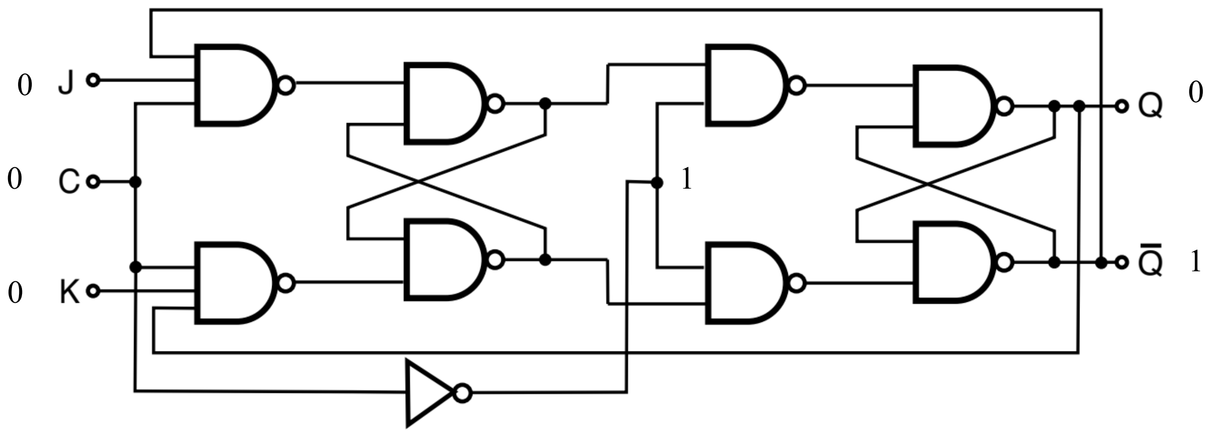
ICS Answer Sheet #9

Sakar Gopal Gurubacharya
s.gurubacharya@jacobs-university.de

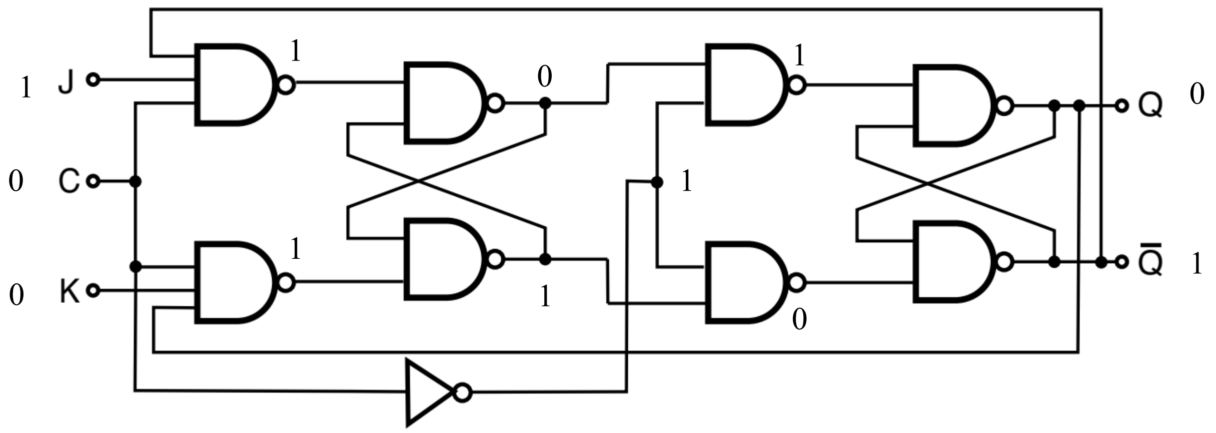
Problem 9.1: JK flip-flops

Initial gates with all inputs as 0, clock input as 0 and circuit's output as 0.

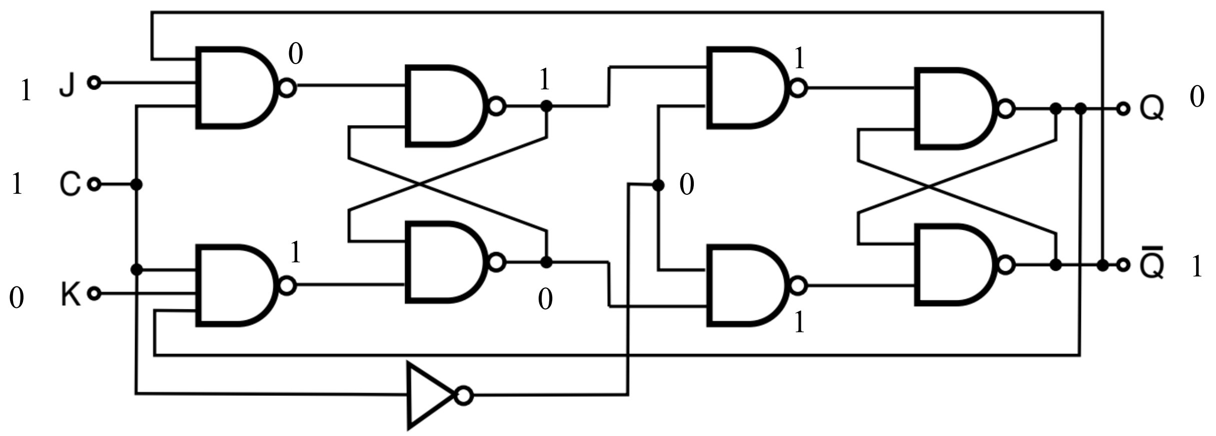


a. Suppose J transitions to 1 and C transitions to 1 soon after. Create a copy of the drawing and indicate for each line whether it carries a 0 or a 1.

J to 1



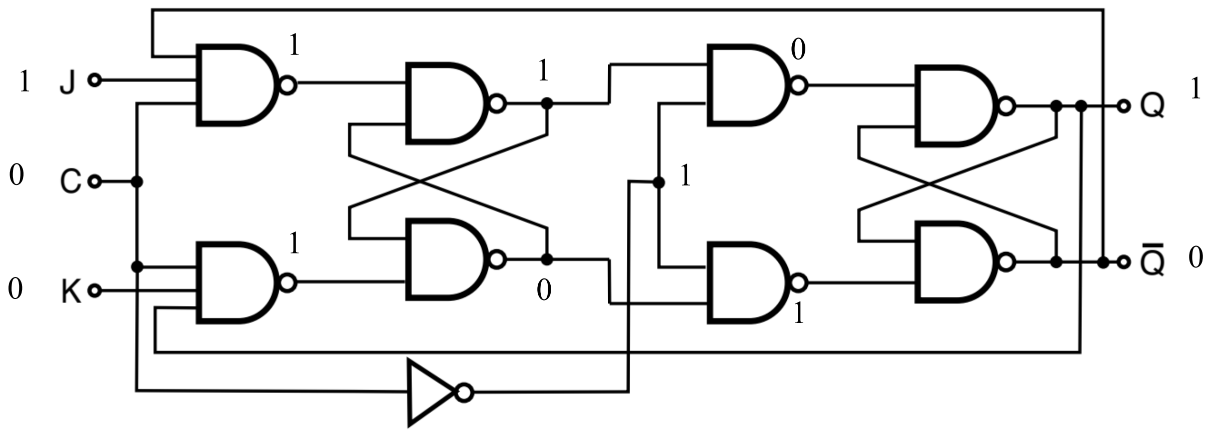
J and C to 1



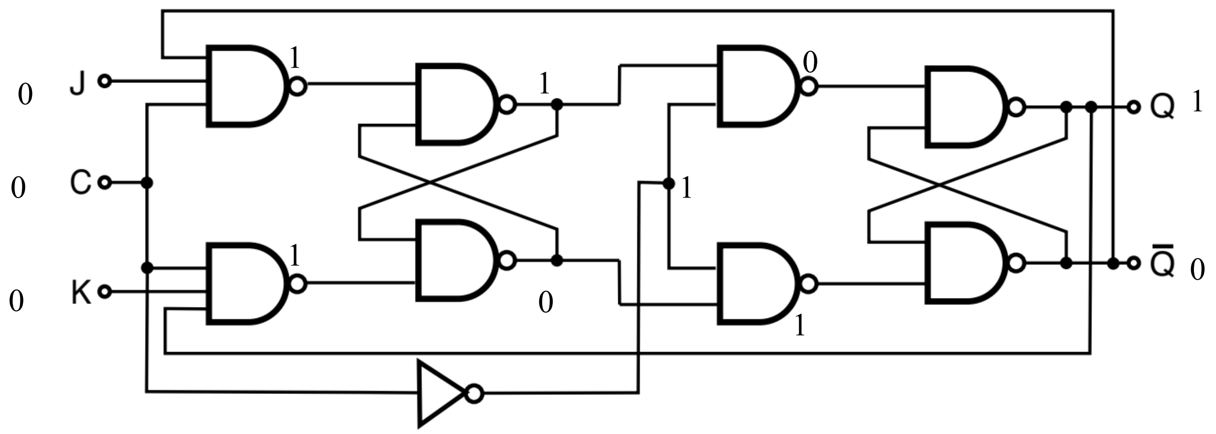
The circuit's output remains 0

b. **Some time later, C transitions back to 0 and soon after J transitions to 0 as well. Create another copy of the drawing and indicate for each line whether it carries a 0 or a 1.**

C to 0



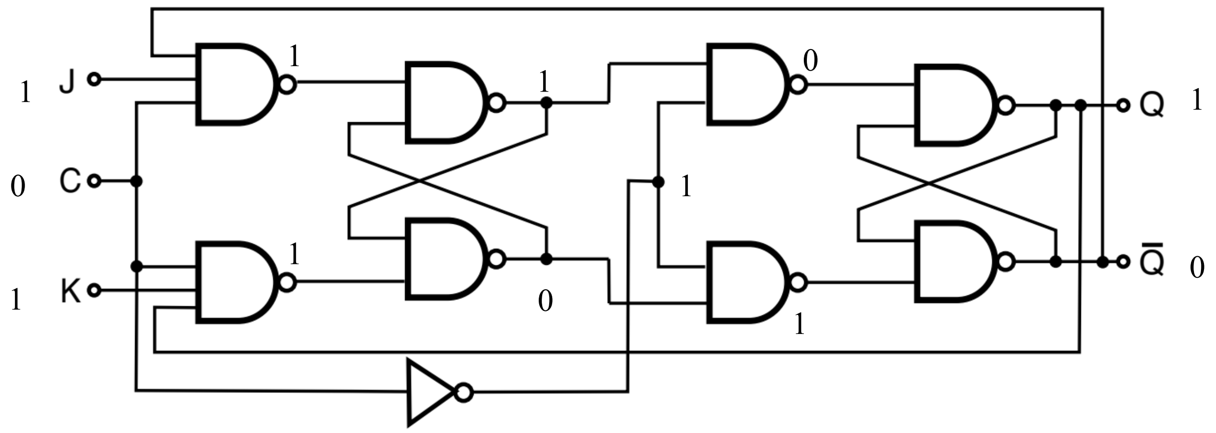
C and J to 0



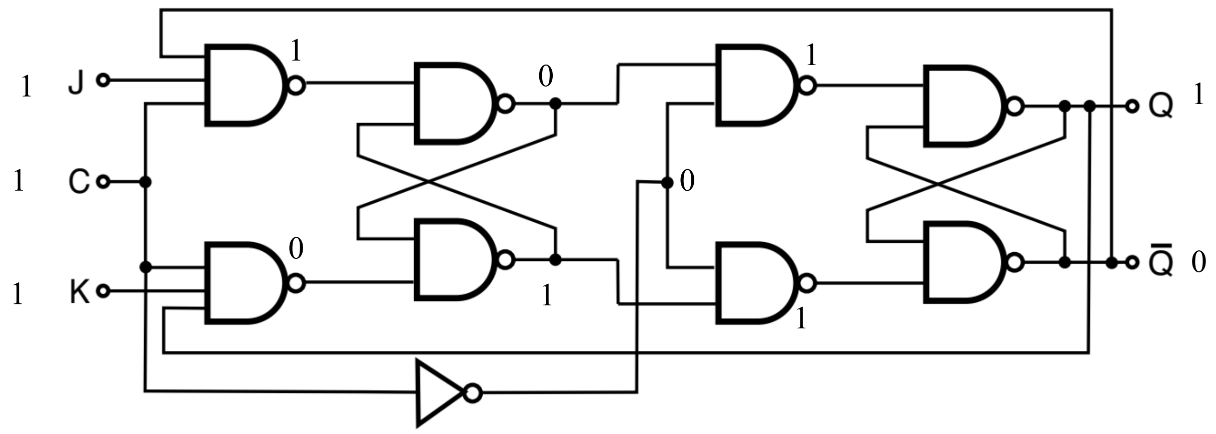
The circuit's output changes to 1

c. **Some time later, J and K both transition to 1 and C transitions to 1 soon after. Create another copy of the drawing and indicate for each line whether it carries a 0 or a 1.**

J and K to 1



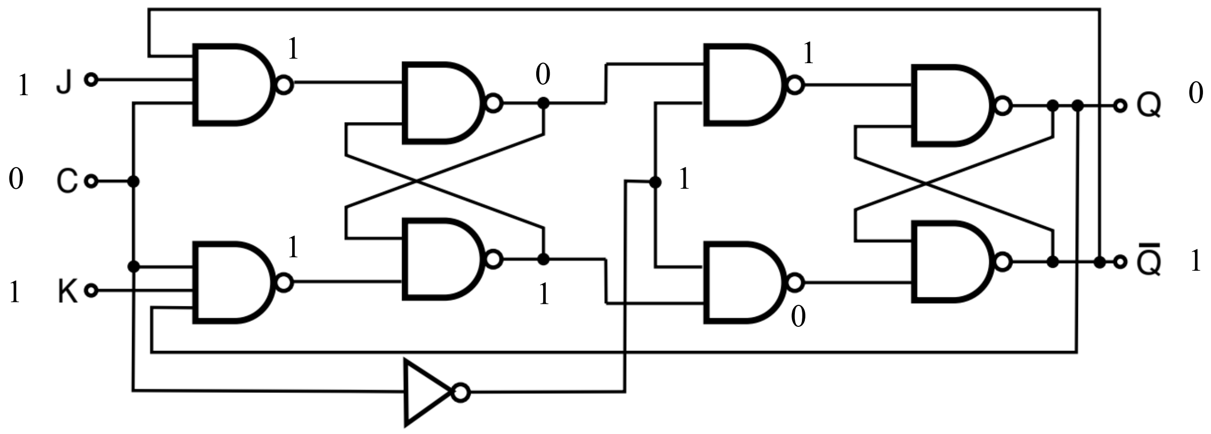
J, K and C to 1



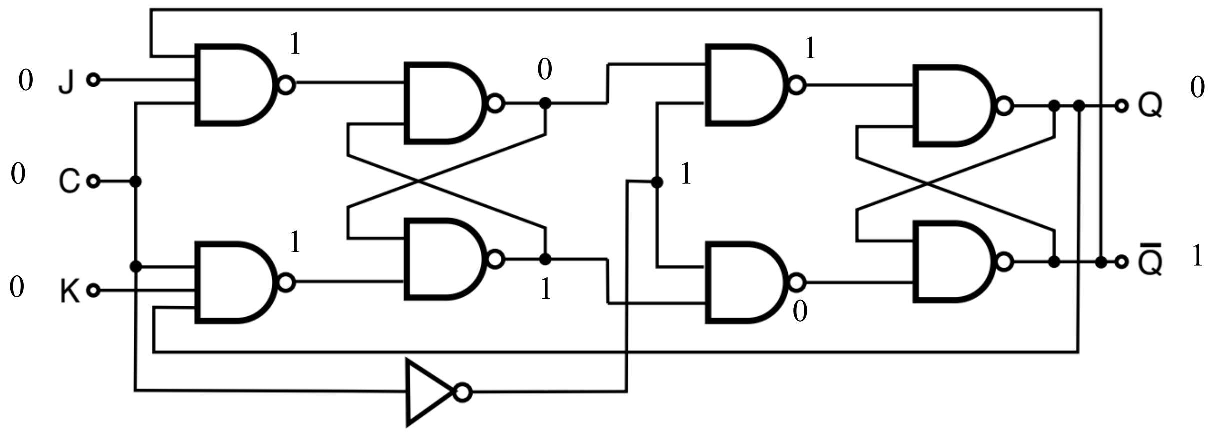
The circuit's output remains 1

d. **Finally, C transitions back to 0 and soon after J and K both transition to 0 as well. Create another copy of the drawing and indicate for each line whether it carries a 0 or a 1.**

C to 0



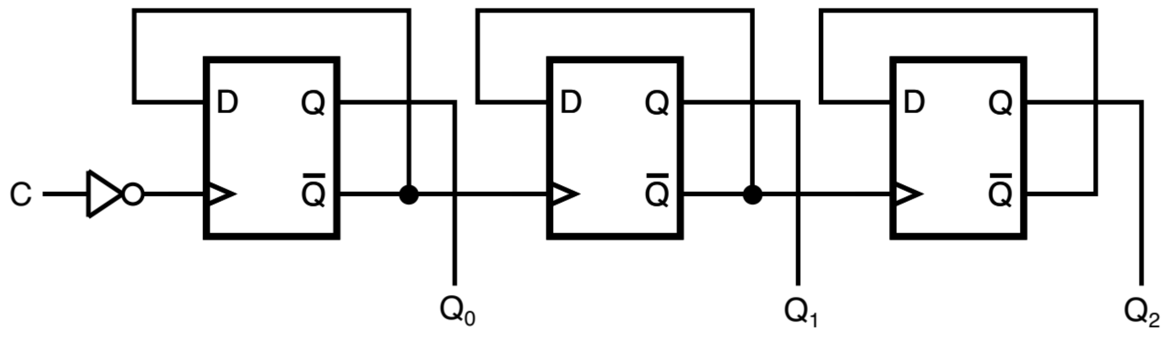
C, J and K to 0



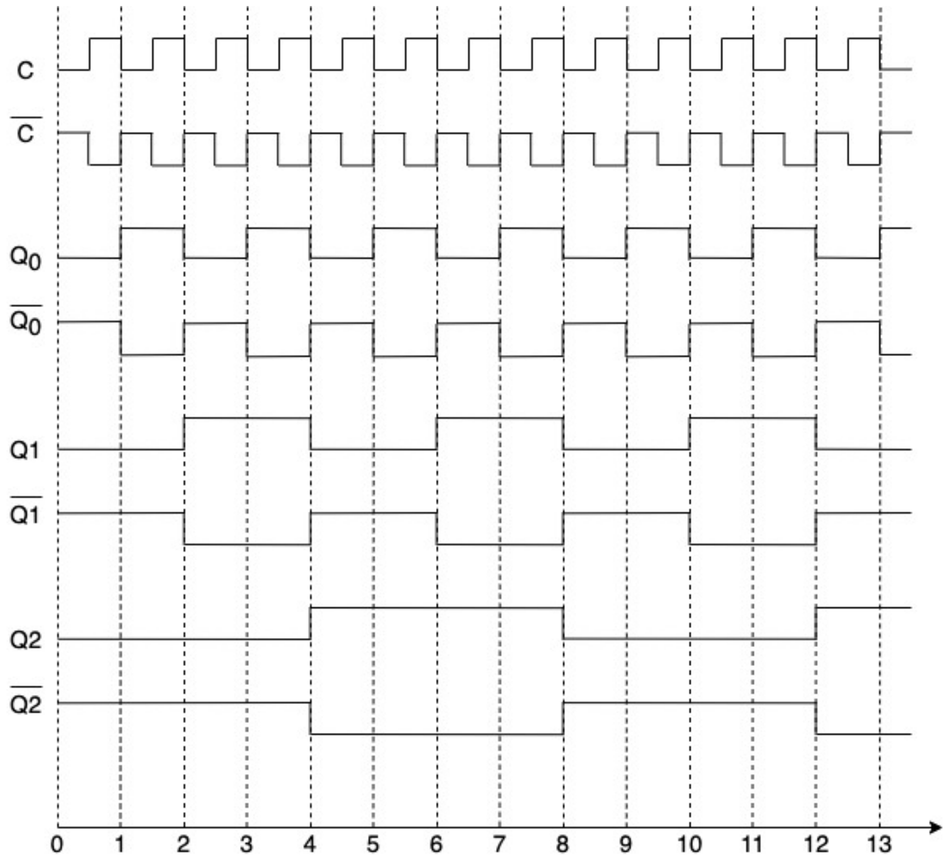
The circuit's output changes back to 0

Problem 9.2: ripple counter using d flip-flops

The following circuit shows a 3-bit ripple counter consisting of three positive edge triggered D flip-flops and a negation gate on the clock input C.



a. **Complete the following timing diagram. Assume that gate delays are very short compared to the speed of the clock signal (i.e., you can ignore the impact of gate delays).**



Timing diagram made from draw.io (app.diagrams.net)

b. **Can you make ripple counters arbitrary "long" or is there a limit on the number of D flip flops that can be chained? Explain.**

In theory, there won't be a limit. The limit would be infinite, you cannot make an infinitely long ripple counter, an n-bit ripple counter.

As we can notice from the timing diagram above, after each positive-edge triggered D-flipflop iteration, the active time increases in an exponential manner.

$\neg$ C lasts for ½ time units.

$\neg$ Q_0 lasts for 1 time units.

$\neg$ Q_1 lasts for 2 time units.

$\neg$ Q_2 lasts for 4 time units.

and so on…

From here we can see that the lasting time units increases at 2^n, where n denotes the n-bit ripple counter.

If the D flip flops ripple counters were chained to infinite numbers, then the time units of the active and inactive period would also be infinite.

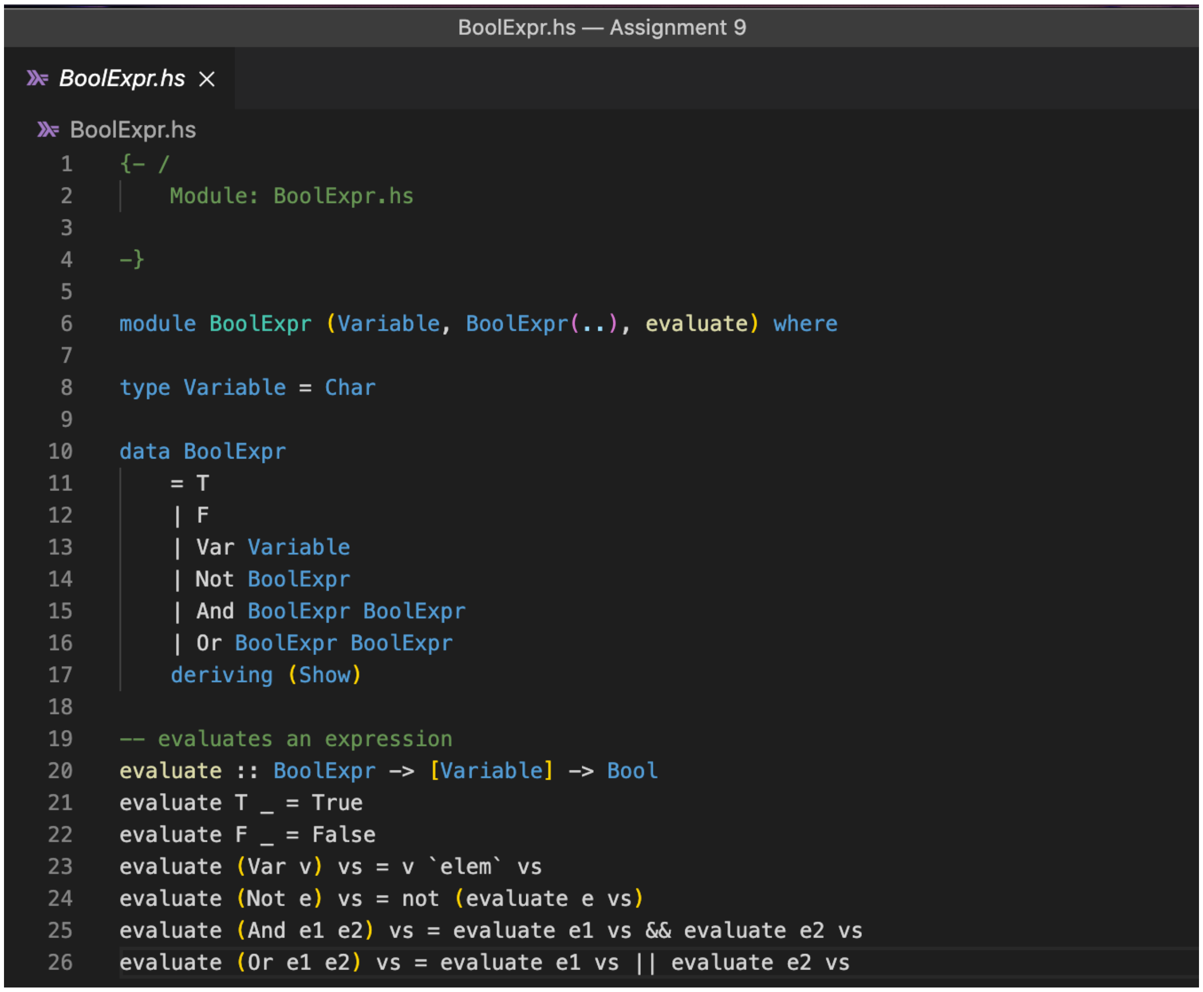
If gate delays were to be taken into account, the outputs in all Q_n will have a delay of n * D flip flop

So, the limit to the chain can be put at **infinite** and any finite value for n would be a valid chain.

Problem 9.3: boolean expressions (haskell)

*(**BoolExpr.txt** and **var_sub_truth.txt** present in the same .zip file)*

*(Given code: **BoolExpr.hs**)*



BoolExpr.hs — Assignment 9

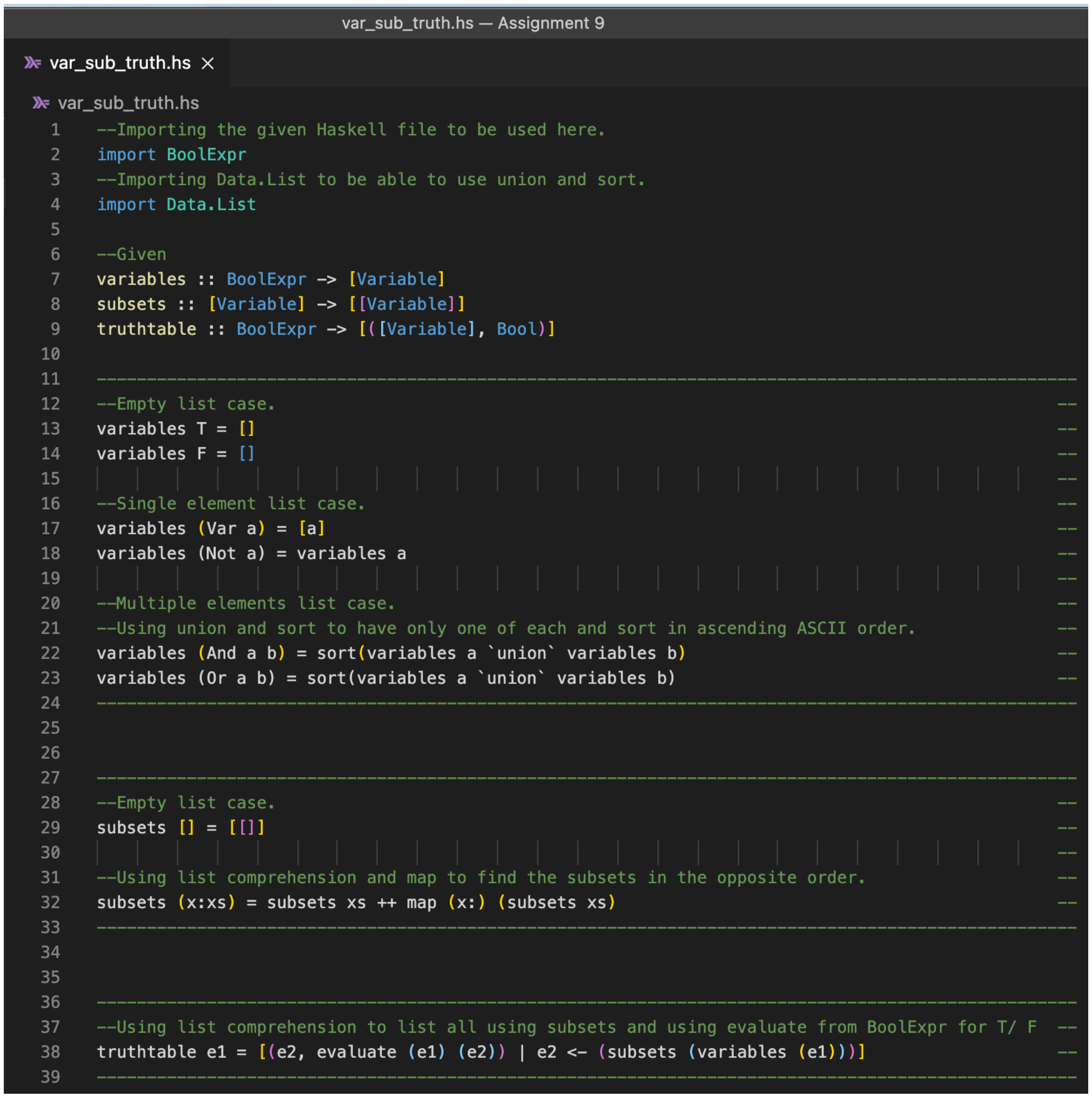
BoolExpr.hs

```haskell
{- /
    Module: BoolExpr.hs

-}

module BoolExpr (Variable, BoolExpr(..), evaluate) where

type Variable = Char

data BoolExpr
    = T
    | F
    | Var Variable
    | Not BoolExpr
    | And BoolExpr BoolExpr
    | Or BoolExpr BoolExpr
    deriving (Show)

-- evaluates an expression
evaluate :: BoolExpr -> [Variable] -> Bool
evaluate T _ = True
evaluate F _ = False
evaluate (Var v) vs = v `elem` vs
evaluate (Not e) vs = not (evaluate e vs)
evaluate (And e1 e2) vs = evaluate e1 vs && evaluate e2 vs
evaluate (Or e1 e2) vs = evaluate e1 vs || evaluate e2 vs
```

a. **Implement a function variables :: BoolExpr -> [Variable], which returns the list of variables that appear in a boolean expression. Feel free to use the Haskell union function to en- sure that there are no duplicates in the list and the Haskell sort function (defined in Data.List) to ensure that the variables are returned in a defined order.**

b. **Implement a function subsets :: [Variable] -> [[Variable]], which returns all subsets of the set of variables passed to the function. Use this function to implement truthtable :: BoolExpr -> [([Variable], Bool)], which returns the entire truth table.**

(var_sub_truth.hs)



```haskell
--Importing the given Haskell file to be used here.
import BoolExpr
--Importing Data.List to be able to use union and sort.
import Data.List

--Given
variables :: BoolExpr -> [Variable]
subsets :: [Variable] -> [[Variable]]
truthtable :: BoolExpr -> [([Variable], Bool)]

-------------------------------------------------------------------------------------------------
--Empty list case.                                                                             --
variables T = []                                                                               --
variables F = []                                                                               --
                                                                                               --
--Single element list case.                                                                    --
variables (Var a) = [a]                                                                        --
variables (Not a) = variables a                                                                --
                                                                                               --
--Multiple elements list case.                                                                 --
--Using union and sort to have only one of each and sort in ascending ASCII order.             --
variables (And a b) = sort(variables a `union` variables b)                                    --
variables (Or a b) = sort(variables a `union` variables b)                                     --
-------------------------------------------------------------------------------------------------


-------------------------------------------------------------------------------------------------
--Empty list case.                                                                             --
subsets [] = [[]]                                                                              --
                                                                                               --
--Using list comprehension and map to find the subsets in the opposite order.                  --
subsets (x:xs) = subsets xs ++ map (x:) (subsets xs)                                           --
-------------------------------------------------------------------------------------------------


-------------------------------------------------------------------------------------------------
--Using list comprehension to list all using subsets and using evaluate from BoolExpr for T/ F  --
truthtable e1 = [(e2, evaluate (e1) (e2)) | e2 <- (subsets (variables (e1)))]                 --
-------------------------------------------------------------------------------------------------
```