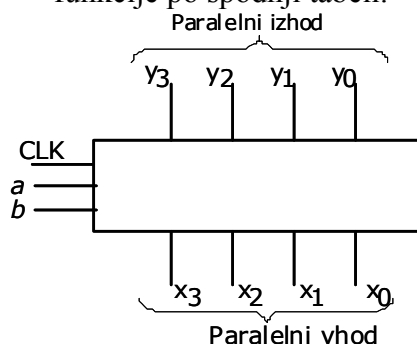


# RAZVOJ DIGITALNIH SISTEMOV

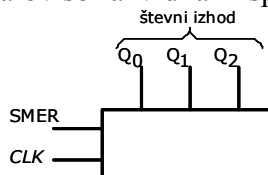
2. kolokvij  
17.1. 2020

1. Realizirajte JK-flip flop z uporabo RS flip flopa in logičnih vrat.
2. Z uporabo univerzalnih logičnih modulov prikažite sintezo univerzalnega 4-bitnega pomikalnega registra, ki ima dva funkcijska vhoda  $a$  in  $b$  in opravlja funkcije po spodnji tabeli.

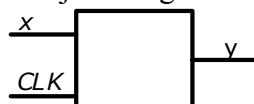


| $a$ | $b$ | <i>funkcija</i>                          |
|-----|-----|--|
| 0   | 0   | briše vsebino registra (CLEAR)           |
| 0   | 1   | rotira vsebino eno mesto desno (ROR)     |
| 1   | 0   | negira vsebino (CPL)                     |
| 1   | 1   | vpiše vsebino s paralelnega vhoda (LOAD) |

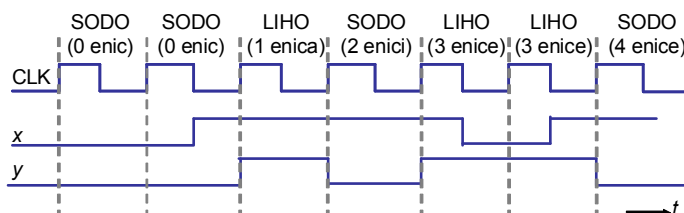
3. Prikažite sintezo sinhronnega dvosmernega 3-bitnega števca z uporabo T flip-flopov: Zapišite tabelo prehajanja stanj in določite enačbe flip-flopov. Števec ima vhod SMER, ki določa smer štetja: Če je SMER='0', števec šteje naraščajoče, sicer padajoče. Imena signalov so razvidna iz spodnje slike.



4. Realizirajte generator lihe parnosti (paritete) kot Moore-ov avtomat končnih stanj, ki šteje število enic v serijskem zaporedju bitov  $x$  na vhodu: Izhod vezja  $y$  naj bo '1', ko je na vhodu liho število enic in '0' ko je na vhodu sodo število enic. Ob resetu avtomata je število enic na vhodu sodo (nič enic). Za realizacijo uporabite D flip-flope, prožene na sprednji rob signala ure  $CLK$ .



Primer delovanja generatorja parnosti povzema spodnja slika:



## Rešitev 1. naloge

Za vezje JK-FF narišemo pravilnostno tabelo, pri čemer na vhodni strani zberemo vhode J, K in trenutno stanje  $Q(t)$ , na izhodni pa naslednje stanje  $Q(t+1)$ . JK-FF opravlja štiri funkcije (HOLD, RESET, SET, INVERT) glede na kombinacijo vhodnih signalov, medtem ko RS-FF opravlja samo dve (RESET, SET).

| J | K | $Q(t)$ | $Q(t+1)$ | R | S | funkcija RS | funkcija JK |
|---|---|--------|----------|---|---|-------------|-------------|
| 0 | 0 | 0      | 0        | 0 | 0 | HOLD        | HOLD        |
| 0 | 0 | 1      | 1        | 0 | 0 | HOLD        | HOLD        |
| 0 | 1 | 0      | 0        | X | 0 | HOLD/RESET  | RESET       |
| 0 | 1 | 1      | 0        | 1 | 0 | RESET       | RESET       |
| 1 | 0 | 0      | 1        | 0 | 1 | SET         | SET         |
| 1 | 0 | 1      | 1        | 0 | X | HOLD/SET    | SET         |
| 1 | 1 | 0      | 1        | 0 | 1 | SET         | INVERT      |
| 1 | 1 | 1      | 0        | 1 | 0 | RESET       | INVERT      |

Iz tabele narišemo Veitchev diagram za vhoda R in S v odvisnosti od vhodov J, K in trenutnega stanja  $Q(t)$ .

**R:**

|     |        |   |   |   |
|-----|--------|---|---|---|
|     | $J$    |   |   |   |
| $K$ | 0      | 1 | 1 | X |
|     | 0      | 0 | 0 | 0 |
|     | $Q(t)$ |   |   |   |

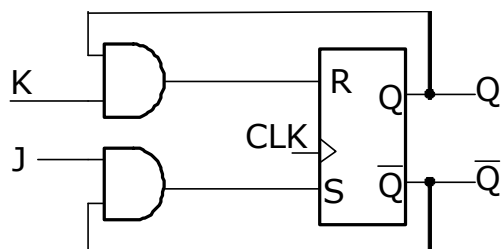
$$R = K \cdot Q(t)$$

**S:**

|     |        |   |   |   |
|-----|--------|---|---|---|
|     | $J$    |   |   |   |
| $K$ | 1      | 0 | 0 | 0 |
|     | 1      | X | 0 | 0 |
|     | $Q(t)$ |   |   |   |

$$S = J \cdot \overline{Q(t)}$$

Vezje narišemo:



Čas pisanja je 60 minut. Vsaka naloga je vredna 10 točk. Na list z rešitvami se podpišite in napišite še vpisno številko ter kateri predmet pišete. Rezultati bodo objavljeni na <http://estudent.fri.uni-lj.si/fe.html>

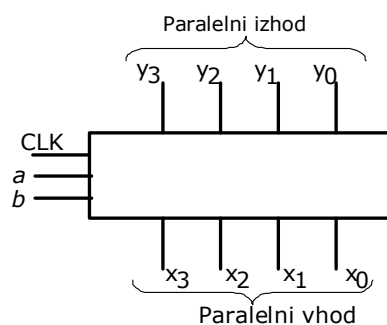
Rešitev 2. naloge:

Naloga zahteva realizacijo univerzalnega registra s funkcijami

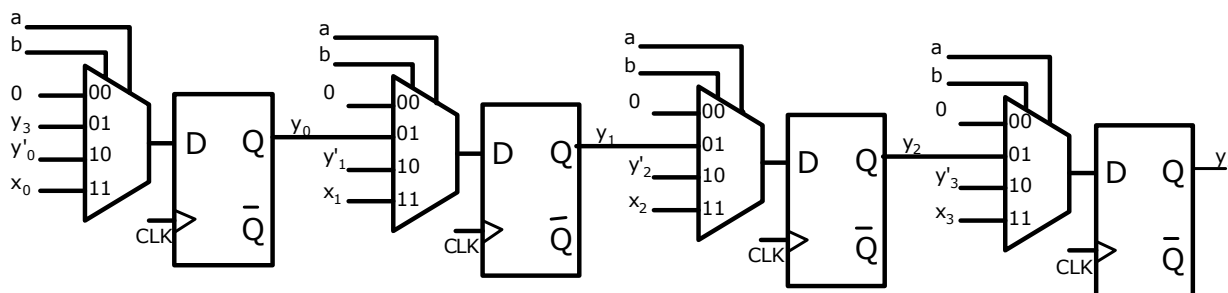
| $a$ | $b$ | $funkcija$                               |
|-----|-----|--|
| 0   | 0   | briše vsebino registra (CLEAR)           |
| 0   | 1   | rotira vsebino eno mesto desno (ROR)     |
| 1   | 0   | negira vsebino (CPL)                     |
| 1   | 1   | vpiše vsebino s paralelnega vhoda (LOAD) |

Vsako od operacij izpišemo v pravilnostno tabelo v kateri združimo funkcijska bita  $a$ ,  $b$  in trenutno stanje na  $i$ -tem mestu registra  $y_i(t)$ . Realizacija z D flip-flopi nam analizo močno poenostavi, zaradi enačbe D flip-flopa:  $D = y_{i+1}(t)$

| $a$ | $b$ | $y_i(t)$ | $y_i(t+1)$   | $D$          |
|-----|-----|----------|--------------|--------------|
| 0   | 0   | 0        | 0            | 0            |
| 0   | 0   | 1        | 0            | 0            |
| 0   | 1   | 0        | $y_{i-1}(t)$ | $y_{i-1}(t)$ |
| 0   | 1   | 1        | $y_{i-1}(t)$ | $y_{i-1}(t)$ |
| 1   | 0   | 0        | $y_i'$       | $y_i'$       |
| 1   | 0   | 1        | $y_i'$       | $y_i'$       |
| 1   | 1   | 0        | $x_i$        | $x_i$        |
| 1   | 1   | 1        | $x_i$        | $x_i$        |



Register izvaja rotacijo, torej nima serijskega vhoda in izhoda, ampak LSB bit  $y_3$  vodimo na MSB bit  $y_0$ . Naloga zahteva realizacijo z 4/1 izbiralniki, s katerimi ločimo 4 operacije registra.



Čas pisanja je 60 minut. Vsaka naloga je vredna 10 točk. Na list z rešitvami se podpišite in napišite še vpisno številko ter kateri predmet pišete. Rezultati bodo objavljeni na <http://estudent.fri.uni-lj.si/fe.html>

Rešitev 3. naloge:

Postopek sinteze zahteva, da zapišemo tabelo prehajanja stanj števca:

| SMER | Q <sub>2</sub> | Q <sub>1</sub> | Q <sub>0</sub> | Q <sub>2</sub> | Q <sub>1</sub> | Q <sub>0</sub> | T <sub>2</sub> | T <sub>1</sub> | T <sub>0</sub> |
|------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| 0    | 0              | 0              | 0              | 0              | 0              | 1              | 0              | 0              | 1              |
| 0    | 0              | 0              | 1              | 0              | 1              | 0              | 0              | 1              | 1              |
| 0    | 0              | 1              | 0              | 0              | 1              | 1              | 0              | 0              | 1              |
| 0    | 0              | 1              | 1              | 1              | 0              | 0              | 1              | 1              | 1              |
| 0    | 1              | 0              | 0              | 1              | 0              | 1              | 0              | 0              | 1              |
| 0    | 1              | 0              | 1              | 1              | 1              | 0              | 0              | 1              | 1              |
| 0    | 1              | 1              | 0              | 1              | 1              | 1              | 0              | 0              | 1              |
| 0    | 1              | 1              | 1              | 0              | 0              | 0              | 1              | 1              | 1              |
| 1    | 0              | 0              | 0              | 1              | 1              | 1              | 1              | 1              | 1              |
| 1    | 0              | 0              | 1              | 0              | 0              | 0              | 0              | 0              | 1              |
| 1    | 0              | 1              | 0              | 0              | 0              | 1              | 0              | 1              | 1              |
| 1    | 0              | 1              | 1              | 0              | 1              | 0              | 0              | 0              | 1              |
| 1    | 1              | 0              | 0              | 0              | 1              | 1              | 1              | 1              | 1              |
| 1    | 1              | 0              | 1              | 1              | 0              | 0              | 0              | 0              | 1              |
| 1    | 1              | 1              | 0              | 1              | 0              | 1              | 0              | 1              | 1              |
| 1    | 1              | 1              | 1              | 1              | 1              | 0              | 0              | 0              | 1              |

Normalna analiza bi zahtevala, da narišemo Veitch–eve diagrame za štiri spremenljivke za vsak vhod T–FF, vendar ker so T–FF po svoji naravi primerni za realizacijo števec, so praviloma njihove vhodne enačbe zelo enostavne. Iz tabele prehajanja stanj števca določimo enačbe T–FF:

Iz stolpca T<sub>0</sub> se vidi, da je T<sub>0</sub>='1'. Kot zanimivost omenimo dejstvo, ki se ponavlja pri večini sinhronih števec: Če namesto '1' na T<sub>0</sub> vodimo nek zunanji signal, nam to omogoča štetje števca (ENABLE oz. EN).

Iz stolpca T<sub>1</sub> se vidi, da se ponavlja vzorec 01, če je SMER='0' in 10, če je SMER='1'.

| SMER | T <sub>1</sub>   |
|------|------------------|
| 0    | Q <sub>0</sub>   |
| 1    | Q <sub>0</sub> ' |

kar lahko kratko zapišemo kot:

$$T_1 = \text{SMER} \cdot \overline{Q_0} + \overline{\text{SMER}} \cdot Q_0 = \text{SMER} \oplus Q_0$$

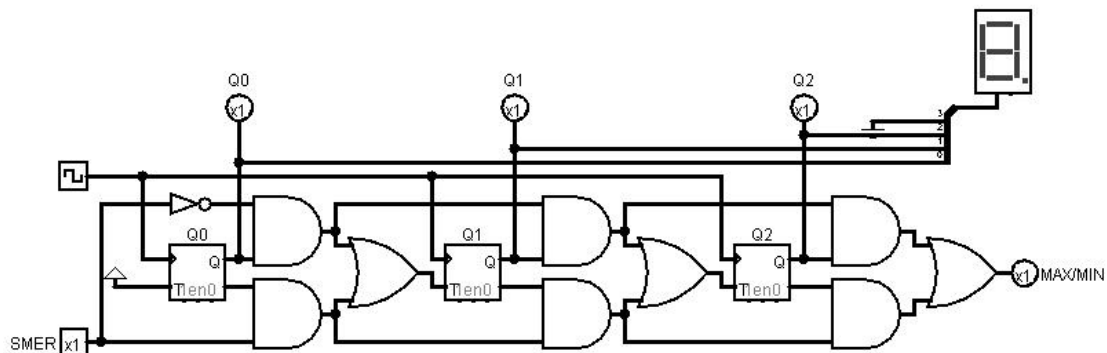
Za T<sub>2</sub> se da enostavno ugotoviti realizacijo iz Veitch–evega diagrama:

|                |   |   |   |
|----------------|---|---|---|
| SMER           |   |   |   |
| Q <sub>2</sub> | 1 | 0 | 0 |
|                | 0 | 0 | 1 |
|                | 0 | 0 | 1 |
|                | 1 | 0 | 0 |
| Q <sub>1</sub> |   |   |   |
| Q <sub>0</sub> |   |   |   |

$$T_2 = \text{SMER} \cdot \overline{Q_1} \cdot \overline{Q_0} + \overline{\text{SMER}} \cdot Q_1 \cdot Q_0$$

V enačbi za T<sub>2</sub> poiščemo podobnosti z enačbo za T<sub>1</sub>: Enačba za T<sub>1</sub> vsebuje konjunkciji SMER·Q<sub>0</sub>' in SMER'·Q<sub>0</sub>, ki sta vsebovani tudi v enačbi za T<sub>2</sub>, kar nam dodatno poenostavi realizacijo števca. Obenem nam taka realizacija nakazuje osnovno strukturo, ki jo lahko s ponavljanjem razširimo v večbitni dvosmerni sinhroni števec.

Primer podobnega vezja 4-bitnega dvojiškega dvosmernega števca, ki ima še vzporedno nalaganje je 74191<sup>1</sup>. Če boste primerjali našo realizacijo in realizacijo v podatkovnem listu, boste opazili, da je v dejanski realizaciji 74191 precej več večvhodnih AND vrat: Delno je razlog za to v dodani logiki za vzporedno nalaganje, delno pa tudi zato, da zagotovimo enakomerno zakasnitev med posameznimi stopnjami števca.



Ko narišemo vezje dvosmernega števca, zelo spominja na združitev sinhronnega števca za štetje navzgor in sinhronnega števca za štetje navzdol: Če bi števec vseboval samo zgornja AND vrata (vezanih neposredno na T vhod – brez OR) bi bil to števec navzgor, če pa samo spodnja AND bi bil števec navzdol. Signal SMER določa katera AND vrata so omogočena:

- zgornja AND vrata, ko je SMER='0' – štejemo naraščajoče,
- spodnja AND vrata, ko je SMER='1' – štejemo padajoče.

Zahteve naloge so s tem izpolnjene. V nadaljevanju bomo pokazali še dodajanje ostalih krmilnih signalov (EN - omogočanje štetja, RCO, MAX/MIN, LOAD - vzporedno nalaganje).

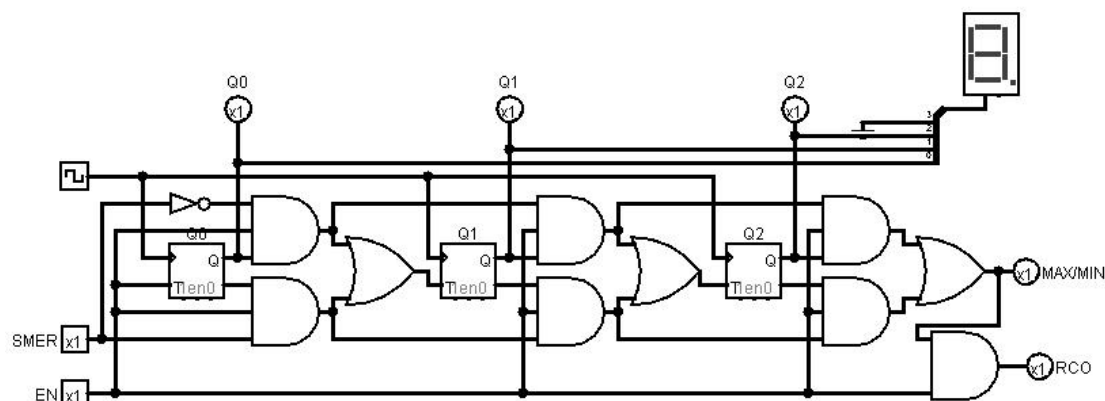
Signal EN je signal za omogočanje štetja – dokler je EN='0' se vsebina vseh T-FF ne spreminja, ampak ohranja trenutno stanje. Iz sheme števca je razviden tudi način razširitve števca na večje število bitov.

Pri tovrstnih števcih želimo realizirati tudi signal za proženje naslednjih stopenj števca RCO (ang.. ripple carry out) poimenovan včasih tudi TC (ang. terminal count), oz. tudi RC (ang. ripple clock). RCO je signal, ki postane '1', ko števec preide iz najvišjega stanja (v našem primeru "111") v stanje "000" pri štetju navzgor in ko preide iz najnižjega stanja "000" v najvišje stanje ("111") pri štetju navzdol:

| SMER | RCO                          |
|------|------------------------------|
| 0    | $Q_0 \cdot Q_1 \cdot Q_2$    |
| 1    | $Q_0' \cdot Q_1' \cdot Q_2'$ |

Tak signal uporabljamo pri realizaciji večbitnih števcov tako, da izdelane 3 bitne števice vezemo kaskadno – torej da signal RCO vezemo na EN signal naslednjega vezja. Za realizacijo takega signala bi narisali enako kombinacijo AND in OR vrat še na izhodu  $Q_2$ , kot kaže naslednja slika:

<sup>1</sup> <http://www.alldatasheet.com/view.jsp?Searchword=74191>



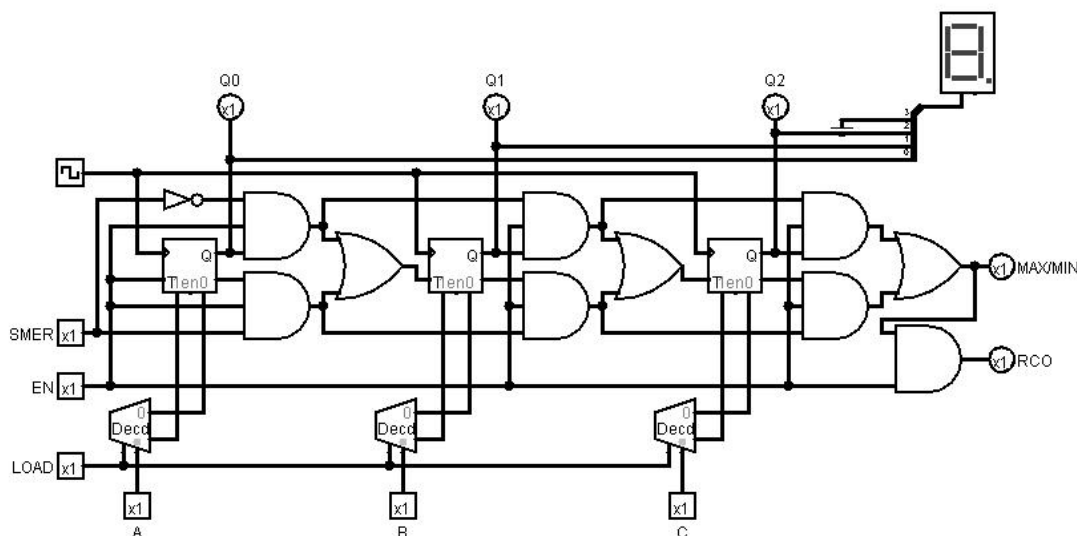
Vezje se nahaja v Logisim predlogah rešenih nalog na domači strani predmeta:  
Logisim\counter\counter\_up\_down\_3\_bit\_using\_T\_FF\_with\_enable.circ

Večina števecv je realizirana v 4-bitni zasnovi, tako da glede na vrednost RCO signala ločimo dve skupini števecv:

- desetiški (BCD) števc, katerih RCO se postavi na '1' takrat, ko števec preide iz stanja "1001" v "0000" in
- dvojiški (binarni), katerih RCO se postavi na '1' takrat, ko števec preide iz stanja "1111" v "0000". Več o delovanju RCO najdete v opisu delovanja števecv 74161<sup>2</sup>.

Števču dodamo še signal za asinhrono vzporedno nalaganje (LOAD), ki v aktivnem stanju (LOAD='1') asinhrono nastavi vrednosti T-FF na vrednosti vhodov za nalaganje C, B, A. V števcu 74191<sup>3</sup> je izveden z enostavnim dekodeerjem, ki postavi ustrezni vhod za asinhrono postavljanje T-FF (ang. preset) na '1', če je vrednost vhoda za nalaganje '1'. Če je vrednost vhoda za nalaganje '0', potem se '1' postavi na asinhroni vhod za brisanje T-FF (ang. clear).

Vezje se nahaja v Logisim predlogah rešenih nalog na domači strani predmeta:  
Logisim\counter\counter\_up\_down\_3\_bit\_using\_T\_FF\_with\_enable\_with\_load.circ



<sup>2</sup> <http://www.alldatasheet.com/view.jsp?Searchword=74161>

<sup>3</sup> <http://www.alldatasheet.com/view.jsp?Searchword=74191>

Čas pisanja je 60 minut. Vsaka naloga je vredna 10 točk. Na list z rešitvami se podpišite in napišite še vpisno številko ter kateri predmet pišete. Rezultati bodo objavljeni na <http://estudent.fri.uni-lj.si/fe.html>

#### Rešitev 4. naloge:

Postopek sinteze zahteva, da realiziramo avtomat končnih stanj Moore-ove izvedbe: Najprej bomo razvili diagram prehajanja stanj, ki opisuje delovanje vezja za liho preverjanje parnosti. Vezje je lahko v enem od dveh stanj: v sekvenci je bilo do tega trenutka liho ali sodo število enic. Kadar je na vhodu 1, je potrebno preklopiti v drugo stanje. Na primer, če je bilo do tega trenutka prisotnih liho število enic in je trenutni vhod 1, potem bomo imeli sedaj sodo število enic. Če pa bo na vhodu 0, ostane v istem stanju. Narisani diagram prehajanja stanj ima dve stanji, ki označujeta trenutno število enic na vhodu – torej LIHO in SODO. Izhod zapišemo pod stanjem (LIHO='1', SODO='0'). Vrednosti na vhodu  $x$  povzročajo spreminjanje stanj, ki so označene z usmerjenimi povezavami. Če je se na vhodu pojavi '0' (ne glede na to v katerem stanju smo) ostanemo v tem stanju: Jasno – saj štejemo samo '1'. Če smo v stanju LIHO in se na vhodu pojavi '1', preidemo v SODO. Če smo v stanju SODO in se na vhodu pojavi '1', preidemo v LIHO. Povedano povzema spodnji diagram prehajanja stanj

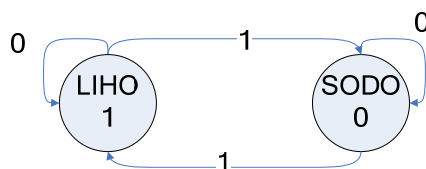


Diagram prehajanja stanj opišemo s tabelo prehajanja stanj:

| vhod<br>$x$ | trenutno<br>stanje<br>$Q(t)$ | naslednje<br>stanje<br>$Q(t+1)$ |
|-------------|------------------------------|---------------------------------|
| 0           | SODO                         | SODO                            |
| 0           | LIHO                         | LIHO                            |
| 1           | SODO                         | LIHO                            |
| 1           | LIHO                         | SODO                            |

Če stanja kodiramo glede na njihov izhod LIHO '1' in SODO '0', potem dobimo novo aplikacijsko tabelo prehajanja stanj.

| $x$ | $Q(t)$ | $Q(t+1)$ | $D$ | $y$ |
|-----|--------|----------|-----|-----|
| 0   | 0      | 0        | 0   | 0   |
| 0   | 1      | 1        | 1   | 0   |
| 1   | 0      | 1        | 1   | 1   |
| 1   | 1      | 0        | 0   | 1   |

Iz tabele prehajanja stanj avtomata določimo enačbo za D-FF. Potrebno število FF je 1, saj sta stanji samo dve.

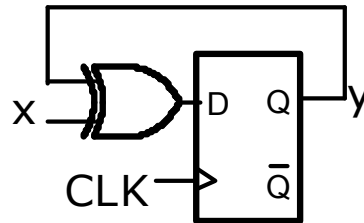
Iz aplikacijske tabele sledi:

$$D = Q(t+1)$$

$$Q(t+1) = x \cdot \overline{Q(t)} + \overline{x} \cdot Q(t) = x \oplus Q(t)$$

$$y = Q(t)$$

Izvedba vezja je:



Realizirali smo T-FF, prožen na sprednji rob signala ure.

Delovanje vezja si lahko ogledate v predlogah Logisim na domači strani predmeta:

Logisim\ff\T\_ff\_using\_D\_ff\_and\_xor.circ