Λύσεις Ιουνίου 2024

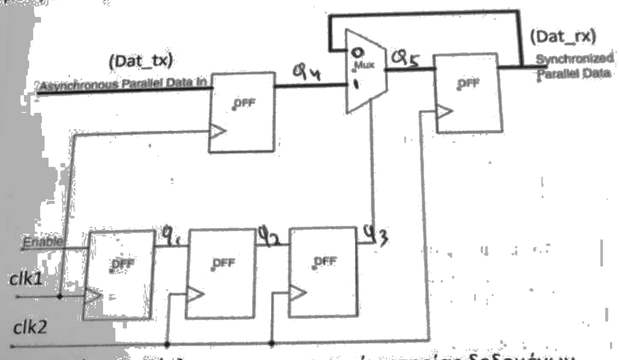
Τρίτη, 18 Ιουνίου 2024

Θέμα 1: Το παρακάτω κύκλωμα υλοποιεί ένα κύκλωμα συγχρονισμού για μία αρτηρία δεδομένων που μεταφέρει δεδομένα μεταξύ δύο διαφορετικών περιοχών ρολογιού, οι οποίες χρονίζονται από τα σήματα clk1 και clk2, αντίστοιχα.

α) Υλοποιήστε το κύκλωμα αυτό αποκλειστικά με συμπεριφορική SystemVerilog. (2 μονάδες)

β) Υλοποιήστε εκ νέου το κύκλωμα αυτό με SystemVerilog όπου ο πολυπλέκτης και το D-FF υλοποιούνται σα χωριστά υποκυκλώματα συμπεριφορικά ενώ το υπόλοιπο κύκλωμα υλοποιείται από στιγμιότυπα (instances) των κυκλωμάτων αυτών σύμφωνα με το Σχήμα 1. (1 μονάδα)

γ) Εξηγήστε σύντομα πότε το κύκλωμα αυτό μπορεί να μη δειγματοληπτεί σωστά. Σε αυτή την περίπτωση περιγράψτε σύντομα μία εναλλακτική λύση που δεν πάσχει από τους περιορισμούς αυτούς. (1 μονάδα)



Σχήμα 1. Κύκλωμα συγχρονισμού αρτηρίας δεδομένων.

Προτεινόμενη διάρκεια εξέτασης: 45 λεπτά

α), β) Δες αρχείο .sv

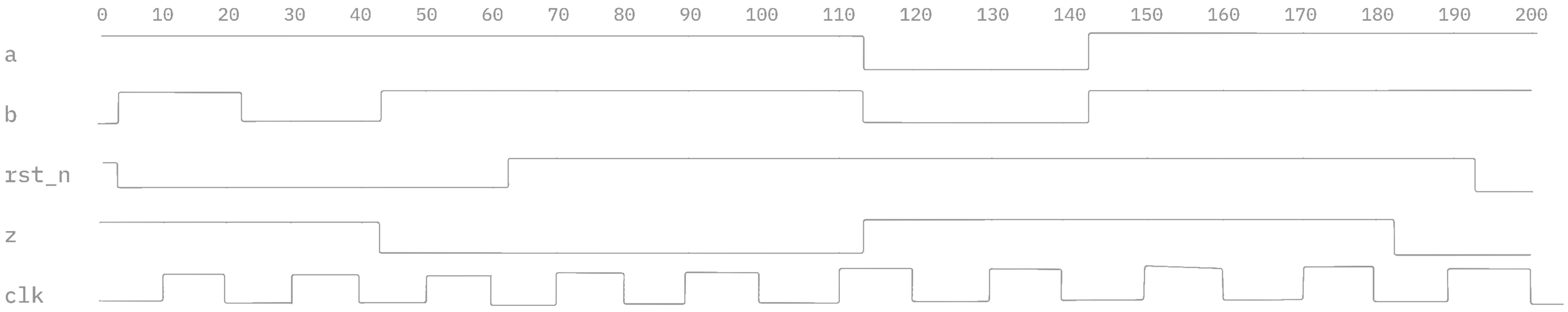
γ) Στην συγκεκριμένη άσκηση έχουμε clock domain crossing, δηλαδή μεταφέρουμε δεδομένα από μία περιοχή ρολογιού (clk1) σε μια άλλη (clk2).

Το πρόβλημα εδώ είναι ότι το q1 μπορεί να αλλάξει τιμή σε μια χρονική στιγμή που το clk2 μεταβαίνει στο posedge. Στην ουσία έχουμε time violation και μεταβαίνουμε σε μια κατάσταση "μετασταθερότητας" που το σήμα q2 δεν είναι ούτε 0 ούτε 1.

Με το DFF3 (με έξοδο q3) δίνουμε χρόνο στο κύκλωμα να επιλύσει αυτήν την αβεβαιότητα, αλλά πάλι υπάρχει πιθανότητα να μην έχει επιλυθεί μέχρι την χρονική στιγμή που το q3 (δηλ το Enable) εισέρχεται στο mux. Αν το q3 βρίσκεται ακόμα σε metastable state, τότε δεν ξέρουμε τι θα γίνει στο mux.

Για να επιλυθεί αυτό το πρόβλημα, η καλύτερη λύση είναι να δημιουργήσουμε μια *Asynchronous FIFO* με write clock το clk1, και read clock το clk2. Έτσι μπορούμε να μεταφέρουμε δεδομένα από το ένα clock domain στο άλλο με ασφάλεια.

Εναλλακτικά, μπορούμε να προσθέσουμε παραπάνω FF μετά το q3 για να του δώσουμε παραπάνω χρόνο, αλλά δεν θεωρείται ιδανική λύση (απλά αυξάνουμε το MTBF, δεν θεωρείται robust solution).



p1: a |-> b[*3]

```
p1 FAIL @ 30ns / started @ 10ns
p1 FAIL @ 30ns / started @ 30ns
p1 PASS @ 90ns / started @ 50ns
p1 PASS @ 110ns / started @ 70ns
p1 FAIL @ 130ns / started @ 90ns
p1 FAIL @ 130ns / started @ 110ns
p1 PASS @ 190ns / started @ 150ns
```

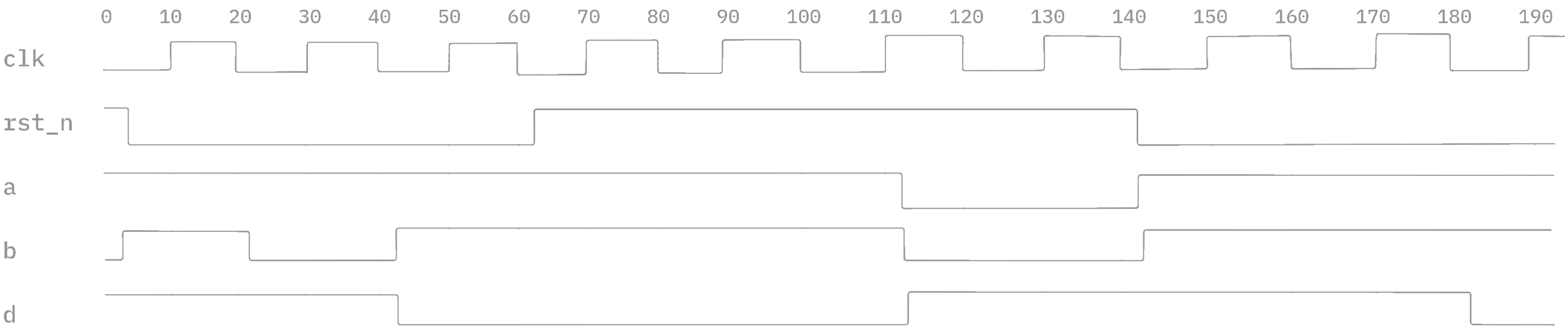
p2: disable iff(!rst_n) a |-> b[=2]

```
p2 PASS @ 90ns / started @ 70ns
p2 PASS @ 110ns / started @ 90ns
p2 PASS @ 150ns / started @ 110ns
p2 PASS @ 170ns / started @ 150ns
p2 PASS @ 190ns / started @ 170ns
```

p3: disable iff(!rst_n) a |-> b[->2:3] ##1 $fell(z)

```
p3 FAIL @ 130ns / started @ 70ns
p3 FAIL @ 170ns / started @ 90ns
p3 PASS @ 190ns / started @ 110ns
p3 PASS @ 190ns / started @ 150ns
```

Λύσεις Ιουνίου 2023



p1: a |-> b[->2]

p1 PASS @ 50ns / started @ 10ns
p1 PASS @ 70ns / started @ 30ns
p1 PASS @ 70ns / started @ 50ns
p1 PASS @ 90ns / started @ 70ns
p1 PASS @ 110ns / started @ 90ns
p1 PASS @ 150ns / started @ 110ns
p1 PASS @ 170ns / started @ 150ns
p1 PASS @ 190ns / started @ 170ns

p2: disable iff(!rst_n) a |-> b[*2]

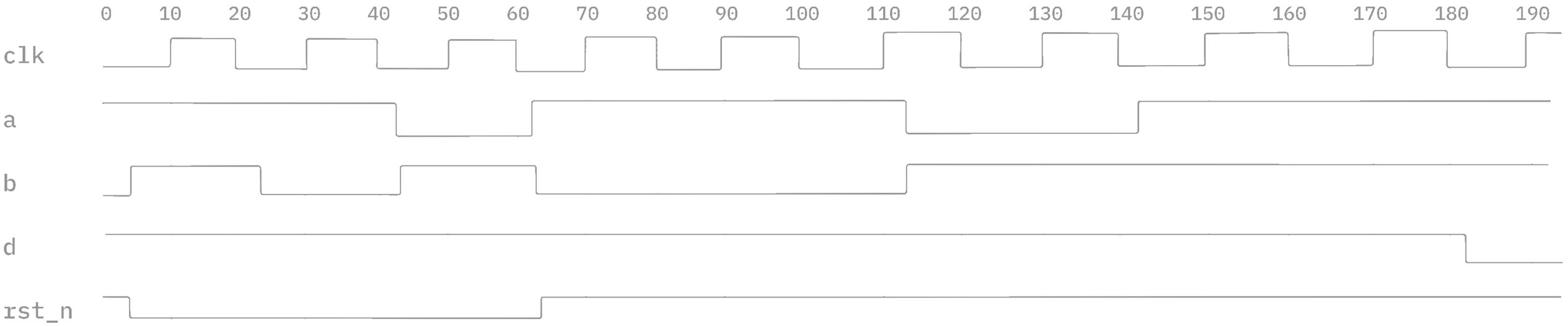
p2 PASS @ 90ns / started @ 70ns
p2 PASS @ 110ns / started @ 90ns
p2 FAIL @ 130ns / started @ 110ns

p3: disable iff(!rst_n) a |-> b[->2:3] ##1 $fell(d)

p3 FAIL @ 130ns / started @ 70ns

(όταν γίνεται το rst_n=0 την ώρα που περιμένουμε
κάποιο assertion, απευθείας ακυρώνεται χωρίς PASS/FAIL)

Λύσεις Ιουνίου 2022



p1: a |-> b[*2]

```
p1 FAIL @ 30ns / started @ 10ns
p1 FAIL @ 30ns / started @ 30ns
p1 FAIL @ 70ns / started @ 70ns
p1 FAIL @ 90ns / started @ 90ns
p1 FAIL @ 110ns / started @ 110ns
p1 PASS @ 170ns / started @ 150ns
p1 PASS @ 190ns / started @ 170ns
```

p2: disable iff(!rst_n) a |-> b[*2]

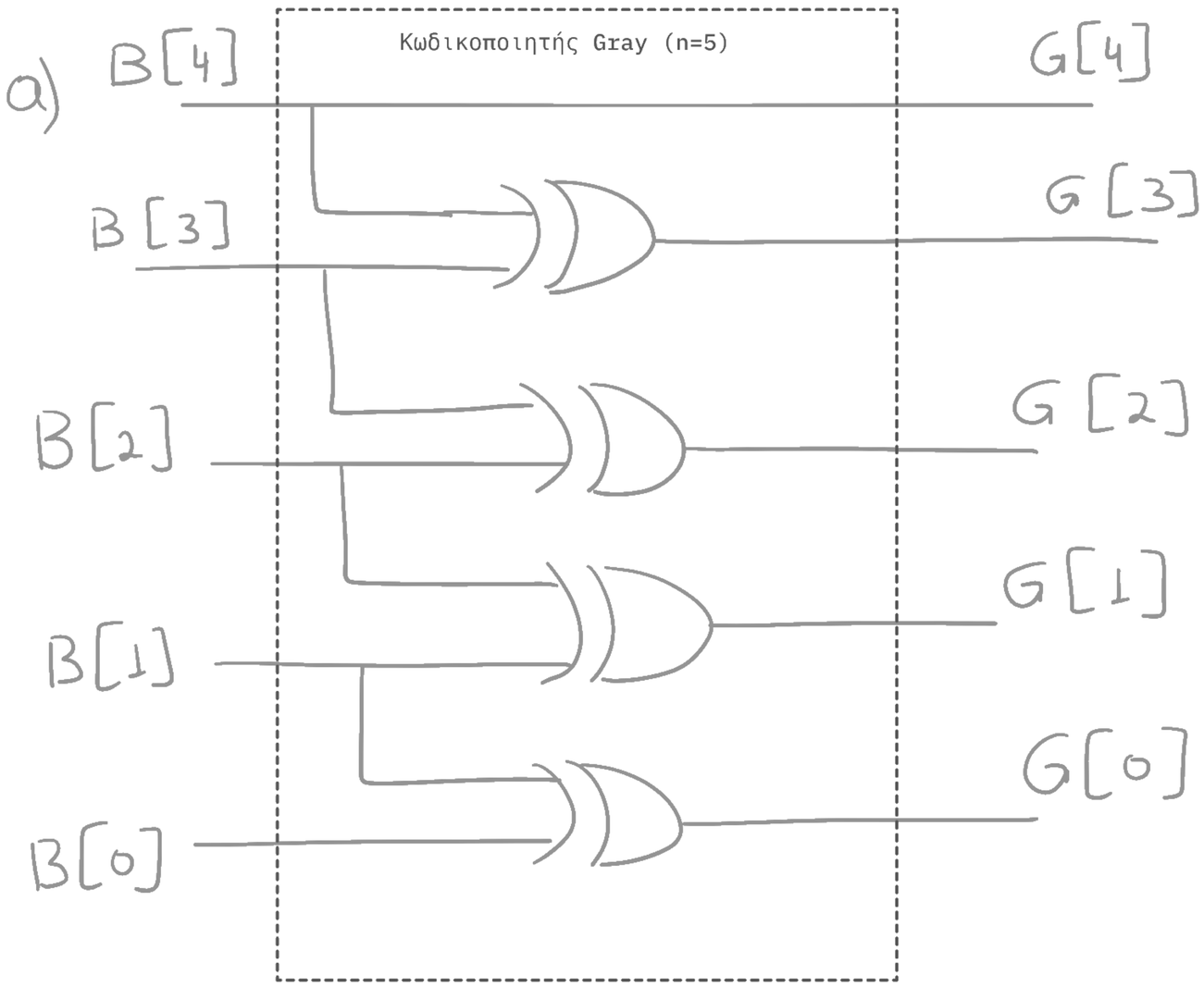
```
p2 FAIL @ 70ns / started @ 70ns
p2 FAIL @ 90ns / started @ 90ns
p2 FAIL @ 110ns / started @ 110ns
p2 PASS @ 170ns / started @ 150ns
p2 PASS @ 190ns / started @ 170ns
```

p3: disable iff(!rst_n) a |-> b[*2:3] ##1 $fell(d)

```
p3 FAIL @ 70ns / started @ 70ns
p3 FAIL @ 90ns / started @ 90ns
p3 FAIL @ 110ns / started @ 110ns
p3 PASS @ 190ns / started @ 150ns
```

Λύσεις Σεπ. 2021

α)

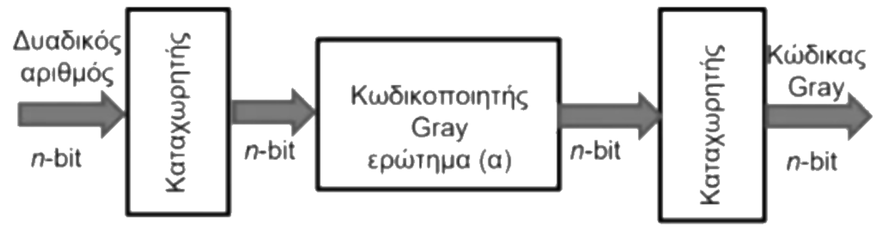


<u>Θέμα 2:</u> Οι λογικές εξισώσεις μετατροπής ενός δυαδικού αριθμού (B) σε κώδικα Gray (G) είναι οι παρακάτω:

$$G[n-1] = B[n-1] \quad (1)$$
$$G[i] = B[i] \oplus B[i+1], \text{ για } 0 \le i < n\text{-}1 \quad (2)$$

(α) Για n = 5, σχεδιάστε το σχηματικό διάγραμμα του κωδικοποιητή αυτού με βασικές λογικές πύλες (π.χ. AND, OR, NOT, XOR, κλπ) όπου φυσικά το ζητούμενο είναι το κύκλωμα να περιέχει όσο το δυνατό λιγότερες πύλες. (**3 βαθμοί**)

(β) Αν ο δυαδικός αριθμός προέρχεται από έναν καταχωρητή και η κωδικοποιημένη έξοδος αποθηκεύεται σε έναν άλλο καταχωρητή όπως φαίνεται στο Σχήμα 1, υπολογίστε τη μέγιστη συχνότητα λειτουργίας του κυκλώματος όπου δίνονται οι παρακάτω παράμετροι:
$t_{clk->Q}$ = 150 ps, t_{suff} = 180 ps, t_{dAND} = 50 ps, t_{dNAND} = 40 ps, t_{dNOR} = 40 ps, t_{dOR} = 50 ps, t_{dNOT} = 20 ps, t_{dXOR} = 100 ps (**2 βαθμοί**)



Σχήμα 1. Κύκλωμα κωδικοποίησης Gray όπου οι είσοδοι και έξοδοι αποθηκεύονται σε καταχωρητές

β) $f_{max} = \frac{1}{t_{pdmax}} = \frac{1}{t_{clk\to Q} + t_{dXOR} + t_{suff}} = \frac{1}{150+100+180} = \frac{1}{430ps} \Leftrightarrow$

πιο αργή διαδρομή στον κωδ/τη Gray

$\Leftrightarrow f_{max} = 0,00232 \cdot 10^{-12} \simeq \boxed{f_{max} = 2,32\ GHz}$