

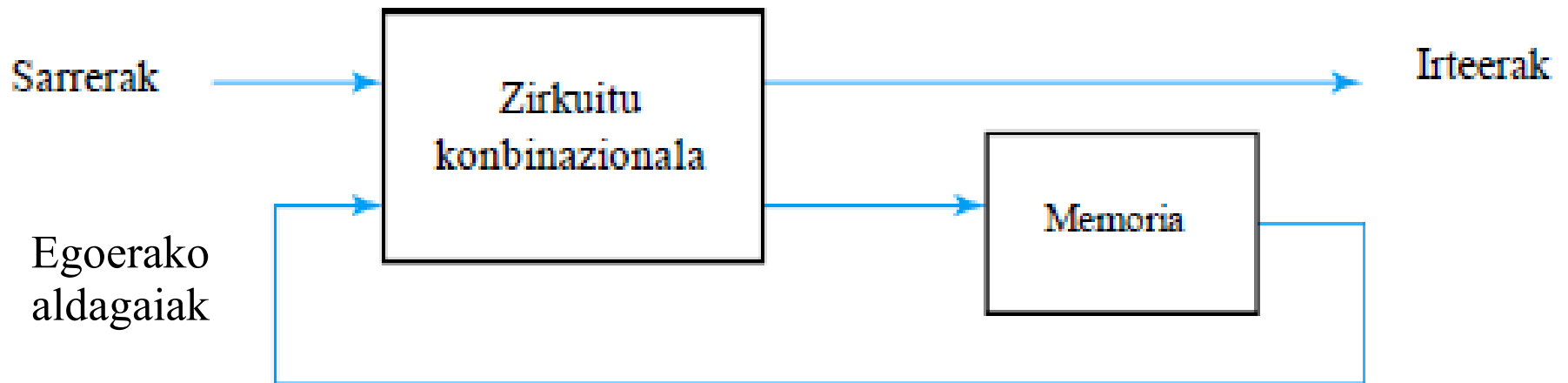
4. gaia:
Bloke sekuentzialak

Zirkuitu sekuentziala

- Zirkuitu konbinazionalen bidez ezinezkoa da balio bitarrak denboran mantentzea edo balio bitarren zerrendak sortzea
- Halako funtzioak zirkuitu sekuentzialek betetzen dituzte, eta memoria eta mikroprozesadoreen oinarriak dira
- Zirkuitu sekuentzialetan, irteera oraingo eta iraganeko sarrerako balioen menpe dago

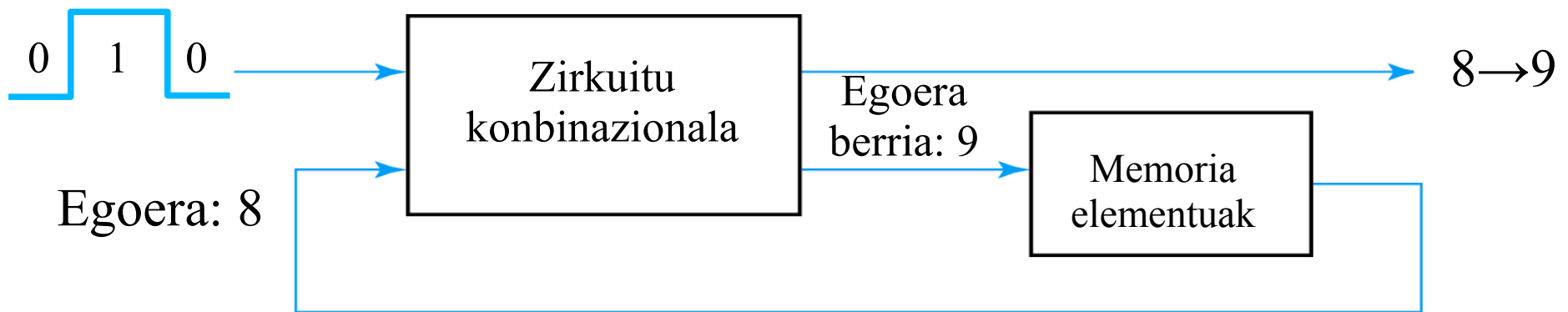
Zirkuitu sekuentziala: egoera

- Zirkuitu hauek oraingo sarreraren funtzioak sortzen dituzte, berriro sarrerara bidaliak direnak → Berrelikadura
- Berrelikatu diren seinaleak egoera seinaleak deitzen dira, eta horien oraingo balioen konbinazioa, zirkuituaren egoera



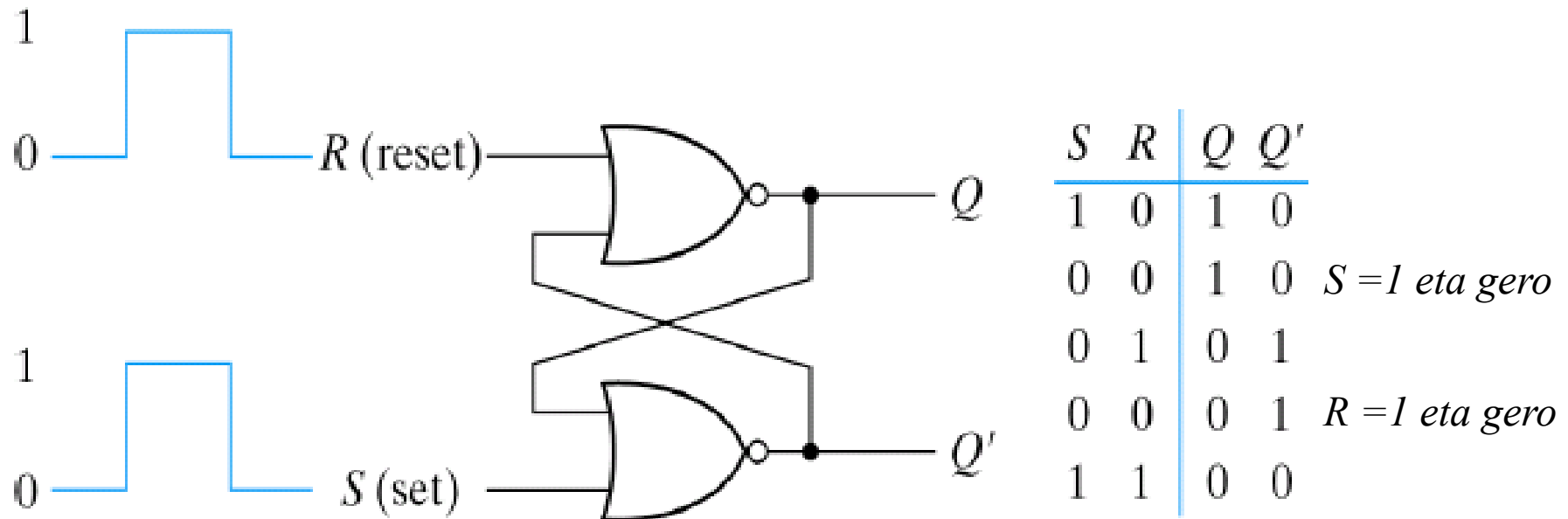
Zirkuitu sekuentziala: egoera

- Zirkuituaren egoera aldatzen denean, irteera alda daiteke, nahiz eta sarrerako balioak aldatu ez
- Egoera aldagaia mantendu egin behar da, nahiz eta sarrera aldatu, bestela zirkuituaren egoera galduko genuke ➔ Memoria elementuak



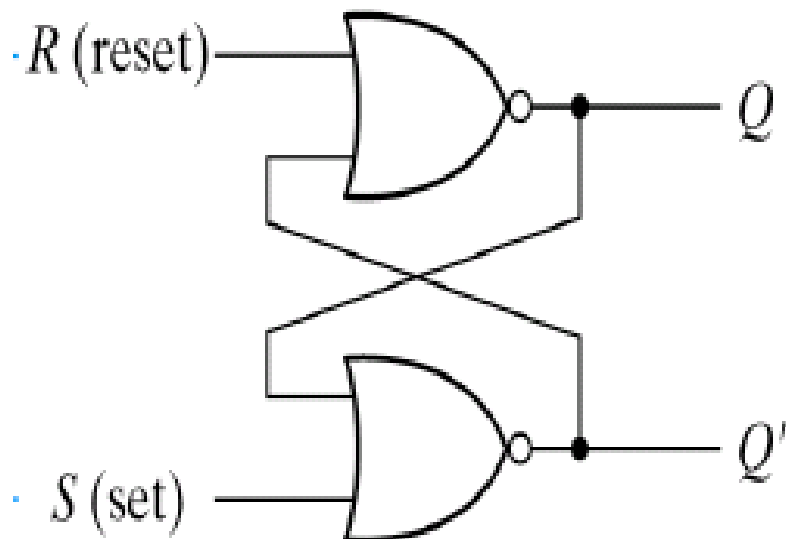
Bi egonkor asinkronoa (latch) S-R

- Latch S-R memoria elementu bat da, bit baten balioa mantendu dezakeela etengabe
- S sarrerako 1-ek irteera 1-era aldarazten du, eta R sarrerako 1-ek irteera 0-ra aldarazten du



Bi egonkor asinkronoa (latch) S-R

- Nahiz eta sarrerak 0-ra itzuli, irteerako balioa (egoera) mantentzen da sarreraren bat berriro 1-era aldatu arte
- Bi sarrerak 1-ean badaude, mantentzeko ezinezkoa den egoera berri bat (Q eta $Q' = 0$) agertzen da → Egoera ez egonkorra

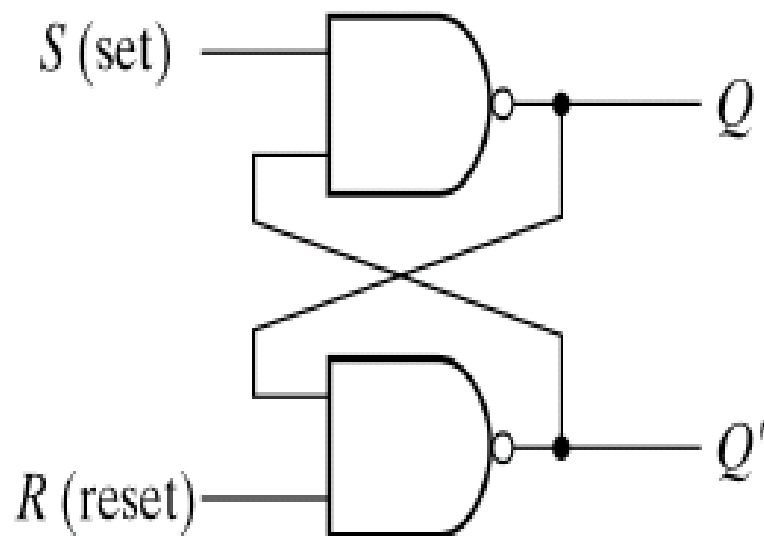


Q	S	R	Q^*	Q^{*}
0	0	0	0	1
0	0	1	0	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	0
1	0	1	0	1
1	1	0	1	0
1	1	1	0	0

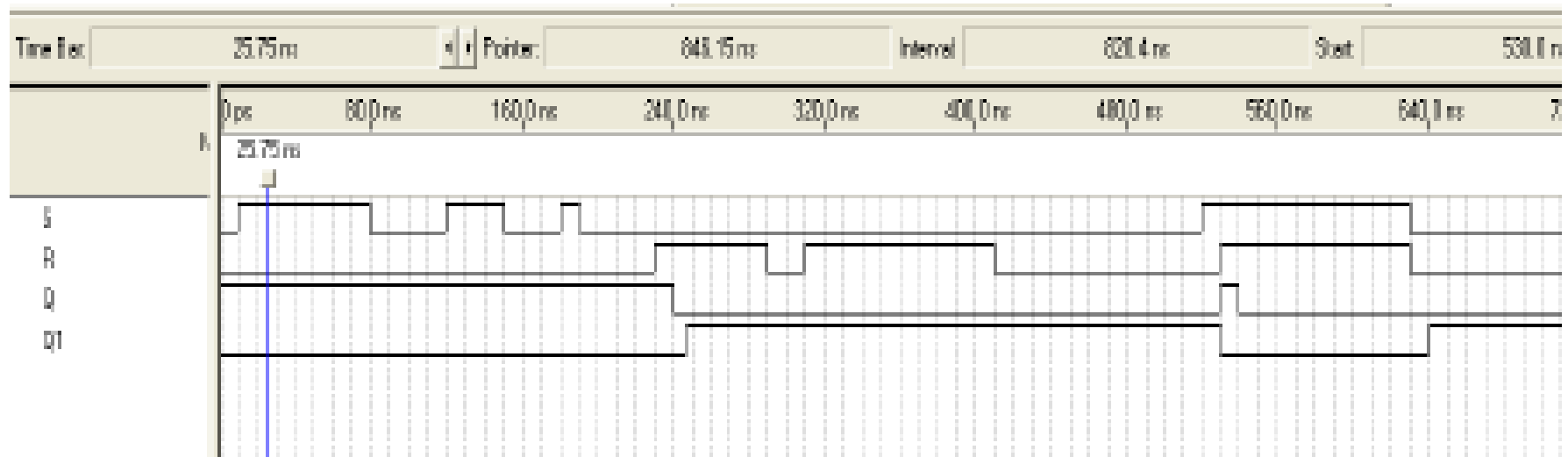
Q : oraingo egoera
 Q^* : hurrengo egoera

Bi egonkor asinkronoa (latch) S-R

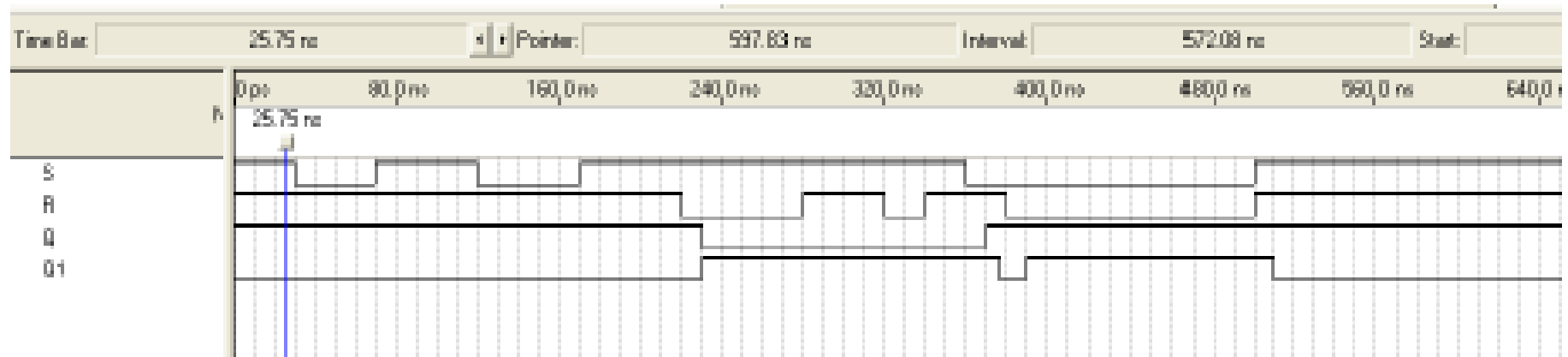
- S-R latch osatu daiteke bi NAND ateen bidez
- Zirkuitu honetan sarrerak maila baxukoak dira \rightarrow S eta R 0-en bidez aktibatzen dira
- Egoera ezegonkorra sarrera biak 0-an daudenean agertzen da



Q	S	R	Q^*	$Q^{*'} $
0	0	0	1	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	1
1	0	1	1	0
1	1	0	0	1
1	1	1	1	0



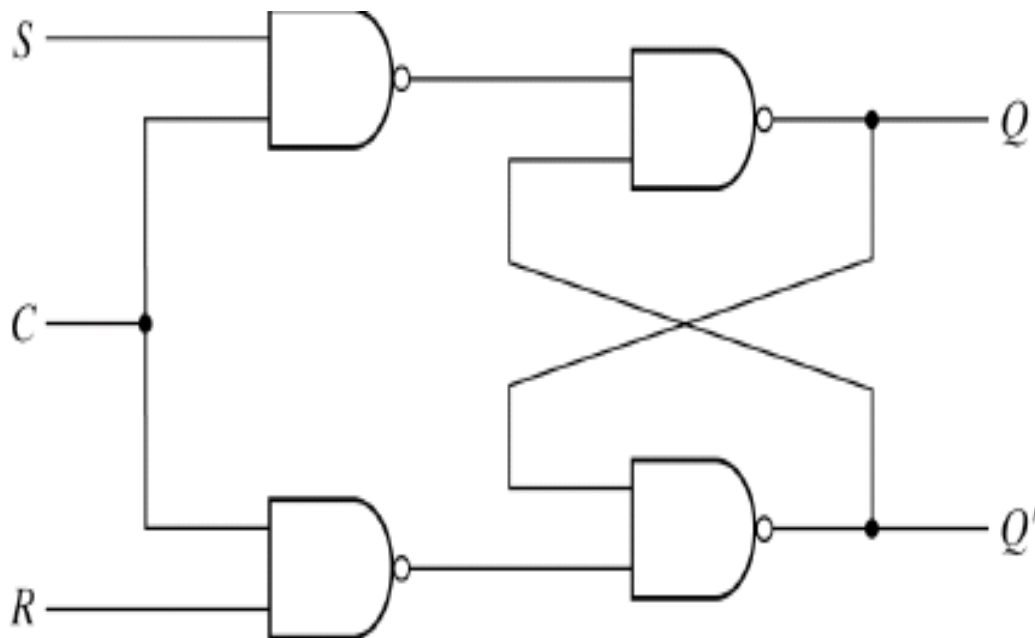
SR latch NOR atez egindako kronograma



SR latch NAND atez egindako kronograma

Bi egonkor asinkronoa (latch) S-R

- Egoera aldaketak bata bestearen atzetik ager daitezke, kontrolik gabe
- Hori kontrola dezakegu, egoera aldaketa denboran murriztuz
- Zirkuitu honetan, egoera aldaketa gerta daiteke kontrol seinalea 1-ean dagoen bitartean bakarrik



C	S	R	Q^*	$Q^{*'} $
0	X	X	Q	Q'
1	0	0	Q	Q'
1	0	1	0	1
1	1	0	1	0
1	1	1	1	1

Biegonkor asinkronoa (latch)S-R

KONTROL SARRERARIK GABE

```
library ieee;  
use ieee.std_logic_1164.all;
```

```
ENTITY simSR IS  
PORT ( S,R:    IN  std_logic;  
Q, Q1:  OUT std_logic);  
END simSR;
```

```
ARCHITECTURE a OF simSR IS  
SIGNAL B, B1: std_logic;  
BEGIN
```

```
b1<=S NOR b;  
b<=R NOR b1;  
Q<=B;  
Q1<=B1;  
END a;
```

KONTROL SARRERAREKIN

```
library ieee;  
use ieee.std_logic_1164.all;
```

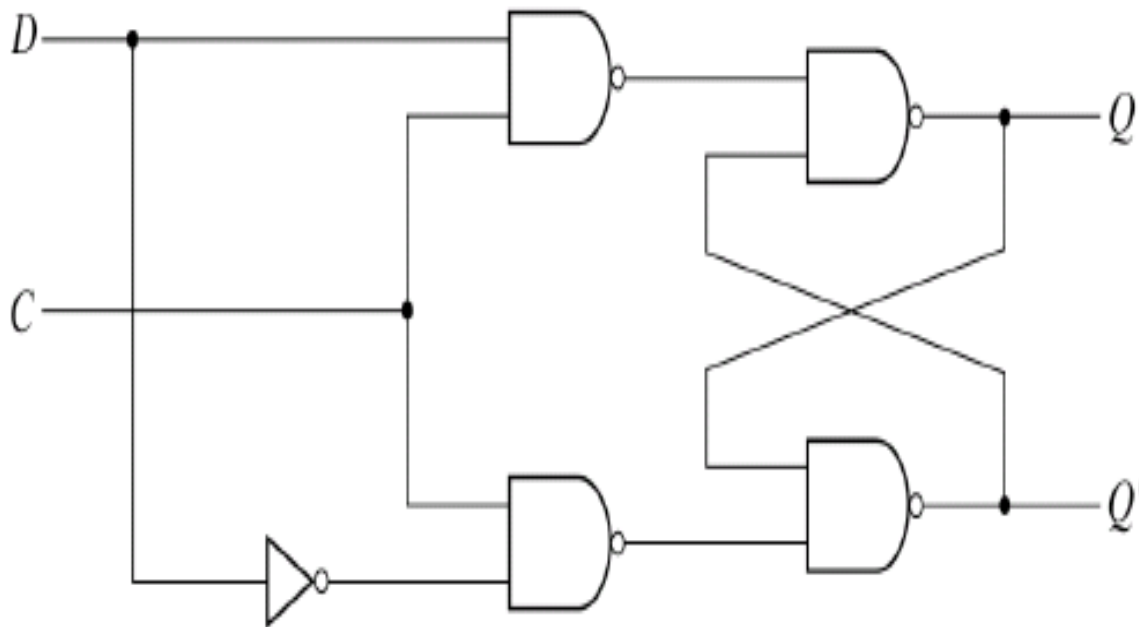
```
ENTITY simSRcont IS  
PORT ( S, R, C: IN  std_logic;  
Q, Q1:      OUT std_logic);  
END simSRcont;
```

```
ARCHITECTURE a OF simSRcont IS  
SIGNAL B, B1: std_logic;  
BEGIN
```

```
b  <=(S NAND C) NAND b1;  
b1 <=(R NAND C) NAND b;  
Q  <=B;  
Q1 <=B1;  
END a;
```

Bi egonkor asinkronoa (latch) D

- Zirkuitu honetan ez dago egoera ezegonkorrik
- Sarrerako balioa irteeran errepikatzen da, kontrol seinalea 0-an ez badago → Azken egoeraren memoria



C	D	Q^*	$Q^{*'} $
0	X	Q	Q'
1	0	0	1
1	1	1	0

Biegonkor asinkronoa (latch) D

```
library ieee;  
use ieee.std_logic_1164.all;
```

```
ENTITY simDcont IS  
PORT ( D, C:   IN  std_logic;  
Q, Q1:       OUT std_logic);  
END simDcont;
```

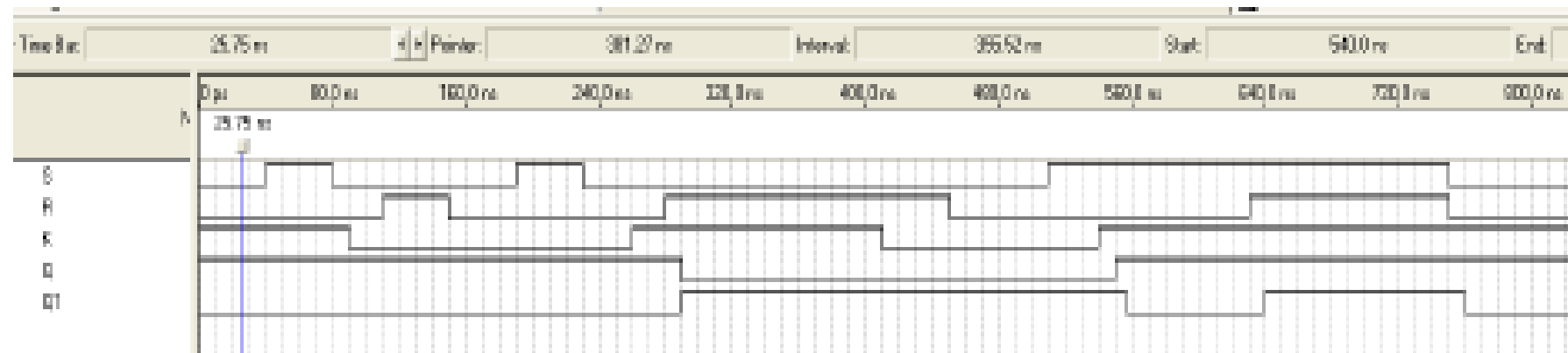
```
ARCHITECTURE a OF simDcont  
IS  
SIGNAL B, B1: std_logic;  
BEGIN
```

```
PROCESS (D, C)  
BEGIN
```

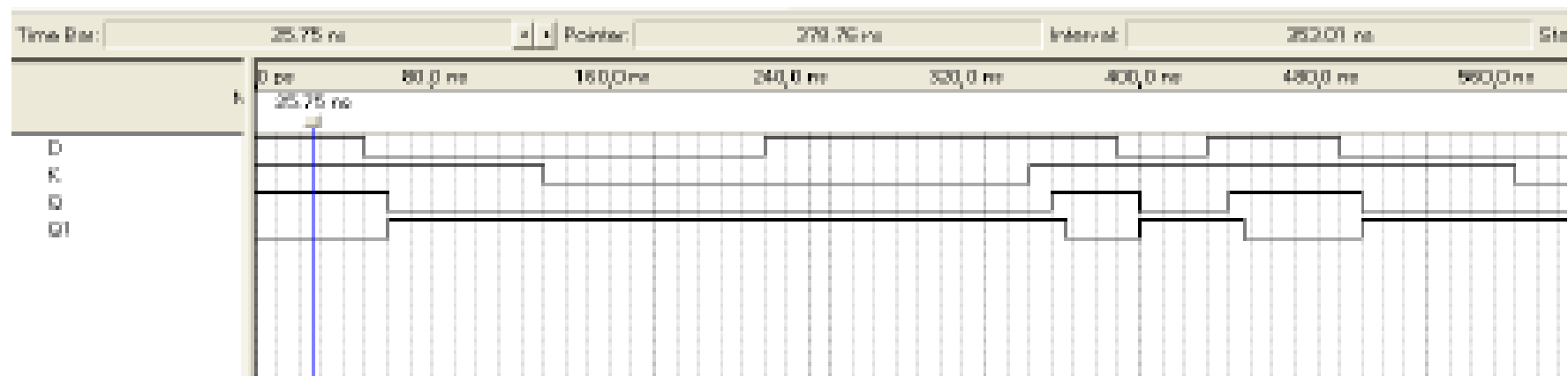
```
IF C='0' THEN  
    B<=B;  
    B1<=B1;
```

```
ELSE  
    B<=D;  
    B1<= NOT D;  
END IF;  
END PROCESS;
```

```
    Q  <=B;  
    Q1 <=B1;  
END a;
```



Latch SR kontrola dueneko kronograma



Latch D kontrola dueneko kronograma

Mailako eta ertzeko aktibazioa

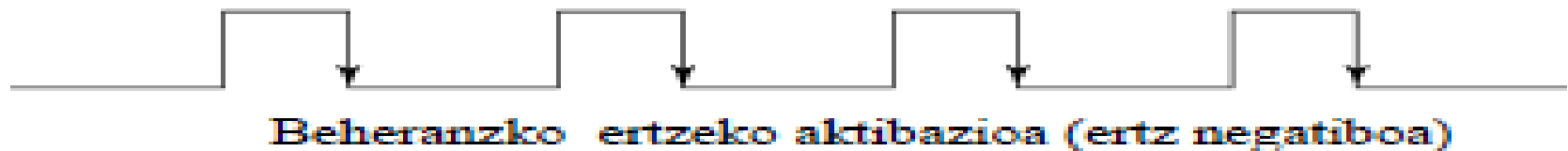
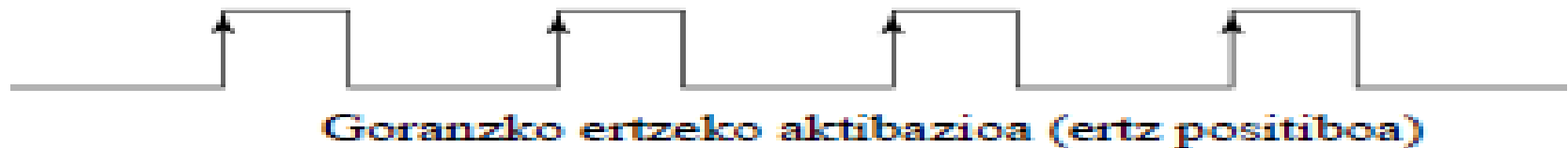
- Kontrol seinalerako (mailako aktibazioa) denbora-tarte bat definitu egin behar da, bertan egoera aldaketak gerta daitezkeela
- Aktibazio denbora-tarte horren bitartean, egoera aldaketak bata bestearen atzean gerta daitezke, beraz arretaz definitu egin behar da



Maila altuko aktibazioa

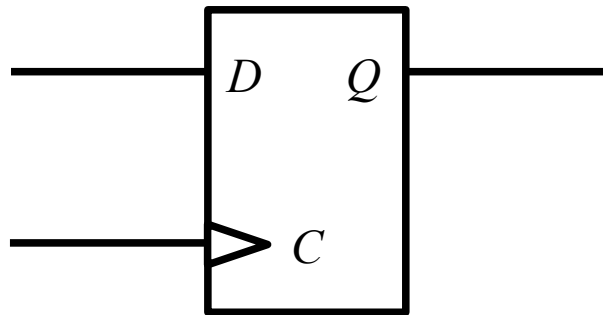
Mailako eta ertzeko aktibazioa

- Denbora-tarte hori gutxitu daiteke seinale bat balioa aldatzeko behar duena izateko bakarrik → Ertzeko aktibazioa
- Maiztasun konstanteko bada aktibazio seinalea, egoera aldaketak maiztasun horrekin sinkronizaturik gertatuko dira → Erlojua



Biegonkor sinkronoa (flip-flop) D

- Latch D kontrol sarrera dueneko baliokidea da, baina ertzeko aktibazioa daukana
- Erlojuko ertz aktiboa gertatzen denean, sarrerako balioa irteeran agertzen da



C	D	Q^*	$Q^{*'} $
X	X	Q	Q'
\uparrow	0	0	1
\uparrow	1	1	0

\uparrow : C 0-tik 1-era aldatzen denean

Biegonkor sinkronoa (flip-flop) D

```
library ieee;
use ieee.std_logic_1164.all;

ENTITY simFFD IS
PORT ( D, CLK:    IN  std_logic;
      Q, Q1:      OUT std_logic);
END simFFD;

ARCHITECTURE a OF simFFD IS
BEGIN

PROCESS (D, CLK)
BEGIN
```

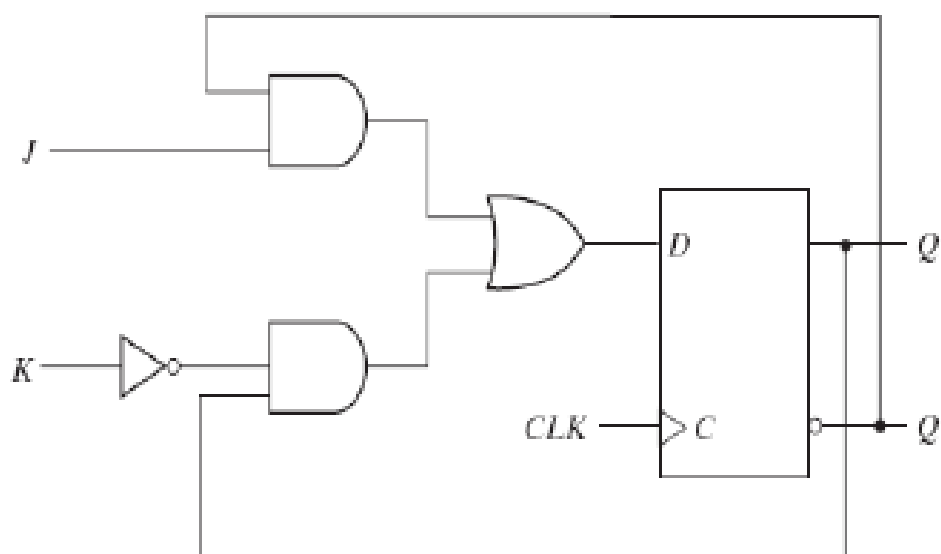
```
IF CLK'EVENT AND CLK='1' THEN
Q<=D;
Q1<=NOT D;
END IF;

END PROCESS;

END a;
```

Biegonkor sinkronoa (flip-flop) JK

- D flip-flopeko aldaketa bi sarrerekin, horren funtzioa da D sarrera
- SR-ren portaera dauka, baina sarrera biak 1-ean daudenean, irteerak aldatzen du azken balioa



C	J	K	Q^*	$Q^{*'} $
X	X	X	Q	Q'
↑	0	0	Q	Q'
↑	0	1	0	1
↑	1	0	1	0
↑	1	1	Q'	Q

$$D=J*Q'+K'*Q$$

Biegonkor sinkronoa (flip-flop) JK

Pre, Clr SARRERA ASINKRONOEKIN

```
library ieee;  
use ieee.std_logic_1164.all;
```

```
ENTITY simFFJK IS  
PORT ( pre, clr, J, K, CLK: IN  
std_logic;  
Q, Q1: OUT std_logic);  
END simFFJK;
```

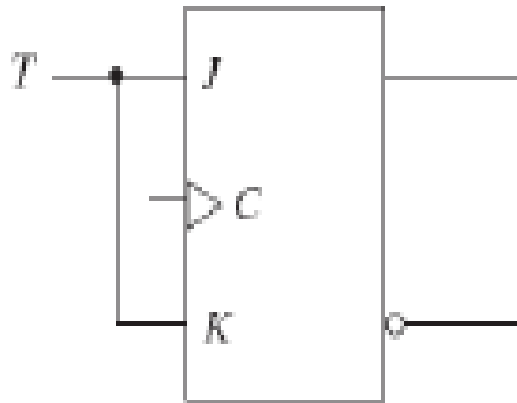
```
ARCHITECTURE a OF simFFJK IS  
SIGNAL D, B, B1: std_logic;  
BEGIN
```

```
PROCESS (pre, clr, D, CLK)  
BEGIN  
IF pre='1' THEN  
    B<='1';  
    B1<='0';
```

```
ELSIF clr='1' THEN  
    B<='0';  
    B1<='1';  
ELSIF CLK'EVENT AND CLK='1' THEN  
    B<=D;  
    B1<=NOT D;  
END IF;  
END PROCESS;  
  
D<= (J AND B1) OR ((NOT K)  
AND B);  
Q<= B;  
Q1<= B1;  
  
END a;
```

T flip-flop

- JK-ren aldaketa sarrera bakarrekin
- T sarrera 1-ean dagoen bitartean, irteerako balioa aldatzen da erloju ertz bat gertatzen denean
- Kontagailuak osatzeko erabiltzen da, aldagai bitar baten aldaketa zifra bitar bateko kontaketa da eta



C	T	Q^*	$Q^{*'} $
X	X	Q	Q'
↑	0	Q	Q'
↑	1	Q'	Q

Biegonkor sinkronoa (flip-flop) T

```
library ieee;
use ieee.std_logic_1164.all;

ENTITY simFFT IS
PORT ( T, CLK:      IN  std_logic;
      Q, Q1:      OUT std_logic);
END simFFT;
```

```
ARCHITECTURE a OF simFFT IS
SIGNAL B: std_logic;
BEGIN
```

```
PROCESS (T, CLK)
BEGIN
```

```
IF CLK'EVENT AND CLK='1' THEN
    IF T='1' THEN
        B<=NOT B;
    END IF;
END IF;

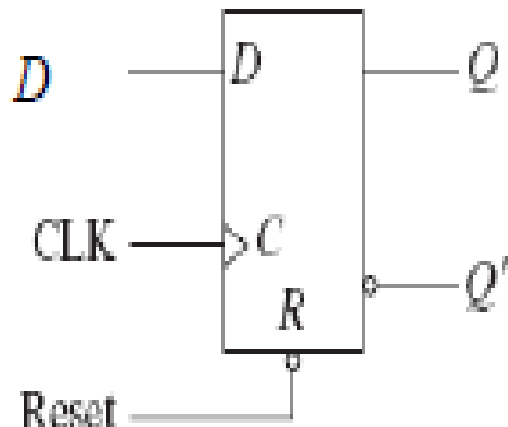
END PROCESS;

Q<=      B;
Q1<=    NOT B;

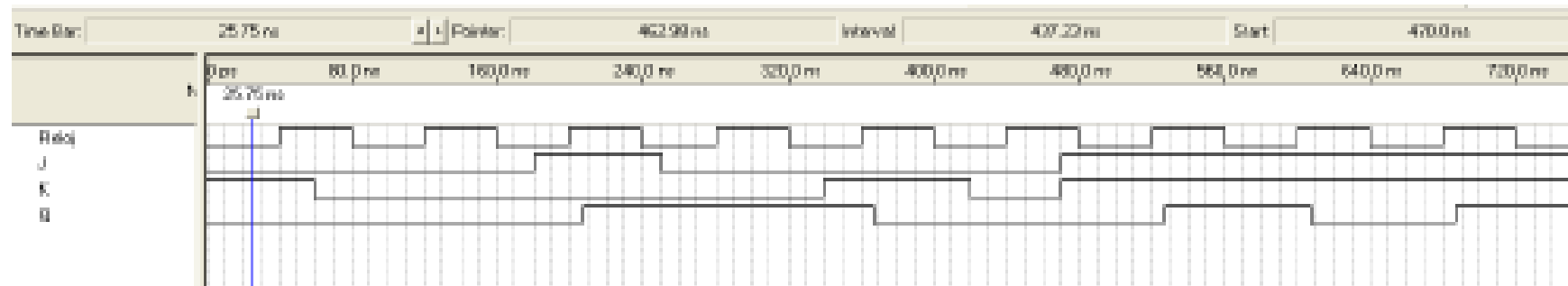
END a;
```

Sarrera asinkronoak

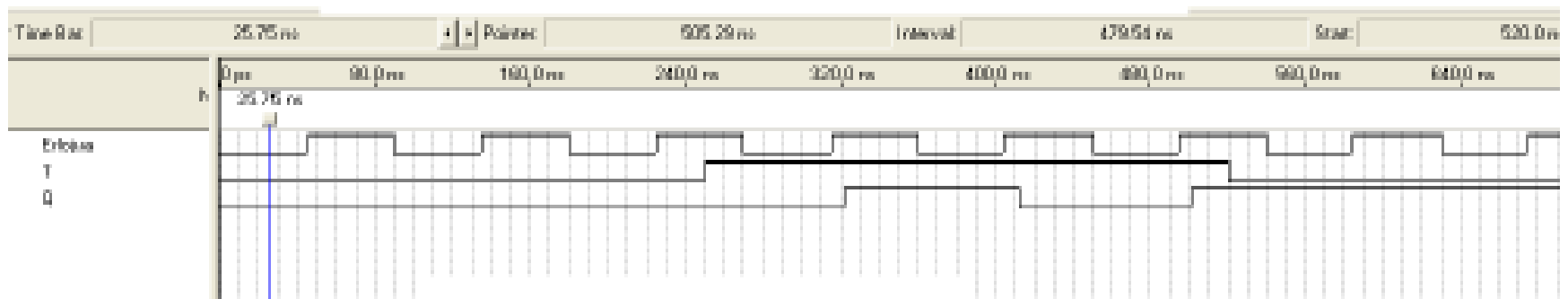
- Flip-flopen diseinuan latch-ena sar daiteke → Sarrera aldatzen denean, irteera aldatzen da
- R sarrera asinkronoak 0 egoerara aldarazten du flip-flopa, eta S sarrera asinkronoak 1-era
- Sarrera asinkronoek, flip-flopen sarrera sinkronoen aurrean lehentasuna daukate



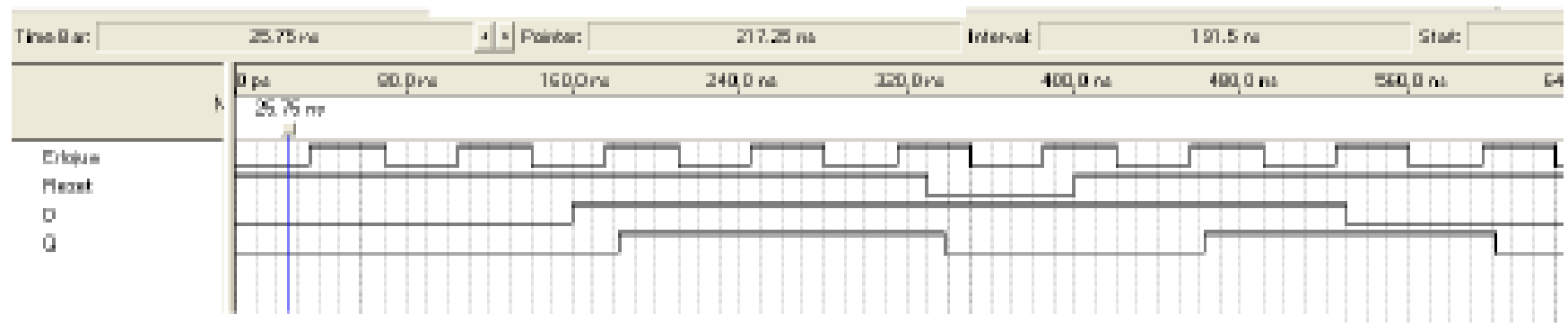
R	C	D	Q^*	$Q^{*'} $
0	X	X	0	1
1	X	X	Q	Q'
1	↑	0	0	1
1	↑	1	1	0



Ertz aktibatuniko JK flip-floppen kronograma



T flip-floppen kronograma

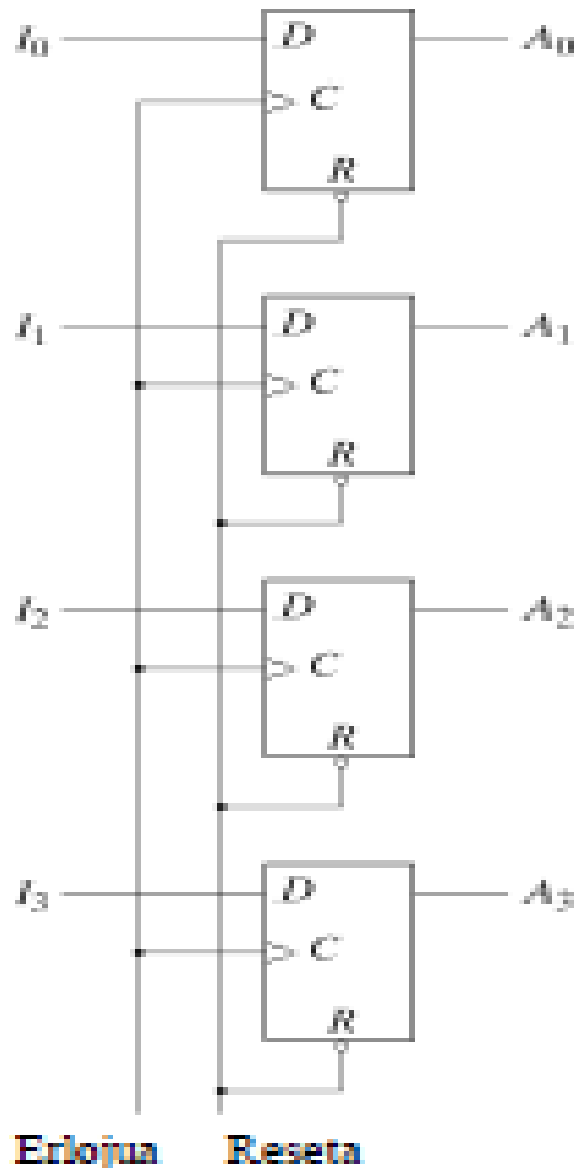


D flip-flop reset asinkrono sarrera duen kronograma

Erregistro eta kontagailuak

- Memoria elementuen aplikazio errazenak erregistro eta kontagailuak dira
- Erregistroek datu bitar baten balioa mantentzen dute (biltegitatzeko erregistroa) eta horrekin eragiketa bat burutzen dute (desplazamenduko erregistroa)
- Kontagailuek, erlojuko ertz bat gertatzen denean, aurretik definitutako sekuentzia bateko balio berri bat sortzen dute

Biltegiratze-erregistroa



- I sarreran dagoen datu bitarra mantenduko da, erlojuko ertz bat gertatzen denetik hurrengoa agertu arte
- Denbora-tarte horren bitartean, nahiz eta I aldatu, A irteeran sartutako balioa mantenduko da
- Reset seinalea aktibatzen bada, A 0-ra itzuliko da

Biltegiratze-erregistroa

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned;

ENTITY registro IS
PORT(
    clk: IN STD_LOGIC;
    d:   IN STD_LOGIC;
    q:   OUT STD_LOGIC);
END registro;
```

```
ARCHITECTURE a OF registro IS

BEGIN
```

```
PROCESS (CLK)
BEGIN

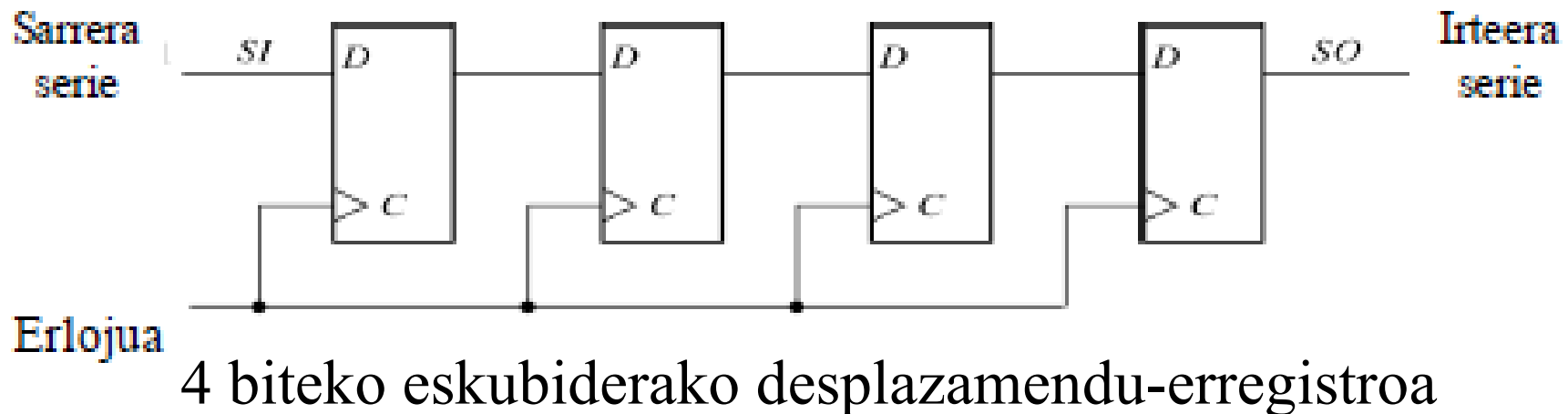
    IF clk'EVENT AND clk='1' THEN
        q<=d;
    END IF;

END PROCESS;

END a;
```

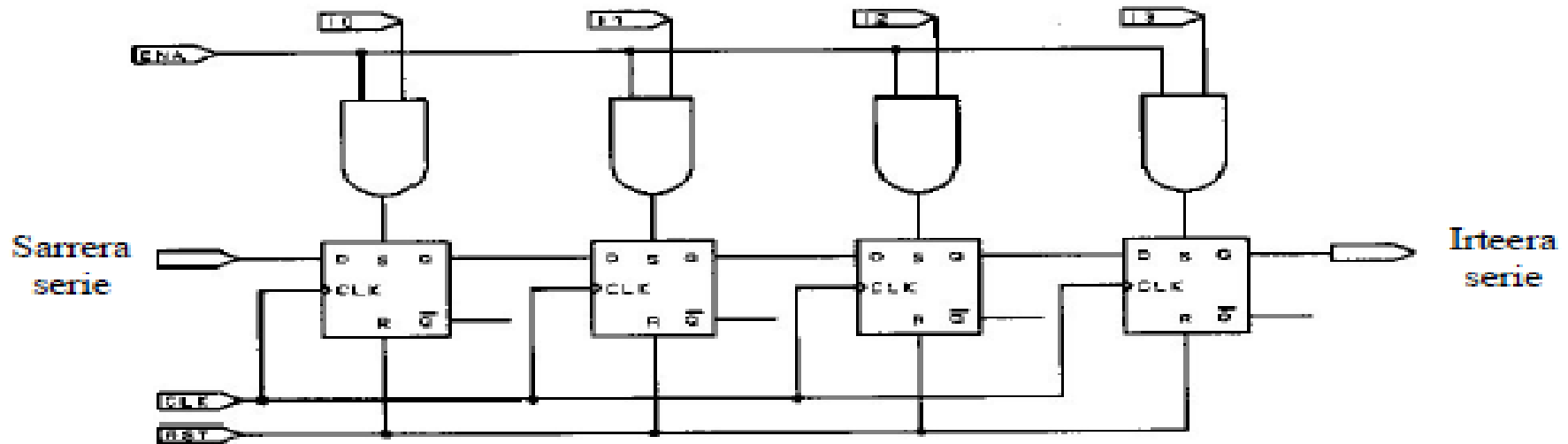
Desplazamendu-erregistroa

- Desplazamenduko erregistroek sartutako balioa aldatzen du, datuaren bit bakoitza hurrengo lekuan idatziz
- Datuaren bitak banan banan idatzi daitezke erregistroan (serie formatua) edo bit guztiak erloju periodo berean (paralelo formatua)



Desplazamendu-erregistroa

- Zenbait zirkuitutan, desplazamendua izan daiteke ezkerrera, edo aukeratu daiteke ea desplazamendua ezkerrera edo eskubidera den
- Zirkuitu honetan sarrera serie edo paralelo aukera daiteke



Sarrera serie eta sarrera paralelo daukan 4 biteko eskubiderako desplazamendu-erregistroa

Desplazamendu-erregistroa

```
library ieee;
use ieee.std_logic_1164.all;

ENTITY regdesp IS
PORT ( Es, CLK:   IN  std_logic;
      E:    IN  std_logic_vector (3 DOWNT0 0);
      K:    IN  std_logic_vector (1 DOWNT0 0);
      S:    OUT std_logic_vector (3 DOWNT0 0));
END regdesp;

ARCHITECTURE a OF regdesp IS
SIGNAL B: std_logic_vector (3 DOWNT0 0);
BEGIN

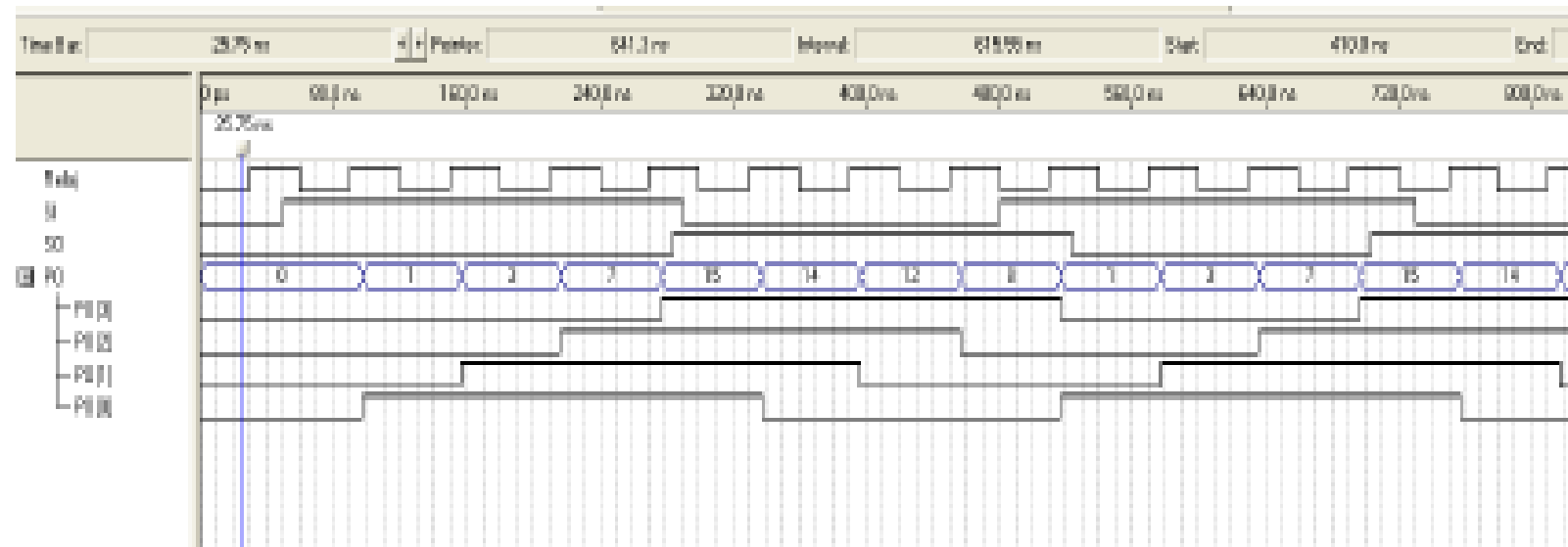
PROCESS (K, E, CLK)

BEGIN
```

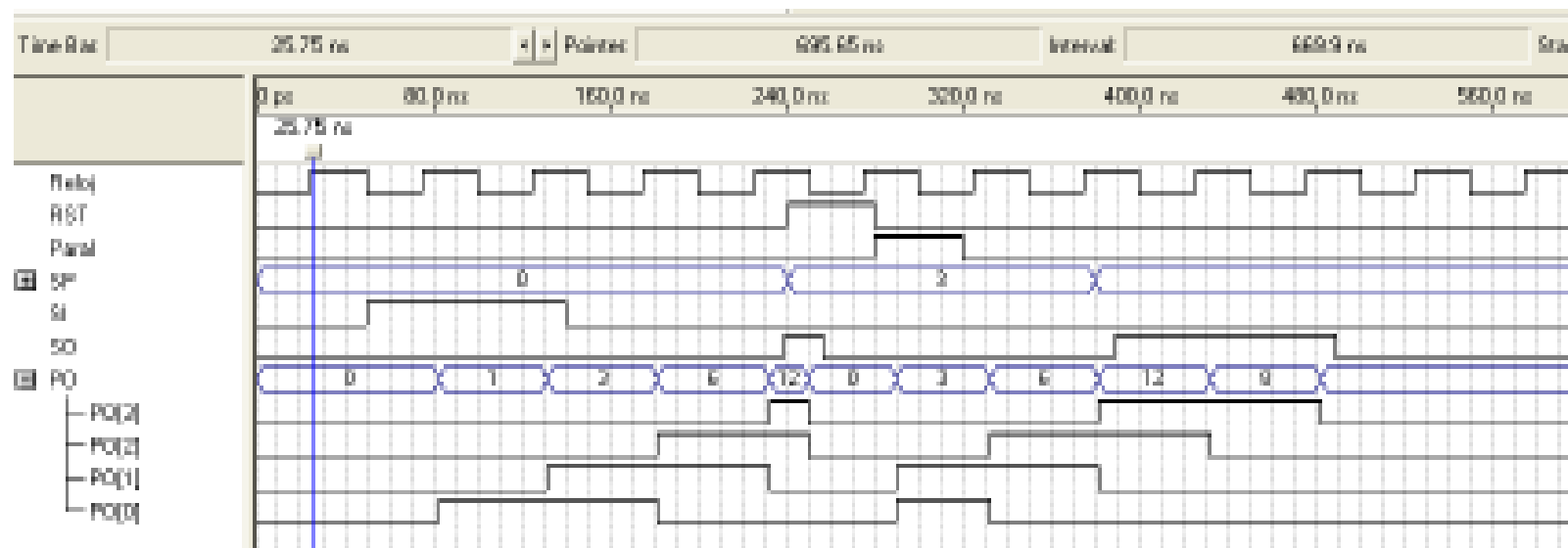
```
IF K="11" THEN B<=E;
ELSIF CLK'EVENT AND CLK='1' THEN
    CASE K IS
        WHEN "01"=> B(3)<=B(2);
                    B(2)<=B(1);
                    B(1)<=B(0);
                    B(0)<=Es;
        WHEN "10"=> B(3)<=Es;
                    B(2)<=B(3);
                    B(1)<=B(2);
                    B(0)<=B(1);
        WHEN OTHERS=> B<=B;
    END CASE;
END IF;
END PROCESS;

S<=    B;
END a;
```

<i>K</i>	<i>Funtzionamendua</i>
00	Biltegitratzea
01	Ezkerrera desp.
10	Eskuinera desp.
11	Karga paraleloa



4 biteko eskubiderako desplazamendu-erregistroaren kronograma

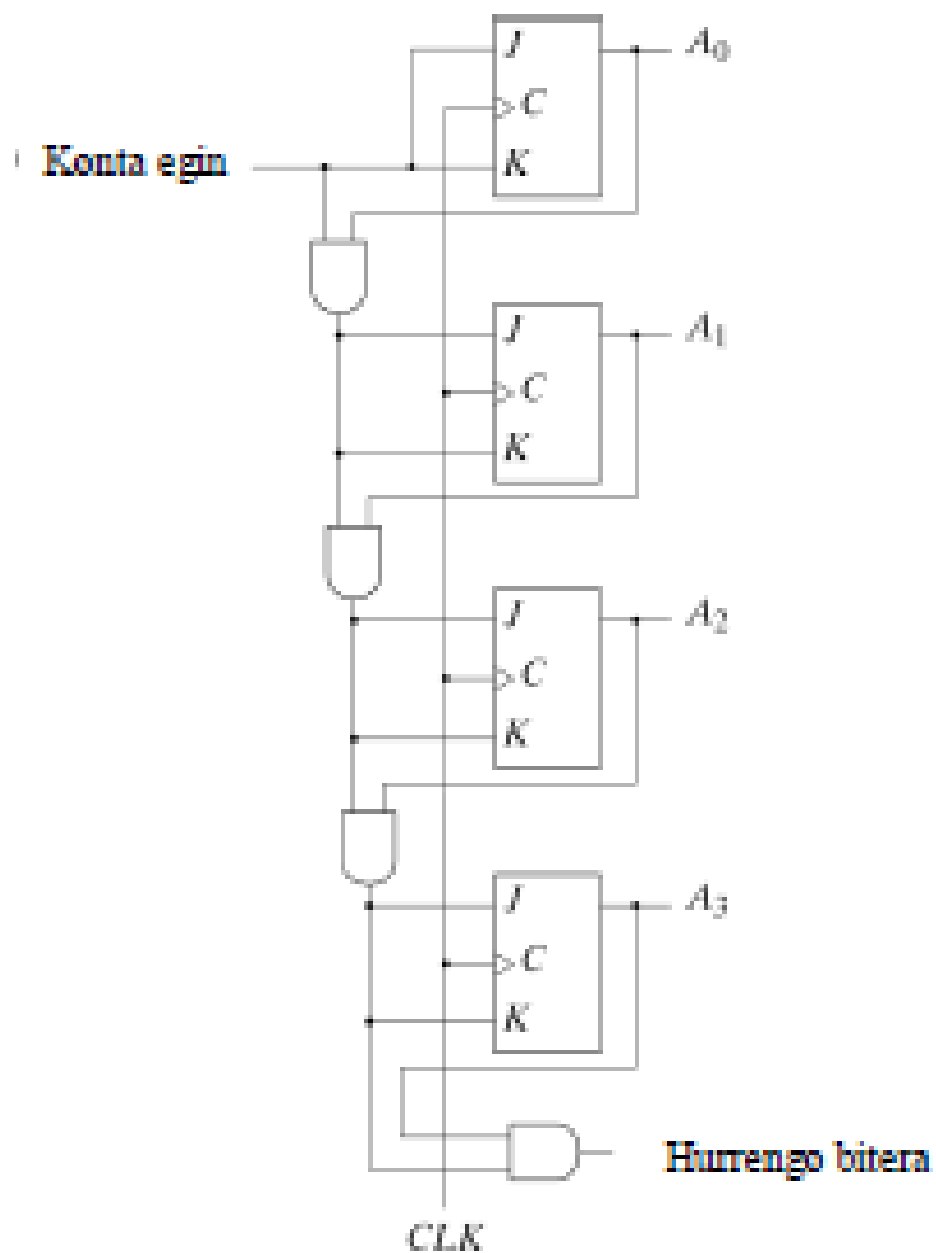


Sarrera paraleloko 4 biteko eskubiderako desplazamendu-erregistroaren kronograma

Kontagailuak

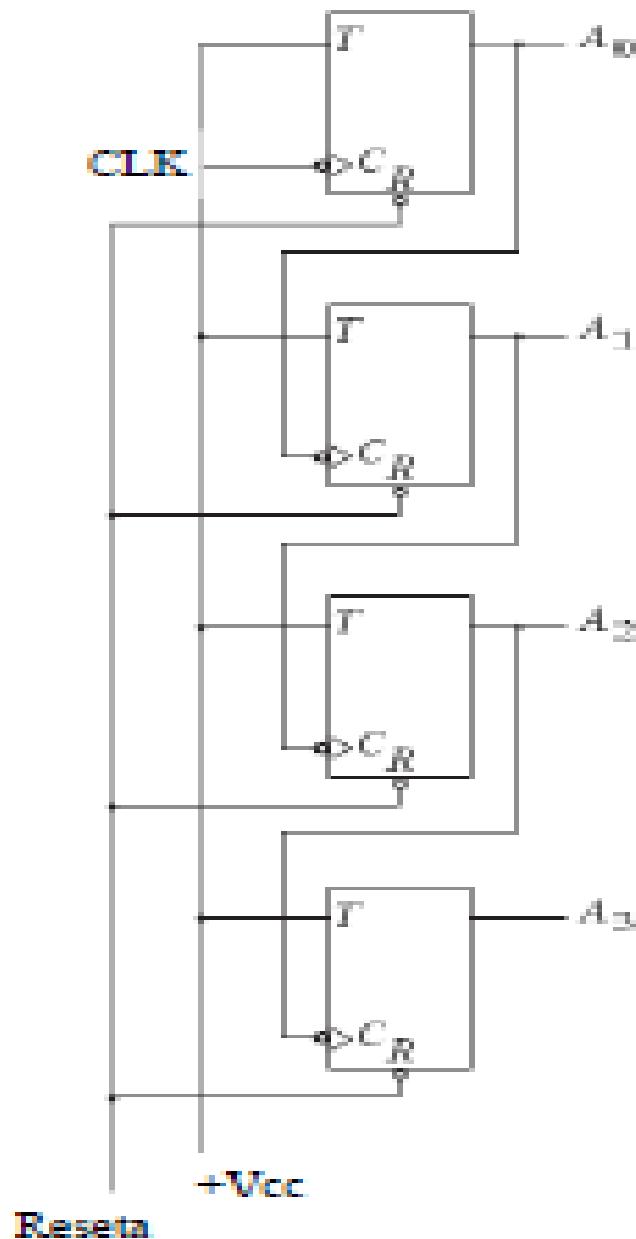
- Kontagailuek, erlojuko ertz bat gertatzen denean, aurretik definitutako sekuentzia bateko balio berri bat sortzen dute
- Sekuentzia bitarra goranzkoa edo beheranzkoa izan daiteke, baina baita BCD sekuentzia (0-tik 9-raino) ager daiteke
- Sortutako datuaren bit guztiak momentu berean aldatzen badira, kontagailu sinkronoa deitzen da
- Horrelakoak ez badira, kontagailu asinkronoak dira

Kontagailu sinkronoa



- Kontagailu honetako A irteeran, CLK ertz bat gertatzen denean, goranzko sekuentzia bitarreko balio berri bat agertzen da
- Zifra bat aldatzen da aurreko zifra guztiak bat direnean
- Sekuentzia gelditzen da, Konta egin seinalea 0-ra aldatzen bada

Kontagailu asinkronoa



- Flip-flop bakoitzeko erloju seinalea ezberdina da, beraz irteera momentu ezberdinetan aldatzen da
- CLK seinaleko ertzekin agertzen da goranzko sekuentzia bitarra, baina badaude bitartean balio okerrak
- Reset seinalea aktibatzen denean, CLK ertzari itxaron gabe, A 0-ra itzultzen da

Kontagailua

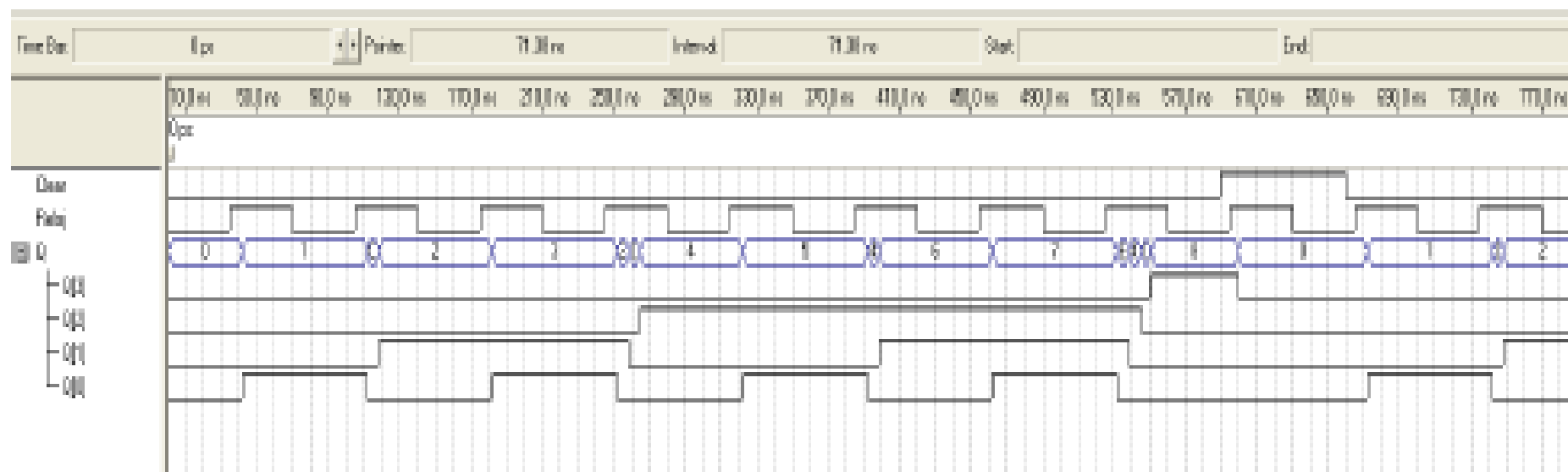
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
ENTITY conta IS PORT (
    CLK: IN std_logic;
    E:      IN  std_logic_vector (3
DOWNT0 0);
    K:      IN  std_logic_vector (1
DOWNT0 0);
    S:      OUT std_logic_vector (3
DOWNT0 0));
END conta;
```

```
ARCHITECTURE a OF conta IS
signal B: std_logic_vector (3 DOWNT0
0);
BEGIN
```

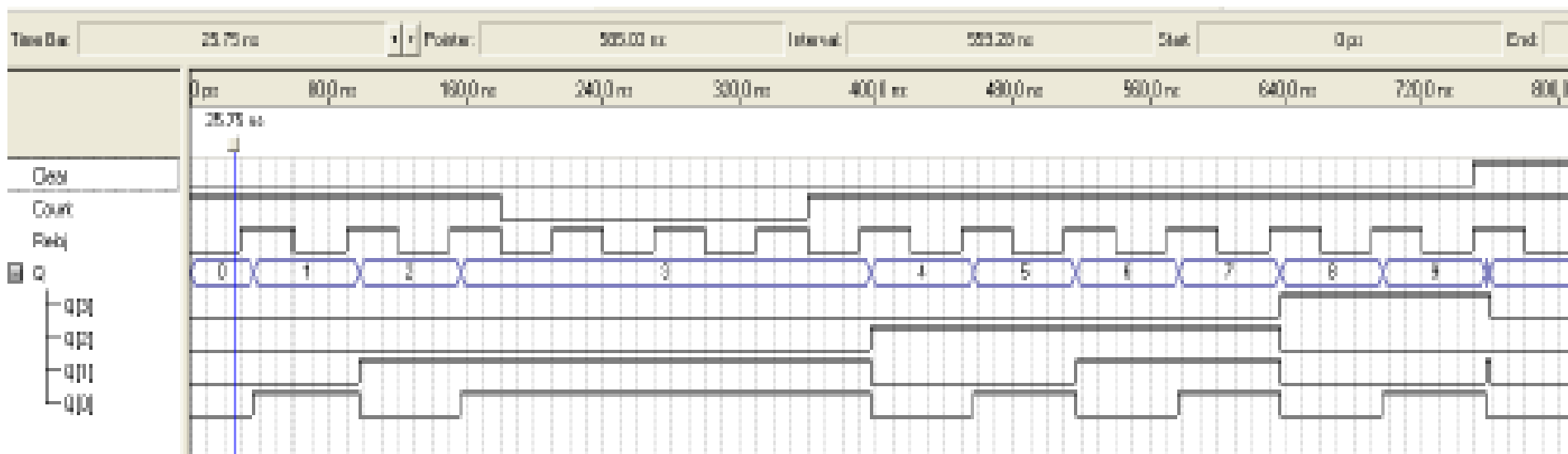
```
PROCESS (E, K, CLK)
BEGIN
    IF K = "11" THEN
        B <= E;
    ELSIF (CLK'EVENT AND CLK = '1')
    THEN
        CASE K IS
            WHEN "01" => B<= B+1;
            WHEN "10" => B<= B-1;
            WHEN OTHERS => B<= B;
        END CASE;
    END IF;
END PROCESS;
```

```
S<=B;
END a;
```

<i>K</i>	<i>Funtzionamendua</i>
00	Biltegitratzea
01	Goranzkoa
10	Beherantzkoa
11	Karga paraleloa



4 biteko kontagailu asinkronoen kronograma



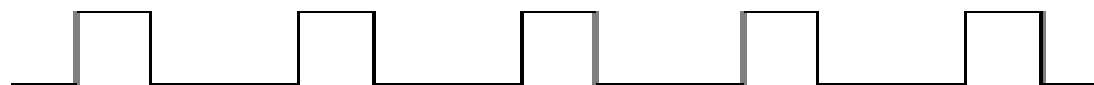
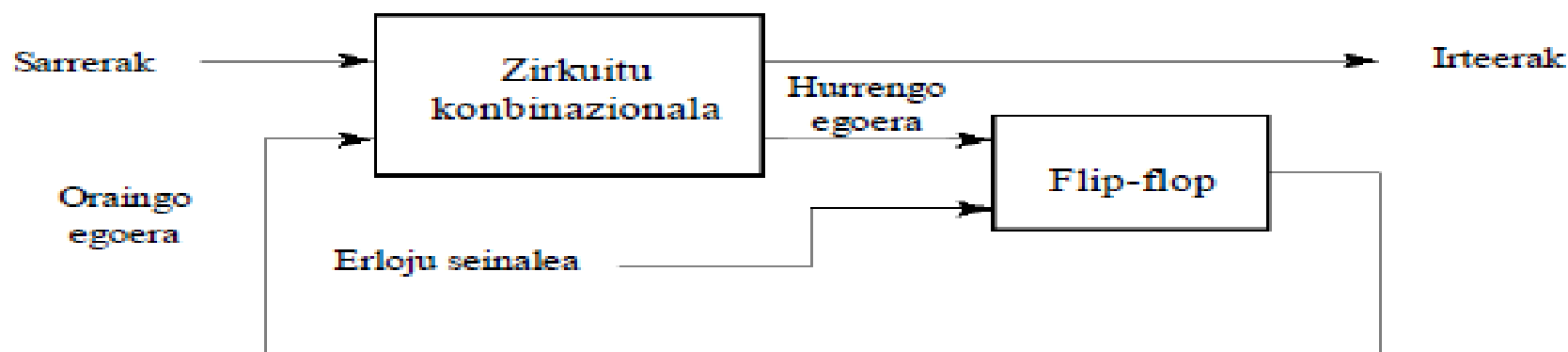
4 biteko kontagailu sinkronoen kronograma

Sistema sekuentzial sinkronoen analisi eta sintesia

- Sistema sekuentzial baten memoria elementuen flip-flop guztiak, erloju seinale berdina daukatenean, sistema sekuentzial sinkrono dela esaten dugu
- Sistemaren ikerketa, bere funtzionamendua deskribatzeko, analisi da
- Sistemaren funtzionamenduko deskripzio batetik hasita, zirkuituaren diseinua, sintesia da

Sistema sekuentzial sinkronoa

- Sistema sekuentzialetan, irteerako balioa sarrera eta oraingo egoeraren balioen menpe dago
- Irteerarekin batera definitzen da hurrengo egoeraren balioa, eta flip-flopetan gordetzen da
- Egoera aldatzeko unea, erloju seinalea aldatzen den une berdina da

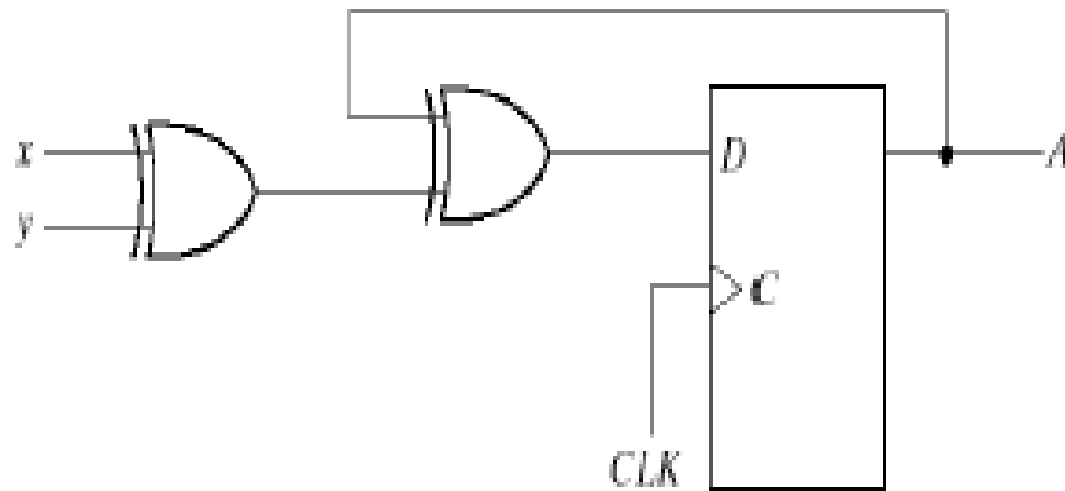


Erloju seinalearen kronograma

Sistema sekuentzialen analisia

- Sistemaren analisia egiteko, egoera berriaren balioa edozein sarrera eta oraingo egoera baliorako ezagutu egin behar da ➔ Hurrengo egoerako ekuazioak
- Honen bidez eta flip-flop ekuazioen bidez, egoera aldaketa (trantsizioak) aukera guztiak ezagutzen ditugu, sarrerako edozein baliorako

Sistema sekuentzialen analisia



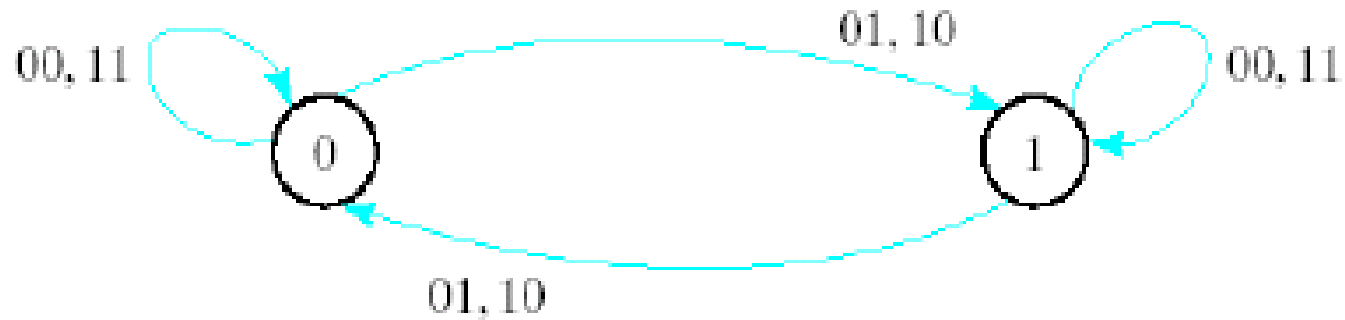
Hurrengo egoera: $D = X \oplus Y \oplus A$
Flip-flop: $Q^* = D$
Irteera: $A = Q$

Oraingo egoera	Sarrerak		Hurrengo egoera
A	x	y	A
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1
Egoera-taula			

Sistema sekuentzialen analisia

- Egoera diagramaren bidez, era grafikoan sistema sekuentzial trantsizio guztiak irudikatzen dira, bere irteerako balioekin batera → Irteerako ekuazioak
- Egoera diagraman irudikatu daiteke sistemaren portaera guztia, edozein sarrerako balioen segidaren aurrean (hasierako egoera ezagutzen bada)

Sistema sekuentzialen analisia

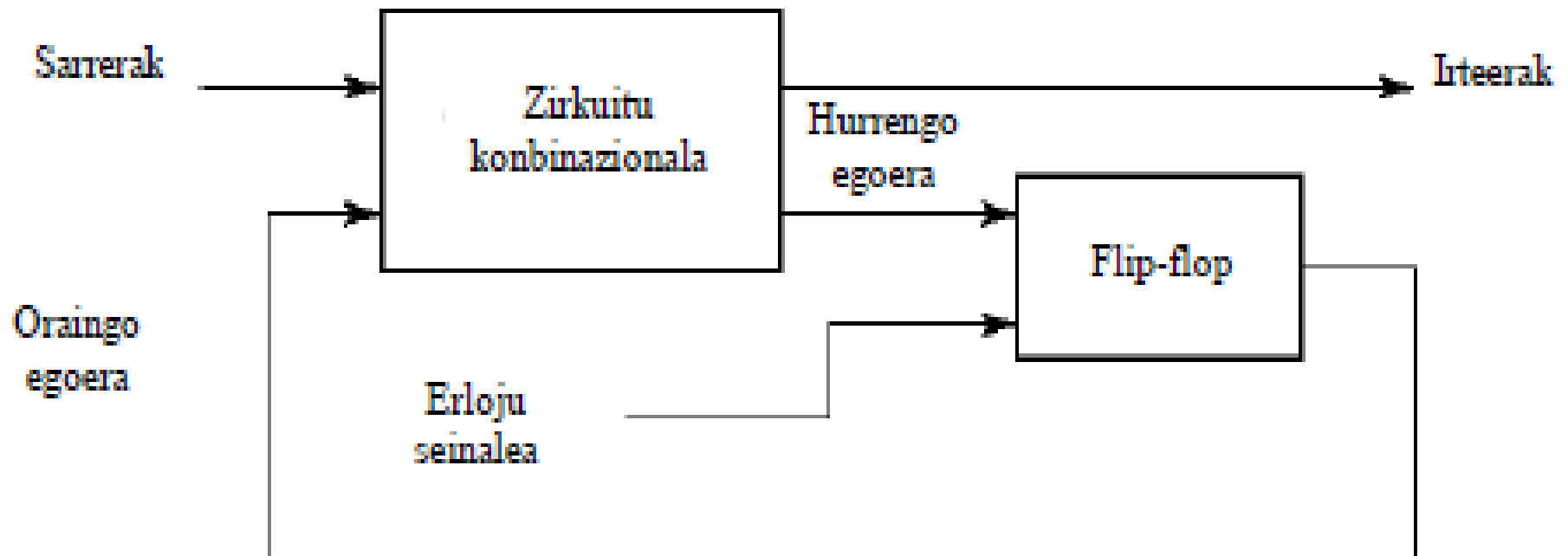


X	0	0	1	1	0	0	0
Y	0	1	1	0	0	1	0
A	0	0	1	1	0	0	1
Oraingo egoera	0	0	1	1	0	0	1

Egoera aldaketak erloju seinalearen ertzeko unetan gertatzen dira, sarrerako aldaketa momentua ez du inolako eraginik

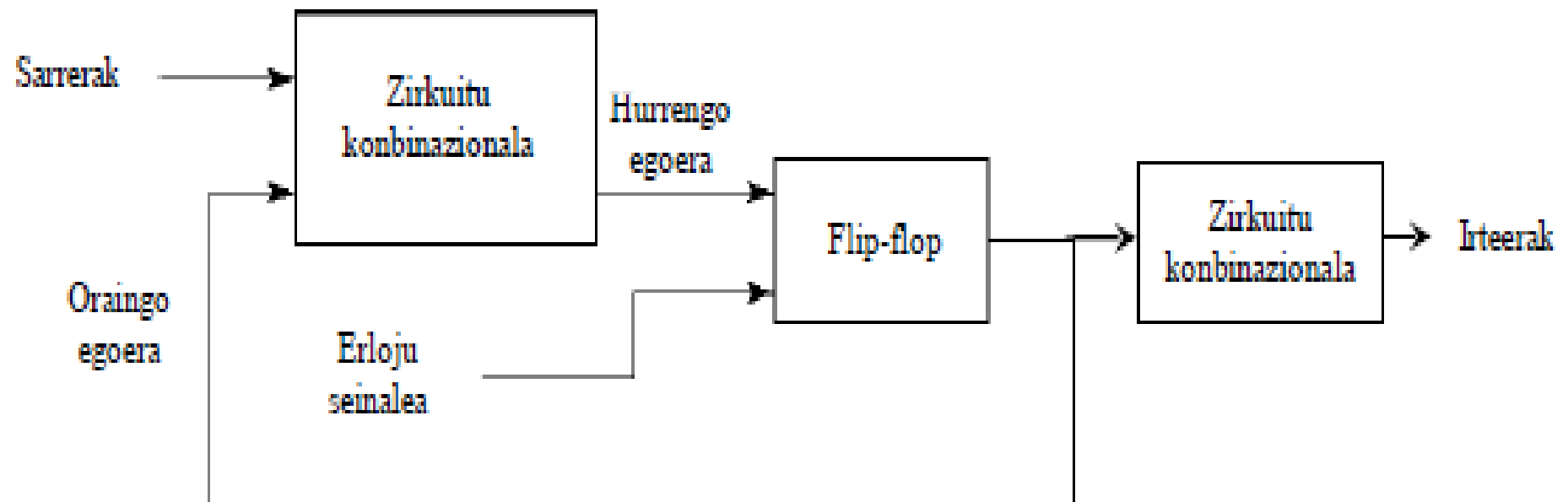
Sistema sekuentzialen diseinua: Mealy eredua

- Sistema sekuentziala diseinatzeko, irteerari buruzko bi eredu kontsideratu dezakegu
- Irteerako funtzioak, oraingo egoera eta sarrerako balioen menpe daude → Mealy eredua



Sistema sekuentzialen diseinua: Moore eredua

- Diseinua errazago bihur dezakegu, irteerako funtzioak bakarrik oraingo egoeraren menpekoak definitzen baditugu → Moore eredua
- Eredua honetan, lehenengo egoera aldaketa gertatzen da, eta gero irteerako aldaketa agertuko da



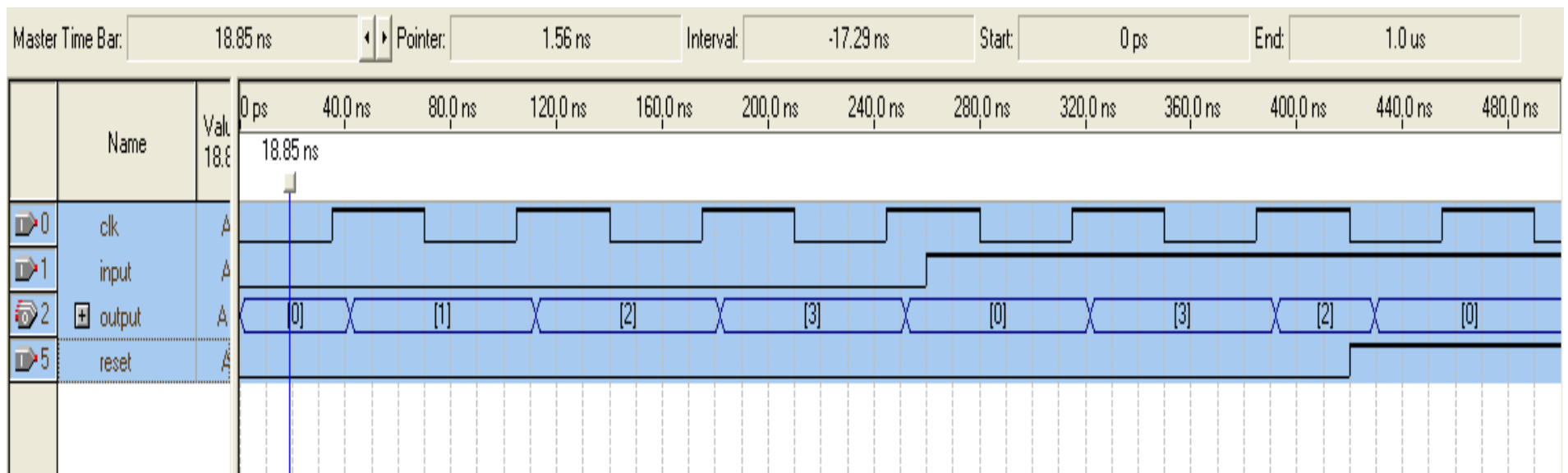
Sistema sekuentzialen diseinua

- Sistema sekuentzial bat diseinatzeko (Moore eredua erabiliko dugu), egoera diagrama definitu egin behar dugu, sistema portaeraren deskripzioa jarraituz
- Diagraman trantsizio aukera guztiak agertuko dira, eta deskribatutako portaera osoa gauzatzeko, behar ditugun egoera guztiak
- Oso egokia da, deskripzioaren aukera guztiak ulertzeko, aurreikusitako portaeraren kronogramaren bat marraztea

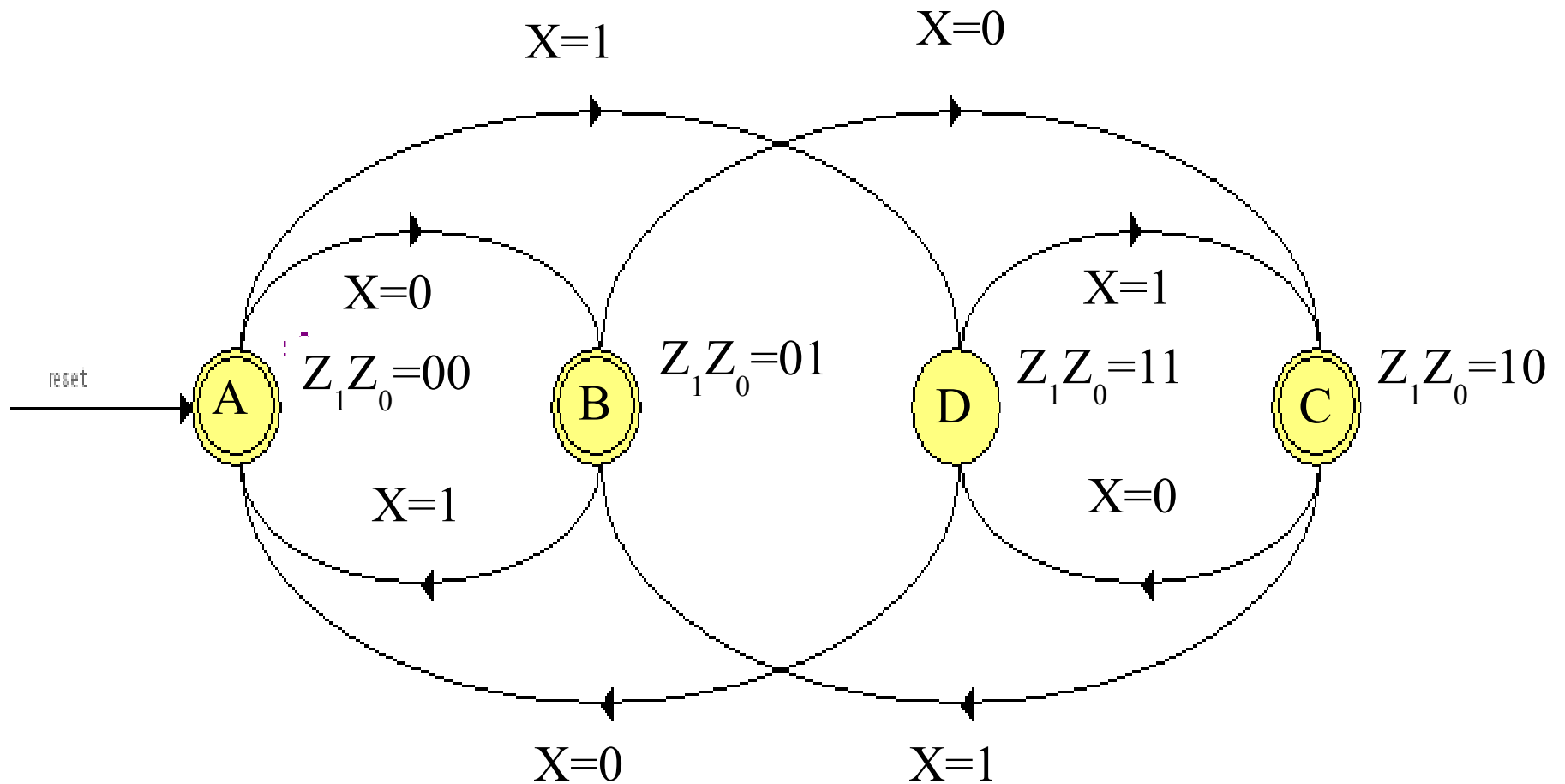
Sistema sekuentzialen diseinua

Portaeraren deskripzioa

- 2 biteko zirkuitu kontagailu bat diseinatu. Sarrerako seinalea $X=0$ denean kontaketa goranzkoa izango da, eta beheranzkoa $X=1$ denean



Sistema sekuentzialen diseinua



Egoera diagrama

Sistema sekuentzialen diseinua

- Informazio hau egoera taularen itxuran idatziko dugu, lerro bakoitzean oraingo egoera bat eta zutabe bakoitzean sarrerako aukera bat; irteerako balioak idatziko ditugu, lerro bakoitzean (egoera bakoitzean) bat
- Egoerak, aldagai bitarretan kodifikatzen dira (egoeren esleipena), gure egoera aldagaiak izango dira horiek

Sistema sekuentzialen diseinua

<i>Oraingo egoera</i>	$X=0$	$X=1$	Z_1	Z_0
A	B	D	0	0
B	C	A	0	1
C	D	B	1	0
D	A	C	1	1

Egoera taula

<i>Oraingo egoera</i>	Q_1	Q_0
A	0	0
B	0	1
C	1	0
D	1	1

Egoeraren esleipena

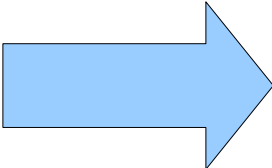
Q_1Q_0	$X=0$	$X=1$	Z_1	Z_0
00	01	11	0	0
01	10	00	0	1
10	11	01	1	0
11	00	10	1	1

$$Q_1^*Q_0^*$$

Sistema sekuentzialen diseinua

- Hurrengo egoeraren balio (Q^*) eta flip-floppen ekuazioen bidez, dagozkien flip-floppen sarrerako balioak (D, J-K) definitzen dira
- D flip-flopak erabiltzeko errazagoak dira, baina J-K flip-floppen bidez, zirkuitu errazagoak sortzen dira
- Balio hauekin, hurrengo egoera eta irteerako ekuazioak definituko ditugu, eta horrela sortutako zirkuitua gure deskripzioa jarraituko du

Sistema sekuentzialen diseinua

C	J	K	Q^*	$Q^{*'} $		Q	Q^*	J	K
X	X	X	Q	Q'		0	0	0	X
↑	0	0	Q	Q'		0	1	1	X
↑	0	1	0	1		1	0	X	1
↑	1	0	1	0		1	1	X	0
↑	1	1	Q'	Q					

JK flip-flopa : egia taula eta bertsio laburra

$Q_1 Q_0$	X=0		X=1		Z_1	Z_0
00	0X	1X	1X	1X	0	0
01	1X	X1	0X	X1	0	1
10	X0	1X	X1	1X	1	0
11	X1	X1	X0	X1	1	1

$J_1 K_1 J_0 K_0$

Sistema sekuentzialen diseinua

$Q_1 Q_0$	Z_1	Z_0
00	0	0
01	0	1
10	1	0
11	1	1

$$Z_1 = Q_1$$

$$Z_0 = Q_0$$

$$J_1 = Q_0 \cdot X' + Q_0' \cdot X = Q_0 \oplus X$$

$$K_1 = Q_0 \cdot X' + Q_0' \cdot X = Q_0 \oplus X$$

$$J_0 = 1$$

$$K_0 = 1$$

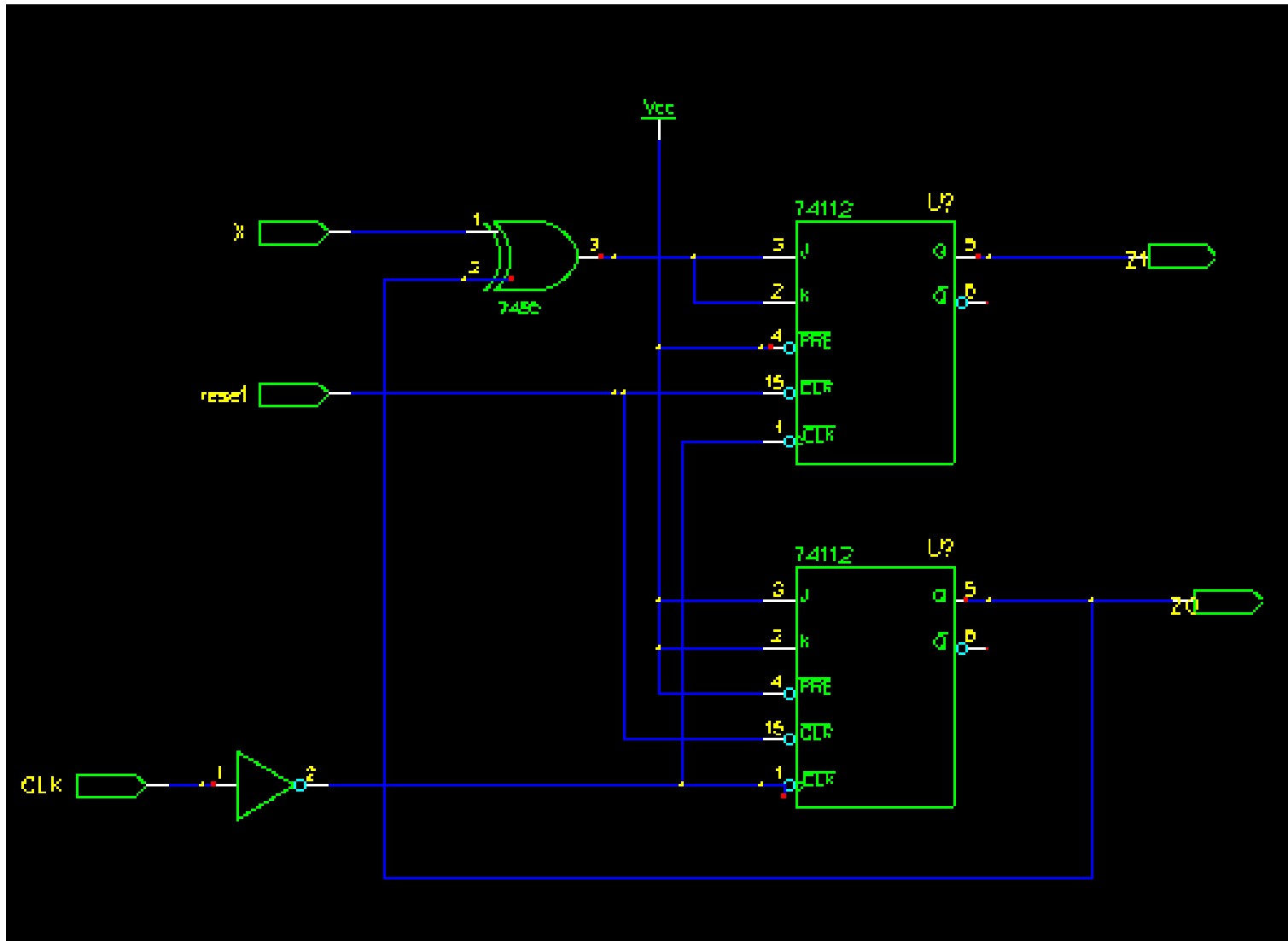
		X	
		0	1
Q_1	00		1
	01	1	
	11	X	X
	10	X	X
		J_1	
		X	

		X	
		0	1
Q_1	00	X	X
	01	X	X
	11	1	
	10		1
		K_1	
		X	

		X	
		0	1
Q_1	00	1	1
	01	X	X
	11	X	X
	10	1	1
		J_0	

		X	
		0	1
Q_1	00	X	X
	01	1	1
	11	1	1
	10	X	X
		K_0	

Sistema sekuentzialen diseinua



$$Z_1 = Q_1$$

$$Z_0 = Q_0$$

$$J_1 = Q_0 \oplus X$$

$$K = Q_0 \oplus X$$

$$J_0 = 1$$

$$K_0 = 1$$

$$\text{CLR} = \text{RESET}'$$

Sistema sekuentzialen diseinua

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;
```

```
ENTITY moore IS
```

```
PORT(
```

```
    clk:    IN  std_logic;
```

```
    x:      IN  std_logic;
```

```
    reset : IN  std_logic;
```

```
    Z      : OUT std_logic_vector(1  
    DOWNTO 0));
```

```
END moore;
```

```
ARCHITECTURE a OF moore IS
```

```
TYPE estado IS (s0, s1, s2, s3);
```

```
SIGNAL cuenta : estado;
```

```
BEGIN
```

```
PROCESS (clk, reset)
```

```
BEGIN
```

```
IF reset = '1' THEN cuenta <= s0;
```

```
ELSIF (clk'EVENT AND clk='1') THEN
```

```
    CASE cuenta IS
```

```
        WHEN s0=>
```

```
            IF x = '0' THEN cuenta <= s1;
```

```
            ELSE cuenta <= s3;
```

```
        END IF;
```

```
        WHEN s1=>
```

```
            IF X = '0' THEN cuenta <= s2;
```

```
            ELSE cuenta <= s0;
```

```
        END IF;
```

Sistema sekuentzialen diseinua

```
WHEN s2=>
  IF x = '0' THEN cuenta <= s3;
    ELSE cuenta <= s1;
  END IF;
WHEN s3 =>
  IF x = '0' THEN cuenta <= s0;
    ELSE cuenta <= s2;
  END IF;
END CASE;
```

END IF;

END PROCESS;

```
PROCESS (cuenta)
BEGIN
```

```
  CASE cuenta IS
    WHEN s0 => z <= "00";
    WHEN s1 => z <= "01";
    WHEN s2 => z <= "10";
    WHEN s3 => z <= "11";
  END CASE;
```

END PROCESS;

END a;