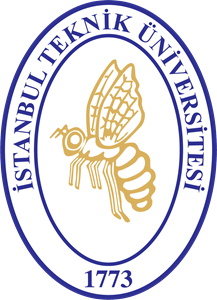
**DIGITAL SYSTEM DESIGN APPLICATIONS**

**(CRN: 11275)**

**THE REPORT OF PROJECT - 1**



Faculty of Electrical and Electronics Engineering

Electronics and Communication Engineering

Yusuf Tekin - 040200043

Bora Kıran – 040210069

**Hamming Weight Calculator using Parallel Counters**

**Algorithm Explanation:** The design slices 32-bit inputs into 4-bit length pieces. Each part goes into a custom counter that counts the number of ones (hamming weight). Outputs of the custom counters are summed into the total hamming weight. To be able to design such mechanism, these steps applied in order:

1. The first step is creating a stimulus text file via generating numbers with python. The script generated 98 random inputs and to be able check the least and the most important bits precisely, all 0s and all 1s values are added manually.
2. The second step is designing the architecture of the overall system. Dividing the 32-bit input into 4-bit parts would be the best option since the parallel working blocks would take up less space rather than using summing operators for all 32-bits. Only downside is that the summing part of the calculated hamming weights would increase in complexity, but the summing is handled by summing operator which uses summing primitives.
3. To be able to check the system with generated binary numbers, the simulation created with the self-checking properties. In the testbench, with the use of operators as “$fopen, $fclose, $feof…” the binary and outputs files are assigned to the proper pointers. Using a “while” loop until the end of the files, all the values within the stimulus text are checked as inputs and if the values are same with the values in the outputs text file, it would be stated in TCL Console as correct.

A diagram of a company

Description automatically generated

4-bit Hammering Weight Block

Verilog Code - Part 1

`timescale 1ns **/** 1ps

**module** HA**(**

**input** x**,**

**input** y**,**

**output** cout**,**

**output** sum**);**

**assign** cout **=** x **&** y**;**

**assign** sum **=** x**^**y**;**

**endmodule**

**module** FA**(**

**input** x**,**y**,**cin**,**

**output** cout**,**sum

**);**

**wire** c0**,**c1**,**s0**;**

HA ha0**(**x**,**y**,**c0**,**s0**);**

HA ha1**(**s0**,**cin**,**c1**,**sum**);**

**assign** cout **=** c1 **|** c0**;**

**endmodule**

**(\*** DONT\_TOUCH **=** "TRUE" **\*)**

**module** counter\_4bit**(**

**input** **[**3**:**0**]** in**,**

**output[**2**:**0**]** o

**);**

**wire** c0**,**s0**,**c1**,**s1**,**c2**;**

HA ha0**(**in**[**1**],**in**[**0**],**c0**,**s0**);**

HA ha1**(**in**[**3**],**in**[**2**],**c1**,**s1**);**

HA ha2**(**s1**,**s0**,**c2**,**o**[**0**]);**

FA fa0**(**c1**,**c0**,**c2**,**o**[**2**],**o**[**1**]);**

**endmodule**

**(\*** DONT\_TOUCH **=** "TRUE" **\*)**

**module** main **(**

**input** **[**31**:**0**]** in**,**

**output** **[**5**:**0**]** out

**);**

**wire** **[**2**:**0**]** block\_output0**;**

**wire** **[**2**:**0**]** block\_output1**;**

**wire** **[**2**:**0**]** block\_output2**;**

**wire** **[**2**:**0**]** block\_output3**;**

**wire** **[**2**:**0**]** block\_output4**;**

**wire** **[**2**:**0**]** block\_output5**;**

**wire** **[**2**:**0**]** block\_output6**;**

**wire** **[**2**:**0**]** block\_output7**;**

Verilog Code - Part 2

counter\_4bit b0**(**in**[**31**:**28**],**block\_output0**);**

counter\_4bit b1**(**in**[**27**:**24**],**block\_output1**);**

counter\_4bit b2**(**in**[**23**:**20**],**block\_output2**);**

counter\_4bit b3**(**in**[**19**:**16**],**block\_output3**);**

counter\_4bit b4**(**in**[**15**:**12**],**block\_output4**);**

counter\_4bit b5**(**in**[**11**:**8**],**block\_output5**);**

counter\_4bit b6**(**in**[**7**:**4**],**block\_output6**);**

counter\_4bit b7**(**in**[**3**:**0**],**block\_output7**);**

**assign** out**=**block\_output0**+**block\_output1**+**block\_output2**+**block\_output3**+**block\_output4**+**block\_output5**+**block\_output6**+**block\_output7**;**

**endmodule**

Testbench Code - Part 1

`timescale 1ns **/** 1ps

**module** main\_tb**;**

main uut**(**

**.**in**(**data\_in**),**

**.**out**(**count\_ones**)**

**);**

**wire** **[**5**:**0**]** count\_ones**;**

**reg** **[**31**:**0**]** data\_in**;**

**reg** **[**31**:**0**]** test\_data**;**

**reg** **[**5**:**0**]** expected\_ones**;**

**integer** input\_file**,** output\_file**;**

**integer** status**;**

**integer** error**;**

**initial** **begin**

input\_file **=** $fopen**(**"stimulus\_input.txt"**,** "r"**);**

error **=** 0**;**

**if(**input\_file **==** 0**)** **begin**

$display**(**"\nError: Failed to open the simulation input file.\n"**);**

$stop**;**

**end**

output\_file **=** $fopen**(**"output.txt"**,** "r"**);**

**if(**output\_file **==** 0**)** **begin**

$display**(**"\nError: Failed to open the simulation output file.\n"**);**

$stop**;**

**end**

Testbench Code - Part 2

**#**10**;**

**while** **(!**$feof**(**input\_file**)** **&&** **!**$feof**(**output\_file**))** **begin**

status **=** $fscanf**(**input\_file**,** "%b\n"**,** test\_data**);**

**if(**status **==** 0**)** **begin**

$display**(**"\nError: simulation input cannot read\n"**);**

$stop**;**

**end**

status **=** $fscanf**(**output\_file**,** "%d\n"**,** expected\_ones**);**

**if(**status **==** 0**)** **begin**

$display**(**"\nError: simulation output cannot read\n"**);**

$stop**;**

**end**

data\_in **=** test\_data**;**

**#**5**;**

**if(**count\_ones **!==** expected\_ones**)** **begin**

$display**(**"\nUnexpected value: Input = %b | Expected\_Ones = %d | Found Value: %d\n"**,** data\_in**,** expected\_ones**,** count\_ones**);**

error **=** error **+** 1**;**

**end** **else** **begin**

$display**(**"Correct value: Input = %b | Ones = %d"**,** data\_in**,** count\_ones**);**

**end**

**end**

**if(**error **==** 0**)** **begin**

$display**(**"\nALL CORRECT, DESIGN PASSED ALL THE TESTS SUCCESFULLY!!\n"**);**

**end** **else** **begin**

$display**(**"\nTEST COMPLETED WITH %d ERRORS\n"**,**error**);**

**end**

$fclose**(**input\_file**);**

$fclose**(**output\_file**);**

**#**10**;**

$finish**();**

**end**

**endmodule**

A screenshot of a graph

Description automatically generated

Utilization Summary

A screenshot of a computer

Description automatically generated

Path Delays - Setup

A screenshot of a computer

Description automatically generated

Path Delays - Hold

A diagram of a computer

Description automatically generated

Technology Schematic

A diagram of a computer system

Description automatically generated with medium confidence

RTL Schematic

A green numbers on a black background

Description automatically generated

Behavioral Simulation

A close up of a computer screen

Description automatically generated

TCL Console Output

A screenshot of a computer

Description automatically generated

Post-Implementation Timing Simulation

**Work Package Table:**

* Stimulus Text – Bora Kıran
* Design – Bora Kıran and Yusuf Tekin
* Simulation Code - Yusuf Tekin and Bora Kıran
* Implementation – Yusuf Tekin
* Report – Yusuf Tekin and Bora Kıran

**References:**

1- Behrooz Parhami, Computer Arithmetic Algorithms and Hardware Designs, 2nd ed. 2010, pp. 164–167.

2- “Hamming weight,” Wikipedia. https://en.wikipedia.org/wiki/Hamming\_weight