Due Date: Sunday October 27, 11:59pm– Submit this editable **Word** document NOT pdf

0 / 32

EEDG/CE 6370
Design and Analysis of Reconfigurable Systems
Homework 6
High-Level Synthesis with CyberWorkBench

Student Names

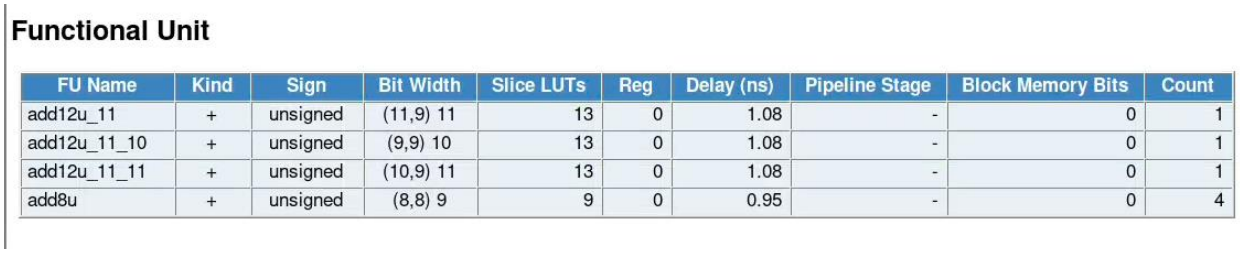
Name – Prathamesh Sanjay Gadad

NetID – psg220003

PART 1 – ave8.c Synthesis

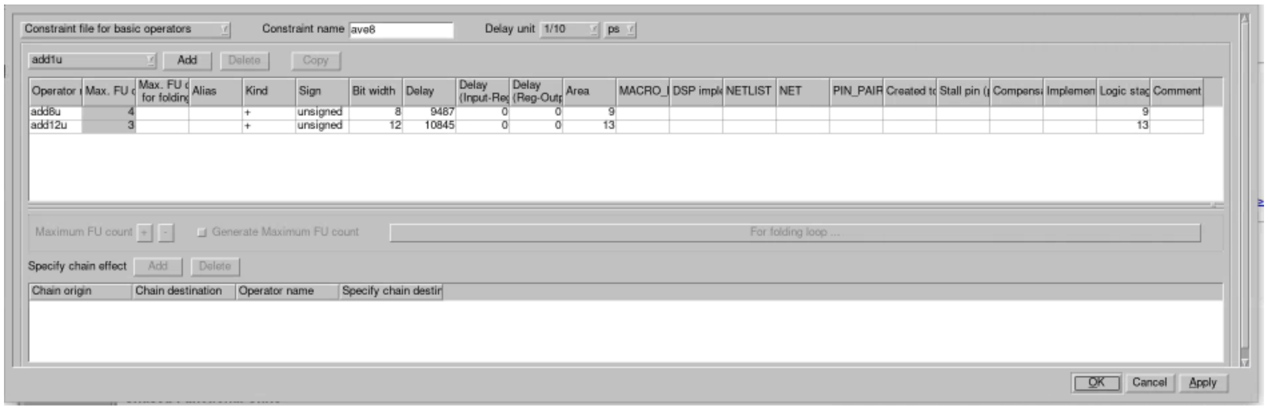
a.) Synthesize the ave8.c design. Annotate from the resource constraint file (FCNT) the number and type of Functional Units (FUs) used (include screenshots from the reports). Explain what type and why these FUs are needed. Create a short YouTube video showing the process of creating the CWB project and synthesis.

Marks
6



Functional Unit

FU Name	Kind	Sign	Bit Width	Slice LUTs	Reg	Delay (ns)	Pipeline Stage	Block Memory Bits	Count
add12u_11	+	unsigned	(11,9) 11	13	0	1.08	-	0	1
add12u_11_10	+	unsigned	(9,9) 10	13	0	1.08	-	0	1
add12u_11_11	+	unsigned	(10,9) 11	13	0	1.08	-	0	1
add8u	+	unsigned	(8,8) 9	9	0	0.95	-	0	4



Constraint file for basic operators — Constraint name: ave8 — Delay unit: 1/10 ps

add1u

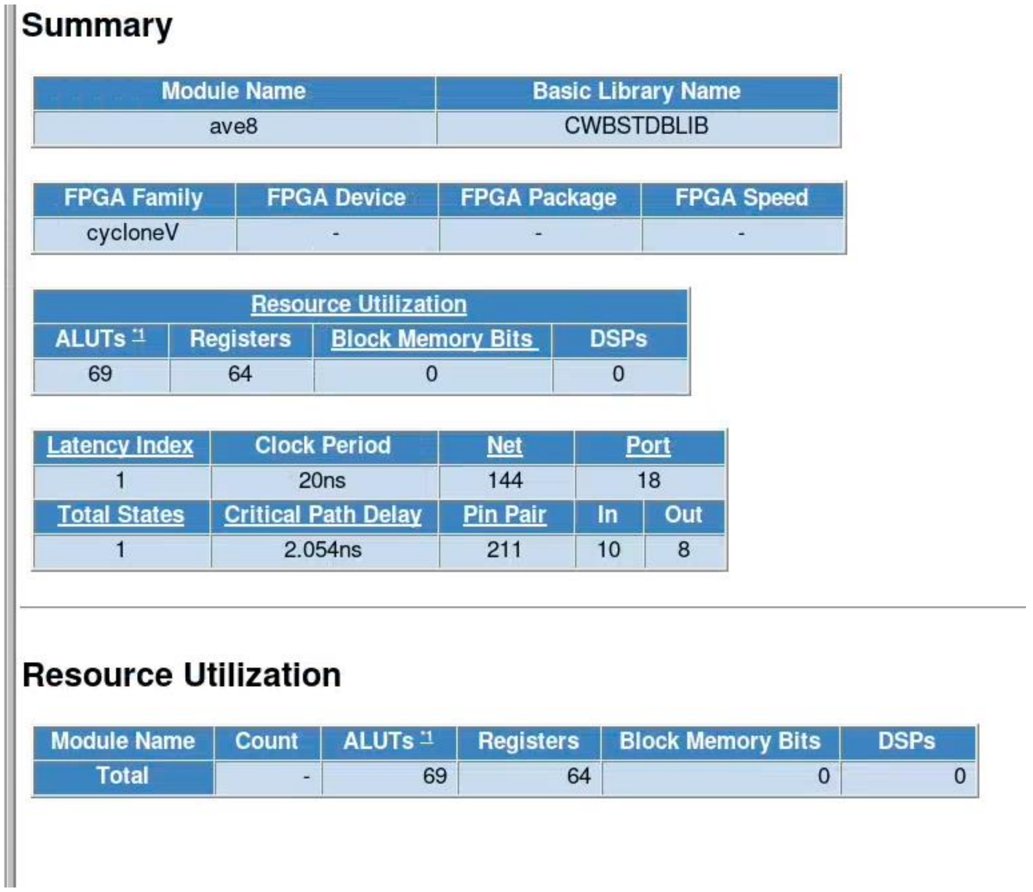
Operator	Max. FU	Max. FU for folding	Alias	Kind	Sign	Bit width	Delay	Delay (Input-Reg	Delay (Reg-Outp	Area	MACRO_I	DSP impl	NETLIST	NET	PIN_PAIR	Created to	Stall pin	Compens	Implemen	Logic stag	Comment
add8u	4			+	unsigned	8	9487	0	0	9										9	
add12u	3			+	unsigned	12	10845	0	0	13										13	

YouTube Video link: https://youtu.be/NMbRhlsUvGw

Functional Units (FUs) are required to perform the shifting operation. In hardware, dividing by 8 involves shifting three times. Since the input is 8-bit, 8-bit adders are necessary. Adding two 8-bit numbers produces a 9-bit result, and likewise, adding two 9-bit numbers results in a 10-bit output. To handle this, we will need a few 12-bit adders. Using these adders, we can successfully achieve the desired output.

b.) Report from the QoR file the size of the circuit in terms of number of LUTs and registers. Report also the latency of the synthesized circuit and the critical path and maximum frequency. Include screenshot.

Marks
2



Summary

Module Name	Basic Library Name
ave8	CWBSTDBLIB

FPGA Family	FPGA Device	FPGA Package	FPGA Speed
cycloneV	-	-	-

Resource Utilization			
ALUTs [*1]	Registers	Block Memory Bits	DSPs
69	64	0	0

Latency Index	Clock Period	Net	Port	
1	20ns	144	18	
Total States	Critical Path Delay	Pin Pair	In	Out
1	2.054ns	211	10	8

Resource Utilization

Module Name	Count	ALUTs [*1]	Registers	Block Memory Bits	DSPs
Total	-	69	64	0	0

Maximum Frequency = 1/2.054^{-10} = 486.85 MHz c.) Perform Logic synthesis using Quartus Prime and compare the area results in terms of LUTs. You might call Quartus from within CWB as shown in the lab sheet or manually create a project in Quartus and include the RTL file generated by CWB, and an SDC file. Discuss if they match or not and why they do/don't. Shown Quartus results screenshots.

Marks
4

RTL File generated by CWB

```
// verilog_out version 6.89.1
// options:  veriloggen -EE ave8_E.IFF
// bdlpars options:  -EE -I../.. -I/proj/cad/cwb-6.1/linux_x86_64/include -
info_base_name ave8_sw ../../ave8_sw.c -process=ave8
// bdltran options:  -EE -c2000 -s -Zresource_fcnt=GENERATE -
Zresource_mcnt=GENERATE -Zdup_reset=YES -Zfolding_sharing=inter_stage -EE -lb
/proj/cad/cwb-6.1/packages/cycloneV.BLIB -lfl /proj/cad/cwb-
6.1/packages/cycloneV.FLIB +lfl ave8-auto.FLIB +lfl ave8-amacro-auto.FLIB -lfc
ave8-auto.FCNT +lfc ave8-amacro-auto.FCNT -lml ave8-auto.MLIB -lmc ave8-
auto.MCNT ave8.IFF
// timestamp_0: 20241026175117_05843_42761
// timestamp_5: 20241026175118_05934_15518
// timestamp_9: 20241026175119_05934_82456
// timestamp_C: 20241026175119_05934_88733
// timestamp_E: 20241026175119_05934_92381
// timestamp_V: 20241026175120_05987_89016

module ave8 ( in0 ,ave8_ret ,CLOCK ,RESET );
input [0:7] in0 ; // line#=../../ave8_sw.c:20
output      [0:7] ave8_ret ;  // line#=../../ave8_sw.c:20
input       CLOCK ;
input       RESET ;
wire  [0:8] add12u_11_101i2 ;
wire  [0:8] add12u_11_101i1 ;
wire  [0:9] add12u_11_101ot ;
wire  [0:8] add12u_11_111i2 ;
wire  [0:9] add12u_11_111i1 ;
wire  [0:10]      add12u_11_111ot ;
wire  [0:8] add12u_111i2 ;
wire  [0:10]      add12u_111i1 ;
wire  [0:10]      add12u_111ot ;
wire  [0:7] add8u4i2 ;
wire  [0:7] add8u4i1 ;
wire  [0:8] add8u4ot ;
wire  [0:7] add8u3i2 ;
wire  [0:7] add8u3i1 ;
wire  [0:8] add8u3ot ;
wire  [0:7] add8u2i2 ;
wire  [0:7] add8u2i1 ;
wire  [0:8] add8u2ot ;
wire  [0:7] add8u1i2 ;
wire  [0:7] add8u1i1 ;
wire  [0:8] add8u1ot ;
reg   [0:7] RG_buffer ; // line#=../../ave8_sw.c:16
reg   [0:7] RG_buffer_1 ;     // line#=../../ave8_sw.c:16
reg   [0:7] RG_buffer_2 ;     // line#=../../ave8_sw.c:16
reg   [0:7] RG_buffer_3 ;     // line#=../../ave8_sw.c:16
reg   [0:7] RG_buffer_4 ;     // line#=../../ave8_sw.c:16
reg   [0:7] RG_buffer_5 ;     // line#=../../ave8_sw.c:16
reg   [0:7] RG_buffer_6 ;     // line#=../../ave8_sw.c:16
reg   [0:7] ave8_ret_r ;      // line#=../../ave8_sw.c:20

ave8_add12u_11_10 INST_add12u_11_10_1
( .i1(add12u_11_101i1) ,.i2(add12u_11_101i2) ,
      .o1(add12u_11_101ot) ); // line#=../../ave8_sw.c:39
ave8_add12u_11_11 INST_add12u_11_11_1
( .i1(add12u_11_111i1) ,.i2(add12u_11_111i2) ,
      .o1(add12u_11_111ot) ); // line#=../../ave8_sw.c:39
ave8_add12u_11 INST_add12u_11_1
( .i1(add12u_111i1) ,.i2(add12u_111i2) ,.o1(add12u_111ot) );      //
line#=../../ave8_sw.c:28,35,39
ave8_add8u INST_add8u_1 ( .i1(add8u1i1) ,.i2(add8u1i2) ,.o1(add8u1ot) );
      // line#=../../ave8_sw.c:28,35,39
ave8_add8u INST_add8u_2 ( .i1(add8u2i1) ,.i2(add8u2i2) ,.o1(add8u2ot) );
      // line#=../../ave8_sw.c:39
ave8_add8u INST_add8u_3 ( .i1(add8u3i1) ,.i2(add8u3i2) ,.o1(add8u3ot) );
      // line#=../../ave8_sw.c:39
ave8_add8u INST_add8u_4 ( .i1(add8u4i1) ,.i2(add8u4i2) ,.o1(add8u4ot) );
      // line#=../../ave8_sw.c:39
assign      ave8_ret = ave8_ret_r ; // line#=../../ave8_sw.c:20
assign      add8u2i1 = RG_buffer_4 ;      // line#=../../ave8_sw.c:28,39
assign      add8u2i2 = RG_buffer_5 ;      // line#=../../ave8_sw.c:28,39
assign      add12u_11_111i1 = add12u_11_101ot ; // line#=../../ave8_sw.c:39
assign      add12u_11_111i2 = add8u2ot ;  // line#=../../ave8_sw.c:39
always @ ( posedge CLOCK or posedge RESET )     //
line#=../../ave8_sw.c:28,35,39,43,46
      if ( RESET )
            ave8_ret_r <= 8'h00 ;
      else
            ave8_ret_r <= add12u_111ot [0:7] ;
always @ ( posedge CLOCK or posedge RESET )     // line#=../../ave8_sw.c:32
      if ( RESET )
            RG_buffer <= 8'h00 ;
      else
            RG_buffer <= in0 ;
always @ ( posedge CLOCK or posedge RESET )     // line#=../../ave8_sw.c:28
      if ( RESET )
            RG_buffer_1 <= 8'h00 ;
      else
            RG_buffer_1 <= RG_buffer ;
always @ ( posedge CLOCK or posedge RESET )     // line#=../../ave8_sw.c:28
      if ( RESET )
            RG_buffer_2 <= 8'h00 ;
      else
            RG_buffer_2 <= RG_buffer_1 ;
```

```
always @ ( posedge CLOCK or posedge RESET )     // line#=../../ave8_sw.c:28
      if ( RESET )
            RG_buffer_3 <= 8'h00 ;
      else
            RG_buffer_3 <= RG_buffer_2 ;
always @ ( posedge CLOCK or posedge RESET )     // line#=../../ave8_sw.c:28
      if ( RESET )
            RG_buffer_4 <= 8'h00 ;
      else
            RG_buffer_4 <= RG_buffer_3 ;
always @ ( posedge CLOCK or posedge RESET )     // line#=../../ave8_sw.c:28
      if ( RESET )
            RG_buffer_5 <= 8'h00 ;
      else
            RG_buffer_5 <= RG_buffer_4 ;
always @ ( posedge CLOCK or posedge RESET )     // line#=../../ave8_sw.c:28
      if ( RESET )
            RG_buffer_6 <= 8'h00 ;
      else
            RG_buffer_6 <= RG_buffer_5 ;
assign      add8u1i1 = RG_buffer_6 ;      // line#=../../ave8_sw.c:28,35,39
assign      add8u1i2 = in0 ;  // line#=../../ave8_sw.c:28,35,39
assign      add12u_111i1 = add12u_11_111ot ;    //
line#=../../ave8_sw.c:28,35,39
assign      add12u_111i2 = add8u1ot ;     // line#=../../ave8_sw.c:28,35,39
assign      add8u3i1 = RG_buffer_2 ;      // line#=../../ave8_sw.c:28,39
assign      add8u3i2 = RG_buffer_3 ;      // line#=../../ave8_sw.c:28,39
assign      add8u4i1 = RG_buffer ;  // line#=../../ave8_sw.c:28,39
assign      add8u4i2 = RG_buffer_1 ;      // line#=../../ave8_sw.c:28,39
assign      add12u_11_101i1 = add8u4ot ;  // line#=../../ave8_sw.c:39
assign      add12u_11_101i2 = add8u3ot ;  // line#=../../ave8_sw.c:39

endmodule

module ave8_add12u_11_10 ( i1 ,i2 ,o1 );
input [0:8] i1 ;
input [0:8] i2 ;
output      [0:9] o1 ;

assign      o1 = ( { 1'h0 , i1 } + { 1'h0 , i2 } ) ;

endmodule

module ave8_add12u_11_11 ( i1 ,i2 ,o1 );
input [0:9] i1 ;
input [0:8] i2 ;
output      [0:10]      o1 ;

assign      o1 = ( { 1'h0 , i1 } + { 2'h0 , i2 } ) ;

endmodule

module ave8_add12u_11 ( i1 ,i2 ,o1 );
input [0:10]      i1 ;
input [0:8] i2 ;
output      [0:10]      o1 ;

assign      o1 = ( i1 + { 2'h0 , i2 } ) ;

endmodule

module ave8_add8u ( i1 ,i2 ,o1 );
input [0:7] i1 ;
input [0:7] i2 ;
output      [0:8] o1 ;

assign      o1 = ( { 1'h0 , i1 } + { 1'h0 , i2 } ) ;

endmodule
```

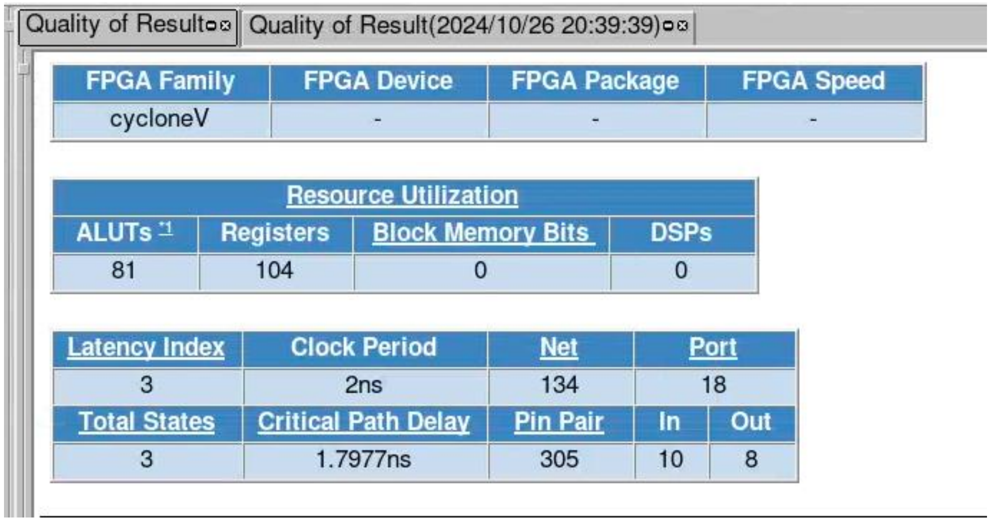
SDC File generated by CWB

create_clock CLOCK -name CLOCK -period 20.000000

The results differ because the report from CWB provides the overall resources needed for the design without optimization, whereas Quartus Prime reports the required resources after optimization has been applied.

d.) What is the latency (in clock cycles) if you set the target synthesis frequency to 500 MHz (in project properties). Include screenshot. Explain why.

Marks
2



Quality of Result | Quality of Result(2024/10/26 20:39:39)

FPGA Family	FPGA Device	FPGA Package	FPGA Speed
cycloneV	-	-	-

Resource Utilization			
ALUTs *1	Registers	Block Memory Bits	DSPs
81	104	0	0

Latency Index	Clock Period	Net	Port	
3	2ns	134	18	
Total States	Critical Path Delay	Pin Pair	In	Out
3	1.7977ns	305	10	8

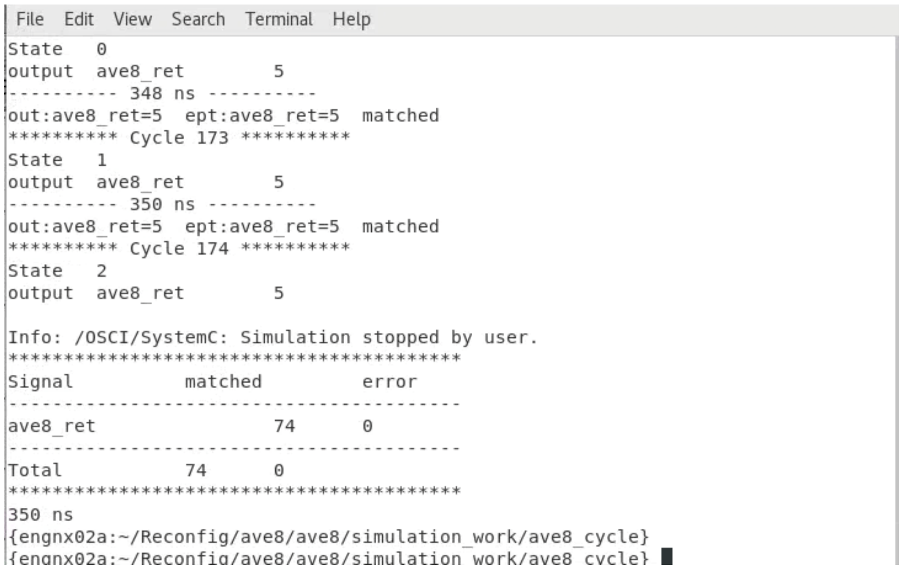
As we increase the clock frequency the clock period gets reduced and hence the latency will increase since the operations cannot be complete in a single step and hence the graph slides the operations into next step.

PART 2 – ave8.c Verification

a.) Perform a cycle-accurate simulation using the untimed test vectors used for the software simulation and make sure that the simulation outputs match for the two versions with HLS frequency 100MHz and 500Mhz. Show the result (paste console window)

Marks
4

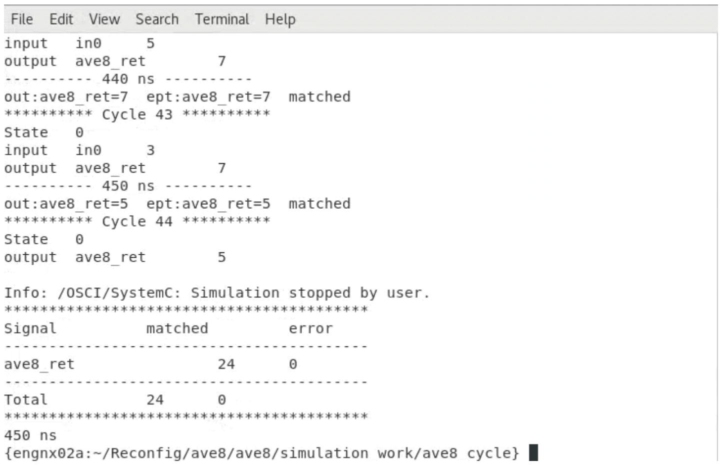
For 500 MHz Simulation


File Edit View Search Terminal Help

```
State   0
output  ave8_ret        5
---------- 348 ns ----------
out:ave8_ret=5  ept:ave8_ret=5  matched
********** Cycle 173 **********
State   1
output  ave8_ret        5
---------- 350 ns ----------
out:ave8_ret=5  ept:ave8_ret=5  matched
********** Cycle 174 **********
State   2
output  ave8_ret        5

Info: /OSCI/SystemC: Simulation stopped by user.
*****************************************
Signal          matched         error
-----------------------------------------
ave8_ret                74      0
-----------------------------------------
Total           74      0
*****************************************
350 ns
{engnx02a:~/Reconfig/ave8/ave8/simulation_work/ave8_cycle}
{engnx02a:~/Reconfig/ave8/ave8/simulation_work/ave8_cycle} █
```

For 100 MHz Simulation



```
input   in0     5
output  ave8_ret        7
---------- 440 ns ----------
out:ave8_ret=7  ept:ave8_ret=7  matched
********** Cycle 43 **********
State   0
input   in0     3
output  ave8_ret        7
---------- 450 ns ----------
out:ave8_ret=5  ept:ave8_ret=5  matched
********** Cycle 44 **********
State   0
output  ave8_ret        5

Info: /OSCI/SystemC: Simulation stopped by user.
*****************************************
Signal          matched         error
-----------------------------------------
ave8_ret                24      0
-----------------------------------------
Total           24      0
*****************************************
450 ns
{engnx02a:~/Reconfig/ave8/ave8/simulation work/ave8 cycle}
```

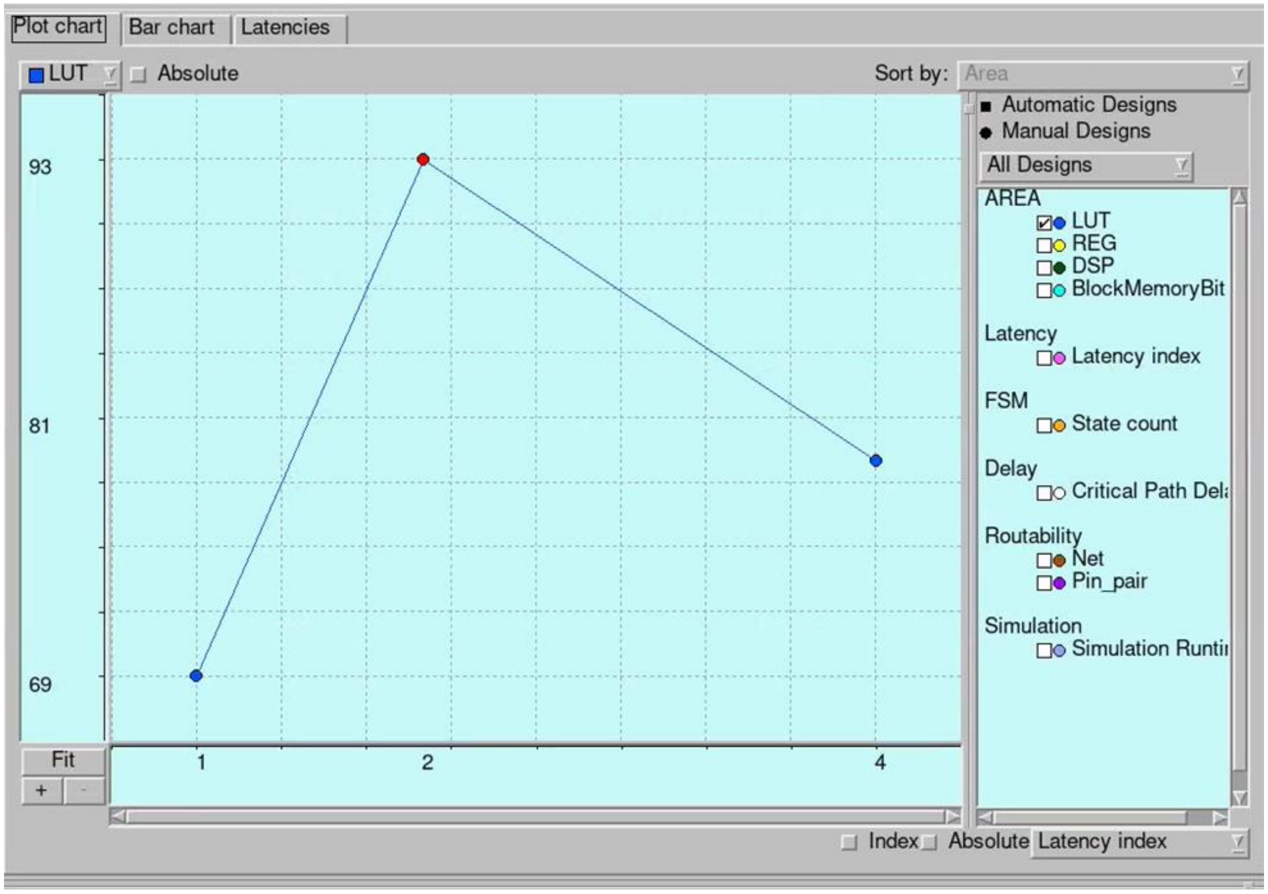
At 100 MHz, since the latency is 1, the outputs are correctly matched. However, at 500 MHz, the latency increases to 3, and a Finite State Machine (FSM) is generated. In this case, the output of the current state depends on the output of the previous state, leading to only a few matches, while the rest result in errors.

PART 3 ave8.c Design Space Exploration

a.) Reduce the number of FUs in the Resource Constraint file (FCNT) from the maximum number obtained in the initial design (50Mhz) to half that number and to only 1 FU. Annotate the area in terms if LUTs and the latency of each of the 3 designs in a table. **Plot** the graph of **area vs. latency** of the 3 designs (y-axis area, x-axis latency). Discuss the results. Are they what you expected? (Yes/No)

Marks
6

Design	ALUTs	Latency	Critical path
Max FUs	69	1	2.054ns
Half FUs	93	2	4.0816ns
One single FU	79	4	3.5975ns

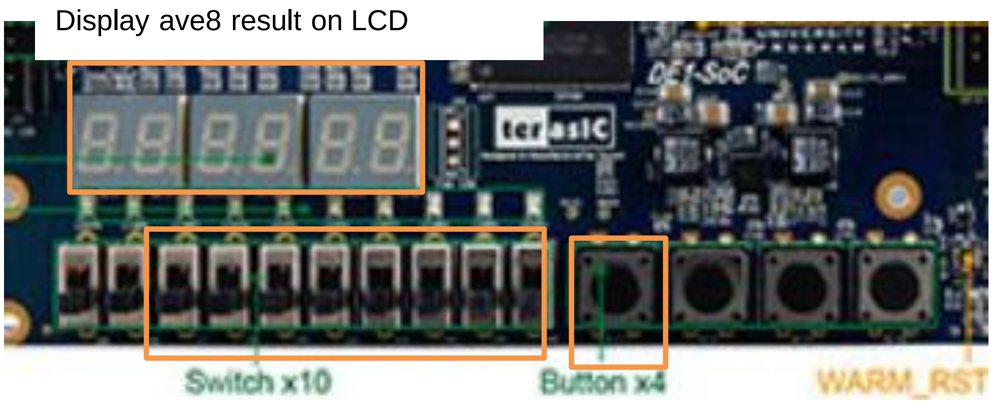


The results are as expected because, in the Max FUs Design, the design uses the exact number of resources and implements accordingly. In the Half FUs Design, due to a shortage of resources, a 2-stage FSM is created and implemented. Similarly, in the One Single FU Design, the design lacks sufficient resources, so a 4-stage FSM is generated and implemented.

PART 4 FPGA Prototyping

Prototype the design on the DE1-SoC board such that the switches act as a new unsigned 8-bit value to be read into the circuit each time that a button is pressed. The average of the last 8 values is displayed on the 7-segment displays as shown. Add, to the report the new synthesizable C code and create a **YouTube video** showing that the design works.

Marks
8



Include the modified ave8.c file that follows the specifications indicated and that displays the results on the LCD display and a video showing that the design is working.

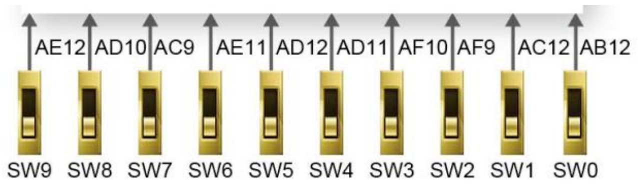


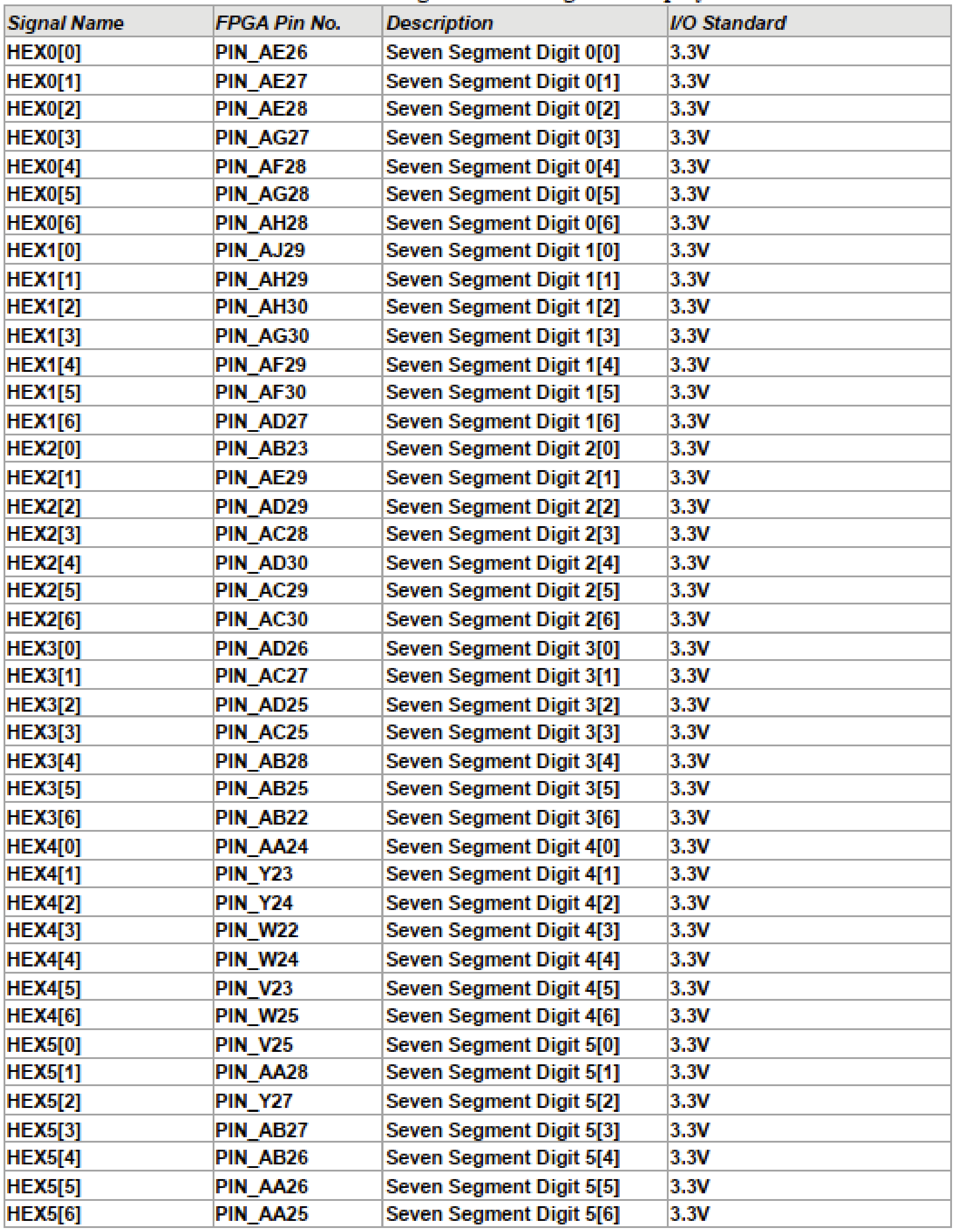
Table 3-6 Pin Assignment of Slide Switches

Signal Name	*FPGA Pin No.*	*Description*	*I/O Standard*
SW[0]	PIN_AB12	Slide Switch[0]	3.3V
SW[1]	PIN_AC12	Slide Switch[1]	3.3V
SW[2]	PIN_AF9	Slide Switch[2]	3.3V
SW[3]	PIN_AF10	Slide Switch[3]	3.3V
SW[4]	PIN_AD11	Slide Switch[4]	3.3V
SW[5]	PIN_AD12	Slide Switch[5]	3.3V
SW[6]	PIN_AE11	Slide Switch[6]	3.3V
SW[7]	PIN_AC9	Slide Switch[7]	3.3V
SW[8]	PIN_AD10	Slide Switch[8]	3.3V
SW[9]	PIN_AE12	Slide Switch[9]	3.3V

Table 3-7 Pin Assignment of Push-buttons

Signal Name	*FPGA Pin No.*	*Description*	*I/O Standard*
KEY[0]	PIN_AA14	Push-button[0]	3.3V
KEY[1]	PIN_AA15	Push-button[1]	3.3V
KEY[2]	PIN_W15	Push-button[2]	3.3V
KEY[3]	PIN_Y16	Push-button[3]	3.3V

Table 3-9 Pin Assignment of 7-segment Displays



Signal Name	*FPGA Pin No.*	*Description*	*I/O Standard*
HEX0[0]	PIN_AE26	Seven Segment Digit 0[0]	3.3V
HEX0[1]	PIN_AE27	Seven Segment Digit 0[1]	3.3V
HEX0[2]	PIN_AE28	Seven Segment Digit 0[2]	3.3V
HEX0[3]	PIN_AG27	Seven Segment Digit 0[3]	3.3V
HEX0[4]	PIN_AF28	Seven Segment Digit 0[4]	3.3V
HEX0[5]	PIN_AG28	Seven Segment Digit 0[5]	3.3V
HEX0[6]	PIN_AH28	Seven Segment Digit 0[6]	3.3V
HEX1[0]	PIN_AJ29	Seven Segment Digit 1[0]	3.3V
HEX1[1]	PIN_AH29	Seven Segment Digit 1[1]	3.3V
HEX1[2]	PIN_AH30	Seven Segment Digit 1[2]	3.3V
HEX1[3]	PIN_AG30	Seven Segment Digit 1[3]	3.3V
HEX1[4]	PIN_AF29	Seven Segment Digit 1[4]	3.3V
HEX1[5]	PIN_AF30	Seven Segment Digit 1[5]	3.3V
HEX1[6]	PIN_AD27	Seven Segment Digit 1[6]	3.3V
HEX2[0]	PIN_AB23	Seven Segment Digit 2[0]	3.3V
HEX2[1]	PIN_AE29	Seven Segment Digit 2[1]	3.3V
HEX2[2]	PIN_AD29	Seven Segment Digit 2[2]	3.3V
HEX2[3]	PIN_AC28	Seven Segment Digit 2[3]	3.3V
HEX2[4]	PIN_AD30	Seven Segment Digit 2[4]	3.3V
HEX2[5]	PIN_AC29	Seven Segment Digit 2[5]	3.3V
HEX2[6]	PIN_AC30	Seven Segment Digit 2[6]	3.3V
HEX3[0]	PIN_AD26	Seven Segment Digit 3[0]	3.3V
HEX3[1]	PIN_AC27	Seven Segment Digit 3[1]	3.3V
HEX3[2]	PIN_AD25	Seven Segment Digit 3[2]	3.3V
HEX3[3]	PIN_AC25	Seven Segment Digit 3[3]	3.3V
HEX3[4]	PIN_AB28	Seven Segment Digit 3[4]	3.3V
HEX3[5]	PIN_AB25	Seven Segment Digit 3[5]	3.3V
HEX3[6]	PIN_AB22	Seven Segment Digit 3[6]	3.3V
HEX4[0]	PIN_AA24	Seven Segment Digit 4[0]	3.3V
HEX4[1]	PIN_Y23	Seven Segment Digit 4[1]	3.3V
HEX4[2]	PIN_Y24	Seven Segment Digit 4[2]	3.3V
HEX4[3]	PIN_W22	Seven Segment Digit 4[3]	3.3V
HEX4[4]	PIN_W24	Seven Segment Digit 4[4]	3.3V
HEX4[5]	PIN_V23	Seven Segment Digit 4[5]	3.3V
HEX4[6]	PIN_W25	Seven Segment Digit 4[6]	3.3V
HEX5[0]	PIN_V25	Seven Segment Digit 5[0]	3.3V
HEX5[1]	PIN_AA28	Seven Segment Digit 5[1]	3.3V
HEX5[2]	PIN_Y27	Seven Segment Digit 5[2]	3.3V
HEX5[3]	PIN_AB27	Seven Segment Digit 5[3]	3.3V
HEX5[4]	PIN_AB26	Seven Segment Digit 5[4]	3.3V
HEX5[5]	PIN_AA26	Seven Segment Digit 5[5]	3.3V
HEX5[6]	PIN_AA25	Seven Segment Digit 5[6]	3.3V

YouTube Video: https://youtu.be/ElvDrV8zdJA

C Code for Moving Average

```
/***********************************************
**
**   ave8.c
**
** Description: The following program computes the average
**                      of the 8 numbers being read
**
************************************************/
#ifdef C
#include "stdio.h"
#include "stdlib.h"
#endif


/* Global variables */
unsigned char storeArray[8]  = {0, 0, 0, 0, 0, 0, 0, 0};


  // Cyber func=process
  unsigned long int ave8(unsigned char in0, bool toggle){

   /* Local variables declaration */
    int  movingAverage, sum,  i;
    unsigned long int delay = 10000000;
    unsigned long int d;
    unsigned long int mappedSegment;
    unsigned long int toDisplay = 0b11111111111111111111111111111111;

 /* Shift data to accommodate new input to be read */
    if (toggle == 0) {
        for (i = 7; i > 0; i--) {
            storeArray[i] = storeArray[i- 1];
            // Cyber scheduling_block
            for (d=0; d<delay; d++) {
          $
        }
    }

  /* Read new input into buffer */
    storeArray[0] = in0;
  }
  /* Set first element of sum to compute the average => can save 1 loop
iteration */
    sum= storeArray[0];

   /* Add up all the numbers in the buffer */
      for (i= 1; i< 8; i++) {
            sum += storeArray[i];
        }
 /* Compute the average by dividing by 8 -> In HW  a divide by 8 (/8) = shift
3 times */
       movingAverage = sum / 8;
 /*Switch Case for Seven Segment */
    switch (movingAverage) {
      case 0   : mappedSegment = 0b00000011111111111111111111111111;
      break;
      case 1   : mappedSegment = 0b10011111111111111111111111111111;
      break;
      case 2   : mappedSegment = 0b00100101111111111111111111111111;
      break;
      case 3   : mappedSegment = 0b00001101111111111111111111111111;
      break;
      case 4   : mappedSegment = 0b10011001111111111111111111111111;
      break;
      case 5   : mappedSegment = 0b01001001111111111111111111111111;
      break;
      case 6   : mappedSegment = 0b01000001111111111111111111111111;
      break;
      case 7   : mappedSegment = 0b00011111111111111111111111111111;
      break;
      case 8   : mappedSegment = 0b00000001111111111111111111111111;
      break;
      case 9   : mappedSegment = 0b00001001111111111111111111111111;
      break;
      case 10 : mappedSegment = 0b00000011001111111111111111111111;
      break;
      case 12 : mappedSegment = 0b00100101001111111111111111111111;
      break;
      case 14 : mappedSegment = 0b10011001001111111111111111111111;
      break;
      case 16 : mappedSegment = 0b01000001001111111111111111111111;
      break;
      case 18 : mappedSegment = 0b00000001001111111111111111111111;
      break;
      case 20 : mappedSegment = 0b00000010010010111111111111111111;
      break;
      case 22 : mappedSegment = 0b00100100010010111111111111111111;
      break;
      case 24 : mappedSegment = 0b10011000010010111111111111111111;
      break;
      case 26 : mappedSegment = 0b01000000010010111111111111111111;
      break;
      case 28 : mappedSegment = 0b00000000010010111111111111111111;
      break;
      case 30 : mappedSegment = 0b00000010000110111111111111111111;
      break;
      case 32 : mappedSegment = 0b00100100000110111111111111111111;
      break;
      default   : mappedSegment = 0b11111111111111111111111111111111;
    }
    toDisplay = mappedSegment;

  /* Output the newly computed average to the output port */
```

```
       return(toDisplay);


}

/*--------------------------------------------------------
 * ANSI-C test bench
 *--------------------------------------------------------  */
#ifdef C                // ifdef pre-compiler directive to separate SW from HW
C
int main(){

    FILE *fp_i, *fp_o;
    int i;
    unsigned char in0;
    int out0;

    if((fp_i = fopen("indata.txt", "r")) == NULL){
      printf(" \"indata.txt \" could not be opened.\n");
      exit(1);
    }
    if((fp_o = fopen("out0000000data.txt", "w")) == NULL){
      printf(" \"outdata.txt \" could not be opened.\n");
      exit(1);
    }

    for (;;){

        if (fscanf(fp_i, "%c", &in0) == EOF) break;
         out0=ave8(in0);          // Main computational function
         fprintf(fp_o, "%d\n", out0);
    }

    fclose(fp_i);
    fclose(fp_o);
}
#endif
```