2023 Digital IC Design

Homework 3: Arithmetic Expression Calculator

1. Introduction

This assignment aims for you to create an Arithmetic Expression Calculator (AEC) with addition, subtraction, and multiplication functions. The AEC circuit must follow the rules of arithmetic expression. The input expressions consist of numeric operands, operators (+, -, *, =), and parentheses. The specification and function of the circuit is detailed in the following sections.

2. Design Specifications

2.1 Block Overview

Fig.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	Input ready indication signal. When this signal is high, the expression to be calculated will be inputted from the '*acsii_in*' port.
ascii_in	I	8	Each operand or operator of the expressions will be inputted with ASCII representation.
valid	O	1	Output valid signal. When this signal is high,

			testbench will check the output *result* signal.
result	O	7	Result of the arithmetic expression.

2.3 Basic Principles and Restrictions

Three basic principles of arithmetic expressions:

1. Calculate multiplication and division before addition and subtraction. (There is no divide operation in the test pattern of this homework.)
2. Perform operations inside parentheses first.
3. Calculate from left to right.

To simplify calculation, there are three restrictions in this homework:

1. The input will only consist of numbers 0 to 15, four operators: +, -, *, =, and parentheses. The inputs are represented in ASCII codes, so you have to convert ASCII codes to their correct meaning in your design.
2. All input values are positive. 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.
3. The input expression string will not exceed 16 characters, and it will always end with the "=" operator.

Table II. ASCII Table (Reference)

Represent	ASCII code	Represent	ASCII code	Represent	ASCII code
0	48	8	56	(	40
1	49	9	57	)	41
2	50	a (number 10)	97	*	42
3	51	b (number 11)	98	+	43
4	52	c (number 12)	99	-	45
5	53	d (number 13)	100	=	61
6	54	e (number 14)	101		
7	55	f (number 15)	102		

2.4 Function Description

2.4.1 Infix to Postfix Method

In human calculation, we are accustomed to interpreting and calculating equations using infix notation. However, it's difficult for computers to comprehensively process and handle the entire equation due to the limitations of the algorithm. One method to solve this problem is converting the notation to postfix and then performing the operation.

The infix to postfix operation can be performed with the following steps:

1. Create an empty stack to hold operators.
2. Scan the infix notation from left to right.
   (1) If the current token is an operand (a number), append it to the output string.
   (2) If the current token is an operator ["+", "-", "*"], the following rules should be applied:
      a. Pop the top token from the stack if its precedence is higher than or equal to the precedence of the current token. Append the token popped out to the output string.
      b. Repeat step (2.a) until the precedence of the top token of the stack is lower than the precedence of the current token or a left parenthesis is encountered.
      c. Push the current token onto the stack.
   (3) If the current token is a left parenthesis, push it onto the stack.
   (4) If the current token is a right parenthesis, pop operators from the stack and append them to the output string until a left parenthesis is found. Discard the left and right parentheses.
3. After scanning the infix string, pop out all the tokens in the stack. Append the popped-out tokens to the output string.
4. The resulting output string is in postfix notation.

2.4.2 Infix to Postfix Example

Below is an example of converting an expression in infix notation "3 + 4 * (2 - 1) " to postfix notation "3 4 2 1 - * +".

Input Token	Output	Stack	Action
3	3		Append 3 to output
+		+	Push '+' onto stack
4	3 4	+	Append 4 to output
*	3 4	+ *	Push '*' onto stack
(	3 4	+ * (	Push '(' onto stack
2	3 4 2	+ * (	Append 2 to output
-	3 4 2	+ * (-	Push '-' onto stack
1	3 4 2 1	+ * (-	Append 1 to output
)	3 4 2 1 -	+ *	Pop stack until '(' is found, append popped out tokens to output

	3 4 2 1 - * +		Pop remaining tokens in stack to output

2.4.3 Calculate Expression in Postfix Notation

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

1. Start by scanning the postfix expression from left to right.
    a. Push each scanned number onto the stack.
    b. When an operator is encountered, pop the top two numbers from the stack. Apply the operator to the popped-out numbers, and push the result back onto the stack.
2. Repeat steps 1 until the entire expression has been scanned.
3. The final value left on the stack is the calculation 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. Push 1 onto the stack
*	3 4	Pop 1 and 4 from the stack, multiply them to get 4. Push 4 onto the stack.
+	7	Pop 3 and 4 from the stack, plus them to get 7. Push 7 onto the stack.

2.5 Timing Specifications

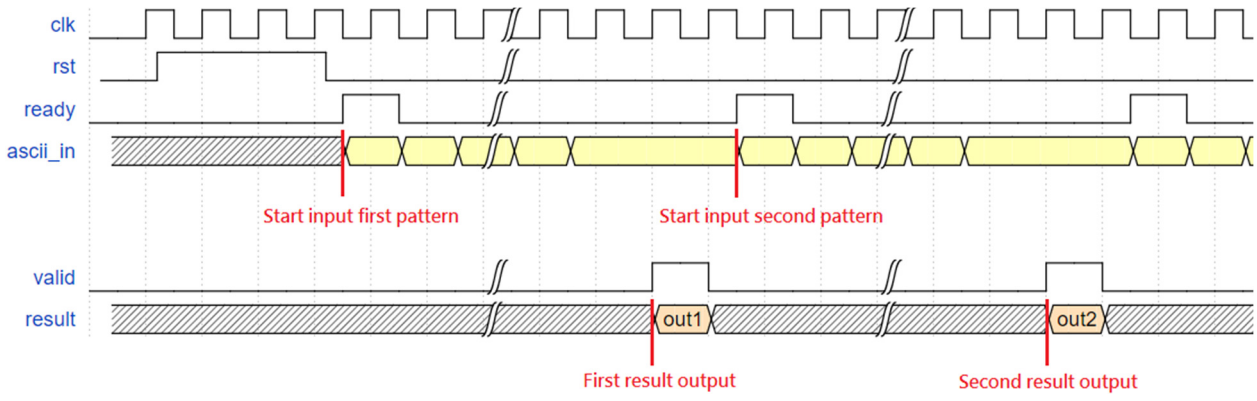


Fig.2. Timing diagram.

1. When the '*ready*' signal is pulled up to high, the expression pattern will start being inputted at the same cycle. The '*ready*' signal will maintain at high for only one cycle.
2. After the testbench finishes inputting, it will begin to wait for a valid output from the AEC. When the AEC completes the calculation, the '*valid*' signal must be pulled up to high. The calculation result should be outputted from the '*result*' port in the same cycle. Then, in the next cycle, the '*valid*' signal has to be pulled back as low to wait for the next pattern from the testbench.
3. When the testbench detects that the 'valid' signal is high, it will begin to compare the 'result' value with the golden data for correctness. The 'ready' signal will then be pulled up as high in the next cycle.

3. File Description

File Name	Description
AEC.v	The top module of your design.
testfixture.sv	The testbench file. In this homework, you are allowed to modify the defined *cycle* and *terminate_cycle* in testbench.
pattern.data	Input data and golden data.

4. Scoring

4.1 Functional simulation [60%]

All of the results 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. (0.6* your score)

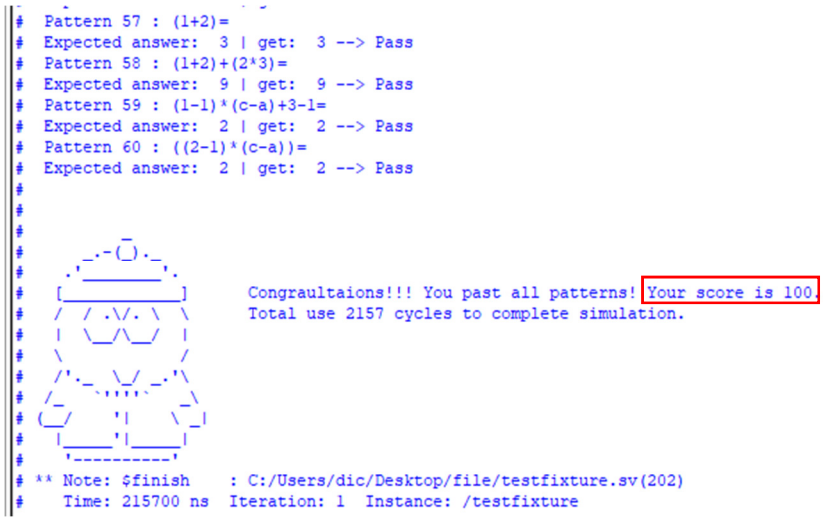


Fig 3. Functional simulation result

4.2 Gate-Level Simulation [20%]

4.2.1 Synthesis

Your code should be synthesizable. After it is synthesized in Quartus, files named *AEC.vo* and *AEC_v.sdo* will be obtained.

DEVICE：Cyclone IV E - EP4CE55F23A7

4.2.2 Simulation

All of the results should be generated correctly using *AEC.vo* and *AEC_v.sdo*, and you will get the following message in ModelSim simulation. If unable to complete all tasks, you will receive partial credit. (0.2* your score)

```
#  Expected answer: 50 | get: 50 --> Pass
#  Pattern 57 : (1+2)=
#  Expected answer:  3 | get:  3 --> Pass
#  Pattern 58 : (1+2)+(2*3)=
#  Expected answer:  9 | get:  9 --> Pass
#  Pattern 59 : (1-1)*(c-a)+3-1=
#  Expected answer:  2 | get:  2 --> Pass
#  Pattern 60 : ((2-1)*(c-a))=
#  Expected answer:  2 | get:  2 --> Pass
#
#
#
#
#
#
#   Congraultaions!!! You past all patterns! Your score is 100.
#   Total use 2157 cycles to complete simulation.
#
#
#
#
#
#
# ** Note: $finish    : C:/Users/dic/Desktop/hw3_pos/testfixture.sv(202)
#    Time: 215700 ns  Iteration: 1  Instance: /testfixture
```

Fig 4. Gate level simulation result

4.3 Performance [20%]

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.

Flow Summary

<<Filter>>

Flow Status	Successful - Mon Mar 27 16:45:45 2023
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition
Revision Name	AMC
Top-level Entity Name	AMC
Family	Cyclone IV E
Device	EP4CE55F23A7
Timing Models	Final
Total logic elements	979 / 55,856 (2 %)
Total registers	439
Total pins	19 / 325 (6 %)
Total virtual pins	0
Total memory bits	0 / 2,396,160 (0 %)
Embedded Multiplier 9-bit elements	1 / 308 (< 1 %)
Total PLLs	0 / 4 (0 %)

Fig 5. Synthesis result

The performance score will be decided by your ranking in all received homework. Only designs that passed gate-level simulation and meet resource limitations will be considered in the ranking. Otherwise, you can't get performance score.

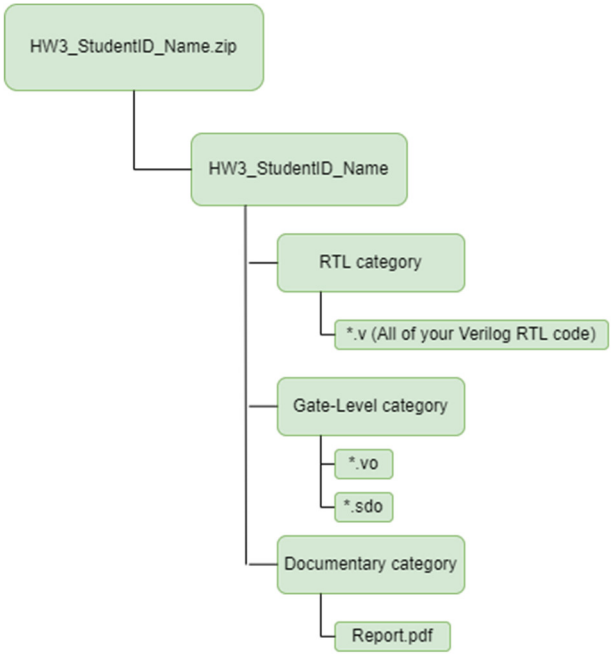
The scoring standard: (The smaller, the better)
Scoring = Area cost * Timing cost
Area cost = Total logic elements + total memory bits + 9*embedded multiplier 9-bit elements
Timing cost = Total cycle used*clock width
*** Total logic elements must not exceed 1500.**

5. Submission

5.1 Submitted files

You should classify your files into three 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
	Gate-Level category
*.vo	Gate-Level netlist generated by Quartus
*.sdo	SDF timing information generated by Quartus
	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, and the flow summary result is necessary in the report. Please fill the field of total logic elements, total memory bits, embedded multiplier 9-bit elements according to the flow summary (as shown in Fig. 5) of your synthesized design. And fill the field of clock cycle used and clock width according to the gate-level simulation results that Modelsim shows. If the filled-in values don't match your synthesized design or gate-level simulation results, you will lose 10 points.

5.3 Note

In this homework, you are allowed to modify the defined *cycle* and *terminate_cycle* in testbench file. 'cycle' decides the clock width that is used to validate your design, and 'terminate_cycle' decides the maximum cycles your circuit takes to complete the simulation. Please do not modify any other content of the testbench.

Please submit your .zip file to folder HW3 in moodle.
Deadline: 2023/04/24 23:55

Please don't design specifically for the test pattern. Otherwise, you will get 0 point. Any plagiarism behavior will also get 0 points.

If you have any problem, please contact TA by email
Ne6114087@gs.ncku.edu.tw