2023 Digital IC Design Homework 3

NAME	林珮玉
Student ID	E24084096

Simulation Result

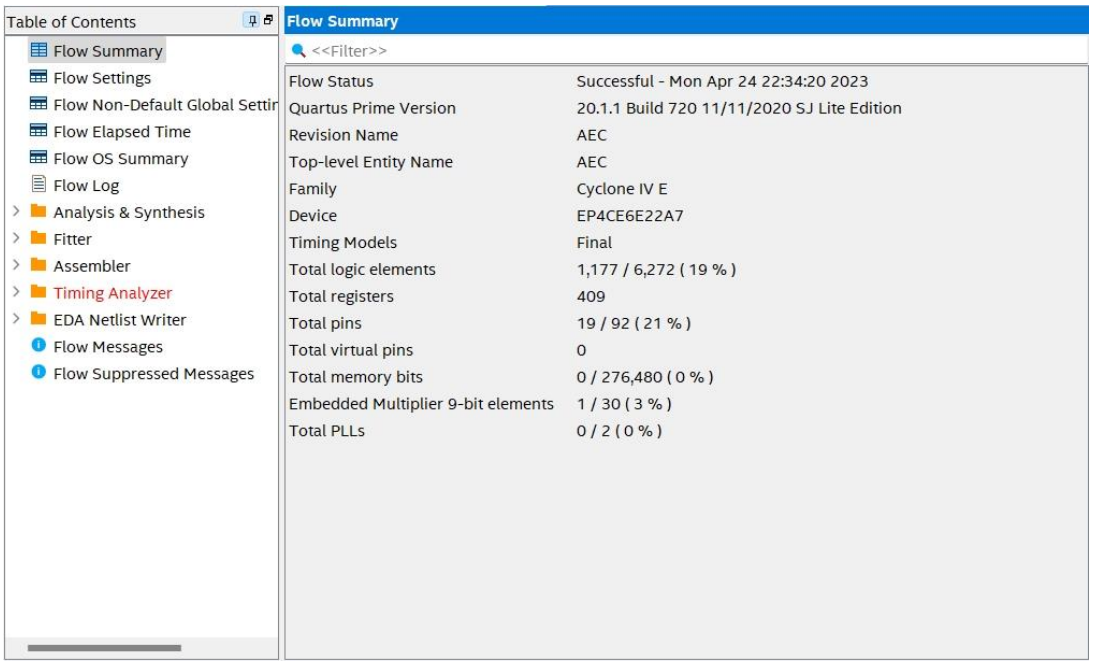
Functional simulation	100	Gate-level simulation	100

```
#  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 2248 cycles to complete simulation.
#   |  \__/\__/  |
#   \            /
#   /'-._  \_/ _.-'\
#  /_    `'''''`    _\
# (__/     '|     \_|
#   |_____'|_____|
#    '----------'
# ** Note: $finish    : C:/DIC/HW3/file/testfixture.sv(191)
#    Time: 224800 ns  Iteration: 1  Instance: /testfixture
# 1
# Break in Module testfixture at C:/DIC/HW3/file/testfixture.sv line 191
```

```
#  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 2248 cycles to complete simulation.
#   |  \__/\__/  |
#   \            /
#   /'-._  \_/ _.-'\
#  /_    `'''''`    _\
# (__/     '|     \_|
#   |_____'|_____|
#    '----------'
# ** Note: $finish    : C:/DIC/HW3/file/testfixture.sv(191)
#    Time: 224800 ns  Iteration: 1  Instance: /testfixture
# 1
# Break in Module testfixture at C:/DIC/HW3/file/testfixture.sv line 191
```

Synthesis Result

Total logic elements	1,177
Total memory bits	0
Embedded multiplier 9-bit elements	1
Total cycle used	2,240
Clock width	100



Description of your design

This Verilog module is an implementation of a postfix calculator using a state machine. The inputs to the module are clk, rst, ascii_in, and ready, and the outputs are valid and result.

In order to achieve the best efficiency, we need reduce the registers and memory length. Therefore, this design consists of a state machine with five states: IDLE, DATA_IN, POSTFIX, CALCULATOR, and RESULT. The state machine transitions are based on the input and output signals of the module.

In the IDLE state, new ascii_in input data are not available until **ready** signal become high. When ready becomes high, the module transitions to the DATA_IN state.

In the DATA_IN state, the module reads the ascii_in input and stores the ASCII code in the deAscii array. If the ascii_in input is the equals sign (=), the module transitions to the POSTFIX state.

In the POSTFIX state, which is the crucial part in this design. In order to reduce the clock cycle, I classify five data into 4 cases, including left parenthesis, right parenthesis, multiply, and or subside, as well as numbers. In situation of left parenthesis, (will be added into stack directly. And when the coming operator is right parenthesis, it will pop out tokens until bumping left parenthesis. The module reads the deAscii array and constructs the postfix expression in the stack array. If the postfix expression is complete and the stack is empty, the module transitions to the CALCULATOR state.

In the CALCULATOR state, the module evaluates the postfix expression in the stack array and stores the result in the result output. When the = symbol is encountered in the outString array, the module transitions to the RESULT state.

In the RESULT state, the module resets the deAscii, outString, and stack arrays, and sets the valid output signal high to indicate that the result is available.

*Scoring = Area cost * Timing cost*

*Area cost = Total logic elements + Total memory bits + 9*Embedded multipliers 9-bit elements*

*Timing cost = Total cycle used * Clock width*

*** Total logic elements must not exceed 1500.**