC**;** --input from FSM; clear average result (avg) register

avg\_ld**:** **in** STD\_LOGIC**;** --input from FSM; load average result (avg) register (used for reset)

i\_clr**:** **in** STD\_LOGIC**;**--input from FSM; clear address index (i) register

i\_ld**:** **in** STD\_LOGIC**;**--input from FSM; load address index (i) register

sum\_clr**:** **in** STD\_LOGIC**;**--input from FSM; clear avg accumulator (sum) register

sum\_ld**:** **in** STD\_LOGIC**;**--input from FSM; load avg accumulator (sum) register (used for reset)

a\_clr**:** **in** STD\_LOGIC**;**--input from FSM; clear average input register (a) register

a\_ld**:** **in** STD\_LOGIC**);**--input from FSM; load average input register (a) register with value from BRAM

**end** **COMPONENT;**

--FSM

**COMPONENT** Lab\_4\_FSM **is**

**Port(**

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

i\_It32**:** **in** STD\_LOGIC**;** --FSM input; high if all 31 memory addresses have been addressed

avg\_clr**:** **out** STD\_LOGIC**;** --clear average result (avg) register

avg\_ld**:** **out** STD\_LOGIC**;** --load average result (avg) register (used for reset)

i\_clr**:** **out** STD\_LOGIC**;**--clear address index (i) register

i\_ld**:** **out** STD\_LOGIC**;**--load address index (i) register

sum\_clr**:** **out** STD\_LOGIC**;**--clear avg accumulator (sum) register

sum\_ld**:** **out** STD\_LOGIC**;**--load avg accumulator (sum) register (used for reset)

a\_clr**:** **out** STD\_LOGIC**;**--clear average input register (a) register

a\_ld**:** **out** STD\_LOGIC**);**--load average input register (a) register with value from BRAM

**end** **COMPONENT;**

**begin**

avg\_bram**:**Lab\_4\_BRAM

**Generic** **Map(**D\_WIDTH **=>** DATA\_W**,**

ADDR\_WIDTH**=>** ADDR\_W**)**

**Port** **Map(** M\_data**=>**data\_bus**,**

M\_addr**=>**addr\_bus**,** --next element address; sent to BRAM

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

BRAM\_en**=>**bram\_on**);**

avg\_logic**:**Lab\_4\_Avg

**Generic** **Map(**D\_WIDTH **=>** DATA\_W**,**

ADDR\_WIDTH**=>** ADDR\_W**,**

SUM\_WIDTH **=>**SUM\_W**)**

**Port** **Map(** clk**=>**clk**,**

M\_data**=>**data\_bus**,**

M\_addr**=>**addr\_bus**,**

average**=>**average\_top**,**

i\_It32**=>**i\_It32\_l**,**

avg\_clr**=>**avg\_clr\_l**,**

avg\_ld**=>**avg\_ld\_l**,**

i\_clr**=>**i\_clr\_l**,**

i\_ld**=>**i\_ld\_l**,**

sum\_clr**=>**sum\_clr\_l**,**

sum\_ld**=>**sum\_ld\_l**,**

a\_clr**=>**a\_clr\_l**,**

a\_ld**=>**a\_ld\_l**);**

avg\_fsm**:** Lab\_4\_FSM

**Port** **Map(**

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

i\_It32**=>**i\_It32\_l**,**

avg\_clr**=>**avg\_clr\_l**,**

avg\_ld**=>**avg\_ld\_l**,**

i\_clr**=>**i\_clr\_l**,**

i\_ld**=>**i\_ld\_l**,**

sum\_clr**=>**sum\_clr\_l**,**

sum\_ld**=>**sum\_ld\_l**,**

a\_clr**=>**a\_clr\_l**,**

a\_ld**=>**a\_ld\_l**);**

**end** Behavioral**;**

1. *Lab\_4\_Avg.vhd*

**library** IEEE**;**

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

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

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

**entity** Lab\_4\_Avg **is**

**Generic(**

D\_WIDTH**:** INTEGER **:=** 8**;** --Data Width

ADDR\_WIDTH**:** INTEGER **:=** 6**;** --Address Width

SUM\_WIDTH**:** INTEGER **:=** 13**);** --Sum bit width; set by user

**Port(**

--CLOCK SIGNALS

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

--DATA I/0

M\_data**:** **in** STD\_LOGIC\_VECTOR**(**D\_WIDTH**-**1 **downto** 0**);** --avg data input, sent from BRAM

M\_addr**:** **out** STD\_LOGIC\_VECTOR**(**ADDR\_WIDTH**-**1 **downto** 0**);**--next element address; sent to BRAM

average**:** **out** STD\_LOGIC\_VECTOR**(**D\_WIDTH**-**1 **downto** 0**);** --computed average of N elements

--CONTROL SIGNALS

i\_It32**:** **out** STD\_LOGIC**;** --input to FSM; high if all 31 memory addresses have been addressed

avg\_clr**:** **in** STD\_LOGIC**;** --input from FSM; clear average result (avg) register

avg\_ld**:** **in** STD\_LOGIC**;** --input from FSM; load average result (avg) register (used for reset)

i\_clr**:** **in** STD\_LOGIC**;**--input from FSM; clear address index (i) register

i\_ld**:** **in** STD\_LOGIC**;**--input from FSM; load address index (i) register

sum\_clr**:** **in** STD\_LOGIC**;**--input from FSM; clear avg accumulator (sum) register

sum\_ld**:** **in** STD\_LOGIC**;**--input from FSM; load avg accumulator (sum) register (used for reset)

a\_clr**:** **in** STD\_LOGIC**;**--input from FSM; clear average input register (a) register

a\_ld**:** **in** STD\_LOGIC**);**--input from FSM; load average input register (a) register with value from BRAM

**end** Lab\_4\_Avg**;**

**architecture** Behavioral **of** Lab\_4\_Avg **is**

**CONSTANT** START\_ADDR**:** STD\_LOGIC\_VECTOR**(**ADDR\_WIDTH**-**1 **downto** 0**)** **:=** std\_logic\_vector**(to\_unsigned(**25**,**ADDR\_WIDTH**));** --starting address of 25

**CONSTANT** NXT\_i**:** STD\_LOGIC\_VECTOR**(**ADDR\_WIDTH**-**1 **downto** 0**)** **:=** std\_logic\_vector**(to\_unsigned(**1**,**ADDR\_WIDTH**));** --amount to incremt each indicie by

**signal** i\_count**:** STD\_LOGIC\_VECTOR**(**ADDR\_WIDTH**-**1 **downto** 0**)** **:=** **(Others** **=>**'0'**);** --counts memory indicies during averaging

**signal** a\_out**:** STD\_LOGIC\_VECTOR**(**D\_WIDTH**-**1 **downto** 0**)** **:=** **(Others** **=>**'0'**);**--data read from a reg

**signal** sum\_in**,**sum\_out**:** STD\_LOGIC\_VECTOR**(**SUM\_WIDTH**-**1 **downto** 0**)** **:=** **(Others** **=>**'0'**);**--data read into sum reg from adder,

--data read from sum reg to adder

**signal** i\_in**,**i\_out**:** STD\_LOGIC\_VECTOR**(**ADDR\_WIDTH**-**1 **downto** 0**)** **:=** **(Others** **=>**'0'**);**--data read from counter to index register

**signal** avg\_in**,**avg\_out**:** STD\_LOGIC\_VECTOR**(**D\_WIDTH**-**1 **downto** 0**)** **:=** **(Others** **=>**'0'**);** --data into address register

**begin**

a\_ld\_clr**:** **process** **(**clk**,**a\_clr**,**a\_ld**)**

**begin**

**if(**a\_clr**=**'1'**)** **then**

a\_out**<=** **(Others** **=>** '0'**);**

**end** **if;**

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

**if(**a\_ld**=**'1'**)** **then**

a\_out**<=**M\_data**;** --load next value form BRAM

**end** **if;**

**end** **if;**

**end** **process;**

sum\_ld\_clr**:** **process** **(**clk**,**sum\_clr**,**sum\_ld**)**

**begin**

**if(**sum\_clr**=**'1'**)** **then**

sum\_out**<=** **(Others** **=>** '0'**);**

**end** **if;**

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

**if(**sum\_ld**=**'1'**)** **then**

sum\_out**<=**sum\_in**;** --load next value form BRAM

**end** **if;**

**end** **if;**

**end** **process;**

i\_ld\_clr**:** **process** **(**clk**,**i\_clr**,**i\_ld**)**

**begin**

**if(**i\_clr**=**'1'**)** **then**

i\_out**<=** **(Others** **=>** '0'**);**

**end** **if;**

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

**if(**i\_ld**=**'1'**)** **then**

i\_out**<=**i\_in**;** --load next value form BRAM

**end** **if;**

**end** **if;**

**end** **process;**

avg\_ld\_clr**:** **process** **(**clk**,**avg\_clr**,**avg\_ld**)**

**begin**

**if(**avg\_clr**=**'1'**)** **then**

avg\_out**<=** **(Others** **=>** '0'**);**

**end** **if;**

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

**if(**avg\_ld**=**'1'**)** **then**

avg\_out**<=**avg\_in**;** --load next value form BRAM

**end** **if;**

**end** **if;**

**end** **process;**

--OUTPUT LOGIC

average**<=**avg\_out**;**

M\_addr**<=**i\_out**+**START\_ADDR**;**--output next element address

i\_It32**<=** '1' **when** i\_out**>**std\_logic\_vector**(to\_unsigned(**31**,**ADDR\_WIDTH**-**1**))** **else**

'0'**;**

--SIGNAL LOGIC

--SUM\_IN

sum\_in**<=**sum\_out**+**a\_out**;**

--AVG\_IN

avg\_in**<=**sum\_out**(**SUM\_WIDTH**-**1 **downto** 5**);** --shift left by 5 bits to divide by 32 elements

--I\_IN

i\_in**<=**i\_out**+**NXT\_i**;**

**end** Behavioral**;**

1. *Lab\_4\_FSM.vhd*

**library** IEEE**;**

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

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

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

**entity** Lab\_4\_FSM **is**

**Port(**

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

i\_It32**:** **in** STD\_LOGIC**;** --FSM input; high if all 31 memory addresses have been addressed

avg\_clr**:** **out** STD\_LOGIC**;** --clear average result (avg) register

avg\_ld**:** **out** STD\_LOGIC**;** --load average result (avg) register (used for reset)

i\_clr**:** **out** STD\_LOGIC**;**--clear address index (i) register

i\_ld**:** **out** STD\_LOGIC**;**--load address index (i) register

sum\_clr**:** **out** STD\_LOGIC**;**--clear avg accumulator (sum) register

sum\_ld**:** **out** STD\_LOGIC**;**--load avg accumulator (sum) register (used for reset)

a\_clr**:** **out** STD\_LOGIC**;**--clear average input register (a) register

a\_ld**:** **out** STD\_LOGIC**);**--load average input register (a) register with value from BRAM

**end** Lab\_4\_FSM**;**

**architecture** Behavioral **of** Lab\_4\_FSM **is**

**CONSTANT** AMT**:** INTEGER **:=**0**;**

**type** state **is** **(**start**,**cleared**,**acc\_avg**,**compute**);**--state types: begin averaging

**signal** cur\_st**:** state **;** --current state, next state; defaults to start

**signal** nxt\_st**:** state**:=**start **;** --current state, next state; defaults to start

**signal** out\_count**:** STD\_LOGIC\_VECTOR**(**4 **downto** 0**)** **:=** std\_logic\_vector**(to\_unsigned(**32**-**AMT**,**5**));**--hold output when done to compute average

**signal** flag**,**test**:** STD\_LOGIC **:=** '0'**;**

**begin**

Change\_st**:process(**clk**)** --get next state when clocked

**begin**

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

cur\_st **<=** nxt\_st**;**

**if(**cur\_st **=** acc\_avg **OR** cur\_st**=**compute**)** **then**

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

**else**

out\_count**<=** **(Others** **=>**'0'**);**

**end** **if;**

**end** **if;**

**end** **process;**

St\_ctrl**:** **process(**cur\_st**,** i\_It32**,**flag**)** --state controller

**begin**

**case** **(**cur\_st**)** **is**

--CLEAR

**when** cleared **=>**

nxt\_st **<=** acc\_avg**;**

--ACC\_AVG

**when** acc\_avg **=>**

**if(**i\_It32 **=** '1'**)then**

nxt\_st**<=** compute**;**

**else**

nxt\_st**<=** acc\_avg**;**

**end** **if;**

**when** compute **=>**

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

nxt\_st**<=**cleared**;**

**else**

nxt\_st**<=**compute**;**

**end** **if;**

**when** start **=>** --return from start

nxt\_st **<=** cleared**;**

**when** **others** **=>**

nxt\_st **<=** cleared**;**

**end** **case;**

**end** **process;**

Out\_ctrl**:process(**cur\_st**)**--output controller, sets control signals based on current stae value

**begin**

**case** **(**cur\_st**)** **is** ---Select output based on current state

--CLEAR

**when** cleared **=>**

avg\_clr**<=**'1'**;**--clear values

i\_clr**<=**'1'**;**

a\_clr**<=**'1'**;**

sum\_clr**<=**'1'**;**

--ACC\_AVG

**when** acc\_avg **=>**

**if(**i\_It32 **=** '1'**)then** --if all elements have been averaged

i\_ld**<=**'0'**;**--task is done, stop loading new values

a\_ld**<=**'0'**;**

**else** --if un-averaged elements remain

avg\_clr**<=**'0'**;**--load next values

avg\_ld**<=**'1'**;**

i\_clr**<=**'0'**;**

i\_ld**<=**'1'**;**

a\_clr**<=**'0'**;**

a\_ld**<=**'1'**;**

sum\_clr**<=**'0'**;**

sum\_ld**<=**'1'**;**

**end** **if;**

**when** compute **=>**

**when** start **=>**

i\_clr**<=**'1'**;**

a\_clr**<=**'1'**;**

**when** **others** **=>** --clear all regs on init/error

avg\_clr**<=**'0'**;**

i\_clr**<=**'0'**;**

a\_clr**<=**'0'**;**

sum\_clr**<=**'0'**;**

**end** **case;**

**end** **process;**

flag **<=** '1' **when** out\_count**=**"00100" **else** -–hold for 3 clock pulses to calculate result

'0'**;**

**end** Behavioral**;**

1. *Lab\_4\_BRAM.vhd*

**library** IEEE**;**

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

**entity** Lab\_4\_BRAM **is**

**Generic(**D\_WIDTH**:** INTEGER **:=** 8**;**

ADDR\_WIDTH**:** INTEGER **:=** 6**);**

**Port(**

M\_data**:** **out** STD\_LOGIC\_VECTOR**(**D\_WIDTH**-**1 **downto** 0**);** --avg data input, sent from BRAM

M\_addr**:** **in** STD\_LOGIC\_VECTOR**(**ADDR\_WIDTH**-**1 **downto** 0**);**--next element address; sent to BRAM

clk**:IN** STD\_LOGIC**;**

BRAM\_en**:** **IN** STD\_LOGIC**);**

**end** Lab\_4\_BRAM**;**

**architecture** Behavioral **of** Lab\_4\_BRAM **is**

**COMPONENT** blk\_mem\_gen\_0 **IS**

**PORT** **(**

clka **:** **IN** STD\_LOGIC**;**

ena **:** **IN** STD\_LOGIC**;**

wea **:** **IN** STD\_LOGIC\_VECTOR**(**0 **DOWNTO** 0**);**

addra **:** **IN** STD\_LOGIC\_VECTOR**(**5 **DOWNTO** 0**);**--M\_addr

dina **:** **IN** STD\_LOGIC\_VECTOR**(**7 **DOWNTO** 0**);** --unused (ram initilzation done with .coe file @ C:\Users\kkeis\Documents\524\_workspace\Lab\_4\_Avg\Lab\_4\_Avg.srcs\BRAM\_init.coe)

douta **:** **OUT** STD\_LOGIC\_VECTOR**(**7 **DOWNTO** 0**)**--M\_data

**);**

**END** **COMPONENT;**

**signal** wr\_en**:** STD\_LOGIC\_VECTOR**(**0 **DOWNTO** 0**):=** **(Others** **=>**'0'**);** --tie wea to low (never write to BRAM during operation

**signal** BRAM\_d\_in**:** STD\_LOGIC\_VECTOR**(**D\_WIDTH**-**1 **downto** 0**)** **:=** **(others** **=>**'0'**);**--tie unused din to '0'

**begin**

avg\_bram**:** blk\_mem\_gen\_0

**Port** **Map(** clka **=>** clk**,**

ena**=>** BRAM\_en**,**

wea**=>**wr\_en**,**

addra**=>**M\_addr**,**

douta**=>**M\_data**,**

dina**=>**BRAM\_d\_in**);**

**end** Behavioral**;**

1. *Tb\_Lab\_4\_Top\_Overflow.vhd*

**library** IEEE**;**

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

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

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

**entity** tb\_Lab\_4\_Top **is**

-- Port ( );

**end** tb\_Lab\_4\_Top**;**

**architecture** Behavioral **of** tb\_Lab\_4\_Top **is**

--Constant Definitions

**CONSTANT** CP**:** TIME **:=** 10ns**;**--100 MHz clock period

**CONSTANT** DATA\_WIDTH**:** integer **:=** 8**;**

**CONSTANT** ADDRESS\_WIDTH**:** integer **:=** 6**;**

**CONSTANT** SUM\_WIDTH**:** integer **:=** 13**;**

--Signal Definitions

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

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

**signal** bram\_reset**,**Of\_det**:** std\_logic **:=** '0'**;** -- active low reset, overflow detection

**signal** avg**:** std\_logic\_vector **(**DATA\_WIDTH**-**1 **downto** 0**)** **:=** **(Others** **=>**'0'**);**

**signal** bad\_avg**:** std\_logic**:=**'0'**;**--indicates bad output; high when overflow occurs

--Design Components

**COMPONENT** Lab\_4\_Top **is**

**Generic(**

DATA\_W**:** INTEGER **:=** 8**;**

ADDR\_W**:** INTEGER **:=** 6**;**

SUM\_W**:** INTEGER **:=** 13**);**

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

bram\_on**:** **in** STD\_LOGIC**;**

Of\_det**:** **out** STD\_LOGIC**;**

average\_top**:** **out** STD\_LOGIC\_VECTOR**(**DATA\_W**-**1 **downto** 0**));**

**end** **COMPONENT;**

**begin**

uut**:** Lab\_4\_Top --Averaging circuit

**Generic** **Map(**DATA\_W **=>** DATA\_WIDTH**,**

ADDR\_W**=>** ADDRESS\_WIDTH**,**

SUM\_W**=>**SUM\_WIDTH**)**

**Port** **Map(** clk **=>** m\_clk**,**

bram\_on**=>**bram\_reset**,**

Of\_det**=>**Of\_det**,**

average\_top **=>**avg**);**

--Clock Processs

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

**begin**

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

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

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

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

**end** **process;**

bram\_r**:process** --Asynch reset

**begin**

**wait** **for** CP**\***2**;**--clear bram for 2 clock pulses to initalize a, sum and avg registers

bram\_reset**<=**'1'**;**

**wait;**

**end** **process;**

--Overflow Processes

Overflow**:process(**Of\_det**,**bram\_reset**)**

**begin**

**if(**bram\_reset**=**'1'**)** **then**

**if(**Of\_det**=**'1'**)** **then**

bad\_avg**<=**'1'**;**

**end** **if;**

**if(**Of\_det**=**'0' **AND** avg**=**0**)** **then**

bad\_avg**<=**'0'**;**

**end** **if;**

**end** **if;**

**end** **process;**

**end** Behavioral**;**

A.7 *Lab\_4\_FSM\_50.vhd*

**library** IEEE**;**

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

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

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

**entity** Lab\_4\_FSM **is**

**Port(**

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

i\_It50**:** **in** STD\_LOGIC**;** --FSM input; high if all 31 memory addresses have been addressed

avg\_clr**:** **out** STD\_LOGIC**;** --clear average result (avg) register

avg\_ld**:** **out** STD\_LOGIC**;** --load average result (avg) register (used for reset)

i\_clr**:** **out** STD\_LOGIC**;**--clear address index (i) register

i\_ld**:** **out** STD\_LOGIC**;**--load address index (i) register

sum\_clr**:** **out** STD\_LOGIC**;**--clear avg accumulator (sum) register

sum\_ld**:** **out** STD\_LOGIC**;**--load avg accumulator (sum) register (used for reset)

a\_clr**:** **out** STD\_LOGIC**;**--clear average input register (a) register

a\_ld**:** **out** STD\_LOGIC**);**--load average input register (a) register with value from BRAM

**end** Lab\_4\_FSM**;**

**architecture** Behavioral **of** Lab\_4\_FSM **is**

**type** state **is** **(**start**,**cleared**,**acc\_avg**,**compute**);**--state types: begin averaging

**signal** cur\_st**:** state **;** --current state, next state; defaults to start

**signal** nxt\_st**:** state**:=**start **;** --current state, next state; defaults to start

**signal** out\_count**:** STD\_LOGIC\_VECTOR**(**5 **downto** 0**)** **:=** std\_logic\_vector**(to\_unsigned(**50**,**6**));**--hold output when done to compute average

**signal** flag**,** sum\_clr\_flag**,**test**:** STD\_LOGIC **:=** '0'**;**

**begin**

Change\_st**:process(**clk**)** --get next state when clocked

**begin**

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

cur\_st **<=** nxt\_st**;**

**if(**cur\_st **=** acc\_avg **OR** cur\_st**=**compute**)** **then**

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

**else**

out\_count**<=** **(Others** **=>**'0'**);**

**end** **if;**

**end** **if;**

**end** **process;**

St\_ctrl**:** **process(**cur\_st**,** i\_It50**,**flag**)** --state controller

**begin**

**case** **(**cur\_st**)** **is**

--CLEAR

**when** cleared **=>**

nxt\_st **<=** acc\_avg**;**

--ACC\_AVG

**when** acc\_avg **=>**

**if(**i\_It50 **=** '1'**)then**

nxt\_st**<=** compute**;**

**else**

nxt\_st**<=** acc\_avg**;**

**end** **if;**

**when** compute **=>**

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

nxt\_st**<=**cleared**;**

**else**

nxt\_st**<=**compute**;**

**end** **if;**

**when** start **=>** --return from start

nxt\_st **<=** cleared**;**

**when** **others** **=>**

nxt\_st **<=** cleared**;**

**end** **case;**

**end** **process;**

Out\_ctrl**:process(**cur\_st**)**--output controller, sets control signals based on current stae value

**begin**

**case** **(**cur\_st**)** **is** ---Select output based on current state

--CLEAR

**when** cleared **=>**

avg\_clr**<=**'1'**;**--clear values

i\_clr**<=**'1'**;**

a\_clr**<=**'1'**;**

sum\_clr**<=**'1'**;**

--ACC\_AVG

**when** acc\_avg **=>**

**if(**i\_It50 **=** '1' **AND** sum\_clr\_flag**=**'0'**)then** --if all elements have been averaged

i\_ld**<=**'0'**;**--task is done, stop loading new values

a\_ld**<=**'0'**;**

**else** --if un-averaged elements remain

avg\_clr**<=**'0'**;**--load next values

avg\_ld**<=**'1'**;**

i\_clr**<=**'0'**;**

i\_ld**<=**'1'**;**

a\_clr**<=**'0'**;**

a\_ld**<=**'1'**;**

sum\_clr**<=**'0'**;**

sum\_ld**<=**'1'**;**

**end** **if;**

**when** compute **=>**

**when** start **=>**

i\_clr**<=**'1'**;**

a\_clr**<=**'1'**;**

**when** **others** **=>** --clear all regs on init/error

avg\_clr**<=**'0'**;**

i\_clr**<=**'0'**;**

a\_clr**<=**'0'**;**

sum\_clr**<=**'0'**;**

**end** **case;**

**end** **process;**

flag **<=** '1' **when** out\_count**=**"000111" **else**

'0'**;**

sum\_clr\_flag **<=** '1' **when** out\_count**=**"000111" **else**

'0'**;**

**end** Behavioral**;**

A.8 *Lab\_4\_FSM\_50.vhd*

**library** IEEE**;**

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

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

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

**entity** Lab\_4\_Avg **is**

**Generic(**

D\_WIDTH**:** INTEGER **:=** 8**;** --Data Width

ADDR\_WIDTH**:** INTEGER **:=** 6**;** --Address Width

SUM\_WIDTH**:** INTEGER **:=** 13**);** --Sum bit width; set by user

**Port(**

--CLOCK SIGNALS

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

--DATA I/0

M\_data**:** **in** STD\_LOGIC\_VECTOR**(**D\_WIDTH**-**1 **downto** 0**);** --avg data input, sent from BRAM

M\_addr**:** **out** STD\_LOGIC\_VECTOR**(**ADDR\_WIDTH**-**1 **downto** 0**);**--next element address; sent to BRAM

average**:** **out** STD\_LOGIC\_VECTOR**(**D\_WIDTH**-**1 **downto** 0**);** --computed average of N elements

sum\_of**:** **out** STD\_LOGIC**;** --detects overflow when computing average

--CONTROL SIGNALS

i\_It50**:** **out** STD\_LOGIC**;** --input to FSM; high if all 31 memory addresses have been addressed

avg\_clr**:** **in** STD\_LOGIC**;** --input from FSM; clear average result (avg) register

avg\_ld**:** **in** STD\_LOGIC**;** --input from FSM; load average result (avg) register (used for reset)

i\_clr**:** **in** STD\_LOGIC**;**--input from FSM; clear address index (i) register

i\_ld**:** **in** STD\_LOGIC**;**--input from FSM; load address index (i) register

sum\_clr**:** **in** STD\_LOGIC**;**--input from FSM; clear avg accumulator (sum) register

sum\_ld**:** **in** STD\_LOGIC**;**--input from FSM; load avg accumulator (sum) register (used for reset)

a\_clr**:** **in** STD\_LOGIC**;**--input from FSM; clear average input register (a) register

a\_ld**:** **in** STD\_LOGIC**);**--input from FSM; load average input register (a) register with value from BRAM

**end** Lab\_4\_Avg**;**

**architecture** Behavioral **of** Lab\_4\_Avg **is**

**CONSTANT** START\_ADDR**:** STD\_LOGIC\_VECTOR**(**ADDR\_WIDTH**-**1 **downto** 0**)** **:=** std\_logic\_vector**(to\_unsigned(**14**,**ADDR\_WIDTH**));** --starting address of 25

**CONSTANT** NXT\_i**:** STD\_LOGIC\_VECTOR**(**ADDR\_WIDTH**-**1 **downto** 0**)** **:=** std\_logic\_vector**(to\_unsigned(**1**,**ADDR\_WIDTH**));** --amount to incremt each indicie by

**CONSTANT** FIFTY**:** INTEGER**:=**50**;**

**signal** fifty\_bit**:** STD\_LOGIC\_VECTOR**(**D\_WIDTH**-**1 **DOWNTO** 0**)** **:=** std\_logic\_vector**(to\_unsigned(**FIFTY**,**D\_WIDTH**));**--constant signal used for division

**signal** sum\_16\_bit**:** STD\_LOGIC\_VECTOR**(**15 **DOWNTO** 0**)** **:=** **(Others** **=>**'0'**);**--16 bit signal used to implement divide by 50 average

**signal** computed\_avg**:** STD\_LOGIC\_VECTOR**(**23 **DOWNTO** 0**)** **:=** **(OTHERS** **=>**'0'**);**

**signal** i\_count**:** STD\_LOGIC\_VECTOR**(**ADDR\_WIDTH**-**1 **downto** 0**)** **:=** **(Others** **=>**'0'**);** --counts memory indicies during averaging

**signal** a\_out**:** STD\_LOGIC\_VECTOR**(**D\_WIDTH**-**1 **downto** 0**)** **:=** **(Others** **=>**'0'**);**--data read from a reg

**signal** sum\_in**,**sum\_out**:** STD\_LOGIC\_VECTOR**(**SUM\_WIDTH **downto** 0**)** **:=** **(Others** **=>**'0'**);**--data read into sum reg from adder,

--data read from sum reg to adder

**signal** i\_in**,**i\_out**:** STD\_LOGIC\_VECTOR**(**ADDR\_WIDTH**-**1 **downto** 0**)** **:=** **(Others** **=>**'0'**);**--data read from counter to index register

**signal** avg\_in**,**avg\_out**:** STD\_LOGIC\_VECTOR**(**D\_WIDTH**-**1 **downto** 0**)** **:=** **(Others** **=>**'0'**);** --data into address register

**signal** t\_val\_in**,**t\_val\_out**:** STD\_LOGIC **:=** '0'**;** --used by div\_gen\_0

**signal** clear\_div**:** STD\_LOGIC**:=**'0'**;**--clears div\_gen inputs on init; internal active low asynchronous reset

**COMPONENT** div\_gen\_0 **IS** --Divider Generator 5.1 IP BLOCK by XILNIX

--https://www.xilinx.com/support/documentation/ip\_documentation/div\_gen/v5\_1/pg151-div-gen.pdf

**PORT** **(**

aclk **:** **IN** STD\_LOGIC**;**

aresetn **:** **IN** STD\_LOGIC**;**

s\_axis\_divisor\_tvalid **:** **IN** STD\_LOGIC**;**

s\_axis\_divisor\_tdata **:** **IN** STD\_LOGIC\_VECTOR**(**7 **DOWNTO** 0**);**

s\_axis\_dividend\_tvalid **:** **IN** STD\_LOGIC**;**

s\_axis\_dividend\_tdata **:** **IN** STD\_LOGIC\_VECTOR**(**15 **DOWNTO** 0**);**

m\_axis\_dout\_tvalid **:** **OUT** STD\_LOGIC**;**

m\_axis\_dout\_tdata **:** **OUT** STD\_LOGIC\_VECTOR**(**23 **DOWNTO** 0**)**

**);**

**END** **COMPONENT;**

**begin**

div\_by\_50**:**div\_gen\_0

**PORT** **MAP** **(**

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

aresetn **=>**clear\_div**,**

s\_axis\_divisor\_tvalid**=>**t\_val\_in**,**

s\_axis\_divisor\_tdata **=>** fifty\_bit**,**

s\_axis\_dividend\_tvalid**=>**t\_val\_in**,**

s\_axis\_dividend\_tdata **=>**sum\_16\_bit**,**

m\_axis\_dout\_tvalid**=>**t\_val\_out**,**

m\_axis\_dout\_tdata **=>** computed\_avg**);**

a\_ld\_clr**:** **process** **(**clk**,**a\_clr**,**a\_ld**,**clear\_div**)**

**begin**

**if(**a\_clr**=**'1' **OR** clear\_div**=**'0'**)** **then**

a\_out**<=** **(Others** **=>** '0'**);**

**end** **if;**

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

**if(**a\_ld**=**'1'**)** **then**

a\_out**<=**M\_data**;** --load next value form BRAM

**end** **if;**

**end** **if;**

**end** **process;**

sum\_ld\_clr**:** **process** **(**clk**,**sum\_clr**,**sum\_ld**)**

**begin**

**if(**sum\_clr**=**'1'**)** **then**

sum\_out**<=** **(Others** **=>** '0'**);**

**end** **if;**

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

**if(**sum\_ld**=**'1'**)** **then**

sum\_out**<=**sum\_in**;** --load next value form BRAM

**end** **if;**

**end** **if;**

**end** **process;**

i\_ld\_clr**:** **process** **(**clk**,**i\_clr**,**i\_ld**)**

**begin**

**if(**i\_clr**=**'1'**)** **then**

i\_out**<=** **(Others** **=>** '0'**);**

**end** **if;**

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

**if(**i\_ld**=**'1'**)** **then**

i\_out**<=**i\_in**;** --load next value form BRAM

**end** **if;**

**end** **if;**

**end** **process;**

avg\_ld\_clr**:** **process** **(**clk**,**avg\_clr**,**avg\_ld**)**

**begin**

**if(**avg\_clr**=**'1'**)** **then**

avg\_out**<=** **(Others** **=>** '0'**);**

**end** **if;**

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

**if(**avg\_ld**=**'1'**)** **then**

avg\_out**<=**avg\_in**;** --load next value form BRAM

**end** **if;**

**end** **if;**

**end** **process;**

--OUTPUT LOGIC

average**<=**avg\_out**;**

M\_addr**<=**i\_out**+**START\_ADDR**;**--output next element address

i\_It50**<=** '1' **when** i\_out**>**std\_logic\_vector**(to\_unsigned(**50**,**ADDR\_WIDTH**))** **else**

'0'**;**

--SIGNAL LOGIC

--SUM\_IN

sum\_in**<=**sum\_out**+**a\_out**;**

sum\_of**<=**sum\_out**(**SUM\_WIDTH**);**--MSB == Overflow detection

--DIVIDEND

sum\_16\_bit**(**SUM\_WIDTH**-**1 **downto** 0**)<=**sum\_out**(**SUM\_WIDTH**-**1 **downto** 0**);**

--AVG\_IN

avg\_in**<=** computed\_avg**(**15 **downto** 8**);** --integer component of average from

--I\_IN

i\_in**<=**i\_out**+**NXT\_i**;**

--clear divisor register on computed average

clear\_div**<=**'0' **when** i\_out **=** START\_ADDR **else**

'1'**;**

**end** Behavioral**;**