**2023 Digital IC Design**

**Homework 3:** **Arithmetic sequence calculator**

1. **Introduction:**

This assignment aims for you to create a calculator with addition, subtraction, and multiplication functions that follows the rules of arithmetic operations. The input method for the expressions will be through a series input. You will need to respond

to the current input signal based on the different states of the circuit.

1. **Design Specifications:**

**2.1 Block Overview:**

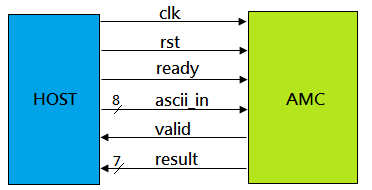
****

Figure 1. System Block Diagram

**2.2 I/O Interface:**

Table I. I/O interface of the design

|  |  |  |  |
| --- | --- | --- | --- |
| Signal Name | I/O | width | Description |
| *clk* | I | 1 | This circuit is a synchronous design triggered at the positive edge of clk. |
| *rst* | I | 1 | Active-high asynchronous reset signal. |
| *ready* | I | 1 | Number series ready indication signal.  When this signal is high, you can obtain a valid numerical input on the 'numIn' port at  next cycle. |
| *ascii\_in* | I | 8 | Number series input data.  (ASCII representation) |
| *valid* | O | 1 | Output valid signal. |
| *result* | O | 7 | Result of arithmetic operations. |

**2.3 Basic Principles and Restrictions:**

Three basic principles of arithmetic operations:

1. Multiply and divide(Unused in this homework) before adding and

subtracting.

1. Perform operations inside parentheses first.
2. Calculate from left to right.

To simplify calculation, the following restrictions have been implemented:

1. The input will only consist of numbers 0 to 15 and five operators: +, -, \*, (,

, ) and =. The input is using ASCII codes, so in your design, you must map

the ASCII codes to their correct meanings.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Represent | ASCII code | Represent | ASCII code | Represent | ASCII code |
| 0 | 48 | 1 | 49 | 2 | 50 |
| 3 | 51 | 4 | 52 | 5 | 53 |
| 6 | 54 | 7 | 55 | 8 | 56 |
| 9 | 57 | a(number 10) | 97 | b(number 11) | 98 |
| c(number12) | 99 | d(number 13) | 100 | e(number 14) | 101 |
| f(number15) | 102 | ( | 40 | ) | 41 |
| \* | 42 | + | 43 | - | 45 |
| = | 61 |  | | | |

Table II. ASCII Table ([Reference](https://www.asciitable.com/))

1. All input values are positive, and there will be no negative numbers during the calculation process, and the maximum value will not exceed 127.

Therefore, you can treat all values as unsigned numbers.

1. The input number string sequence will not exceed 16 elements, and it will

always end with the "=" sign.

**2.4 Function Description:**

**2.4.1 Infix to Postfix method:**

In human calculation, we are accustomed to interpreting and calculating equations using the infix notation. However, for computers, due to limitations in algorithms that prevent them from comprehensively processing and handling the entire equation, it is customary to convert the notation to postfix and then perform

the operation.

You can follow these rules:

1. Create an empty stack to hold operators.
2. Scan the infix notation from left to right.
3. If the current token is an operand (a number), add it to the output string.
4. If it is a operator["+", "-", "\*", "(", ")" ], You should use following rules below :
5. If the current token is a left parenthesis, push it onto the stack.
6. If the current token is a right parenthesis, pop operators from the

stack and add them to the output string until a left parenthesis is found.

Discard the left and right parentheses.

1. If the current token is an operator :

Pop operators from the stack and append them to the output string until either the stack is empty or the top operator has lower precedence than

the current operator.

1. The resulting string is the postfix notation.

**2.4.2 Infix to Postfix Example:**

Below is an example of how to convert the infix notation "*3 + 4 \* (2 - 1)* " to postfix notation "*3 4 2 1 - \* +*"。

|  |  |  |  |
| --- | --- | --- | --- |
| Input Token | Output | Stack | Action |
| 3 | 3 |  | Add 3 to output |
| + |  | + | Push + onto stack |
| 4 | 3 4 | + | Add 4 to output |
| \* | 3 4 | + \* | Push \* onto stack |
| ( | 3 4 | + \* ( | Push ( onto stack |
| 2 | 3 4 2 | + \* ( | Add 2 to output |
| - | 3 4 2 | + \* ( - | Push - onto stack |
| 1 | 3 4 2 1 | + \* ( - | Add 1 to output |
| ) | 3 4 2 1 - | + \* | Pop stack until ( is found, add to output |
|  | 3 4 2 1 - \* + |  | Pop remaining stack elements to output |

**2.4.3 Calculate Postfix Example:**

Here is an explanation of how to calculate the postfix expression :

1. Start by reading the expression from left to right.
2. Push each number onto the stack as it is encountered.
3. When an operator is encountered, pop the top two numbers from the stack, apply the operator to them, and push the result back onto the stack.
4. Repeat steps 2 and 3 until the entire expression has been read.
5. The final value left on the stack is the result of the expression.

Below is an example of how to calculate the postfix expression

"*3 4 2 1 - \* +*" :

|  |  |  |
| --- | --- | --- |
| Input Token | Stack | Action |
| 3 | 3 | Push 3 onto stack |
| 4 | 3 4 | Push 4 onto stack |
| 2 | 3 4 2 | Push 2 onto stack |
| 1 | 3 4 2 1 | Push 1 onto stack |
| - | 3 4 1 | Pop 1 and 2 from the stack, subtract 1 from 2 to get 1, and push 1 onto the stack |
| \* | 3 4 | Pop 1 and 4 from the stack, multiply them to get 4, and push 4 onto the stack. |
| + | 7 | Pop 3 and 4 from the stack, plus them to get 7, and push 7 onto the stack. |

**2.4.4 Data input and result output:**

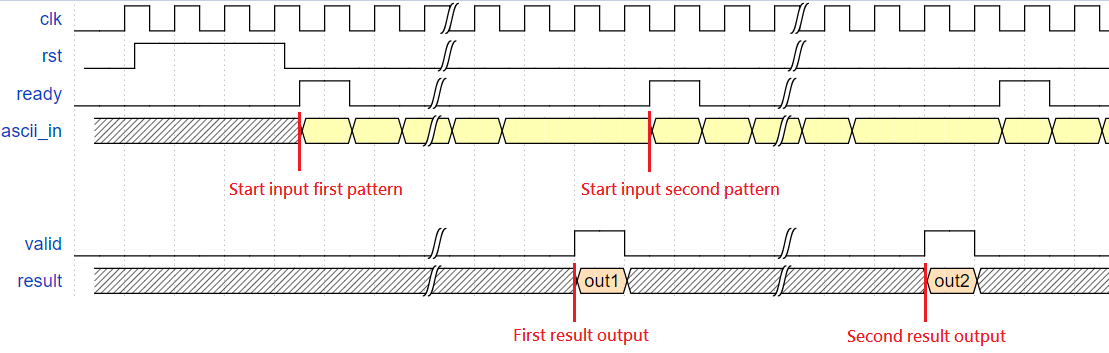
****

Figure 2. System I/O Timing

1. After each 'ready' signal remains high for one cycle, data input will begin

start input at the same cycle.

1. After the host finishes inputting, it will begin to wait for a response from the AMC. When the AMC completes calculation, you must pull the 'valid' signal high and output the calculation result from 'result' in the same cycle. Then,

in the next cycle, you should pull 'valid' back low.

1. When the host detects that 'valid' is high, it will begin to compare the data for correctness and pull 'ready' high in the next cycle, repeating the actions 1, 2,

and 3 until the patterns is completely read.

**3. File Description:**

|  |  |
| --- | --- |
| File Name | Description |
| AMC.v | The top module of your design. |
| testfixture.sv | The testbench file. |
| Pattern.txt | Input data and golden data file. |

**4. Scoring:**

**4.1 Function simulation [ %]**

All of the result should be generated correctly, and you will get the following message in ModelSim simulation. If unable to complete all tasks, you will receive partial credit of 0. \* your score.

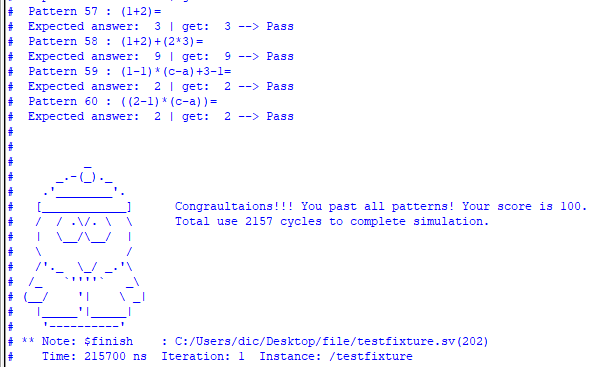


Fig 3. Function simulation result

**4.2 Gate-Level Simulation [ %]**

4.2.1 Synthesis:

Your code should be synthesizable. After it is synthesized in Quartus, a file named AMC.vo will be obtained.

4.2.2 Simulation:

All of the result should be generated correctly using AMC.vo, and you will get the following message in ModelSim simulation. If unable to complete all tasks, you will receive partial credit of 0. \* your score.

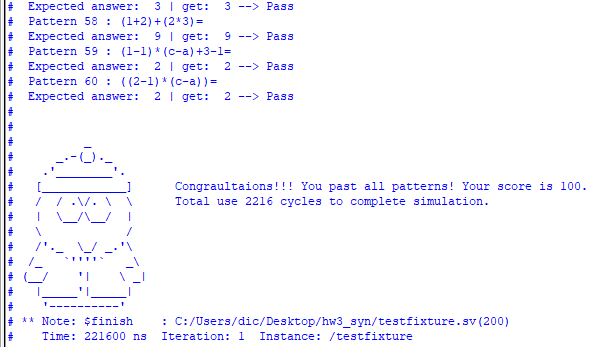


Fig 4. Gate level simulation result

**4.3 Performance [ %]**

The performance is scored by the total logic elements, total memory bit, and embedded multiplier 9-bit element your design used in gate-level simulation and the simulation time your design takes. The score will be decided by your ranking in all received homework. (The smaller, the better)

Scoring = (Total logic elements + total memory bit + 9\*embedded multiplier 9-bit element) × (Total cycle)

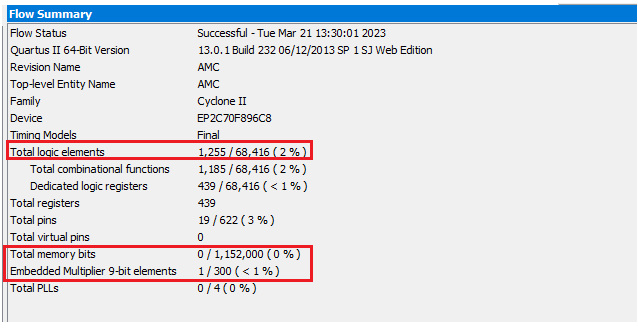


Fig 5. Synthesis result

**5. Submission:**

**5.1 Submitted files**

You should classify your files into two directories and compress them to .zip format. The naming rule is HW3\_studentID\_name.zip. If your file is not named according to the naming rule, you will lose five points.

|  |  |
| --- | --- |
|  | RTL category |
| \*.v | All of your Verilog RTL code |
|  | Documentary category |
| \*.pdf | The report file of your design (in pdf). |

**5.2 Report file**

Please follow the spec of report. You are asked to describe how the circuit is designed as detailed as possible.

**5.3 Note**

Please submit your .zip file to folder HW3 in moodle.

Deadline:

If you have any problem, please contact TA by email

Ne6114087@gs.ncku.edu.tw