Introduction to Computer Science
Jacobs University Bremen
Dr. Jürgen Schönwälder

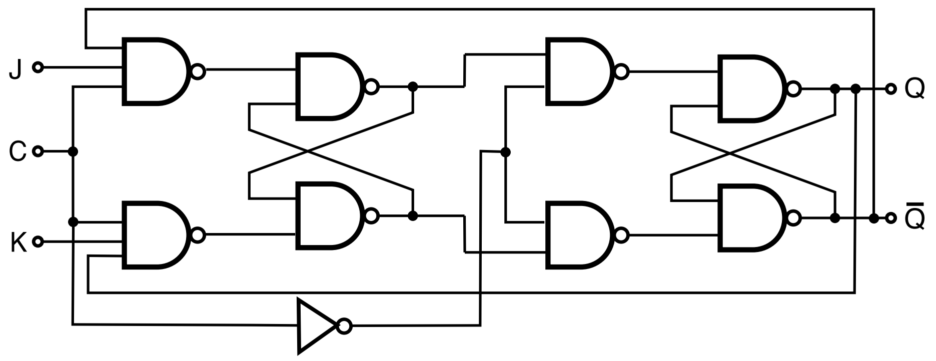
Module: CH-232
Date: 2021-11-05
Due: 2021-11-12

ICS 2021 Problem Sheet #9

Problem 9.1: *JK flip-flops* (1+1+1+1 = 4 points)

JK flip-flops, also colloquially known as jump/kill flip-flops, augment the behaviour of SR flip-flops. The letters J and K were presumably picked by Eldred Nelson in a patent application.

The sequential digital circuit shown below shows the design of a JK flip-flop based on two SR NAND latches. Assume the circuit's output is $Q = 0$ and that the inputs are $J = 0$ and $K = 0$, and that the clock input is $C = 0$. (You can make use of the fact that we already know how an SR NAND latch behaves.)



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.

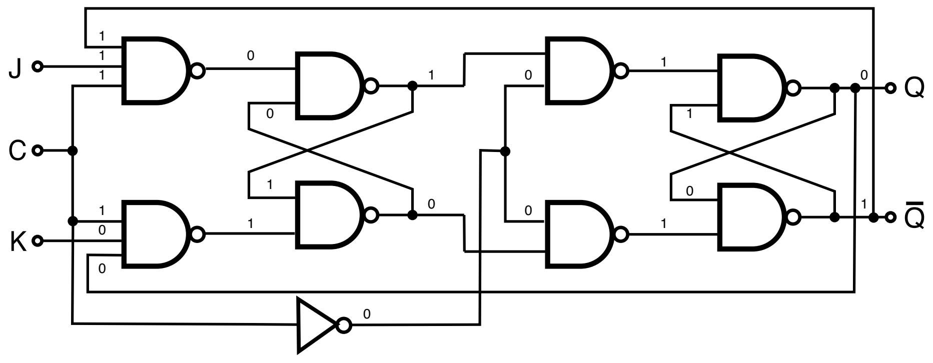
b) Some time later, C transitions back to 0 and soon after J transitions to 0 as well. Create another copy of the drawing and indicate for each line whether it carries a 0 or a 1.

c) 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.

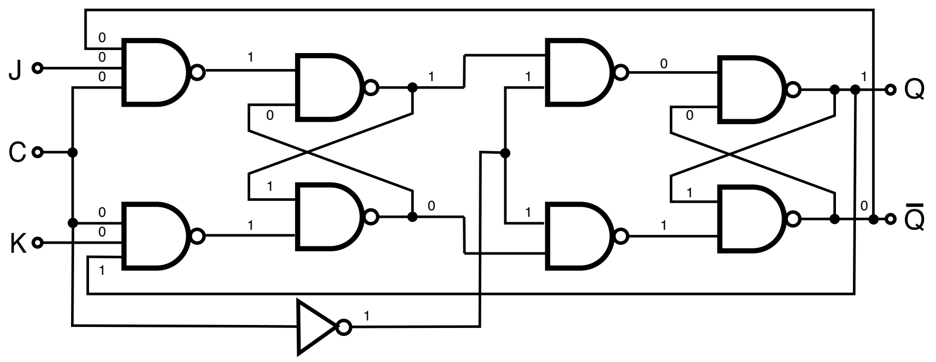
d) Finally, C transitions back to 0 and soon after J and K both transition to 0 as well. Create another copy of the drawing and indicate for each line whether it carries a 0 or a 1.

Solution:

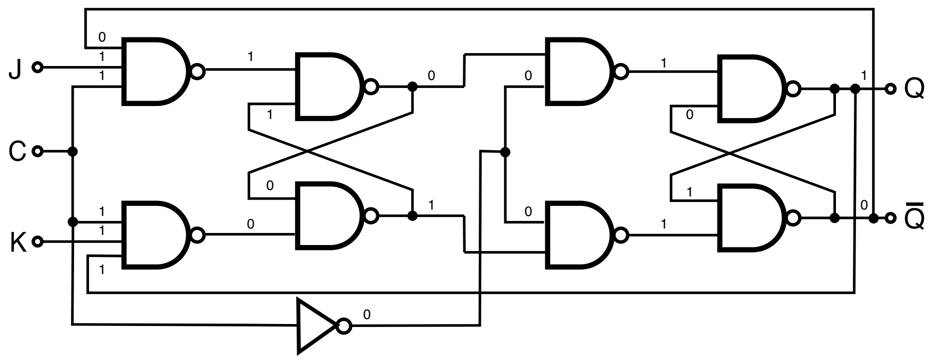
a) $J = 1, K = 0, C = 1$



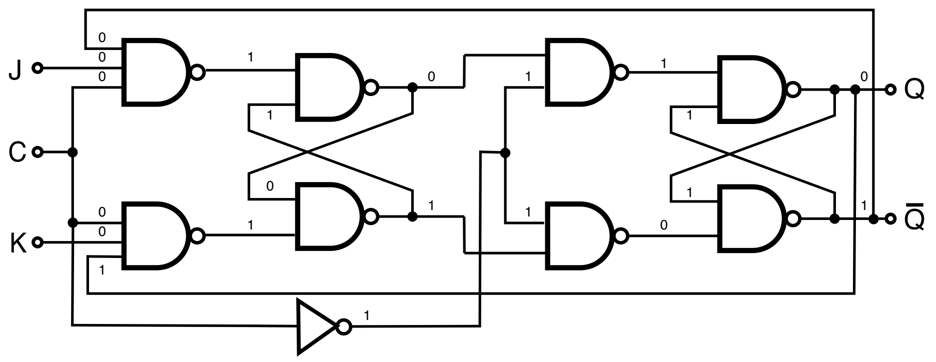
b) $J = 0, K = 0, C = 0$



c) $J = 1, K = 1, C = 1$



d) $J = 0, K = 0, C = 0$



Marking:

a) - *1pt, remove .2pt for each incorrect or missing line annotation, minimum 0pt*

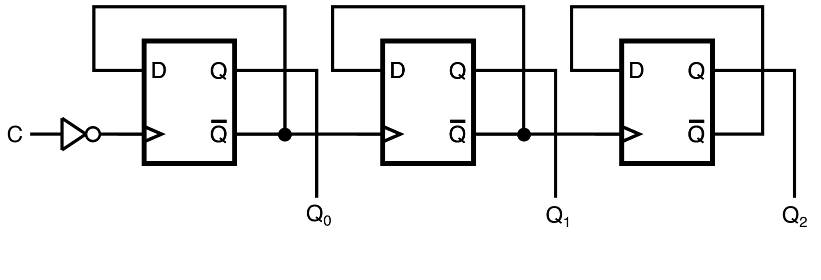
b) - *1pt, remove .2pt for each incorrect or missing line annotation, minimum 0pt*

c) - *1pt, remove .2pt for each incorrect or missing line annotation, minimum 0pt*

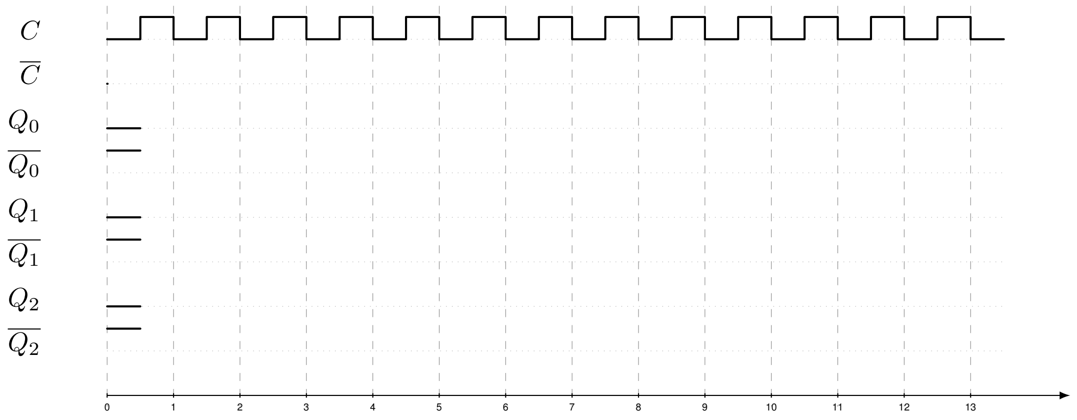
d) - *1pt, remove .2pt for each incorrect or missing line annotation, minimum 0pt*

Problem 9.2: *ripple counter using d flip-flops* (2+1 = 3 points)

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.



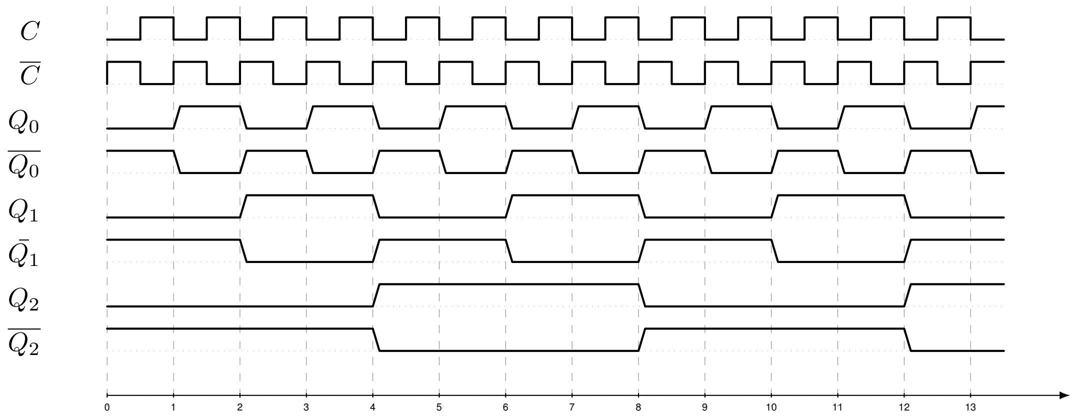
a) Complete the following timing diagram. Assume that gate delays are very short compared to the speed of the clock signal (i.e., you can ignore the impact of gate delays).



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.

Solution:

a) Below is the complete timing diagram:



b) A problem with the ripple counter is that a new arriving bit "ripples" through the circuit and the new counter bits do not arrive all at the same time. In extreme cases, i.e., when the input bits arrive very fast, the ripple counter may even count wrongly.

Marking:

a) - *1pt for getting the frequency division right*
- *1pt for getting positive edge triggering right*

b) - *1pt for a statement that the delay accumulates badly*

Problem 9.3: *integer expression simplification (haskell)* (2+1 = 3 points)

Our goal is to simplify integer expressions that may include constants and variables and which are constructed using a sum and a product operator. For example, we want to write an program that can simplify $(2 \cdot y) \cdot (3 + (2 \cdot 2))$ to $14 \cdot y$. We can represent expressions in Haskell using the following data type:

```haskell
data Exp = C Int          -- a constant integer
         | V String       -- a variable with a name
         | S Exp Exp      -- a sum of two expressions
         | P Exp Exp      -- a product of two expressions
         deriving (Show, Eq)
```

We use the following rules to simplify expressions:

S1 Adding two constants a and b yields a constant, which has the value $a + b$, e.g., $3 + 5 = 8$.

S2 Adding 0 to a variable yields the variable, i.e., $0 + x = x$ and $x + 0 = x$.

S3 Adding a constant a to a sum consisting of a constant b and a variable yields the sum of the $a + b$ and the variable, e.g., $3 + (5 + y) = 8 + y$.

P1 Multiplying two constants a and b yields a constant, which as the value $a \cdot b$, e.g., $3 \cdot 5 = 15$.

P2 Multiplying a variable with 1 yields the variable, i.e., $1 \cdot y = y$ and $y \cdot 1 = y$.

P3 Multiplying a variable with 0 yields the constant 0, i.e., $0 \cdot y = 0$ and $y \cdot 0 = 0$.

P4 Multiplying a constant a with a product consisting of a constant b and a variable yields the product of $a \cdot b$ and the variable, e.g., $3 \cdot (2 \cdot y) = 6 \cdot y$.

The usual associativity rules apply. Note that we have left out distributivity rules.

a) Implement a function `simplify :: Expr -> Expr`, which simplifies expressions that do not contain variables. In other words, `simplify` returns the (constant) value of an expression that does not contain any variables.

b) Extend the function `simplify` to handle variables as described above.

Submit your Haskell code plus an explanation (in Haskell comments) as a plain text file. Below is a template providing a collection of test cases.

```haskell
module Main (main) where

import Test.HUnit

data Exp = C Int        -- a constant integer
         | V String     -- a variable with a name
         | S Exp Exp    -- a sum of two expressions
         | P Exp Exp    -- a product of two expressions
         deriving (Show, Eq)

simplify :: Exp -> Exp
simplify _ = undefined

tI0 = TestList
  [ simplify (C 3) ~?= C 3                        -- 3 = 3
  , simplify (V "y") ~?= V "y"                    -- y = y
  ]

tS1 = TestList
  [ simplify (S (C 3) (C 5)) ~?= C 8              -- 3 + 5 = 8
  ]

tS2 = TestList
  [ simplify (S (C 0) (V "y")) ~?= V "y"          -- 0 + y = y
  , simplify (S (V "y") (C 0)) ~?= V "y"          -- y + 0 = y
  ]

tS3 = TestList
  [ simplify (S (S (C 3) (V "y")) (C 5)) ~?= S (C 8) (V "y")     -- (3 + y) + 5) = 8 + y
  , simplify (S (S (V "y") (C 3) ) (C 5)) ~?= S (C 8) (V "y")    -- (y + 3) + 5) = 8 + y
  , simplify (S (C 3) (S (C 5) (V "y"))) ~?= S (C 8) (V "y")     -- 3 + (5 + y) = 8 + y
  , simplify (S (C 3) (S (V "y") (C 5))) ~?= S (C 8) (V "y")     -- 3 + (y + 5) = 8 + y
  ]
```

```haskell

tS4 = TestList
  [ simplify (S (S (C 3) (C 5)) (C 8)) ~?= C 16                  -- (3 + 5) + 8 = 16
  , simplify (S (C 3) (S (C 5) (C 8))) ~?= C 16                  -- 3 + (5 + 8) = 16
  , simplify (S (C 5) (V "y")) ~?= S (C 5) (V "y")               -- 5 + y = 5 + y
  , simplify (S (V "y") (C 5)) ~?= S (V "y") (C 5)               -- y + 5 = y + 5
  , simplify (S (V "x") (V "y")) ~?= S (V "x") (V "y")           -- x + y = x + y
  ]

tP1 = TestList
  [ simplify (P (C 3) (C 5)) ~?= C 15         -- 3 * 5 = 15
  ]

tP2 = TestList
  [ simplify (P (C 1) (V "y")) ~?= V "y"      -- 1 * y = y
  , simplify (P (V "y") (C 1)) ~?= V "y"      -- y * 1 = y
  ]

tP3 = TestList
  [ simplify (P (C 0) (V "y")) ~?= C 0        -- 0 * y = 0
  , simplify (P (V "y") (C 0)) ~?= C 0        -- y * 0 = 0
  ]

tP4 = TestList
  [ simplify (P (P (C 3) (V "y")) (C 2)) ~?= P (C 6) (V "y")     -- (3 * y) * 2) = 6 * y
  , simplify (P (P (V "y") (C 3) ) (C 2)) ~?= P (C 6) (V "y")    -- (y * 3) * 2) = 6 * y
  , simplify (P (C 3) (P (C 2) (V "y"))) ~?= P (C 6) (V "y")     -- 3 * (2 * y) = 6 * y
  , simplify (P (C 3) (P (V "y") (C 2))) ~?= P (C 6) (V "y")     -- 3 * (y * 2) = 6 * y
  ]

tP5 = TestList
  [ simplify (P (P (C 3) (C 5)) (C 8)) ~?= C 120                 -- (3 * 5) * 8 = 120
  , simplify (P (C 3) (P (C 5) (C 8))) ~?= C 120                 -- 3 * (5 * 8) = 120
  , simplify (P (C 5) (V "y")) ~?= P (C 5) (V "y")               -- 5 * y = 5 * y
  , simplify (P (V "y") (C 5)) ~?= P (V "y") (C 5)               -- y * 5 = y * 5
  , simplify (P (V "x") (V "y")) ~?= P (V "x") (V "y")           -- x * y = x * y
  ]

tM0 = TestList [
  -- (2 * y) * (3 + (2 * 2)) = 14 * y
  simplify (P (P (C 2) (V "y")) (S (C 3) (P (C 2) (C 2)))) ~?= P (C 14) (V "y")
  -- x + (1 + -1) = x
  , simplify (S (V "x") (S (C 1) (C (-1)))) ~?= V "x"
  -- (1 + -1) * x = 0
  , simplify (P (S (C 1) (C (-1))) (V "x")) ~?= C 0
  -- (2 + -1) * x = x
  , simplify (P (S (C 2) (C (-1))) (V "x")) ~?= V "x"
  -- (2 * 2) * (3 + 4) = 28
  , simplify (P (P (C 2) (C 2)) (S (C 3) (C 4))) ~?= C 28
  ]

main = runTestTT $ TestList [tI0, tS1, tS2, tS3, tS4, tP1, tP2, tP3, tP4, tP5, tM0 ]
```

Solution:

```haskell
module Main (main) where

import Test.HUnit

data Exp = C Int          -- a constant integer
         | V String       -- a variable with a name
         | S Exp Exp      -- a sum of two expressions
         | P Exp Exp   -- a product of two expressions
         deriving (Show, Eq)

simplify :: Exp -> Exp
simplify (S a b)
    = case (simplify a, simplify b) of
        (C x, C y) -> C (x + y)
        (C 0, b) -> b
        (a, C 0) -> a
        (C x, S (C y) z) -> S (C (x + y)) z
        (C x, S y (C z)) -> S (C (x + z)) y
        (S (C x) y, C z) -> S (C (x + z)) y
        (S x (C y), C z) -> S (C (y + z)) x
        (a, b) -> S a b
simplify (P a b)
    = case (simplify a, simplify b) of
        (C x, C y) -> C (x * y)
        (C 1, b) -> b
        (a, C 1) -> a
        (C 0, b) -> C 0
        (a, C 0) -> C 0
        (C x, P (C y) z) -> P (C (x * y)) z
        (C x, P y (C z)) -> P (C (x * z)) y
        (P (C x) y, C z) -> P (C (x * z)) y
        (P x (C y), C z) -> P (C (y * z)) x
        (a, b) -> P a b
simplify exp = exp

tI0 = TestList
  [ simplify (C 3) ~?= C 3                          -- 3 = 3
  , simplify (V "y") ~?= V "y"                      -- y = y
  ]

tS1 = TestList
  [ simplify (S (C 3) (C 5)) ~?= C 8                -- 3 + 5 = 8
  ]

tS2 = TestList
  [ simplify (S (C 0) (V "y")) ~?= V "y"            -- 0 + y = y
  , simplify (S (V "y") (C 0)) ~?= V "y"            -- y + 0 = y
  ]

tS3 = TestList
  [ simplify (S (S (C 3) (V "y")) (C 5)) ~?= S (C 8) (V "y")    -- (3 + y) + 5) = 8 + y
  , simplify (S (S (V "y") (C 3) ) (C 5)) ~?= S (C 8) (V "y")   -- (y + 3) + 5) = 8 + y
  , simplify (S (C 3) (S (C 5) (V "y"))) ~?= S (C 8) (V "y")    -- 3 + (5 + y) = 8 + y
  , simplify (S (C 3) (S (V "y") (C 5))) ~?= S (C 8) (V "y")    -- 3 + (y + 5) = 8 + y
  ]

tS4 = TestList
  [ simplify (S (S (C 3) (C 5)) (C 8)) ~?= C 16                 -- (3 + 5) + 8 = 16
  , simplify (S (C 3) (S (C 5) (C 8))) ~?= C 16                 -- 3 + (5 + 8) = 16
  , simplify (S (C 5) (V "y")) ~?= S (C 5) (V "y")              -- 5 + y = 5 + y
  , simplify (S (V "y") (C 5)) ~?= S (V "y") (C 5)              -- y + 5 = y + 5
  , simplify (S (V "x") (V "y")) ~?= S (V "x") (V "y")          -- x + y = x + y
  ]

tP1 = TestList
  [ simplify (P (C 3) (C 5)) ~?= C 15               -- 3 * 5 = 15
  ]

tP2 = TestList
  [ simplify (P (C 1) (V "y")) ~?= V "y"            -- 1 * y = y
  , simplify (P (V "y") (C 1)) ~?= V "y"         -- y * 1 = y
  ]

tP3 = TestList
  [ simplify (P (C 0) (V "y")) ~?= C 0           -- 0 * y = 0
  , simplify (P (V "y") (C 0)) ~?= C 0           -- y * 0 = 0
  ]

tP4 = TestList
  [ simplify (P (P (C 3) (V "y")) (C 2)) ~?= P (C 6) (V "y")     -- (3 * y) * 2) = 6 * y
  , simplify (P (P (V "y") (C 3) ) (C 2)) ~?= P (C 6) (V "y")    -- (y * 3) * 2) = 6 * y
  , simplify (P (C 3) (P (C 2) (V "y"))) ~?= P (C 6) (V "y")     -- 3 * (2 * y) = 6 * y
  , simplify (P (C 3) (P (V "y") (C 2))) ~?= P (C 6) (V "y")     -- 3 * (y * 2) = 6 * y
  ]

tP5 = TestList
  [ simplify (P (P (C 3) (C 5)) (C 8)) ~?= C 120                 -- (3 * 5) * 8 = 120
  , simplify (P (C 3) (P (C 5) (C 8))) ~?= C 120                 -- 3 * (5 * 8) = 120
  , simplify (P (C 5) (V "y")) ~?= P (C 5) (V "y")               -- 5 * y = 5 * y
  , simplify (P (V "y") (C 5)) ~?= P (V "y") (C 5)               -- y * 5 = y * 5
  , simplify (P (V "x") (V "y")) ~?= P (V "x") (V "y")           -- x * y = x * y
  ]

tM0 = TestList [
  -- (2 * y) * (3 + (2 * 2)) = 14 * y
  simplify (P (P (C 2) (V "y")) (S (C 3) (P (C 2) (C 2)))) ~?= P (C 14) (V "y")
  -- x + (1 + -1) = x
  , simplify (S (V "x") (S (C 1) (C (-1)))) ~?= V "x"
  -- (1 + -1) * x = 0
  , simplify (P (S (C 1) (C (-1))) (V "x")) ~?= C 0
  -- (2 + -1) * x = x
  , simplify (P (S (C 2) (C (-1))) (V "x")) ~?= V "x"
  -- (2 * 2) * (3 + 4) = 28
  , simplify (P (P (C 2) (C 2)) (S (C 3) (C 4))) ~?= C 28
  ]

main = runTestTT $ TestList [tI0, tS1, tS2, tS3, tS4, tP1, tP2, tP3, tP4, tP5, tM0 ]
```

Marking:

a) - *1pt for handling flat expressions not including variables*
- *1pt for handling nested expressions not including variables*

b) - *1pt for handling expressions including variables*