Logo

Description automatically generated**EEDG/CE 6370**

0

30

**Design and Analysis of Reconfigurable Systems**

**Homework 6**

**High-Level Synthesis with CyberWorkBench**

**Student Names: Khoa Diep, Dismas Ezechukwu**

**PART 1 – ave8.c Synthesis**

1. Synthesize the ave8.c design. Annotate from Resource constraint file (FCNT) the number and type of Functional Units (FUs) needed to fully parallelize the description (include screenshots from the reports). Explain why these FUs are needed.

|  |
| --- |
| Marks |
| 4 |
|  |

A table with numbers and symbols

Description automatically generated

Due to loop unrolling, adding up all the numbers in the buffer is done in parallel. Using A, B, ..,H as variables for the addition. A + B, C + D, E + F, G + H are done in parallel, hence 4 add8u address. Then the sum of those adders is added up: (A + B) + (C + D), (E + F) + (G + H), hence two additional adders. Then the sum of those is added up: (A + B + C + D) + (E + F + G + H), hence the last adder. Thus creating a tree adder.

1. 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.

|  |
| --- |
| Marks |
| 4 |
|  |

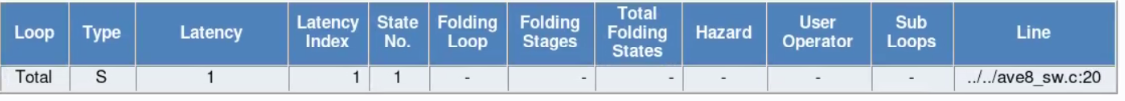
NOTE: This QoR file report was recorded after changing the C file, (fixing the testbench)

A blue and white sign

Description automatically generated

A blue and white rectangular sign with black text

Description automatically generated



LUTs: 69

Registers: 88

Critcal Path: 2.054 ns

Max Frequency: 486.855 MHz

Latency: 1 clock cycle

1. 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.

|  |
| --- |
| Marks |
| 4 |
|  |

LUTs: 43

Registers: 68

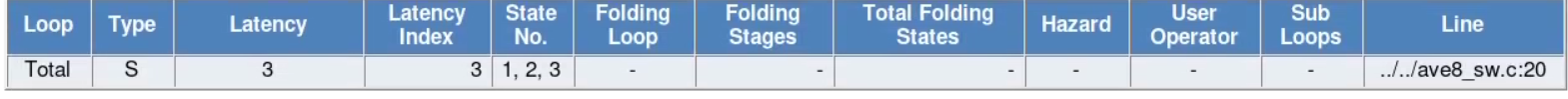
Registers and LUTs for the logic synthesis using Quartus Prime is less compared to the CWB’s QoR file. This is because the CWB uses the generated library for synthesis and to create the circuit, versus the Quartus Prime physically routes the circuit into the board, which is more accurate. Also, the CWB’S QoR file includes the layout of the testbench of the software C file, while the logic synthesis using Quartus Prime does not take the testbench into account.

1. What is the latency (in clock cycles) if we modified the target synthesis frequency to 500 MHz (in project properties)

|  |
| --- |
| Marks |
| 2 |
|  |

A screenshot of a calculator

Description automatically generated



Latency: 3 clock cycles

**PART 2 – ave8.c Verification**

1. 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 version with HLS frequency 100MHz and 500Mhz. Show the result (paste console window)

|  |
| --- |
| Marks |
| 2 |
|  |

NOTE: added random test vector (123) to *ave8\_ret.clv.*

100Mhz Console:

A screenshot of a computer error

Description automatically generated

500Mhz Console:

A screenshot of a computer error

Description automatically generated

For 500Mhz, the circuit does not work properly due to timing violations. 500 MHz is too fast for the synthesized RTL.

**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 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 cycle | 2.054 ns |
| Half Fus | 93 | 2 cycles | 4.08 ns |
| One single FU | 79 | 4 cycles | 3.5975 ns |

NOTE: for half Fus, add12u is limited to 2 from 3 (Max Fus)

Max Fus graph:

A graph with a line in the center

Description automatically generated with medium confidence

Half Fus graph:

A screen shot of a graph

Description automatically generated

One Single Fu Graph:

**A screen shot of a graph

Description automatically generated**

* The worst-case area in terms of ALUTs is the half Fus.
* The worst-case latency is one Fu.
* The worst-case critical path is the half Fus
* The best-case for area, latency, and critical path is Full Fus.
* The best-case was expected, however I expected the worst case for area and critical-path was going to be one Fu instead of half Fus. One Fu was not the worse area and critical path since it has four clock cycles to complete its computation.

**PART 4 FPGA Prototyping**

**A picture containing text, electronics, circuit

Description automatically generated**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.

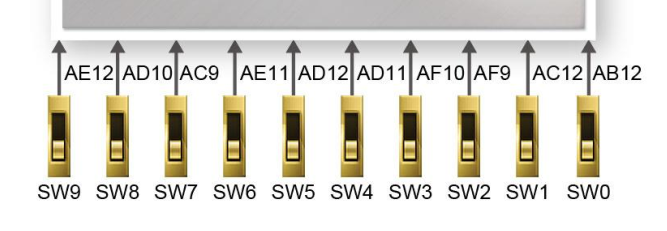
Display ave8 result on LCD

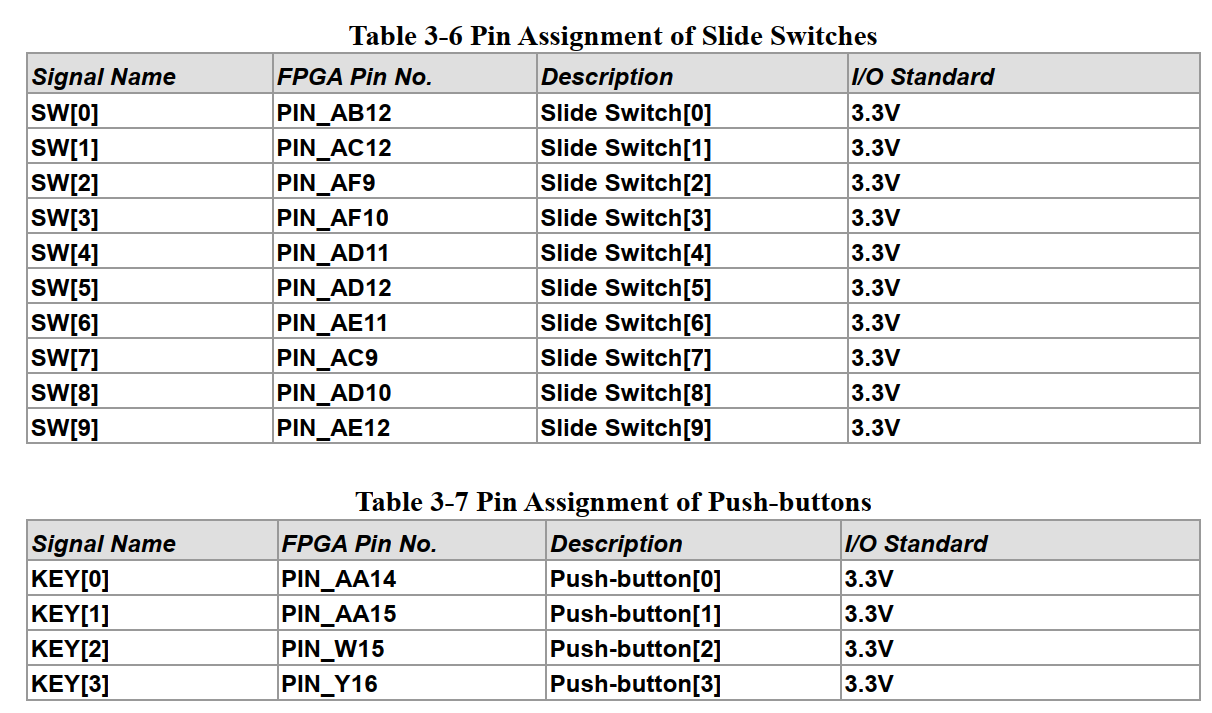
|  |
| --- |
| Marks |
| 8 |
|  |

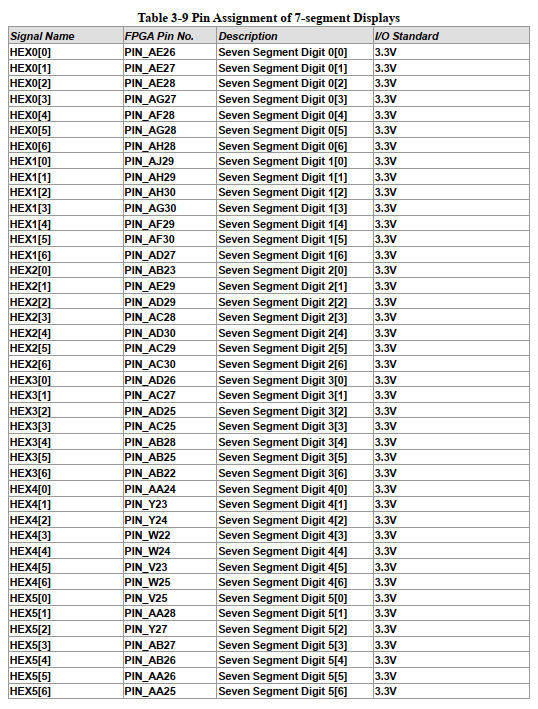
Read the value when button is pressed

New unsigned 8-bit value

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.







We couldn’t get the C code to implement the specified circuitry. Particularly, we had trouble figuring out how to implement the enable button for High Level Synthesis. We tried many designs, but it failed. The seven-segment display was designed with using the values of the output integer values to decode the display.

C Code

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*

\*\* ave8.c

\*\*

\*\* Description: The following program computes the average

\*\* of the 8 numbers being read

\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifdef C

#include "stdio.h"

#include "stdlib.h"

#include <stdbool.h>

#define $

#endif

/\* Global variables \*/

unsigned char buffer[8] = {0, 0, 0, 0, 0, 0, 0, 0};

// Hard code values

unsigned char seven\_segment\_decoder[10] = {128, 242, 72, 96, 98, 36, 4, 240, 0, 32};

// Cyber func=process

unsigned int ave8(unsigned char in0, bool enable){

/\* Local variables declaration \*/

int out0\_v, sum, i;

unsigned int hundred, ten, one;

unsigned char a, b, c;

unsigned int out\_seven\_seg = 0;

/\*

if(warm\_RST){

for (i = 7; i > 0; i--) {

buffer[i] = 0;

}

}

\*/

/\* Read new input into buffer \*/

//buffer[0] = in0;

if(enable){

/\* Shift data to accommodate new input to be read \*/

for (i = 7; i > 0; i--) {

buffer[i] = buffer[i- 1];

}

buffer[0] = in0;

}

//$

//enable=0;

/\* Set first element of sum to compute the average => can save 1 loop iteration \*/

sum= buffer[0];

/\* Add up all the numbers in the buffer \*/

for (i= 1; i< 8; i++) {

sum += buffer[i];

}

/\* Compute the average by dividing by 8 -> In HW a divide by 8 (/8) = shift 3 times \*/

out0\_v= sum / 8;

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

//return(out0\_v);

hundred = out0\_v / 100;

ten = (out0\_v - hundred\*100)/10;

one = (out0\_v - hundred\*100 - ten\*10);

a = seven\_segment\_decoder[hundred];

b = seven\_segment\_decoder[ten];

c = seven\_segment\_decoder[one];

out\_seven\_seg = (a << 16) | (b << 8) | c;

return(out\_seven\_seg);

}

/\*------------------------------------------------------

\* 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 int in0;

unsigned int out0;

bool enable;

enable = true;

if((fp\_i = fopen("indata.txt", "r")) == NULL){

printf(" \"indata.txt \" could not be opened.\n");

exit(1);

}

if((fp\_o = fopen("outdata.txt", "w")) == NULL){

printf(" \"outdata.txt \" could not be opened.\n");

exit(1);

}

for (;;){

if (fscanf(fp\_i, "%u", &in0) == EOF) break;

//printf("New Input%c :", in0);

out0=ave8(in0, enable); // Main computational function

fprintf(fp\_o, "%u\n", out0);

printf("Input: %u Output:%u\n", in0,out0);

}

fclose(fp\_i);

fclose(fp\_o);

}

#endif