LABORATORIUM
PROGRAMOWALNYCH UKŁADÓW
LOGICZNYCH

Ćwiczenie 2
Kontroler wyświetlacza 7-segmentowego

W12IEA-SI0038P

W12EIT-SI0106P

wzn.pwr.edu.pl/materialy-dydaktyczne/PUL

1. CEL ĆWICZENIA

Doskonalenie projektowania układów cyforwych wykorzystujących hierarchiczną budowę na przykładzie kontrolera wyświetlacza 7-segmentowego. Program zajęć obejmuje:

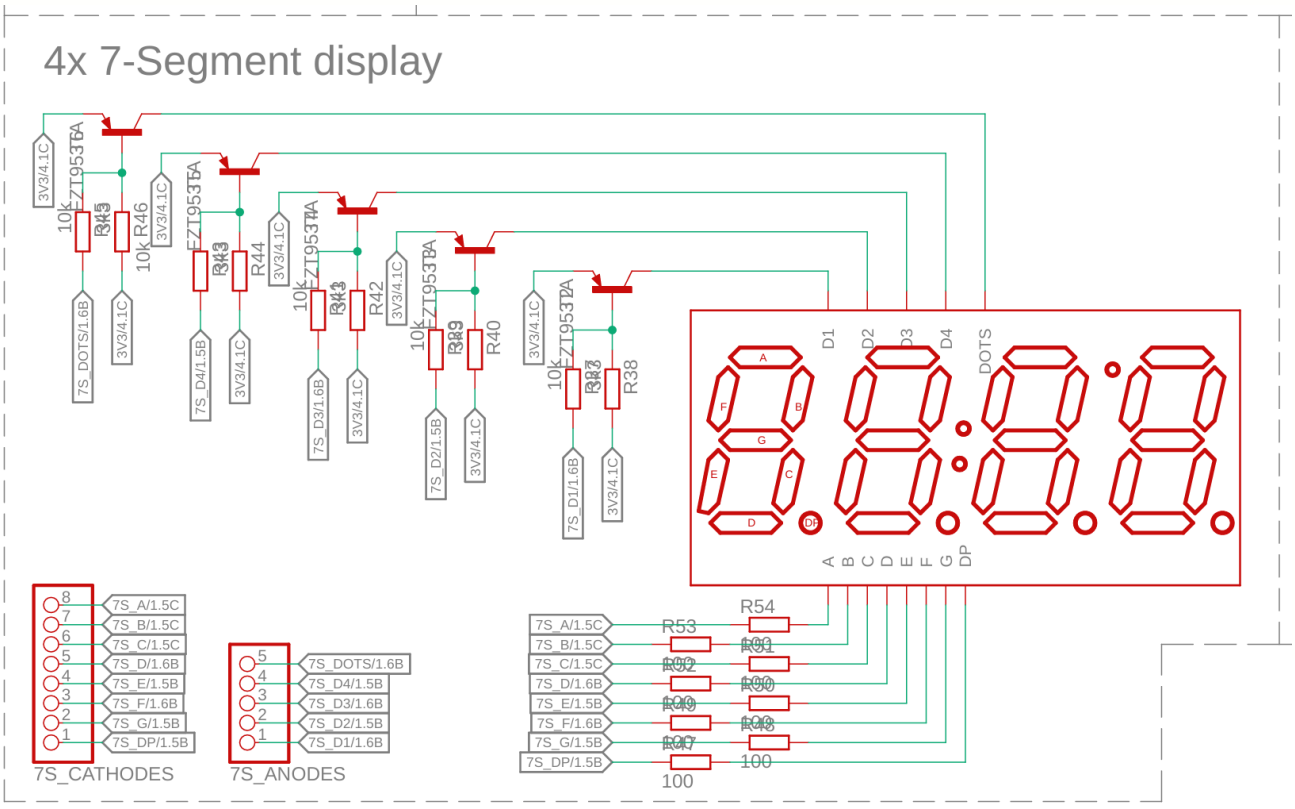
- Podział funkcjonalny układu komunikacji
- Poprawny opis układu w języku VHDL
- Testowanie opisanej struktury w środowisku testowym
- Implementacja układu na makiecie

1.1. WYMAGANIA WSTĘPNE

Obowiązują podstawowe pojęcia związane z projektowaniem układów logicznych. Wymagana umiejętność posługiwania się dokumentacją techniczną w zakresie schematów elektrycznych.

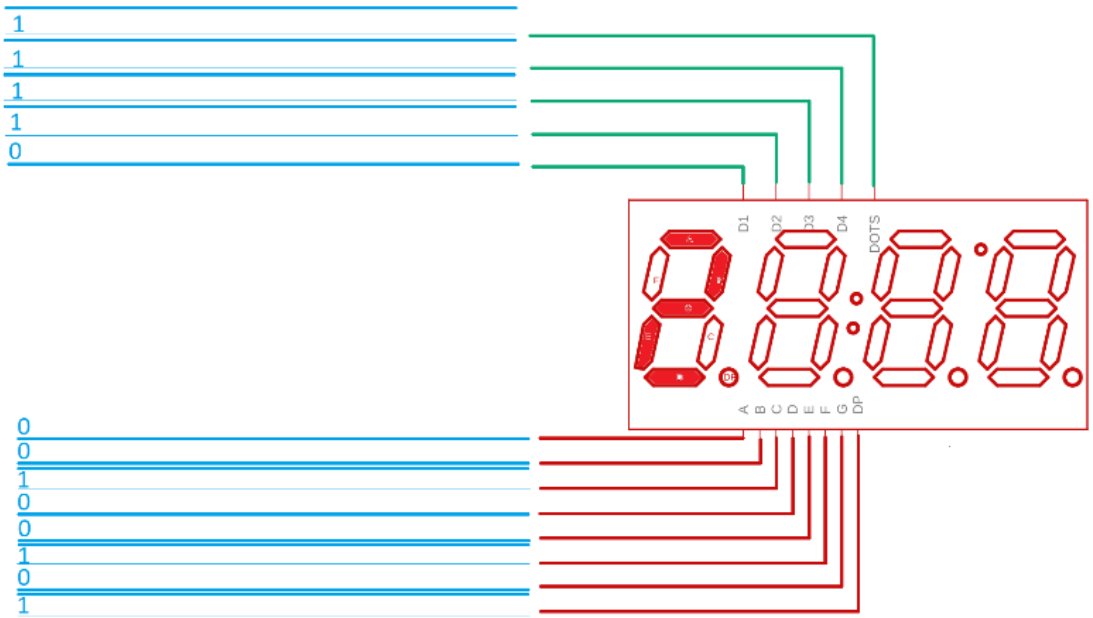
2. OPIS FUNKCJONALNY

Wyświetlacz 7-segmentowy składa się z mozaiki diod elektroluminescencyjnych (LED), ułożonych w segmenty w kształcie, który pozwala na odwzorowanie znaków numerycznych i niektórych liter poprzez wygaszanie i podświetlanie poszczególnych segmentów. Wyświetlacze te są wyposażone w doprowadzenia podłączone wewnątrz do odpowiednich elektrod diod: anod i katod. Sterując przepływem prądu pomiędzy anodą i katodą, podświetlamy segment znaku. Ze względu na dużą liczbę linii wymaganych do podłączenia katod (lub anod, w zależności od rodzaju układu wykonawczego sterownika), katody wszystkich segmentów są ze sobą połączone. O aktywnej, podświetlanej cyfrze decyduje sygnał anody.

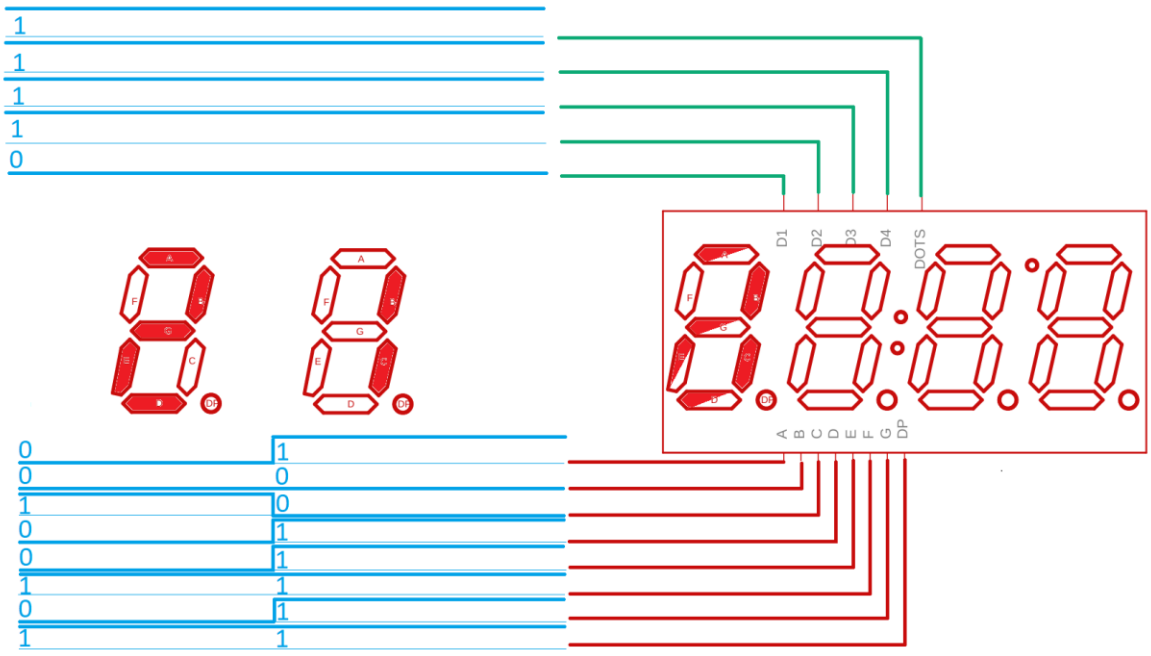


Wyświetlanie znaków na wyświetlaczu 7-segmentowym polega na multipleksowaniu, tj. przełączaniu kolejnych cyfr poprzez przełączanie anod czyli aktywowanie danej cyfry oraz jednoczesną zmianę sygnałów katod. Z punktu widzenia percepcji oka ludzkiego, wykorzystywany jest fakt, że zmieniający się obraz daje człowiekowi wrażenie migotania przy częstotliwości ok. 25-30 Hz. Zatem zmieniając znaki z taką częstotliwością, ludzkie oko będzie je odbierać jako obrazy nieporuszające się. Poniższe przykłady pokazują przykładowe sygnały podawane na wyprowadzenia wyświetlacza i uzyskane wyniki wyświetlania.

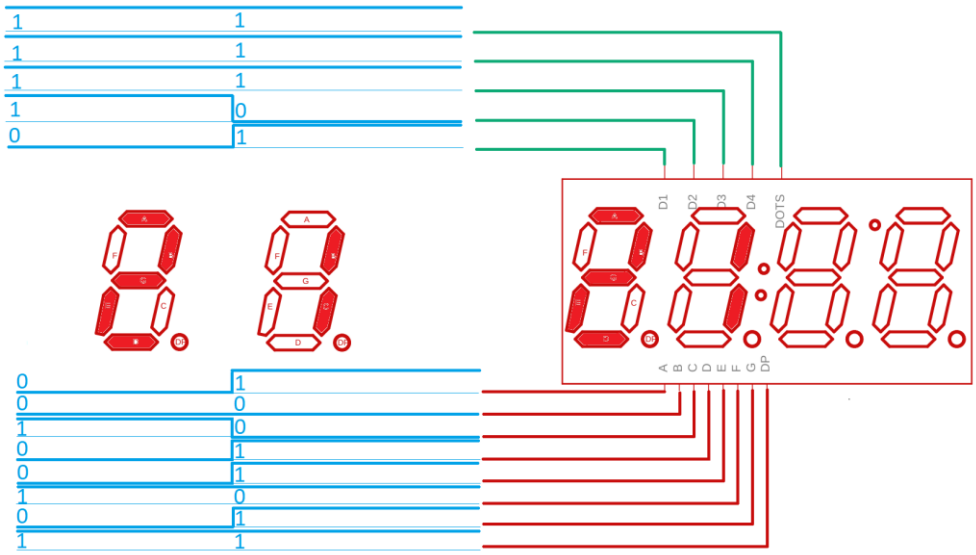
a) Znak nieruchomy wyświetlany na jednej cyfrze wyświetlacza



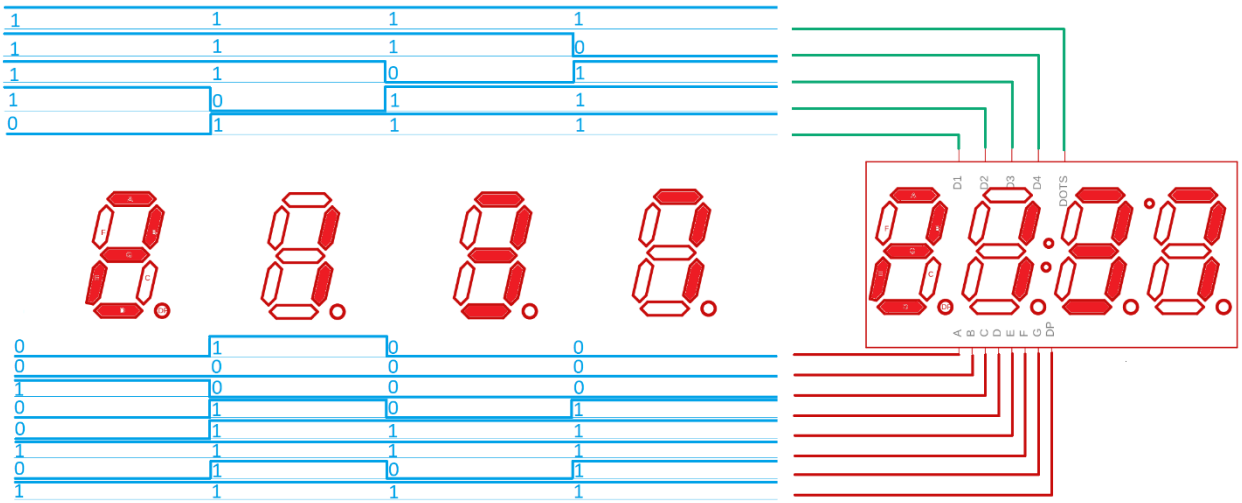
b) Znak zmieniający się, wyświetlany na jednej cyfrze



c) Dwa znaki wyświetlane na dwóch cyfrach

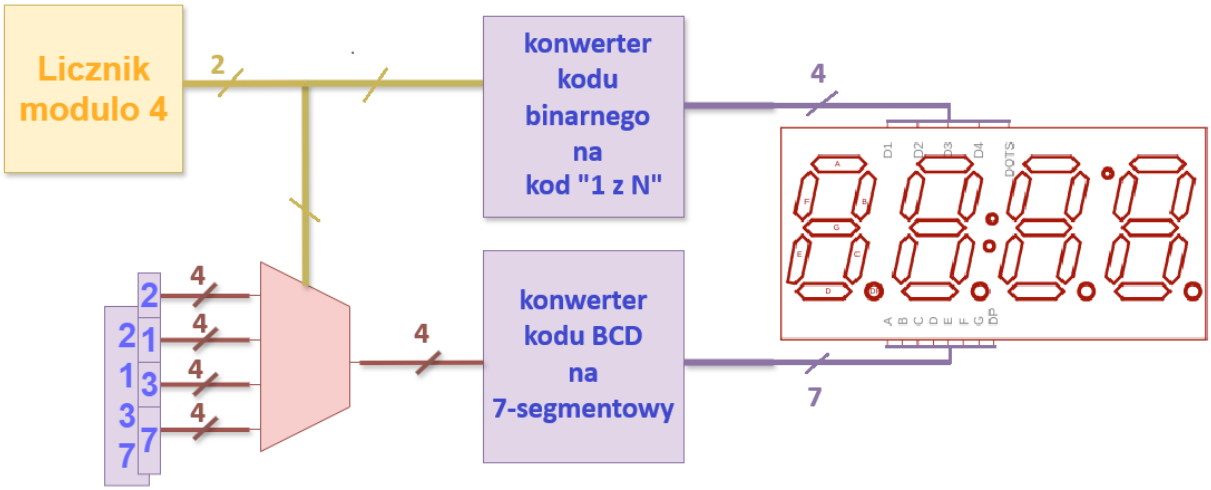


d) Cztery znaki wyświetlane na czterech cyfrach



3. REALIZACJA UKŁADU

Podstawą konwersji cyfry na kod wyświetlacza 7-segmentowego jest kombinacyjny układ dekodera. Ponadto, wykorzystywany jest również demultiplekser, przełączający w każdym z czterech cykli wartość licznika na kod 1 z N (przy czym wartości są zanegowane – sygnał podawany na anody). W tym samym czasie na wejścia konwertera katod podawana jest jedna z cyfr wyświetlanych liczb. Schemat ideowy takiego połączenia dla wszystkich cyfr przedstawia poniższy rysunek.



Przedstawiony układ można utworzyć w podobny, strukturalny sposób. Logika wyświetlacza 7-segmentowego jest jednak na tyle mało skomplikowana, że opis behawioralny konwerterów i multiplekserów pozwala w zwięzły sposób zaimplementować układ, również bez stosowania automatów.

4. ZADANIA DO WYKONANIA

1) Wyświetlić na pierwszej cyfrze wyświetlacza 7-segmentowego naprzemiennie dwa znaki wskazane przez prowadzącego (0-9, A,b,C,d,E,F).
2) Wyświetlić te same znaki na dwóch cyfrach obok siebie.
3) Wyświetlać stan licznika na wszystkich cyfrach (0-9999).

5. „TROUBLESHOOTING" (ROZWIĄZYWANIE PROBLEMÓW)

Nierzadko (a wręcz niezwykle często!) rezultaty implementacji opracowanych układów są niezadowalające. Tzn. kod nie działa na makiecie (czy innym układzie) tak, jak zakładaliśmy. Istnieje kilka metod poprawy tej sytuacji. Po pierwsze, po syntezie powinniśmy zerknąć na wyniki tejże. W oknie *Flow Navigator* Vivado klikamy **SYNTHESIS -> Open Synthesized Design.** Możemy wówczas podejrzeć schemat zsyntezowanej struktury. Następujące rezultaty wskazują na potencjalne problemy:

- Porty wejściowe niewprowadzone do elementów struktury
- Porty wyjściowe podciągnięte do GND (GROUND) lub PWR (POWER)
- Brak spodziewanych elementów struktury (np. zredukowane liczniki lub redukcja przez wykluczające się warunki!)

Protip.

*ZAWSZE uważnie czytajcie wyniki działania zintegrowanego środowiska! Tzn. CZYTAJCIE wyniki syntezy i implementacji: tzw. **logi** – „logs" po angielsku, ostrzeżenia – „warnings" czy **błędy** – „errors" po angielsku*

- Ostrzeżenie zawiera informację „*inferred latch*" – wnioskowany rejestr zatrzaskowy – wróg dobrego przejścia pomiędzy stanami w automacie
- Ostrzeżenie zawiera informację „*signal used in process but not on a sensititvity list*" –brak sygnału na liście wrażliwości procesu kombinacyjnego. Może spowodować „nie zauważenie" pożądanego przez nas warunku

Poprawne wnioskowanie maszyny stanu przez narzędzia syntezy na postawie opisu sprzętu jest sygnalizowane w logu syntezy. (Log nie zawsze jest widoczny. Aby go wywołać, należy kliknąć na pasku menu *Window->Log*). Wówczas, po syntezie jedna z pierwszych linijek *Loga* dotyczy wnioskowanych stanów w naszym automacie. `INFO: [Synth 8-802] inferred FSM for state register 'STAN_OBECNY_reg' in module 'fsm'`.

Jeśli natomiast wydaje nam się, że zrobiliśmy wszystko w porządku, symulacja jest udana (choć symulacja ≠ synteza), warto podejrzeć sygnały odpowiadające za przejścia między stanami w naszym automacie. Jak to zrobić? W układach proceduralnych (µkontrolery) mamy układy debuggera, pozwalające na wstrzymywanie pracy i uruchamianie jej na żądanie w trybie krokowym, podgląd rejestrów a nawet obszarów pamięci. W układach programowalnych taki debugger z prawdziwego zdarzenia możemy zsyntezować i zaimplementować samodzielnie. Jest to jednak dość żmudne i długotrwałe zajęcie. Prostszą metodą jest deugowanie poprzez wystawianie sygnałów na wyprowadzenia zewnętrzne układu (np. niewyokrzystywane piny układów peryferyjnych makiety). Aby zobrazować jak można taki prosty debug osiągnąć, spójrzmy na poniższą filozofię. Tworzymy port wyjściowy, np.:

```
dbg: out std_logic_vector(2 downto 0);
```

Następnie, przypiszmy tym portom sygnały stanu, np.:

```
dbg(0) <= '1' when STAN_OBECNY = IDLE else '0';
dbg(1) <= '1' when STAN_OBECNY = LAUNCH else '0';
dbg(2) <= '1' when STAN_OBECNY = SUSTAIN else '0';
```

Po przypisaniu portów do wyprowadzeń, możemy próbkować sygnały na wyjściu za pomocą układu Analog Discovery w trybie *Logic* i obserwować jak długo automat przebywa w danym stanie i czy jest to zgodne z naszymi założeniami.