

Technika Cyfrowa

Sprawozdanie z ćwiczenia nr 2

Natalia Curzytek, Marta Stanisławska,
Karolina Nitsch, Szymon Kłodowski

Treść zadania

Korzystając tylko z konkretnego jednego typu przerzutników oraz z dowolnych bramek logicznych, proszę zaprojektować czterobitowy licznik działający zgodnie z ciągiem Fibonacciego (z nieobowiązkowym upraszczającym zastrzeżeniem, że wartość "1" powinna się pojawiać tylko raz w cyklu). Po uruchomieniu licznika, w kolejnych taktach zegara powinien on zatem przechodzić po wartościach:

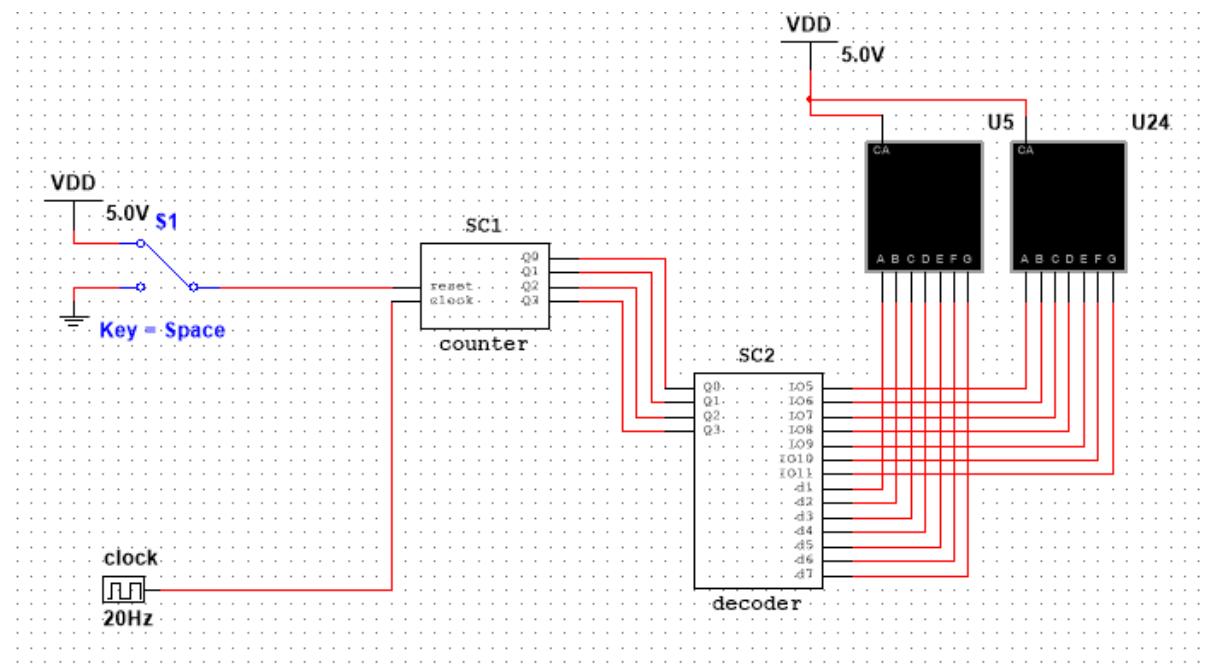
$$0, 1, 2, 3, 5, 8, 13, 0, 1, 2, 3, 5, 8, 13, 0, 1, \dots \text{ itd.}$$

Aktualna wartość wskazywana przez licznik powinna być widoczna na wyświetlaczaх siedmiosegmentowych.

Idea rozwiązania

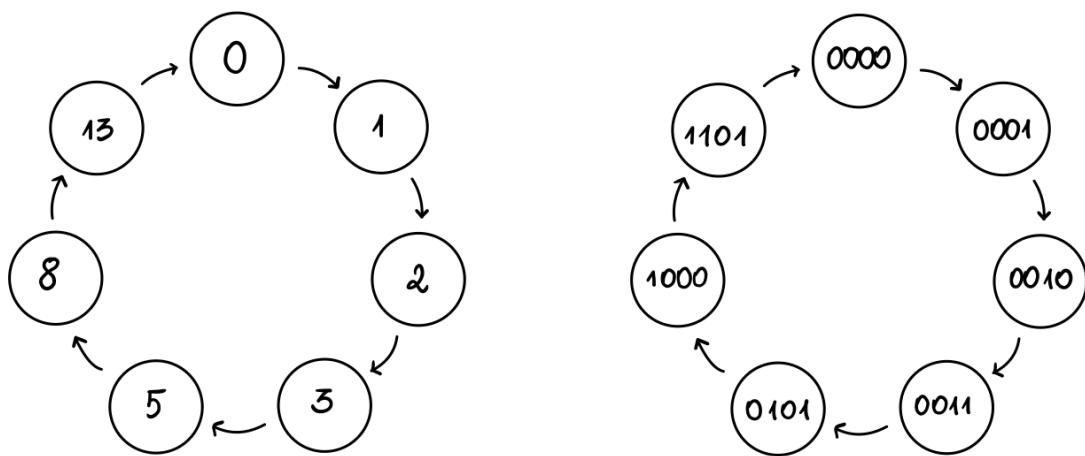
Do rozwiązania zadania wykorzystano przerzutniki typu D. Rozpoczęto od zaprojektowania funkcji wejściowych dla każdego przerzutnika na podstawie analizy przejść stanów. Do uproszczenia funkcji logicznych wykorzystano tabele Karnaugh. Następnie przetestowano projekt za pomocą symulacji w Pythonie, co potwierdziło poprawność działania licznika oraz stworzono układy testujące w środowisku symulacyjnym. Zaprojektowano też dekoder do wyświetlania wartości na wyświetlaczaх siedmiosegmentowych.

Rys. 1. Czarna skrzynka



Schemat licznika

Na rysunku 1 przedstawiono cykliczny diagram przejść stanów 4-bitowego licznika realizującego ciąg Fibonacciego. Szczegółowe przejścia pomiędzy stanami przedstawiono w tabeli 1.



Rys. 2. Cykl stanów licznika realizującego licznik liczb Fibonacciego w układzie 4-bitowym

Tabela 1. Przekształcenia stanów automatu w kolejnych taktach zegara

T(n)				T(n+1)			
Q3	Q2	Q1	Q0	Q3	Q2	Q1	Q0
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	1
0	1	0	1	1	0	0	0
1	0	0	0	1	1	0	1
1	1	0	1	0	0	0	0

Tabele Karnauga dla funkcji wejściowych D-przerzutników

Tabela 2. Tabela Karnaugh dla funkcji wejściowej D3

	Q1 Q0 = 00	Q1 Q0 = 01	Q1 Q0 = 11	Q1 Q0 = 10
Q3 Q2 = 00	0	0	0	0
Q3 Q2 = 01	X	1	X	X
Q3 Q2 = 11	X	0	X	X
Q3 Q2 = 10	1	X	X	X

$$D3 = \overline{Q3}Q2 + Q3\overline{Q2} = Q3 \text{ xor } Q2$$

Rys. 3. Schemat podukładu dla D3

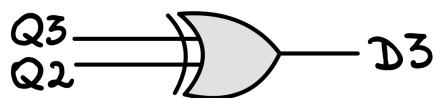


Tabela 3. Tabela Karnaugh dla funkcji wejściowej D2

	Q1 Q0 = 00	Q1 Q0 = 01	Q1 Q0 = 11	Q1 Q0 = 10
Q3 Q2 = 00	0	0	1	0
Q3 Q2 = 01	X	0	X	X
Q3 Q2 = 11	X	0	X	X
Q3 Q2 = 10	1	X	X	X

$$D2 = Q3\overline{Q2} + Q1Q0 = (Q3 \text{ and not}(Q2)) \text{ or } (Q1 \text{ and } Q0)$$

Rys. 4. Schemat podukładu dla D2

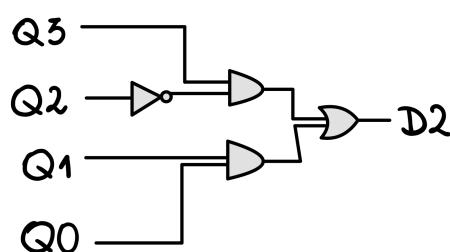


Tabela 4. Tabela Karnaugh funkcji wejściowej D1

	Q1 Q0 = 00	Q1 Q0 = 01	Q1 Q0 = 11	Q1 Q0 = 10
Q3 Q2 = 00	0	1	0	1
Q3 Q2 = 01	X	0	X	X
Q3 Q2 = 11	X	0	X	X
Q3 Q2 = 10	0	X	X	X

$$D1 = \overline{Q2} \overline{Q1} Q0 + Q1 \overline{Q0} = (\text{not}(Q2) \text{ and } \text{not}(Q1) \text{ and } Q0) \text{ or } (Q1 \text{ and } \text{not}(Q0))$$

Rys. 5. Schemat podukładu dla D1

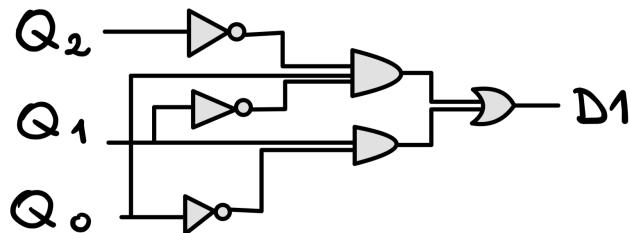
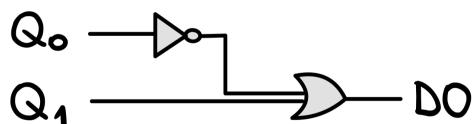


Tabela 5. Tabela Karnaugh funkcji wejściowej D0

	Q1 Q0 = 00	Q1 Q0 = 01	Q1 Q0 = 11	Q1 Q0 = 10
Q3 Q2 = 00	1	0	1	1
Q3 Q2 = 01	X	0	X	X
Q3 Q2 = 11	X	0	X	X
Q3 Q2 = 10	1	X	X	X

$$D0 = \overline{Q0} + Q1 = \text{not}(Q0) \text{ or } Q1$$

Rys. 6. Schemat podukładu dla D0



Sprawdzenie poprawności wyprowadzeń

W celu sprawdzenia poprawności wyprowadzonych wzorów napisany został program w języku python, zaprezentowany poniżej, symulujący działanie projektowanego układu.

1. Implementacja wykorzystanych bramek logicznych

```
def xor2(A, B): return (A and not B) or (not A and B)

def or2(A, B): return A or B

def and2(A, B): return A and B
def and3(A, B, C): return A and B and C

def not1(A): return not A
```

2. Implementacja otrzymanych funkcji wejściowych przerzutników

```
def D3 (Q0, Q1, Q2, Q3): return xor2(Q3, Q2)

def D2 (Q0, Q1, Q2, Q3): return or2(
    and2(Q3, not(Q2)),
    and2(Q1, Q0)
)

def D1 (Q0, Q1, Q2, Q3): return or2 (
    and3(not(Q2), not(Q1), Q0),
    and2(Q1, not(Q0)),
)

def D0 (Q0, Q1, Q2, Q3): return or2( Q1, not(Q0) )
```

3. Test poprawności wyprowadzenia

```
# Stan początkowy
Q3, Q2, Q1, Q0 = 0, 0, 0, 0

# Symulacja 15 kroków
for i in range(15):
    value = (Q3 << 3) | (Q2 << 2) | (Q1 << 1) | Q0
    print(f"Krok {i:2d}: {Q3}{Q2}{Q1}{Q0} → {value}")

    # Oblicz nowe wartości D
    new_Q3 = int(D3(Q0, Q1, Q2, Q3))
    new_Q2 = int(D2(Q0, Q1, Q2, Q3))
    new_Q1 = int(D1(Q0, Q1, Q2, Q3))
    new_Q0 = int(D0(Q0, Q1, Q2, Q3))

    # Aktualizacja stanu
    Q3, Q2, Q1, Q0 = new_Q3, new_Q2, new_Q1, new_Q0
```

Po uruchomieniu programu otrzymaliśmy sekwencję stanów automatu przedstawioną na rysunku 7. Uzyskane wyniki zgadzają się z przedstawionym wcześniej na rysunku 2 cyklem automatu, co wskazuje na poprawność wyprowadzeń.

```
Krok 0: 0000 → 0
Krok 1: 0001 → 1
Krok 2: 0010 → 2
Krok 3: 0011 → 3
Krok 4: 0101 → 5
Krok 5: 1000 → 8
Krok 6: 1101 → 13
Krok 7: 0000 → 0
Krok 8: 0001 → 1
Krok 9: 0010 → 2
Krok 10: 0011 → 3
Krok 11: 0101 → 5
Krok 12: 1000 → 8
Krok 13: 1101 → 13
Krok 14: 0000 → 0
```

Rys. 7. Sekwencja stanów automatu uzyskana w programie testującym wyprowadzone funkcje

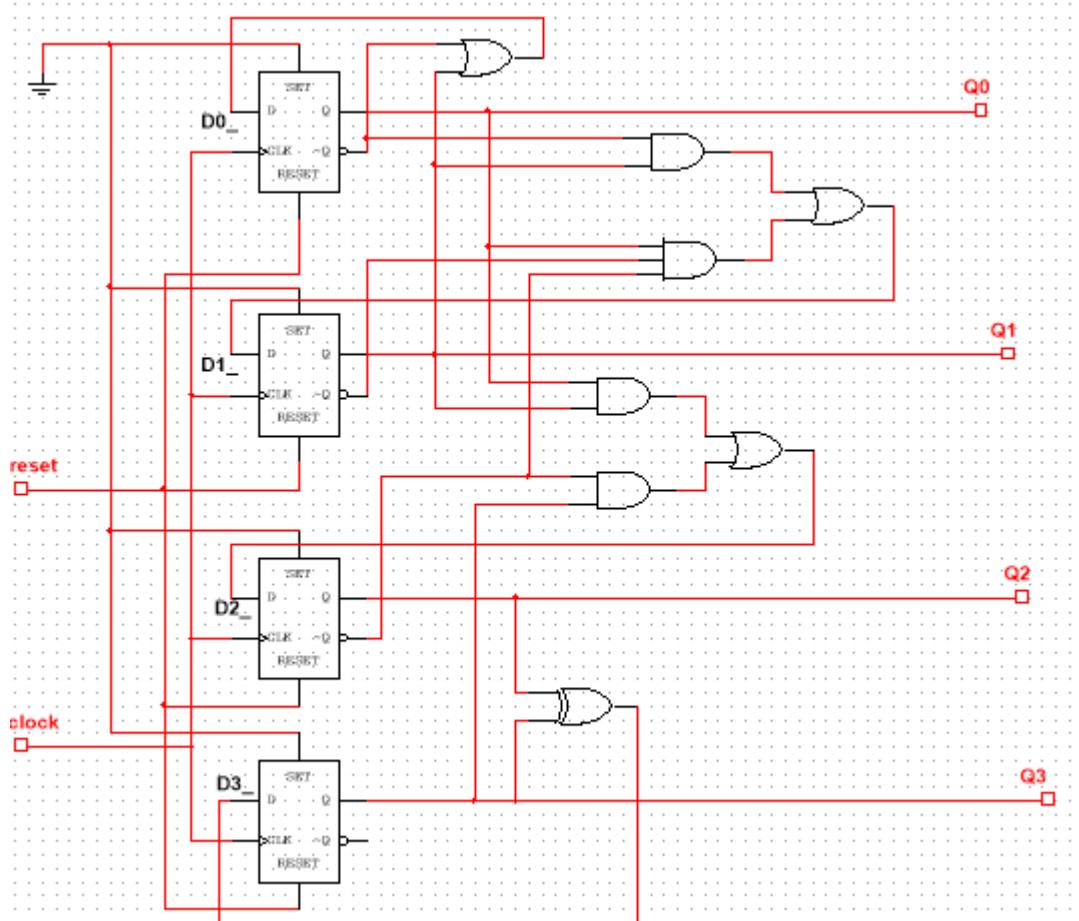
Implementacja układu

Układ został zaimplementowany z wykorzystaniem przerzutników typu D.

Tabela 6. Tabela przejść przerzutnika D

CLK	D	Q(n+1)
↑	1	1
↑	0	0

1. Schemat licznika zbudowanego przy użyciu przerzutników D i bramek logicznych, na podstawie wyprowadzonych wzorów.



Rys. 8. Podukład 'counter'

Przypomnienie wyprowadzonych wcześniej wzorów:

$$D3 = \overline{Q3}Q2 + Q3\overline{Q2} = Q3 \text{ xor } Q2$$

$$D2 = Q3\overline{Q2} + Q1Q0 = (Q3 \text{ and } \text{not}(Q2)) \text{ or } (Q1 \text{ and } Q0)$$

$$D1 = \overline{Q2}\overline{Q1}Q0 + Q1\overline{Q0} = (\text{not}(Q2) \text{ and } \text{not}(Q1) \text{ and } Q0) \text{ or } (Q1 \text{ and } \text{not}(Q0))$$

$$D0 = \overline{Q0} + Q1 = \text{not}(Q0) \text{ or } Q1$$

2. Podkład ‘decoder’

Podkład ‘decoder’ został zaprojektowany w celu wyświetlania liczb 0,1,2,3,5,8,13 w systemie dziesiętnym na wyświetlaczkach 7-segmentowych. Rozdziela liczbę na dwie cyfry przy użyciu bramek logicznych (wyprowadzenia poniżej) a następnie wykorzystuje 2 gotowe dekodery BCD do sterowania poszczególnymi segmentami wyświetlacza. W celu optymalnego zaprojektowania układu sporządzono tabelę prawdy oraz tabele Karnaugh.

Wprowadzenia dla podukładu 'decoder'

D,C,B,A - wejścia do dekodera BCD obsługującego wyświetlacz cyfry jedności.

dA - wejście do dekodera BCD obsługującego wyświetlacz cyfry dziesiątek. Dotyczy tylko najmłodszego bitu, ponieważ cyfra dziesiątek ma przyjmować wartość 0 lub 1. Pozostałe wejścia są na stałe podłączone do masy.

Tabela prawdy

<u>Q₃</u>	<u>Q₂</u>	<u>Q₁</u>	<u>Q₀</u>	<u>D</u>	<u>C</u>	<u>B</u>	<u>A</u>	<u>dA</u>
0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1	0
0	0	1	0	0	0	1	0	0
0	0	1	1	0	0	1	1	0
0	1	0	1	0	1	0	1	0
1	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	1	1

Tabela Karnaugh

1) dA - dotyczy wyświetlacza cyfry dziesiątek

Q₁Q₀	00	01	11	10
Q₃Q₂	00	0	0	0
00	0	0	0	0
01	X	0	X	X
11	X	1	X	X
10	0	X	X	X

$$dA = \overline{Q_3} Q_0$$

2) A

$Q_3 Q_2$	00	01	11	10
00	0	1	1	0
01	X	1	X	X
11	X	1	X	X
10	0	X	X	X

$$A = Q_0$$

3) B

$Q_3 Q_2$	00	01	11	10
00	0	0	1	1
01	X	0	X	X
11	X	1	X	X
10	0	X	X	X

$$B = Q_1 + Q_3 Q_2$$

4) C

$Q_1 Q_0$	00	01	11	10	
$Q_3 Q_2$	00	0	0	0	
01	X	1	X	X	
11	X	0	X	X	
10	0	X	X	X	

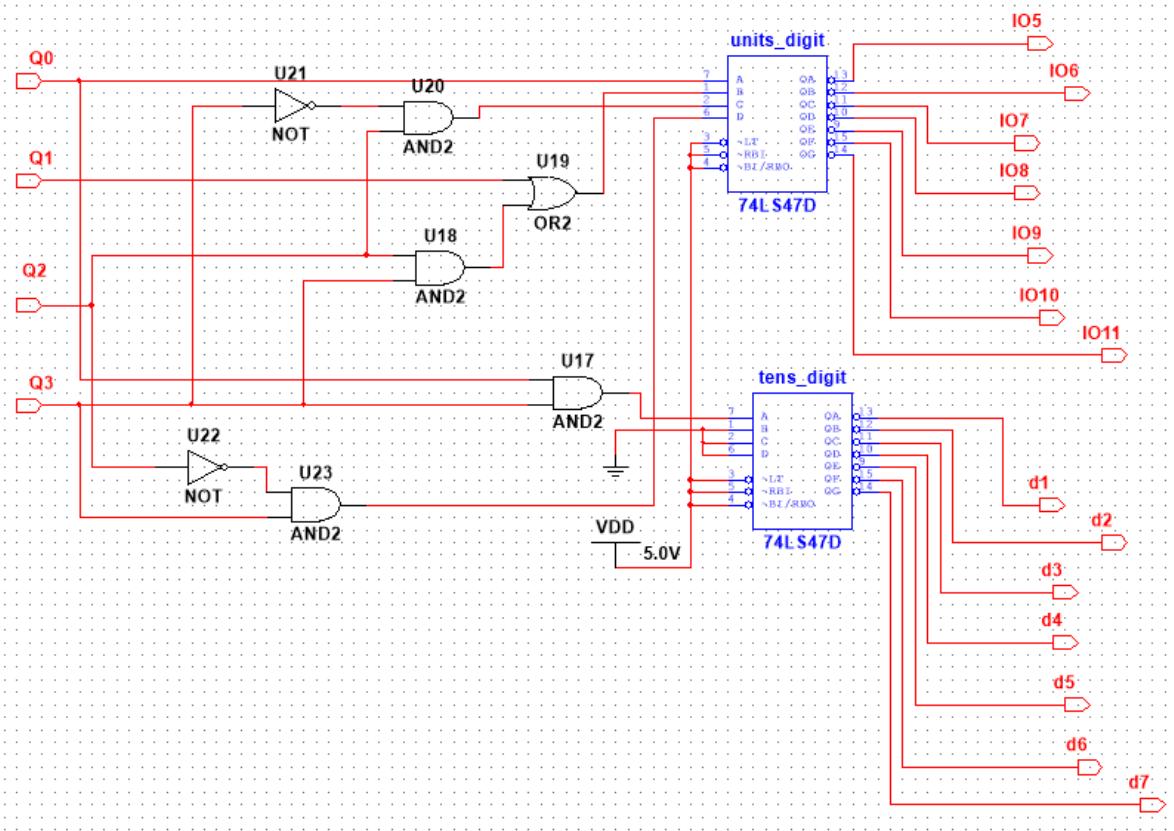
$$C = \overline{Q}_3 Q_2$$

5) D

$\overline{Q}_1 Q_0$	00	01	11	10	
$Q_3 Q_2$	00	0	0	0	
01	X	0	X	X	
11	X	0	X	X	
10	1	X	X	X	

$$D = Q_3 \overline{Q}_2$$

Schemat podukładu 'decoder'



Rys. 9. Podkład 'decoder'

Przypomnienie wyprowadzonych wzorów dla podukładu 'decoder'

$$dA = Q_3 Q_0$$

$$A = Q_0$$

$$B = Q_1 + Q_3 Q_2$$

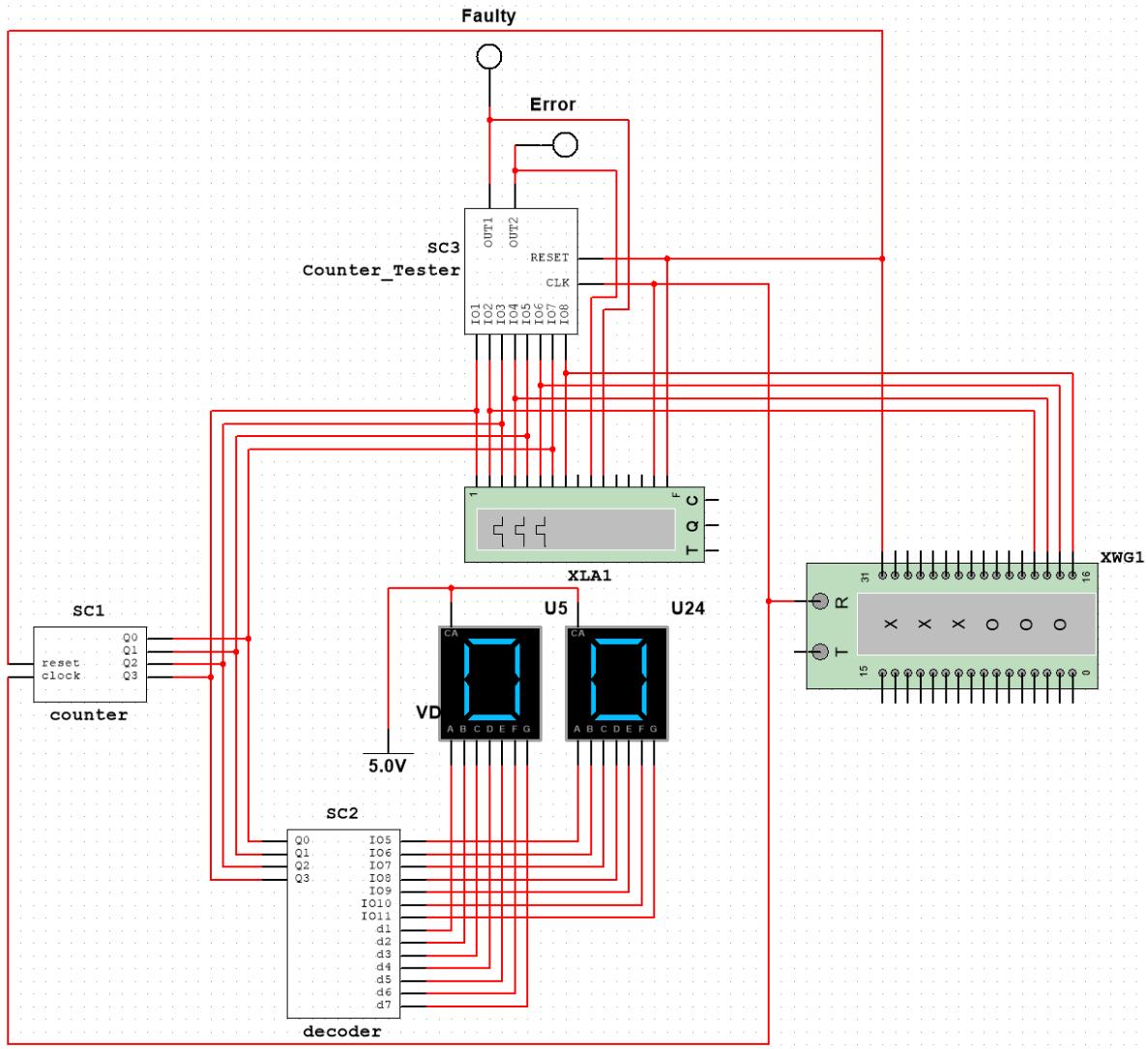
$$C = \overline{Q_3} Q_2$$

$$D = Q_3 \overline{Q_2}$$

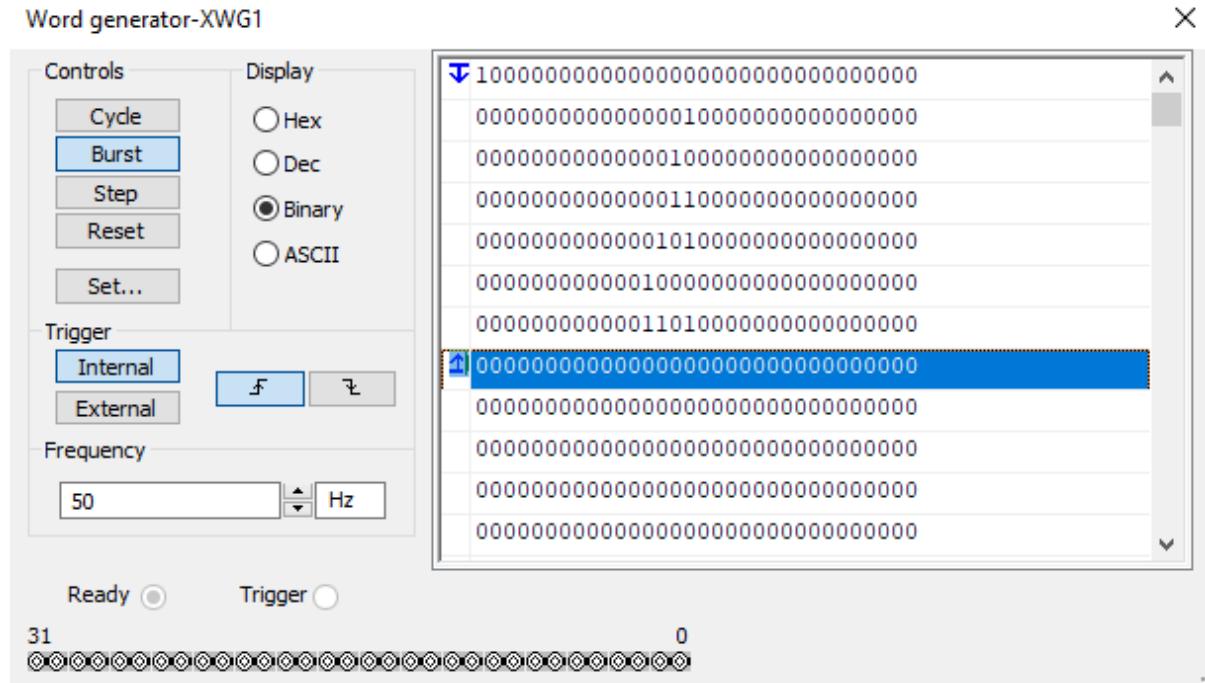
Układ testujący

Żeby przetestować nasze układy stworzyliśmy nowy układ zawierający generator słów i analizator logiczny oraz własny układ testujący poprawność układu.

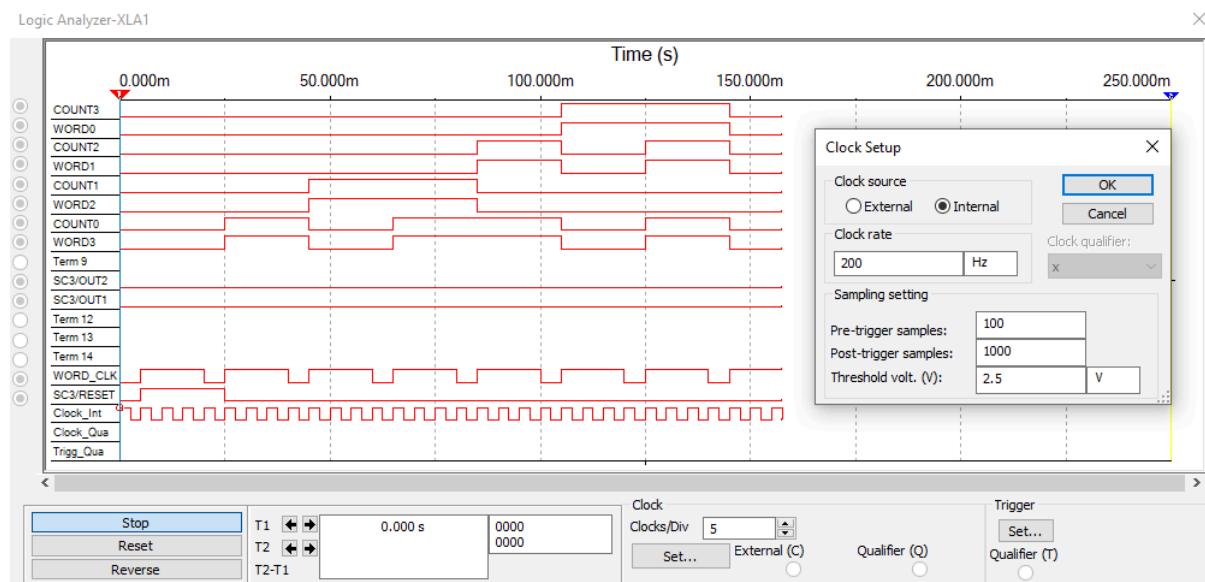
1. Układy z analizatorami logicznymi



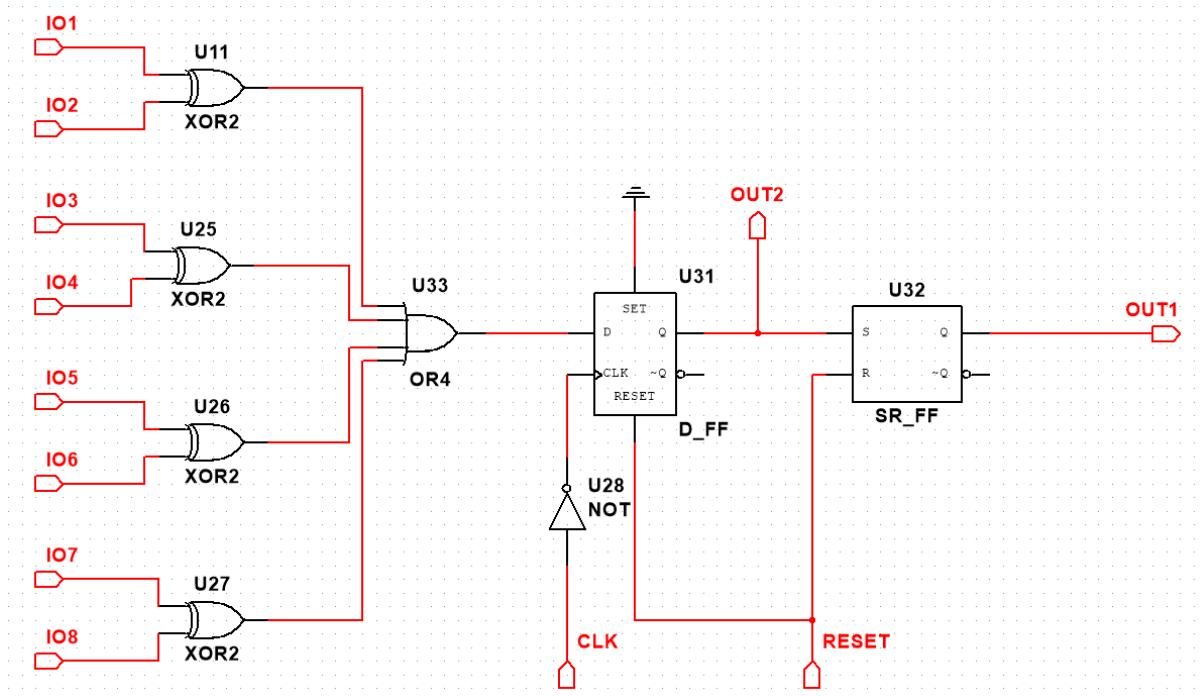
Generator słów - słowa w generatorze są zgodnie z schematem: 1 bit odpowiadający sygnałowi RESET, $11 \times '0'$, 4 bity odpowiadające wyjściu Counter'a, $16 \times '0'$



Analizator logyczny:



Counter_Tester:



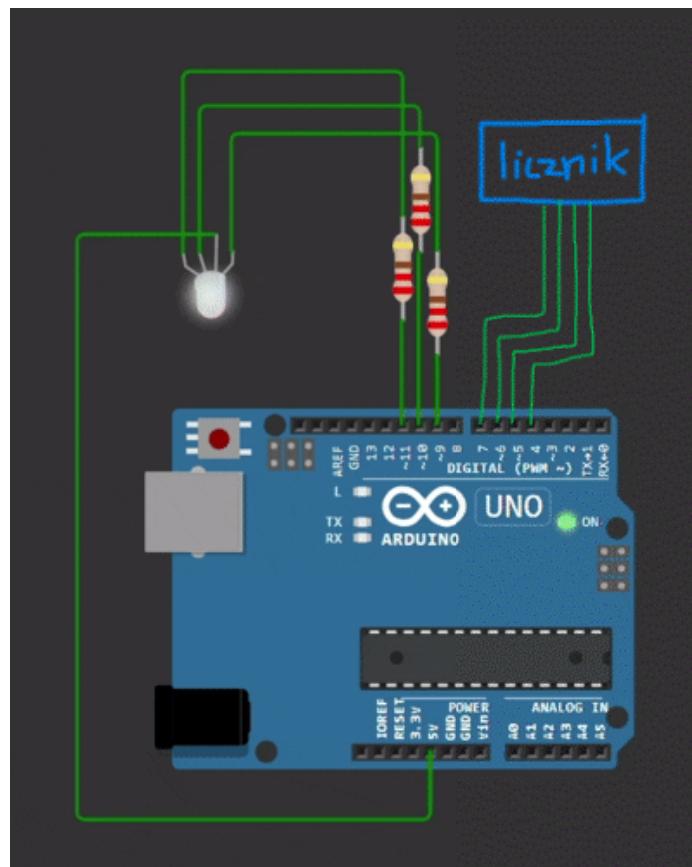
Zastosowania - oddychające światło

“Oddychające światło” to efekt świetlny, w którym intensywność światła zmienia się w czasie. Zaczyna się od zera (ciemność) i powoli nieliniowo rośnie do maksimum, a następnie powoli (np. liniowo) gaśnie z powrotem do zera - i tak w kółko.

W tym przypadku licznik z ciągiem Fibonacciego gwarantuje nieliniowość i daje bardziej “naturalny” efekt (najpierw szybkie małe różnice, a potem większe skoki).

1. Licznik Fibonacciego, w kolejnych taktach zegara podaje liczby, które reprezentują konkretną wartość jasności.
2. Każda wartość zasila układ PWM, czyli sygnał prostokątny o stałej częstotliwości, ale zmiennej szerokości impulsu (duty cycle) [0 -> 0%, 1 -> 8%, 2 -> 15%, ..., 8 -> 62%, 13 -> 100%].
3. Dioda LED będzie świecić z intensywnością odpowiadającą wartości z ciągu.

Ten efekt można wykorzystać między innymi w świątecznych lampkach choinkowych przy rytmicznej zmianie kolorów:



Wnioski

1. Zastosowanie D-przerzutników uprościło projektowanie – ich prosty sposób działania ($Q = D$ w kolejnym taktie zegara) sprawił, że tworzenie funkcji sterujących było prostsze niż w przypadku innych typów przerzutników (np. JK czy T).
2. Projektowanie liczników o niestandardowych sekwencjach stanów daje dużą elastyczność i pokazuje praktyczne aspekty stosowania automatów sekwencyjnych w elektronice cyfrowej. Dzięki temu licznik nie był po prostu licznikiem binarnym, a można było zaprogramować konkretne przejścia – w tym wypadku ciąg Fibonacciego.

Co można było zrobić inaczej? - porównanie możliwych przerzutników

1. (wybrany) **Przerzutnik D (Data / Delay)** – bardzo prosty w użyciu (jedno wejście sterujące), użyteczny do rejestrów, liczników i pamięci; nie pozwala bezpośrednio na tworzenie liczników modulo bez dodatkowej logiki.
2. **Przerzutnik T (Toggle)** – bardzo prosty do projektowania liczników binarnych, umożliwia tworzenie liczników bez dodatkowych bramek; trudniej sterować bardziej skomplikowanymi ciągami (jak Fibonacciego).
3. **Przerzutnik JK** – bardzo uniwersalny, można z niego uzyskać działanie każdego innego przerzutnika przy odpowiednim połączeniu wejść; skomplikowana logika sterująca (trudniej upraszczać funkcje), posiada dwie linie wejściowe (zwiększa liczbę połączeń).
4. **Przerzutnik RS (Set-Reset)** – dobrze nadaje się do prostych układów pamięciowych, posiada asynchroniczne sterowanie; brak zegara utrudnia synchronizację, stan nieokreślony przy $R = S = 1$.

W kontekście zadania: najlepszym wyborem jest przerzutnik D, który wymusza pełną analizę logiczną i projekt funkcji sterujących. Alternatywnym wyborem mógłby być przerzutnik T, jednak nie sprawdza się aż tak dobrze przy skomplikowanych ciągach.