

# Laboratorium 2

## Układ odliczający

Łukasz Kwinta, Kacper Kozubowski, Ida Ciepiela

kwiecień 2024

---

# Spis treści

---

<b>1</b>	<b>Cel zadania</b>	<b>3</b>
<b>2</b>	<b>Idea rozwiązania</b>	<b>3</b>
<b>3</b>	<b>Układ timer</b>	<b>4</b>
3.1	Black box . . . . .	4
3.1.1	Wejścia . . . . .	4
3.1.2	Wyjścia . . . . .	5
3.2	Diagram załączania układów . . . . .	6
3.3	Kontrola działania licznika . . . . .	6
3.3.1	Tablice prawdy . . . . .	7
3.3.2	Wyprowadzenie formuł . . . . .	7
3.3.3	Realizacja formuł . . . . .	8
3.4	Podukład ustawiający czas licznika . . . . .	9
3.4.1	Wejścia i Wyjścia . . . . .	9
3.4.2	Tabele prawdy . . . . .	10
3.4.3	Wyprowadzenie formuł . . . . .	10
3.4.4	Realizacja formuł . . . . .	11
3.5	Podukład kontrolujący przerzutniki . . . . .	12
3.5.1	Wejścia i Wyjścia . . . . .	12
3.5.2	Tabele prawdy . . . . .	12
3.5.3	Wyprowadzenie formuł . . . . .	15
3.5.4	Realizacja formuł . . . . .	17
3.6	Podukład enabler . . . . .	19
<b>4</b>	<b>Przykład implementacji układu w obwodzie</b>	<b>19</b>
<b>5</b>	<b>Testy</b>	<b>20</b>
5.1	Testy podukładów . . . . .	20
5.2	Test timera . . . . .	20
<b>6</b>	<b>Zastosowania</b>	<b>23</b>
<b>7</b>	<b>Wnioski</b>	<b>23</b>

## 1 Cel zadania

---

Korzystając wyłącznie z wybranych przerzutników oraz dowolnych bramek logicznych, proszę zaprojektować czterobitowy układ TIMER, odmierzający ustawiany za pomocą przełączników czas (od 0 do 15).

Po wciśnięciu przycisku STRAT, układ rozpoczyna odmierzanie czasu do tyłu (proszę dobrać częstotliwość tak, aby efekt był dobrze widoczny na ekranie).

Po wyzerowaniu się licznika czasu, układ powinien się zatrzymać i włączyć alarm świetlny wykorzystujący diodę LED. Po ponownym wciśnięciu przycisku START, układ powinien wyłączyć alarm i ponownie rozpocząć odmierzanie ustawionego na przełącznikach czasu. Aktualny wskazywany przez układ czas proszę pokazywać na wyświetlaczach siedmiosegmentowych.

## 2 Idea rozwiązania

---

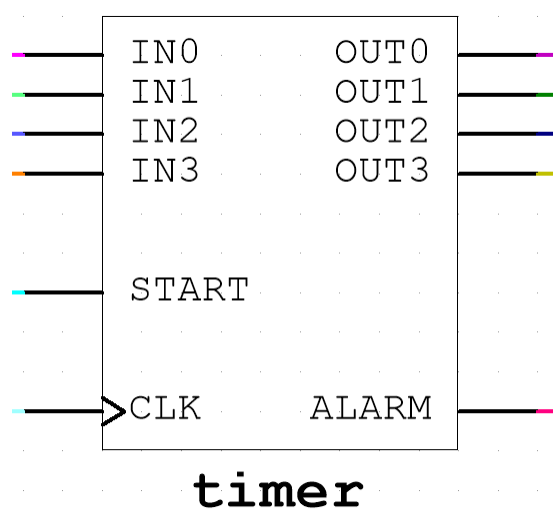
Do rozwiązania zadania wybraliśmy przerzutniki T - z powodu łatwości sterowania takim układem. Do sterowania przerzutnikami w trybie synchronicznym wykorzystaliśmy transkoder uruchamiający wejścia T kolejnych przerzutników na bazie obecnego stanu wyjścia układu. Dodatkowo dodaliśmy pojedynczy sygnał kontrolujący włączenie/wyłączenie układu.

Do początkowego zaprogramowania czasu odliczania na liczniku wykorzystaliśmy możliwość asynchronicznego ustawienia przerzutników w konkretny stan, również tutaj zaprojektowaliśmy transkoder, który porównuje stan przerzutników z wejściem do programowania czasu i odpowiednio ustawia układ.

### 3 Układ timer

#### 3.1 Black box

Pierwszym krokiem w projektowaniu układu było zaprojektowanie czarnej skrzynki i określenie wejść i wyjść układu.



Rysunek 3.1: Czarna skrzynka timera

Poniżej przedstawimy specyfikację wejść i wyjść układu

##### 3.1.1 Wejścia

- **IN<sub>x</sub>** - wejścia programujące czas odliczania licznika - binarny zapis liczby od której licznik powinien zacząć odliczać. 4 wejścia łącznie pozwalają na odliczanie w zakresie 0-15. IN0 oznacza najmniej znaczący bit, IN3 oznacza najbardziej znaczący bit. Wejście jest używane do zaprogramowania w momencie gdy na wejściu **START** pojawi się stan wysoki.

Numer bitu	3	2	1	0
Bit	IN3	IN2	IN1	IN0
Mnożnik	$2^3$	$2^2$	$2^1$	$2^0$

Tabela 3.1: Kodowanie pinów wejściowych

- **START** - wejście aktywujące układ. Stan wysoki oznacza aktywację licznika, stan niski oznacza, że licznik dokończy liczenie do wyzerowania licznika.

Jeśli wejście **START** będzie miało stan wysoki w czasie dojścia licznika do zera, na wyjściu **ALARM** pojawi się puls, po czym licznik zostanie zaprogramowany obecnym wejściem, a następnie uruchomiony ponownie.

Zmiany stanu na wejściu **START** w czasie gdy licznik jest w stanie liczenia, nie mają żadnego efektu.

- **CLK** - wejście zegara stanowiącego podstawę czasu licznika - określa jak szybko następować będą zmiany wyjścia i odliczanie licznika do zera.

### 3.1.2 Wyjścia

- **OUTx** - wyjścia stanowiące kolejne bity aktualnego stanu licznika. Zmiana wartości licznika, następuje na wznoszącym zboczu zegara wejściowego. **OUT0** stanowi najmniej znaczący bit, a **OUT3** najbardziej znaczący bit.

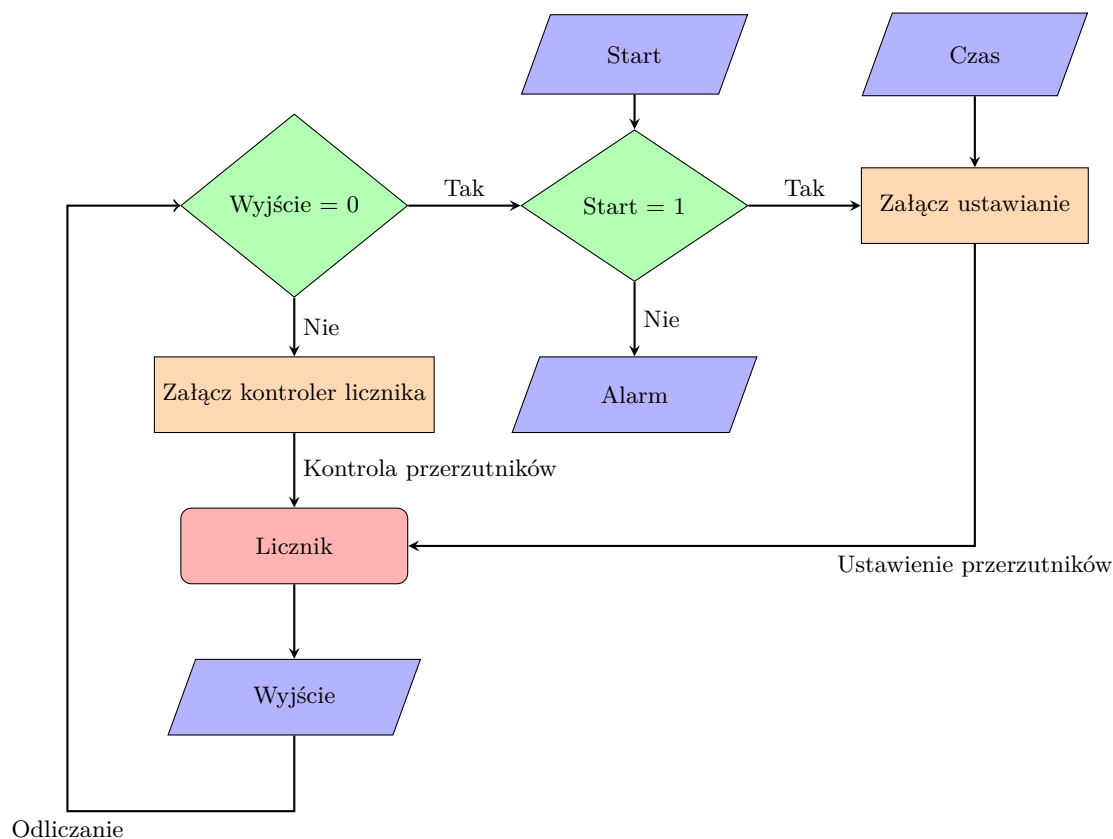
Numer bitu	3	2	1	0
Bit	<b>OUT3</b>	<b>OUT2</b>	<b>OUT1</b>	<b>OUT0</b>
Mnożnik	$2^3$	$2^2$	$2^1$	$2^0$

Tabela 3.2: Kodowanie pinów wejściowych

- **ALARM** - wyjście sygnalizujące zakończenie odliczania licznika. Stan wysoki oznacza, że obecny stan licznika jest równy 0.

### 3.2 Diagram załączania układów

Poniżej rozpisaliśmy diagram zależności stanu załączenia poszczególnych układów od siebie z którego wynikać będą tabele prawdy.



Rysunek 3.2: Diagram załączania układów

### 3.3 Kontrola działania licznika

Na najwyższym poziomie nasz układ `timer` składa się z dwóch podukładów: `timer_setter` - układu ustawiającego czas odliczania oraz `timer_driver` układu kontrolującego wejścia T przerzutników. Na tym samym poziomie znajdują się przerzutniki stanowiące faktyczny licznik oraz implementacja formuł załączających te układy opisanych tutaj.

Dla czytelności poniżej przyjmujemy następujące oznaczenia:

- `EN_SET` - wejście aktywujące w układzie `timer_setter`
- `EN_DRV` - wejście aktywujące w układzie `timer_driver`
- `EQ0` - wyjście układu `timer_driver` mówiące o tym czy obecny stan licznika to 0 (stan wysoki).
- `START` - wejście startowe timera

### 3.3.1 Tablice prawdy

Tabela prawdy wynika z schematu kontroli przedstawionego powyżej.

Wejście		Wyjście	
EQ0	START	EN_SET	EN_DRV
0	0	0	1
0	1	0	1
1	0	0	0
1	1	1	0

Tabela 3.3: Tabela prawdy dla stanów aktywacji podukładów

### 3.3.2 Wyprowadzenie formuł

Dla wyjścia EN\_SET możemy odczytać formułę wprost z tabeli:

$$\text{EN\_SET} = \text{EQ0} \cdot \text{START}$$

Dla wyjścia EN\_DRV możemy pokusić się o próbę optymalizacji formuły przy pomocy tablicy Karnaugh:

		START	
		0	1
EQ0	0	1	1
	1	0	0

Tabela 3.4: Tablica Karnaugh dla formuły aktywującej układ kontrolujący licznik

Możemy z niej odczytać zoptymalizowaną formułę:

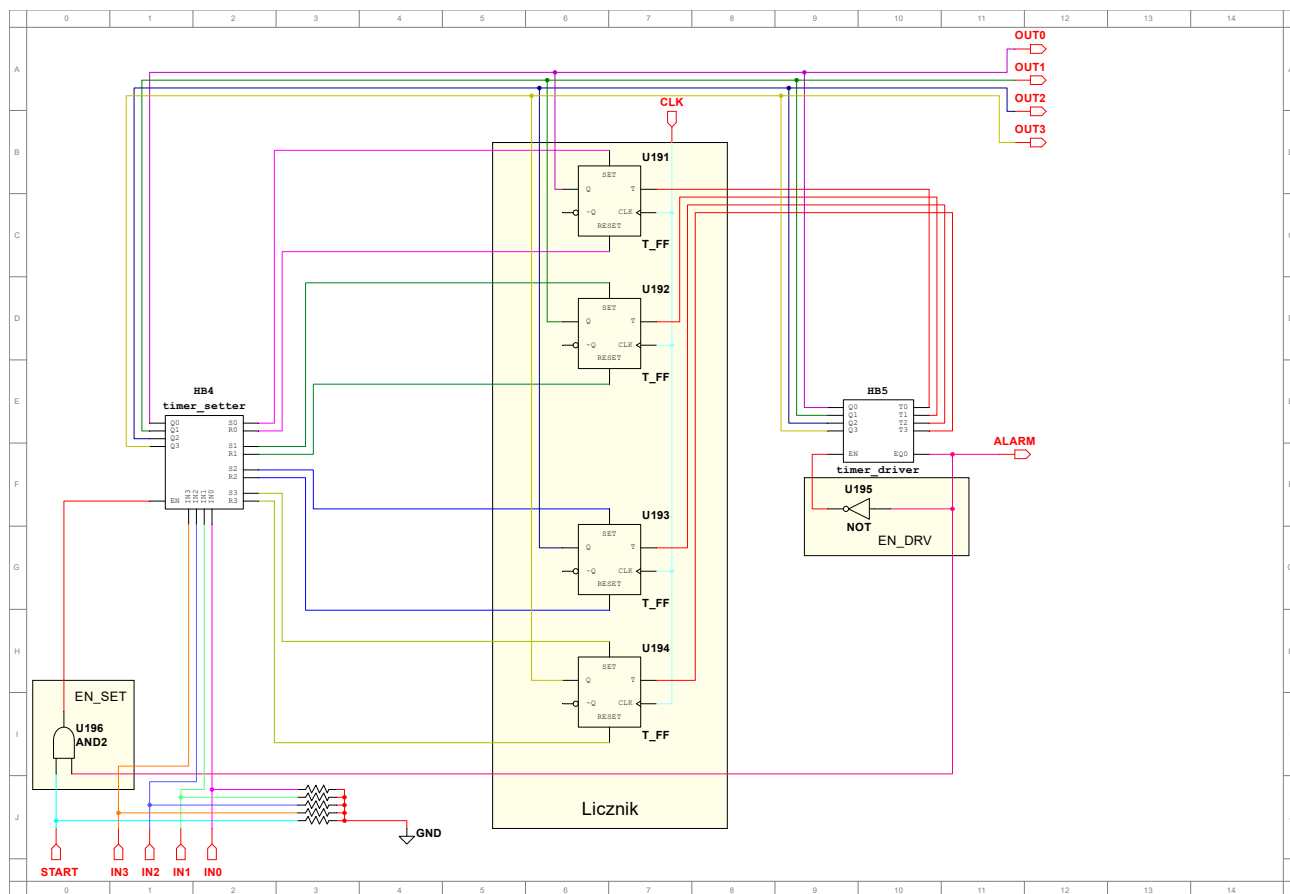
$$\text{EN\_DRV} = \overline{\text{EQ0}}$$

### 3.3.3 Realizacja formuł

Poniżej przedstawiamy realizację wcześniej wyprowadzonych formuł:

$$\text{EN\_SET} = \text{EQ0} \cdot \text{START}$$

$$\text{EN\_DRV} = \overline{\text{EQ0}}$$



Rysunek 3.3: Ogólny schemat timera

Na schemacie znajdują się również rezystory pull-down zabezpieczające wejścia układu przed nieokreślonym stanem wejść.

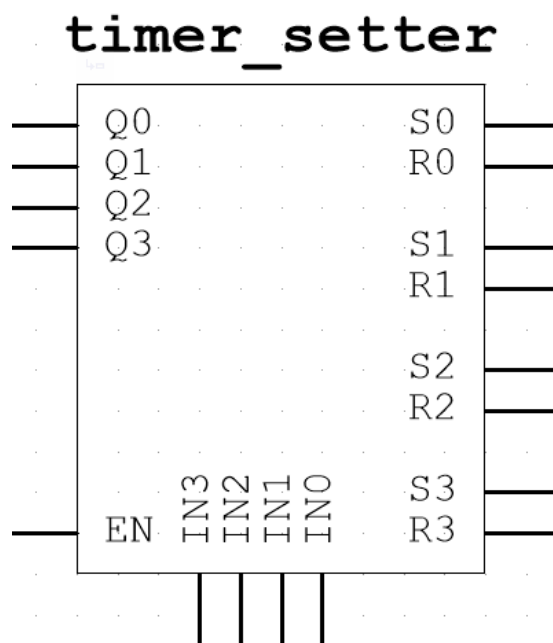


### 3.4 Podukład ustawiający czas licznika

Układ nazwany na naszych schematach `timer_setter` ustawia czas odliczania gdy układ zostanie załączony. Układ ma na celu wysterowanie asynchronicznych wejść przerzutników T poprzez odpowiednie wykonanie operacji SET lub RESET w zależności od obecnego stanu przerzutnika w porównaniu do odpowiadającego bitu programowania.

#### 3.4.1 Wejścia i Wyjścia

Wejścia do układu stanowią bity oznaczające obecny stan poszczególnych wyjść przerzutnika, bity oznaczające stan wejścia programowania timera oraz sygnał załączający układ. Wyjścia natomiast stanowią pary pinów SET i RESET dla poszczególnych przerzutników.



Rysunek 3.4: Czarna skrzynka podukładu `time_setter`

Poniżej opis wejść układu:

- `EN` - wejście aktywujące układ, gdy wejście jest w stanie wysokim, na wyjściach układu pojawiają się odpowiednie wartości
- `Qx` - wejścia obecnego stanu licznika, `Q0` stanowi najmniej znaczący bit obecnego stanu licznika, a `Q3` najbardziej znaczący bit.
- `INx` - wejścia programowania startowego stanu licznika, `IN0` stanowi najmniej znaczący bit wejścia, a `IN3` najbardziej znaczący bit.

Poniżej opis wyjść układu:

- **Sx** - wyjście SET ustawiające odpowiedni przerzutnik T, wartość S0 obliczana jest na podstawie wejść Q0 i IN0, a więc odpowiada ustawieniu przerzutnika T odpowiadającemu najmniej znaczącemu bitowi licznika.
- **Rx** - wyjście RESET resetujący odpowiedni przerzutnik T, wartość R0 obliczana jest na podstawie wejść Q0 i IN0, a więc odpowiada resetowaniu przerzutnika T odpowiadającemu najmniej znaczącemu bitowi licznika.

### 3.4.2 Tabele prawdy

Jako że, układ oblicza każdą parę wyjść dokładnie tak samo na podstawie odpowiadających sobie bitów, tabelę prawdy zapiszemy w postaci sparametryzowanej, tzn. parze wyjściowej **Sx**, **Rx** odpowiadają wejścia **INx**, **Qx** oraz sygnał enable. Finalnie ostateczny układ stanowią 4 powtórzone takie formuły dla każdego z bitów 0,1,2,3.

Tabela prawdy wynika z następujących faktów:

- jeśli  $EN = 0$  to żadne wyjście nie jest aktywne
- jeśli  $INx = Qx$  to nie musimy zmieniać stanu przerzutnika
- w pozostałych przypadkach wykonujemy odpowiednio albo operację SET albo RESET

Wejście			Wyjście	
EN	INx	Qx	Sx	Rx
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	1
1	1	0	1	0
1	1	1	0	0

Tabela 3.5: Tabela prawdy dla układu programującego początkowy stan licznika

### 3.4.3 Wyprowadzenie formuł

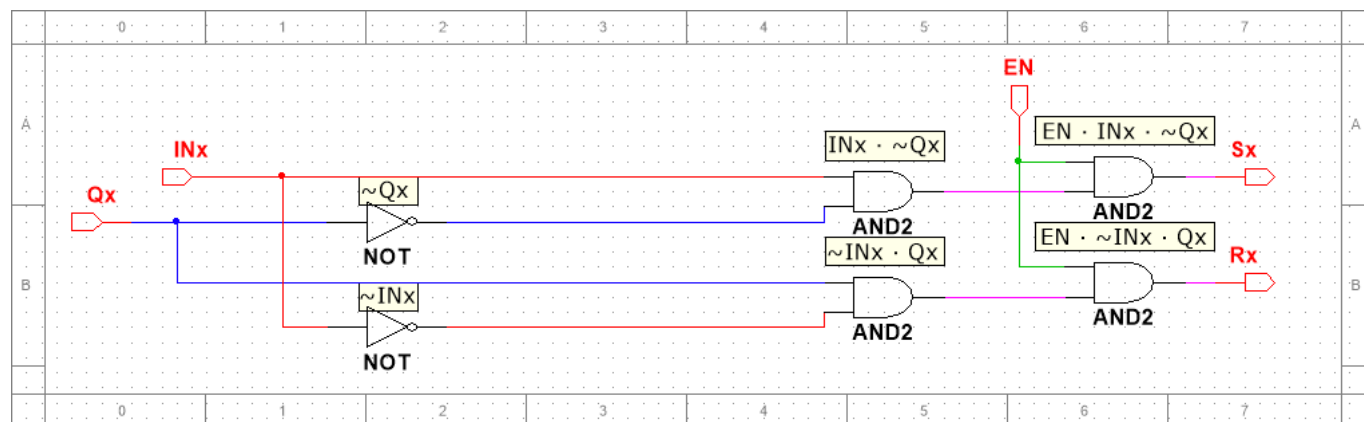
Na podstawie tabeli prawd możemy wyprowadzić formułę na wyjścia **Sx** i **Rx**

$$Sx = EN \cdot INx \cdot \overline{Qx}$$

$$Rx = EN \cdot \overline{INx} \cdot Qx$$

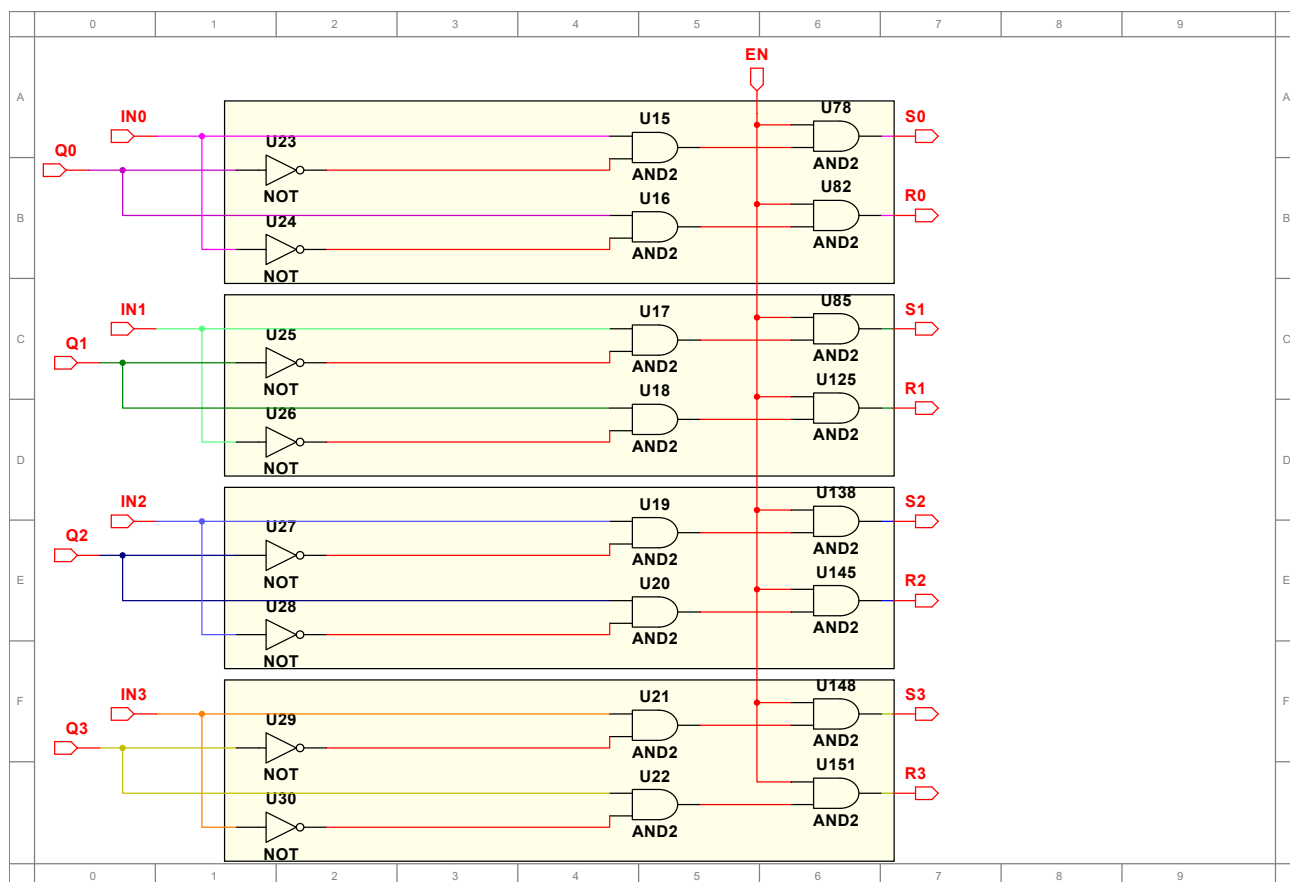
## 3.4.4 Realizacja formuł

Wyżej wymienione formuły można w multisimie przedstawić w następujący sposób:



Rysunek 3.5: Realizacja funkcji logicznych w Multisimie

Układ timer\_setter został zaimplementowany jako czterokrotne powielenie powyższej struktury.



Rysunek 3.6: Schemat podukładu: timer\_setter

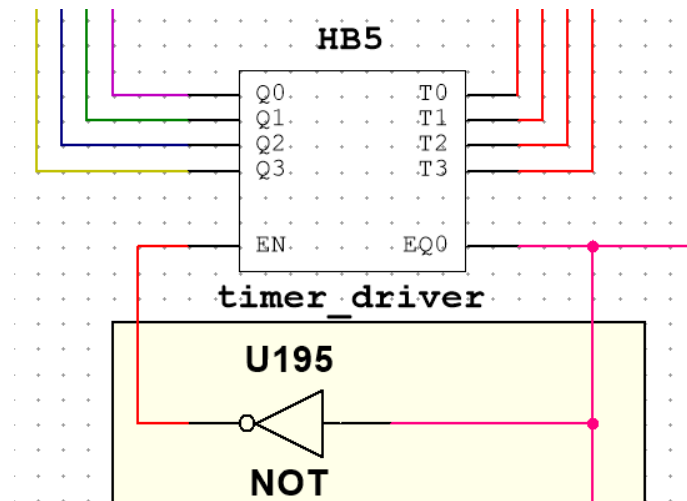
### 3.5 Podukład kontrolujący przerzutniki

Układ nazwany na naszych schematach `timer_driver` kontroluje wejścia  $T$  przerzutników.

#### 3.5.1 Wejścia i Wyjścia

Poniżej znajduje się opis wejść układu:

- EN - Wejście aktywujące układ. Pozwala, aby na wyjściu pojawiały się odpowiednie wartości kiedy jest w stanie wysokim
- Qx - wejścia obecnego stanu licznika, Q0 stanowi najmniej znaczący bit obecnego stanu licznika, a Q3 najbardziej znaczący bit.
- EQ0 - Wyjście informujące o tym czy obecny stan licznika jest równy 0.
- Tx - wyjścia przerzutnika, T0 stanowi najmniej znaczący bit przerzutnika, a T3 najbardziej znaczący bit.



Rysunek 3.7: Czarna skrzynka podukładu: `time_driver`

#### 3.5.2 Tabele prawdy

Tabela prawdy została skonstruowana na podstawie formuły

$$Tx = \text{XOR}(Q_{n_x}, Q_{n_x+1})$$

, gdzie  $Q_{n_x}$  oznacza stan obecny, a  $Q_{n_x+1}$  oznacza stan następny

	Stan Obecny				Stan Następny				Przerzutniki			
	$Q_{n_3}$	$Q_{n_2}$	$Q_{n_1}$	$Q_{n_0}$	$Q_{n_3+1}$	$Q_{n_2+1}$	$Q_{n_1+1}$	$Q_{n_0+1}$	$T_3$	$T_2$	$T_1$	$T_0$
15	1	1	1	1	1	1	1	0	0	0	0	1
14	1	1	1	0	1	1	0	1	0	0	1	1
13	1	1	0	1	1	1	0	0	0	0	0	1
12	1	1	0	0	1	1	0	1	0	1	1	1
11	1	1	0	1	1	0	1	0	0	0	0	1
10	1	0	1	0	1	0	0	1	0	0	1	1
9	1	0	0	1	1	0	0	0	0	0	0	1
8	1	0	0	0	0	1	1	1	1	1	1	1
7	0	1	1	1	0	1	1	0	0	0	0	1
6	0	1	1	0	0	1	0	1	0	0	1	1
5	0	1	0	1	0	1	0	0	0	0	0	1
4	0	1	0	0	0	0	1	1	0	1	1	1
3	0	0	1	1	0	0	1	0	0	0	0	1
2	0	0	1	0	0	0	0	1	0	0	1	1
1	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	x	x	x	x	x	x	x	x

Tabela 3.6: Tabela prawdy z uwzględnieniem stanu następnego

Wtedy tabela prawdy prezentuje się następująco:

Wejście					Wyjście				
EN	Q3	Q2	Q1	Q0	T3	T2	T1	T0	EQ0
1	1	1	1	1	0	0	0	1	0
1	1	1	1	0	0	0	1	1	0
1	1	1	0	1	0	0	0	1	0
1	1	1	0	0	0	1	1	1	0
1	1	1	0	1	0	0	0	1	0
1	1	0	1	0	0	0	1	1	0
1	1	0	0	1	0	0	0	1	0
1	1	0	0	0	1	1	1	1	0
1	0	1	1	1	0	0	0	1	0
1	0	1	1	0	0	0	1	1	0
1	0	1	0	1	0	0	0	1	0
1	0	1	0	0	0	1	1	1	0
1	0	0	1	1	0	0	0	1	0
1	0	0	1	0	0	0	1	1	0
1	0	0	0	1	0	0	0	1	0
1	0	0	0	0	1	1	1	1	1
0	1	1	1	1	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0
0	1	1	0	1	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0
0	1	1	0	1	0	0	0	0	0
0	1	0	1	0	0	0	0	0	0
0	1	0	0	1	0	0	0	1	0
0	1	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1

Tabela 3.7: Tablica prawdy dla podukładu: timer\_driver

## 3.5.3 Wyprowadzenie formuł

Na podstawie poniższej tabeli wyprowadzone zostały Tablice Karnaugh

	Stan Obecny				Przerzutniki			
	$Q_{n_3}$	$Q_{n_2}$	$Q_{n_1}$	$Q_{n_0}$	$T_3$	$T_2$	$T_1$	$T_0$
15	1	1	1	1	0	0	0	1
14	1	1	1	0	0	0	1	1
13	1	1	0	1	0	0	0	1
12	1	1	0	0	0	1	1	1
11	1	1	0	1	0	0	0	1
10	1	0	1	0	0	0	1	1
9	1	0	0	1	0	0	0	1
8	1	0	0	0	1	1	1	1
7	0	1	1	1	0	0	0	1
6	0	1	1	0	0	0	1	1
5	0	1	0	1	0	0	0	1
4	0	1	0	0	0	1	1	1
3	0	0	1	1	0	0	0	1
2	0	0	1	0	0	0	1	1
1	0	0	0	1	0	0	0	1
0	0	0	0	0	x	x	x	x

Tabela 3.8: Tabela prawdy z wyłączeniem pinów EN i EQ0

		$Q_1Q_0$			
		00	01	11	10
$Q_3Q_2$	00	x	0	0	0
	01	0	0	0	0
	11	0	0	0	0
	10	1	0	0	0

Tabela 3.9: Tabela Karnaugh dla przerzutnika  $T_3$ 

Możemy z niej odczytać zoptymalizowaną formułę:

$$T_3 = \overline{Q_2Q_1Q_0}$$

		$Q_1Q_0$			
		00	01	11	10
$Q_3Q_2$	00	x	0	0	0
	01	1	0	0	0
	11	1	0	0	0
	10	1	0	0	0

Tabela 3.10: Tabela Karnaugh dla przerzutnika  $T_2$ 

Możemy z niej odczytać zoptymalizowaną formułę:

$$T_2 = \overline{Q_1}Q_0$$

		$Q_1Q_0$			
		00	01	11	10
$Q_3Q_2$	00	x	0	0	1
	01	1	0	0	1
	11	1	0	0	1
	10	1	0	0	1

Tabela 3.11: Tabela Karnaugh dla przerzutnika  $T_1$ 

Możemy z niej odczytać zoptymalizowaną formułę:

$$T_1 = \overline{Q_0}$$



		$Q_1Q_0$			
		00	01	11	10
$Q_3Q_2$	00	x	1	1	1
	01	1	1	1	1
	11	1	1	1	1
	10	1	1	1	1

Tabela 3.12: Tabela Karnaugh dla przerzutnika  $T_0$ 

Możemy z niej odczytać zoptymalizowaną formułę:

$$T_0 = 1$$

Oprócz bitów licznika na wyjście ma także wpływ EN, czyli bit aktywujący. Aby poprawnie zaimplementować formuły potrzebujemy go uwzględnić

$$T_0 = \text{EN}$$

$$T_1 = \overline{Q_0}\text{EN}$$

$$T_2 = \overline{Q_1Q_0}\text{EN}$$

$$T_3 = \overline{Q_2Q_1Q_0}\text{EN}$$

EQ0 ma za zadanie informować kiedy wszystkie bity stanu obecnego licznika  $Qx$  będą równe 0, więc formuła będzie wyglądać następująco:

$$\text{EQ0} = \overline{Q_0Q_1Q_2Q_3}$$

### 3.5.4 Realizacja formuł

Układ został stworzony na podstawie poniższych formuł

$$T_0 = \text{EN}$$

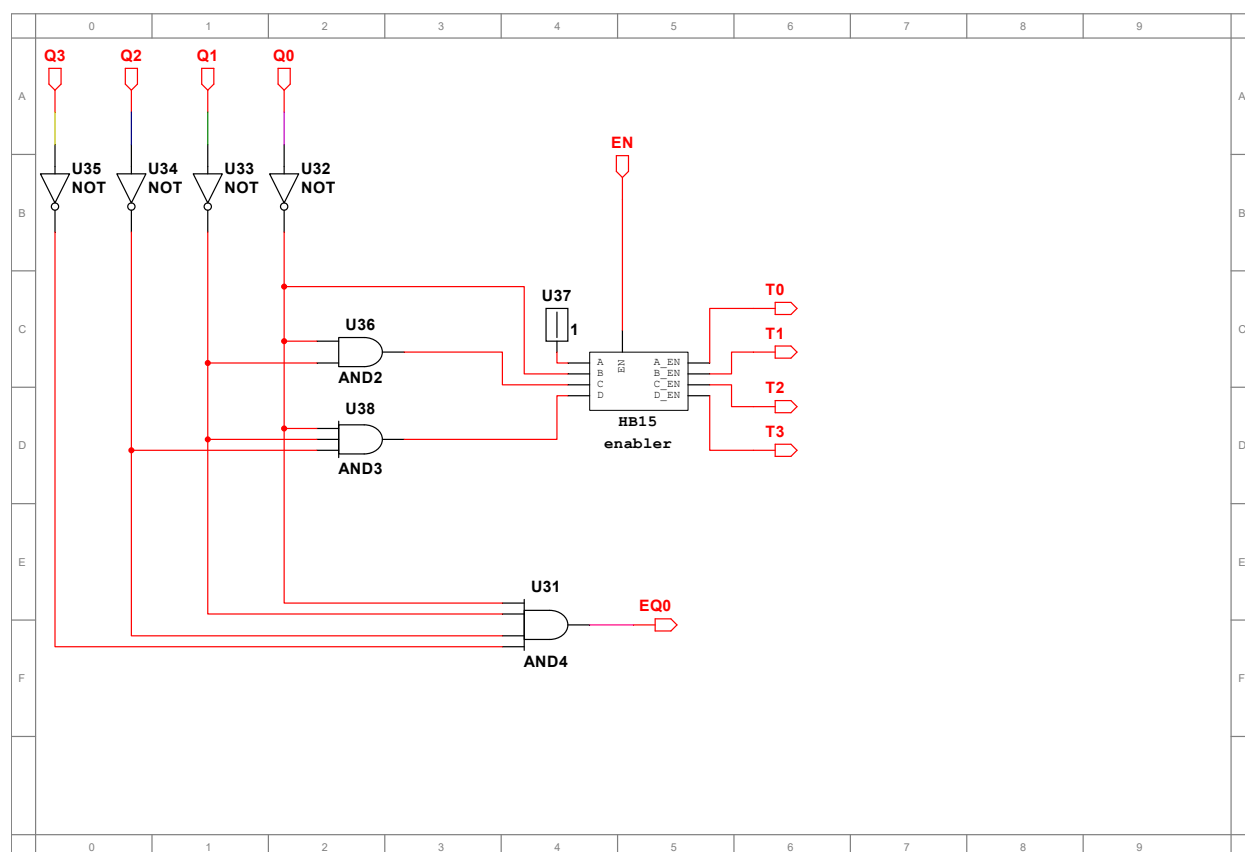
$$T_1 = \overline{Q_0}\text{EN}$$

$$T_2 = \overline{Q_1Q_0}\text{EN}$$

$$T_3 = \overline{Q_2Q_1Q_0}\text{EN}$$

$$\text{EQ0} = \overline{Q_0Q_1Q_2Q_3}$$

Dla czytelności przeprowadzanie operacji AND z każdym z bitów zostało zaimplementowane w podukładzie enabler, który szczegółowo opisany jest w kolejnym paragrafie.



Rysunek 3.8: Schemat podukładu: time\_driver

### 3.6 Podukład enabler

## 4 Przykład implementacji układu w obwodzie

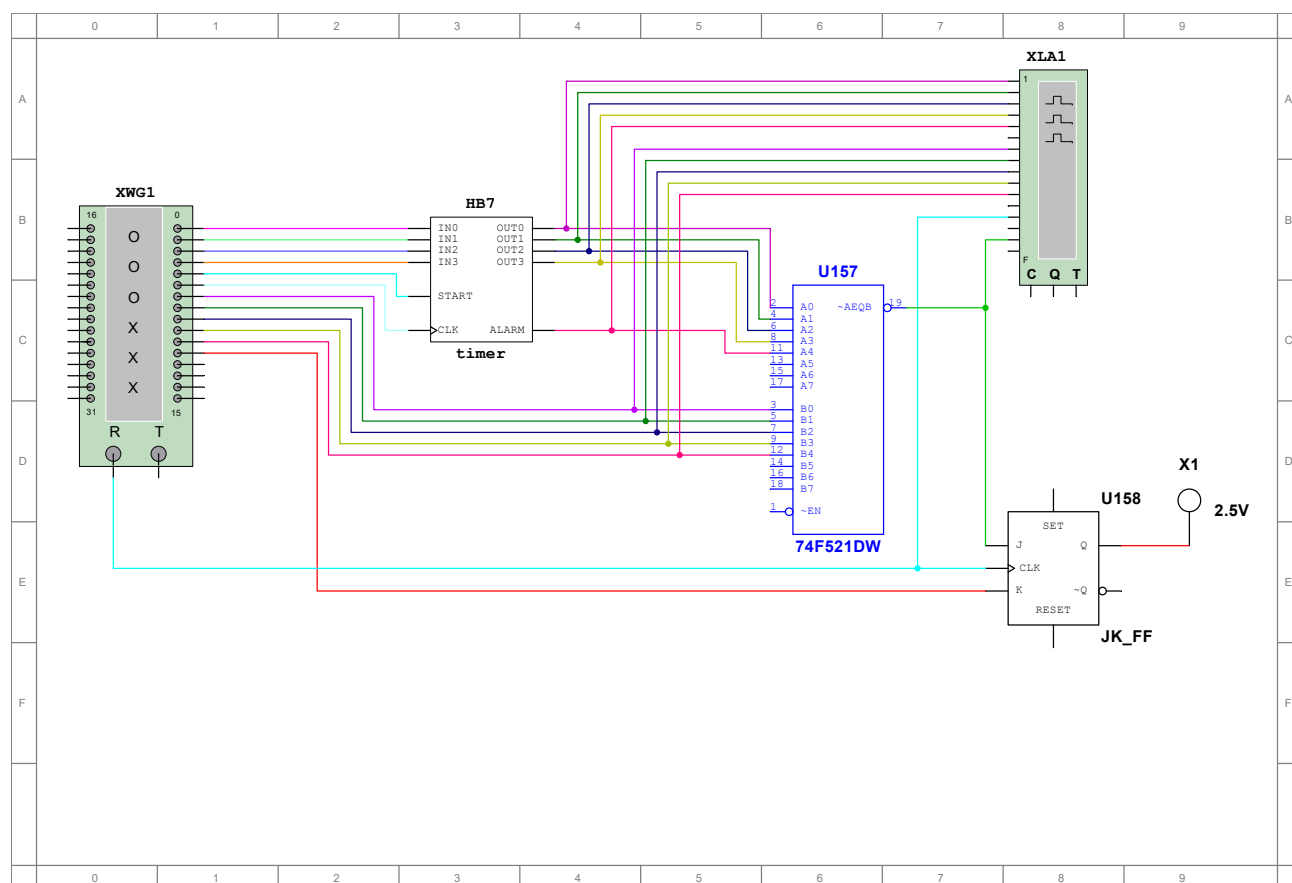
---

## 5 Testy

### 5.1 Testy podukładów

### 5.2 Test timera

Zdecydowaliśmy się przeprowadzić ogólny test układu timer - na najwyższym poziomie abstrakcji. W tym celu zestawiliśmy układ składający się z generatora słów, naszego układu `timer`, komparatora oraz analizatora stanów logicznych. Generator słów obsługuje nasz timer zapewniając mu cykle zegara w kolejnych generowanych słowach oraz generujący dane porównawcze dla każdego cyklu.



Rysunek 5.1: Schemat układu testującego

Zastosowaliśmy przerzutnik JK próbujący wyjście komparatora, gdy generator słów generuje wyjście R(READY) oznaczające, że skończył on generować dane słowo. Pojawienie się stanu wysokiego na wejściu J przerzutnika JK powoduje jego ustawienie, a pojawienie się stanu wysokiego na wyjściu K resetuje ten przerzutnik.

Do wygenerowania pliku wejściowego dla generatora słów, napisaliśmy skrypt w języku python, który symuluje kolejne kroki odliczania timera w zmiennych i zapisuje

je do pliku w formacie, który można zaimportować do multisima.  
Poniżej kod skryptu generującego dane:

```

1  f = open("test_data.dp", "w")
2  f.write("Data:\n")
3
4  #####
5  # |SR_RESET|ALARM|OUT3|OUT2|OUT1|OUT0|CLK|START|IN3|IN2|IN1|IN0| #
6  #####
7
8  class TestOutput:
9      def __init__(self):
10         self.in_data = 0
11         self.start = 0
12         self.clk = 0
13         self.output_data = 0
14         self.alarm = 0
15         self.reset_sr = 0
16
17     def to_bin_string(self):
18         input_binary = str(bin(self.in_data)).removeprefix("0b").rjust(4, '0')
19         start_binary = str(bin(self.start)).removeprefix("0b")
20         clk_binary = str(bin(self.clk)).removeprefix("0b")
21         output_binary = str(bin(self.output_data)).removeprefix("0b").rjust(4, '0')
22         alarm_binary = str(bin(self.alarm)).removeprefix("0b")
23         reset_sr_binary = str(bin(self.reset_sr)).removeprefix("0b")
24
25         return reset_sr_binary + alarm_binary + output_binary + clk_binary + start_binary + input_binary
26
27
28     def to_hex_string(self, pad):
29         hex_val = hex(int(self.to_bin_string(), 2))
30         return hex_val.removeprefix("0x").rjust(pad, '0')
31
32
33     #####
34     # Test cycle to reset JK flip flop #
35     #####
36
37     reset_to = TestOutput()
38     reset_to.alarm = 1
39     reset_to.reset_sr = 1
40     f.write(reset_to.to_hex_string(8) + "\n")
41     f.write(reset_to.to_hex_string(8) + "\n")
42     reset_to.reset_sr = 0
43     f.write(reset_to.to_hex_string(8) + "\n")
44     data_count = 3
45
46     for i in range(16):
47         to = TestOutput()
48         to.in_data = i
49         to.start = 1
50         to.output_data = i
51         to.alarm = int(i == 0)
52
53         f.write(to.to_hex_string(8) + "\n")
54         data_count += 1
55
56         to.in_data = 0
57         to.start = 0
58         to.alarm = int(i == 0)

```

```

59
60     for k in range(i):
61         to.clk = 1
62         to.output_data -= 1
63         if to.output_data == 0:
64             to.alarm = 1
65
66         f.write(to.to_hex_string(8) + "\n")
67         data_count += 1
68
69         to.clk = 0
70
71         f.write(to.to_hex_string(8) + "\n")
72         data_count += 1
73
74     f.write(to.to_hex_string(8) + "\n")
75     data_count += 1
76
77 f.write("Initial:\n")
78 f.write("0000\n")
79 f.write("Final:\n")
80 f.write(str(hex(data_count)).capitalize().removeprefix("0x").rjust(4, '0'))
81
82 f.close()

```

Rysunek 5.2: Skrypt generujący dane do testów, napisany w języku Python

W wyniku testu otrzymujemy informację poprzez zaświecenie się lampki jeśli gdziekolwiek wystąpił błąd oraz przebieg sygnałów z analizatora stanów logicznych:



Rysunek 5.3: Przebieg sygnałów w analizatorze stanów logicznych

## 6 Zastosowania

---

- Sprzętowa obsługa timerów na mikroprocesorach, wraz z mechanizmem sprzętowych przerwań stanowią przydatny mechanizm pozwalający na cykliczne wywoływanie zadanego kodu bez konieczności zużywania czasu procesora na kontrolę upływu czasu. Mechanizm taki ma zastosowanie od wspomnianego cyklicznego wykonywania kodu do sprzężenia ze sobą różnych peryferiów mikroprocesora, pozwalające na automatyczne dokonywanie cyklicznych pomiarów czy generowanie sygnałów, jak np. sygnał Alarm timera podpięty jako sygnał wyzwalający konwersję przetwornika ADC, który odczyt zapisuje bezpośrednio do pamięci poprzez mechanizm DMA czy timer generujący sygnał PWM przez porównanie obecnego stanu licznika z zadaną wartością - ponownie w sprzężeniu z mechanizmem DMA pozwalającym odczytywać kolejne wartości do porównania w każdym okresie timera bezpośrednio z pamięci można nadawać praktycznie dowolny sygnał cyfrowy.

## 7 Wnioski

---

Dzięki przerzutnikom możemy projektować złożone układy z pamięcią swojego stanu.

W toku rozwiązywania postawionego problemu opracowaliśmy kilka możliwych rozwiązań. Ostatecznie zdecydowaliśmy się na przedstawienie tego konkretnego rozwiązania ze względu na jego prostotę i największą niezawodność. Zastosowanie przerzutników działających synchronicznie pozwoliło na zredukowanie ilości hazardów przy odliczaniu czasu.

Innym sposobem było: <tu opis rozwiązanie Kacpra i schemat>

Pierwotnym pomysłem było użycie przerzutników T asynchronicznie - łącząc je w szeregowo dzięki czemu nie było konieczne projektowanie układu kontrolującego przerzutniki. Największą wadą tego rozwiązania było to, że otoczka sterowania tak połączonych przerzutników była trudna do przedstawienia wzorami i uzasadnienia w prosty sposób. Rozwiązanie opierało się na idei generowania kolejnych impulsów synchronizujących kolejne etapy przygotowania i uruchomienia układu.

<tu mój schemat>