

# Logika cyfrowa

Wykład 7: liczniki, moduły sparametryzowane, generowanie

---

Marek Materzok

15 kwietnia 2024

## Moduły sparametryzowane i generowanie

---

# Moduły sparametryzowane

Współdzielenie kodu w przypadku wykorzystania modułów o zbliżonej budowie.

Przykładowo, zamiast poniższego kodu:

```
module mux2_1bit(output o, input a, b, s);
    assign o = s ? a : b;
endmodule

module mux2_2bit(output [1:0] o, input [1:0] a, b, input s);
    assign o = s ? a : b;
endmodule
```

Można napisać:

```
module mux2#(parameter N=1)
    (output [N-1:0] o, input [N-1:0] a, b, input s);
    assign o = s ? a : b;
endmodule
```

## Moduł sparametryzowany z pętlą

Sparametryzowany enkoder priorytetowy:

```
module prio_encoder#(parameter N = 2)(  
    output [N-1:0] o,  
    input [2**N-1:0] i  
);  
    integer k;  
    always_comb begin  
        o = N'bxx;  
        for (k = 0; k < 2**N; k = k + 1)  
            if (i[k]) o = k;  
        end  
    endmodule
```

## Moduł sparametryzowany z pętlą

Sparametryzowany dekodер używający instrukcji pętli:

```
module decoder#(parameter N = 2)(  
    output [2**N-1:0] o,  
    input [N-1:0] i  
);  
    integer k;  
    always_comb begin  
        o = 0;  
        for (k = 0; k < 2**N; k = k + 1)  
            if (i == k) o[k] = 1;  
        end  
    endmodule
```

## Sumator jednobitowy

```
module halfadder(  
    output o, c,  
    input a, b,  
);  
    assign o = a ^ b;  
    assign c = a & b;  
endmodule
```

```
module fulladder(  
    output o, co,  
    input a, b, c  
);  
    logic t, c1, c2;  
    halfadder ha1(t, c1, a, b);  
    halfadder ha2(o, c2, t, c);  
    assign co = c1 | c2;  
endmodule
```

## Sumator czterobitowy

```
module adder4(  
    output [3:0] o,    output c4,  
    input  [3:0] a, b, input  c0  
);  
    logic c1, c2, c3;  
    fulladder fa1(o[0], c1, a[0], b[0], c0);  
    fulladder fa2(o[1], c2, a[1], b[1], c1);  
    fulladder fa3(o[2], c3, a[2], b[2], c2);  
    fulladder fa4(o[3], c4, a[3], b[3], c3);  
endmodule
```

## Sumator n-bitowy – z generowaniem

```
module adder#(parameter W = 4) (  
    output [W-1:0] o,    output cn,  
    input  [W-1:0] a, b, input  c0  
);  
    logic [W:0] c;  
    assign c[0] = c0;  
    assign cn = c[W];  
    genvar i;  
    for (i = 0; i < W; i = i+1)  
        fulladder fa1(o[i], c[i+1], a[i], b[i], c[i]);  
endmodule
```



## Alternatywne rozwiązanie

```
module adder#(parameter W = 4) (  
    output [W-1:0] o,    output cn,  
    input  [W-1:0] a, b, input  c0  
);  
    logic [W:0] c;  
    assign c[0] = c0;  
    assign cn = c[W];  
    fulladder#(W) fa(o, c[1:+W], a, b, c[0:+W]);  
endmodule
```

# Instrukcja pętli a pętla generująca

Podobna składnia, różne działanie:

instrukcja pętli

jest instrukcją

w pętli instrukcja

zmienna `integer`

użycie w blokach `always`

syntezowana po generowaniu

pętla generująca

generuje bloki

w pętli blok (np. `always`, `assign`) lub

instancja modułu

zmienna `genvar`

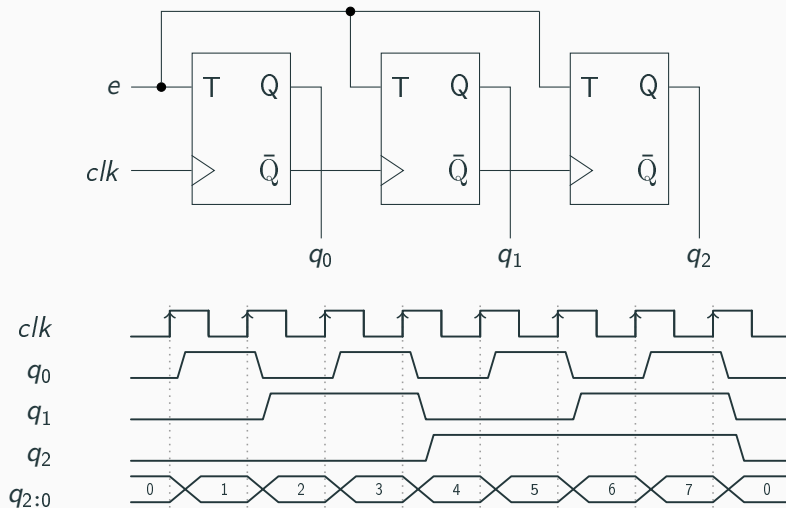
użycie w module, nie w blokach

syntezowana przed blokami

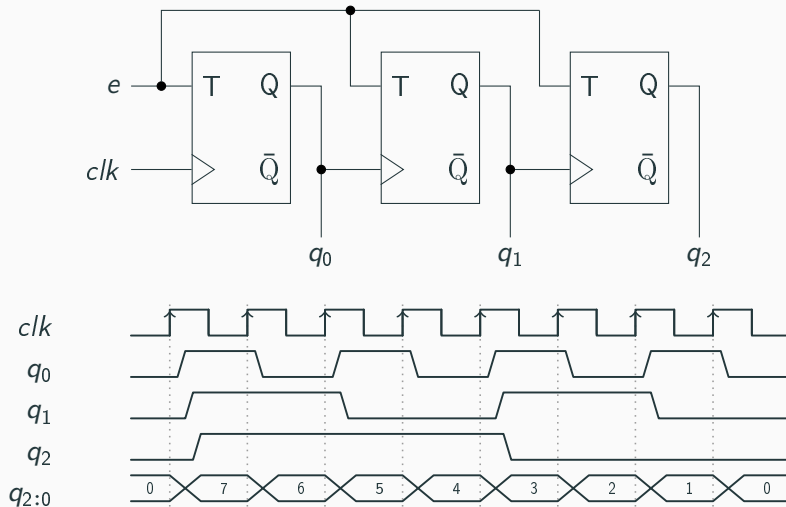
# Liczniki

---

# Licznik asynchroniczny



## Licznik asynchroniczny liczący w tył



## Licznik asynchroniczny – model bramkowy

```
module tff(output q, nq, input t, clk, nrst);
    logic ns, nr, ns1, nr1, j, k;
    nand n1(ns, clk, j), n2(nr, clk, k), n3(q, ns, nq),
        n4(nq, nr, q, nrst), n5(ns1, !clk, t, nq),
        n6(nr1, !clk, t, q), n7(j, ns1, k), n8(k, nr1, j, nrst);
endmodule

module asyncnt#(parameter N = 3)(output [N-1:0] q, input en, clk, nrst);
    genvar n;
    logic [N-1:-1] nq;
    assign nq[-1] = clk;
    for (n = 0; n < N; n = n + 1)
        tff tf(q[n], nq[n], en, nq[n-1], nrst);
endmodule
```

## Licznik synchroniczny – wprowadzenie

Wada licznika asynchronicznego: opóźnienia po zboczu zegara

Cel: zmiana wszystkich bitów w tym samym czasie

## Licznik synchroniczny – wprowadzenie

Wada licznika asynchronicznego: opóźnienia po zboczu zegara

Cel: zmiana wszystkich bitów w tym samym czasie

n	$q_2$	$q_1$	$q_0$
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8	0	0	0



## Licznik synchroniczny – wprowadzenie

Wada licznika asynchronicznego: opóźnienia po zboczu zegara

Cel: zmiana wszystkich bitów w tym samym czasie

n	$q_2$	$q_1$	$q_0$
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8	0	0	0

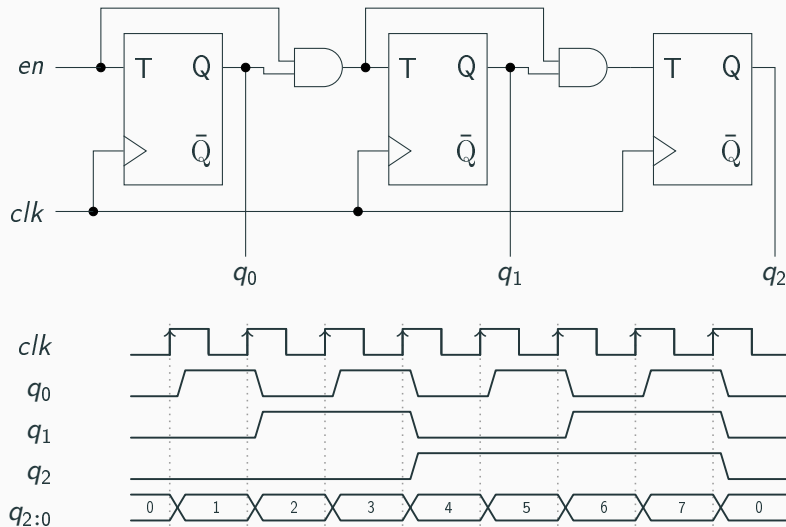
$$T_0 = e$$

$$T_1 = eq_0$$

$$T_2 = eq_0q_1$$

...

# Licznik synchroniczny



## Licznik synchroniczny – model bramkowy

```
module tff(output q, nq, input t, clk, nrst);
    logic ns, nr, ns1, nr1, j, k;
    nand n1(ns, clk, j), n2(nr, clk, k), n3(q, ns, nq),
        n4(nq, nr, q, nrst), n5(ns1, !clk, t, nq),
        n6(nr1, !clk, t, q), n7(j, ns1, k), n8(k, nr1, j, nrst);
endmodule

module syncnt#(parameter N = 3)(output [N-1:0] q, input en, clk, nrst);
    genvar n;
    logic [N-1:0] t;
    assign t = {q & t, en};
    for (n = 0; n < N; n = n + 1)
        tff tf(.q(q[n]), .t(t[n]), .clk(clk), .nrst(nrst));
endmodule
```

## Porównanie liczników

```
module cnts(output [2:0] q1, q2, input en, clk, nrst);  
    asyncnt c1(q1, en, clk, nrst);  
    syncnt c2(q2, en, clk, nrst);  
endmodule
```

Liczniki asynchroniczne

Zegary przerzutników wyzwalane  
przez wyjścia innych przerzutników

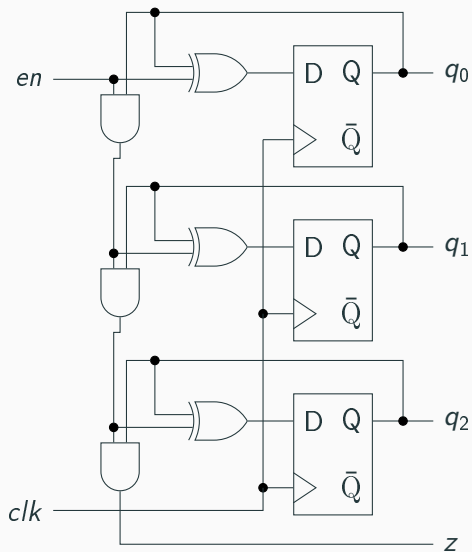
Wyjścia zmieniają się w różnych cza-  
sach

Liczniki synchroniczne

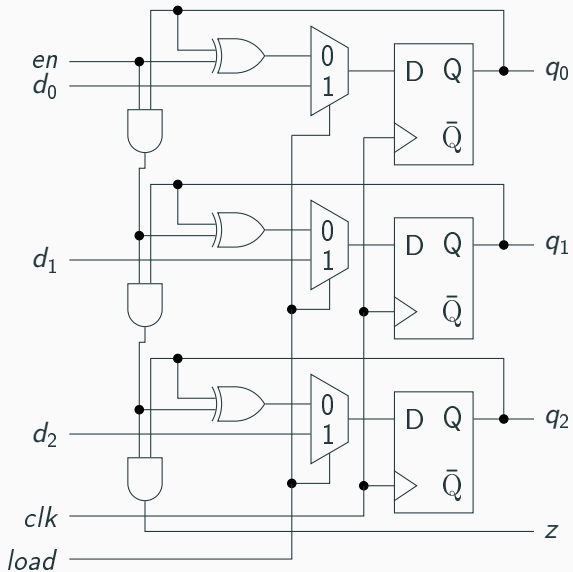
Zegary przerzutników połączone rów-  
nolegle

Wyjścia zmieniają się jednocześnie

## Licznik synchroniczny używający przerzutników D



## Licznik z ładowaniem równoległym

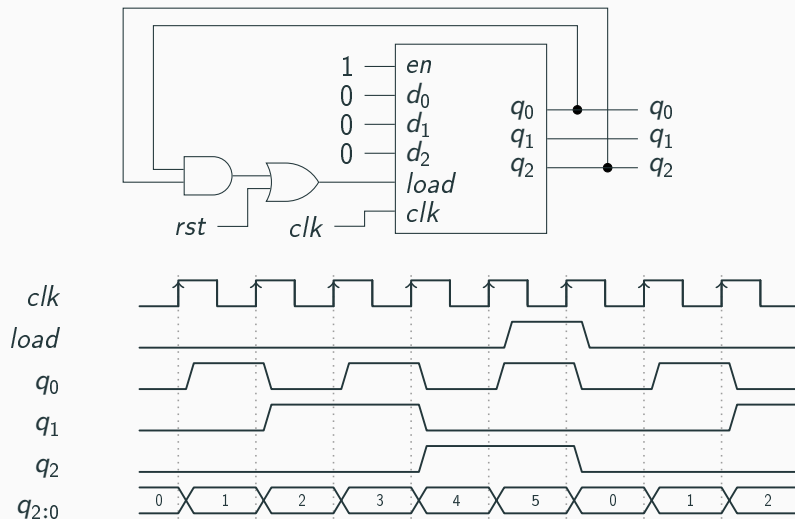


## Licznik z ładowaniem równoległym – model bramkowy

```
module dff(output q, input clk, d);
    logic r, s, nr, ns, q;
    nand gq(q, nr, nq), gnq(nq, ns, q), gr(nr, clk, r),
        gs(ns, nr, clk, s), gr1(r, nr, s), gs1(s, ns, d);
endmodule

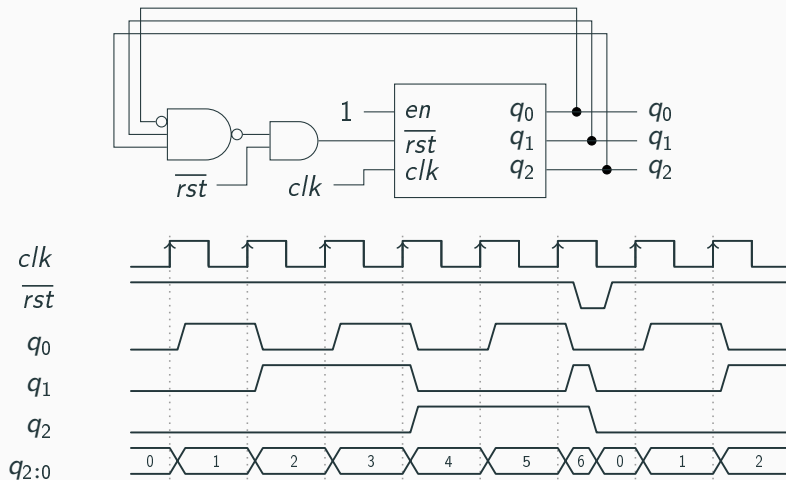
module syncnt_load#(parameter N = 3)(
    output [N-1:0] q, input [N-1:0] d,
    input en, clk, load
);
    genvar n;
    logic [N-1:0] t;
    assign t = {q & t, en};
    for (n = 0; n < N; n = n + 1)
        dff df(q[n], clk, load ? d[n] : q[n] ^ t[n]);
endmodule
```

## Licznik modulo – synchroniczny reset





## Licznik modulo – asynchroniczny reset

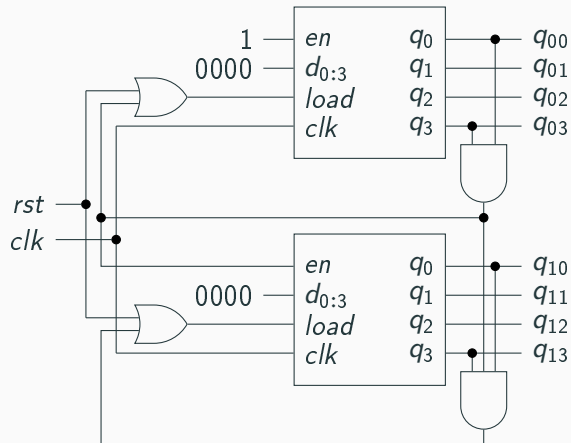


Uwaga na glitche!

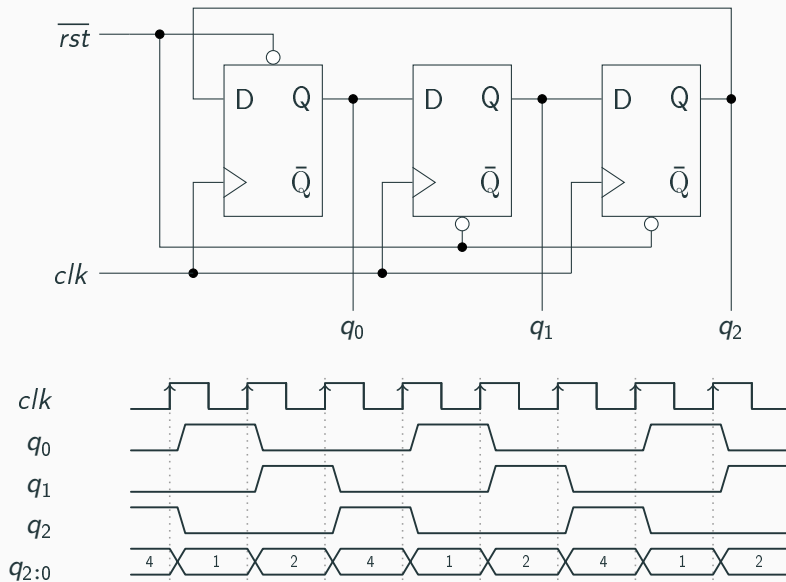
```
module modulo6_sync(output [2:0] q, input clk, rst);
    syncnt_load sc(q, 3'd0, 1'b1, clk,
                  q[0] && q[2] || rst);
endmodule

module modulo6_async(output [2:0] q, input clk, nrst);
    syncnt sc(q, 1'b1, clk,
              nrst && !(q[0] && q[1] && q[2]));
endmodule
```

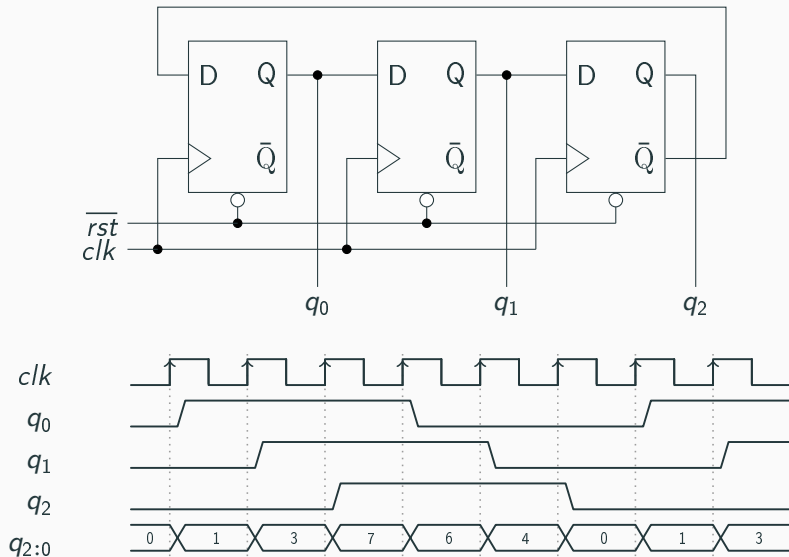
# Licznik BCD



# Licznik pierścieniowy



# Licznik Johnsona



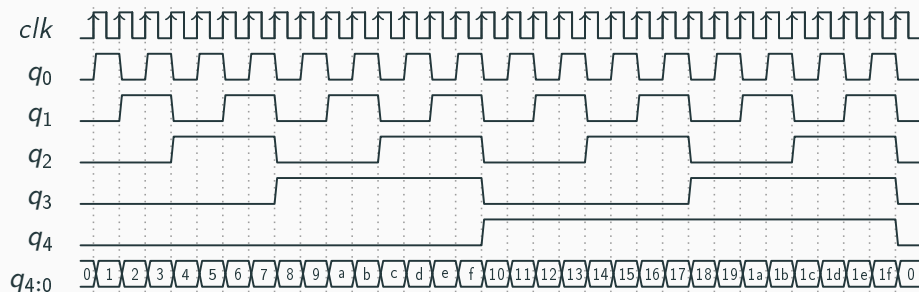
- Liczniki przedstawione na dzisiejszym wykładzie były projektowane w sposób intuicyjny.
- Na dalszych wykładach zostaną pokazane metody, którymi można konstruować liczniki (i inne układy ze stanem) w sposób usystematyzowany.

## Zastosowania liczników

---

# Spowalnianie sygnału zegarowego

Każdy stopień licznika redukuje częstotliwość zegara dwukrotnie:

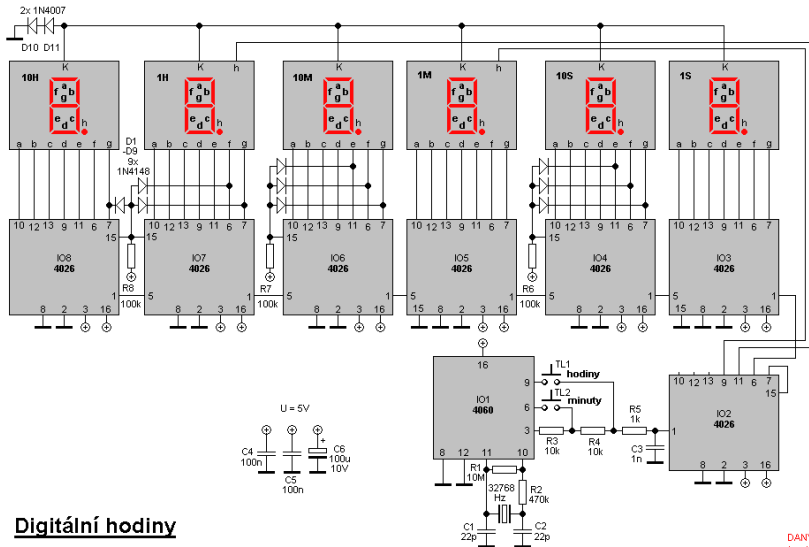




# Zegarek kwarcowy

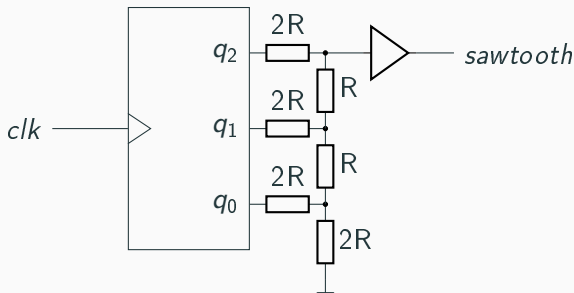
- Kwarc zegarkowy:  $32768 \text{ Hz} = 2^{15} \text{ Hz}$
- Licznikiem 14-bitowym (układ 4060) generujemy  $2 \text{ Hz}$  (sygnał dwukropka)
- 1-bitowym licznikiem redukujemy do  $1 \text{ Hz}$
- Licznikami BCD liczymy sekundy, dziesiątki sekund, minuty, dziesiątki minut, godziny, dziesiątki godzin
- Odpowiednim dekodery sterujemy wyświetlaczami 7-segmentowymi (układ 4026 integruje dekodery z licznikami BCD)
- Układy resetu zapewniają liczenie modulo dla sekund, minut i godzin

# Zegarek kwarcowy



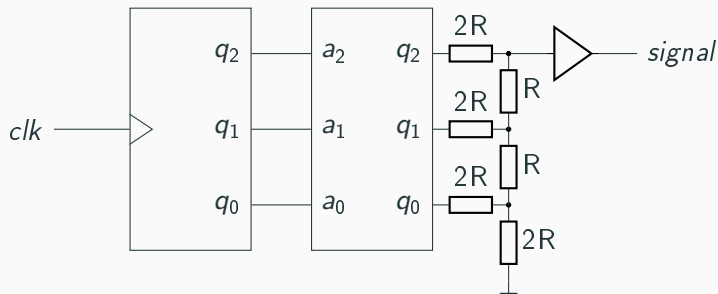
# Generator sygnału piłokształtnego

- Wartość licznika może sterować przetwornikiem cyfrowo-analogowym (DAC – digital to analog converter).
- Prostym przetwornikiem DAC jest tzw. drabinka R-2R.

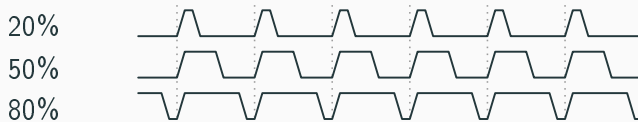


# Generator sygnału o dowolnym kształcie

- Wartość licznika może być użyta jako adres w pamięci ROM.  
(O pamięciach będzie więcej na kolejnych wykładach.)
- Pamięć może przechowywać dowolny sygnał (np. melodyjkę od kartki elektronicznej).

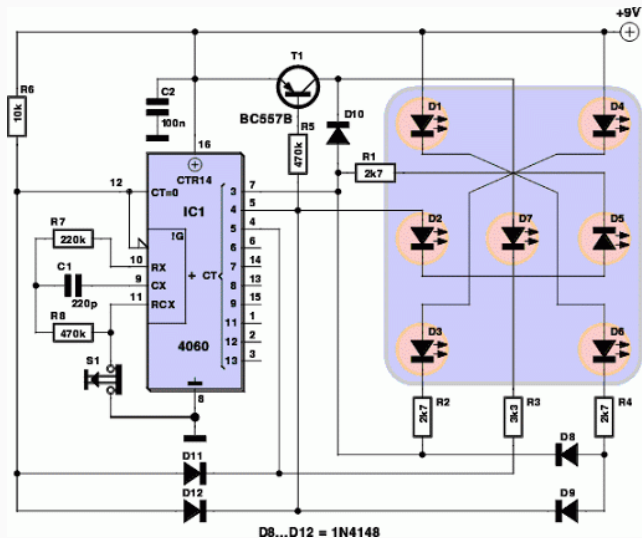


- Sygnał PWM (Pulse Width Modulation) – sygnał prostokątny o stałej częstotliwości, ale zmiennym wypełnieniu:



- Używany m.in. do: zmiany jasności diod LED, sterowania silnikami
- Sposób generowania: układ licznika + układ porównujący

# Elektroniczna kostka do gry



# Układy asynchroniczne i synchroniczne

---

Układy ze stanem (**sekwencyjne**) dzielą się na dwa główne rodzaje:

- **asynchroniczne** i
- **synchroniczne**.



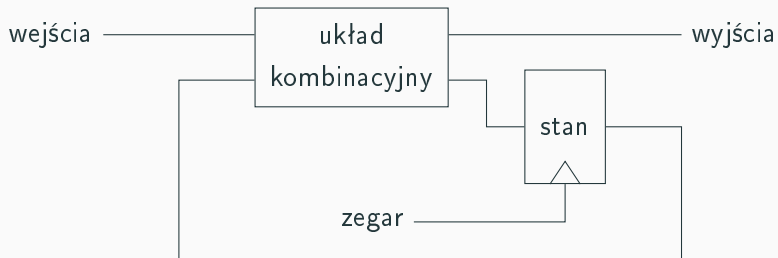
# Układy asynchroniczne

- Przerzutniki są wyzwalane sygnałem pochodzącym z **dowolnego źródła** (w tym z zewnętrznego źródła lub sygnałów danych).
- Przykład – liczniki asynchroniczne: zegar bitu  $n + 1$  pochodzi z  $n$ -tego bitu.
- Zalety:
  - Mogą być mniejsze i efektywniejsze od układów synchronicznych.
- Wady:
  - Mogą być podatne na glitche.
  - Mogą być podatne na wyścigi czasowe oraz naruszenia timingów (czasów ustalania i podtrzymania).
  - W związku z powyższymi: **trudno zapewnić niezawodność**.

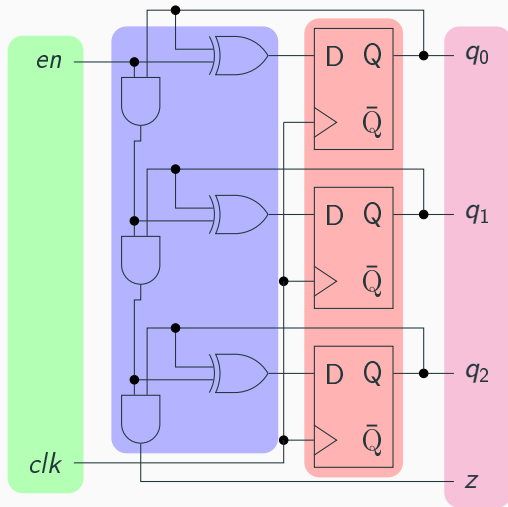
# Układy synchroniczne

- Przerzutniki są wyzwalane **wspólnym** sygnałem zegarowym.
- Forma **dyscypliny projektowej**.
- Przykład – liczniki synchroniczne: wszystkie bity wyzwalane jednym sygnałem.
- Zalety:
  - Prosty model czasu – stan w cyklu  $n + 1$  zależy tylko od stanu w cyklu  $n$  oraz wejść.
  - Odporność na glitche – istotny tylko stan sygnału blisko zbocza zegara.
  - Prosta analiza timingów.
- Wady:
  - Koszt synchronizacji – gdy składowe układu mają ścieżki różnej długości, część układu „nudzi się”.
- W dalszej części wykładu będziemy zajmować się tylko układami synchronicznymi!

# Budowa układu synchronicznego



# Licznik synchroniczny jako układ synchroniczny



- wejścia i zegar
- układ kombinacyjny
- stan
- wyjścia

# Analiza timingu układu synchronicznego

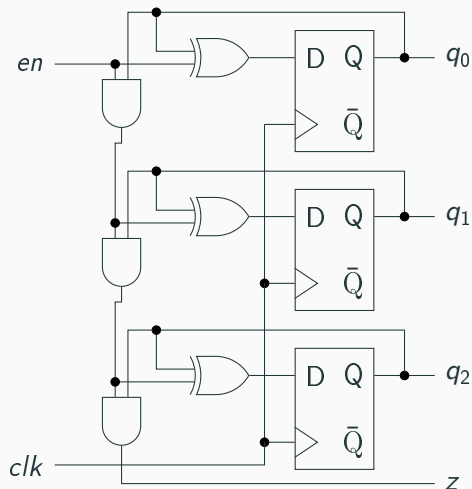
Aby układ synchroniczny działał poprawnie, musi być spełniona nierówność:

$$\frac{1}{f_{clk}} = T_{clk} > t_p + t_{crit} + t_{su}$$

Gdzie:

- $f_{clk}$  – częstotliwość sygnału zegara
- $T_{clk}$  – okres sygnału zegara
- $t_p$  – czas propagacji przerzutnika
- $t_{crit}$  – ścieżka krytyczna układu kombinacyjnego
- $t_{su}$  – czas ustalania przerzutnika

## Analiza timingu układu synchronicznego – przykład



Przyjmijmy, że:

- $t_p^{\text{dff}} = 44 \text{ ns}$
- $t_p^{\text{and}} = 23 \text{ ns}$
- $t_p^{\text{xor}} = 30 \text{ ns}$
- $t_{su}^{\text{dff}} = 20 \text{ ns}$

Minimalny okres licznika:

$$T_{clk} > t_p^{\text{dff}} + 2t_p^{\text{and}} + t_p^{\text{xor}} + t_{su}^{\text{dff}} = 140 \text{ ns}$$

Maksymalny zegar:

$$f_{clk} < T_{clk}^{-1} \approx 7 \text{ MHz}$$