LABORATORIUM
PROGRAMOWALNYCH UKŁADÓW
LOGICZNYCH

Ćwiczenie 1
Strukturyzacja kodu VHDL
Maszyna stanów (FSM)

W12IEA-SI0038P

W12EIT-SI0106P

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

dr inż. Bartosz Świadkowski

dr inż. Piotr Smagowski

1. CEL ĆWICZENIA

Zapoznanie z budową maszyny stanów (automatem) i jego przykładem realizacji w języku VHDL. Program zajęć obejmuje:

- elementy składowe maszyny stanów
- poprawny opis maszyny w języku VHDL
- testowanie opisanej struktury w środowisku testowym
- implementacja układu na makiecie

2. MASZYNA STANÓW – INFORMACJE

2.1. OPIS FUNKCJONALNY

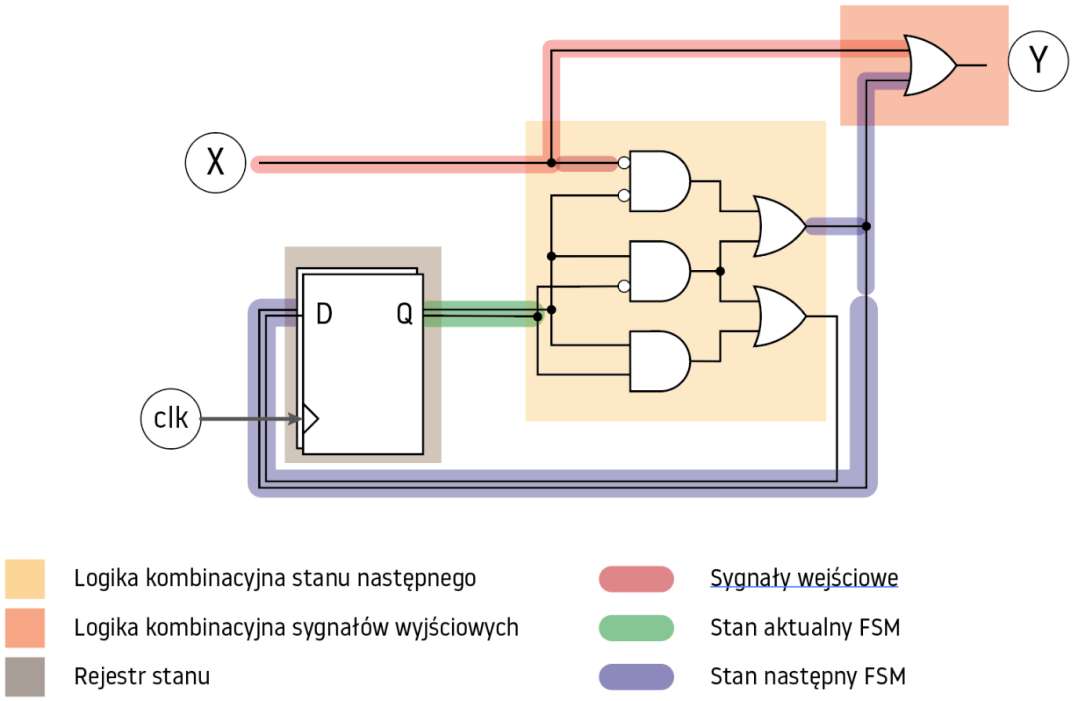
Maszyna stanów (formalnie: automat skończony, FSM, z ang. *Finite State Machine*) jest układem, który może przyjmować określoną z góry liczbę oraz typ (zbiór) „stanów", kombinacji. Ich rodzaj oraz przejścia między nimi są określane w trakcie projektowania FSM. Przejścia mogą być bezwarunkowe lub warunkowe, tj. możliwe tylko po spełnieniu konkretnego warunku. Przykładem maszyny stanów jest automat wydający napoje, który składa się przykładowo ze stanu bezczynności, stanu przyjmowania gotówki oraz stanu wydawania napoju (pod warunkiem dostarczenia odpowiedniej ilości gotówki).

Maszyny stanów opisać można na wiele sposobów, począwszy od równań Boole'a, po przejrzyste grafy – preferowane rozwiązanie. Przykład grafu opisującego układ sygnalizujący naciśnięcie przycisku przez 1 s został przedstawiony poniżej. Poza stanami oczekiwania i sygnalizacji posiada on również dodatkowy stan oczekujący na ustabilizowanie się drgań styków mechanicznego przełącznika. Składa się on z trzech stanów:

1. „Stabilny", z którego przejście jest możliwe do stanu „Opóźnienie" pod warunkiem, że przycisk został naciśnięty (odpowiada stanowi logicznemu wysokiemu na wejściu układu),
2. „Opóźnienie", w którym odmierzany jest czas 20 ms, a następnie, w zależności od stanu przycisku, następuje przejście do stanu „Stabilnego" lub „Niestabilnego",
3. „Niestabilny", który po odczekaniu 1000 ms powraca do stanu „Stabilny".

2.2. REALIZACJA

Automat jest układem synchronicznym, co oznacza, że w swojej strukturze zawiera składowe zależne od zegara – jest to tzw. **rejestr stanu** pamiętający jego aktualny stan. Zbocze zegara wyznacza moment, w którym stan automatu może się zmienić. Następuje wtedy przypisanie stanu „następnego" do sygnału stanu „aktualnego". Do wyznaczenia następnego stanu (czyli określenia, czy i jakie warunki zostały spełnione) potrzebne są dodatkowo układy kombinacyjne (zbudowane z bramek logicznych). Przykładowy schemat prostego automatu o nieokreślonym działaniu przedstawiono na poniższym rysunku.



Na szarym tle zaznaczony został element synchroniczny automatu – rejestr stanu. Na zielono zaznaczono sygnał będący aktualnym stanem, który trafia do logiki kombinacyjnej (czyli

równań/warunków stanu następnego). Wynik tych obliczeń zaznaczono na niebiesko – trafia on do wejścia przerzutników rejestru stanu oraz do logiki kombinacyjnej obliczającej stan wyjść. Wpływ na stan automatu może mieć również sygnał wejściowy zaznaczony na czerwono – widoczne jest jego połączenie zarówno z logiką stanu następnego, jak i sygnałów wyjściowych.

Przy analizie tego i podobnych układów automatów należy pamiętać, że zmiany stanu (co sprowadza się do zapisu wartości do rejestru stanu) następują w takt zegara i jedynie na jego wybranych zboczach (narastających lub opadających).

Opisując strukturę automatu w języku VHDL należy pamiętać, by odzwierciedlić jego strukturę, tj. do osobnego procesu wyodrębnić element synchroniczny (rejestr), a w osobnym zapisać warunki przejść stanów. Podobnie, stan wyjść zaleca się zapisać w osobnych instrukcjach współbieżnych lub procesach. W takim układzie mówi się, że automat został przedstawiony w trzech procesach.

3. ZADANIA DO WYKONANIA

3.1. ZADANIA

3.1.1. Zaimplementować maszynę stanów realizującą tzw. debouncer – układ eliminujący drgania styków. Upewnić się, że układ działa poprawnie przez symulację. Sygnalizować stan przycisku i stanu układu na diodach.

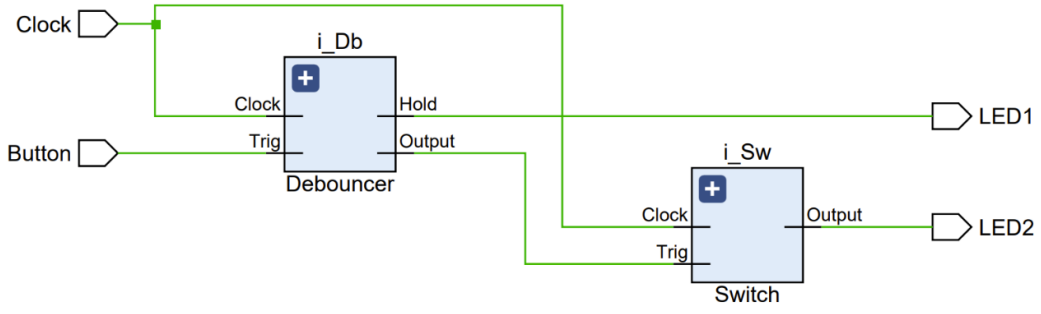
Protip.

Pamiętacie sposób symulacji przedstawiony na zajęciach wprowadzających? Warto z tej metody korzystać, zwłaszcza przy prostych systemach.

3.1.2. Zaimplementować układ przełączający stan wyjścia w sposób wskazany przez prowadzącego, np. zmieniać stan wyjścia (1/0) po każdym naciśnięciu przycisku. Wykorzystać debouncer z poprzedniego zadania. Kod podzielić na dwa komponenty.

3.2. STRUKTURYZACJA KODU

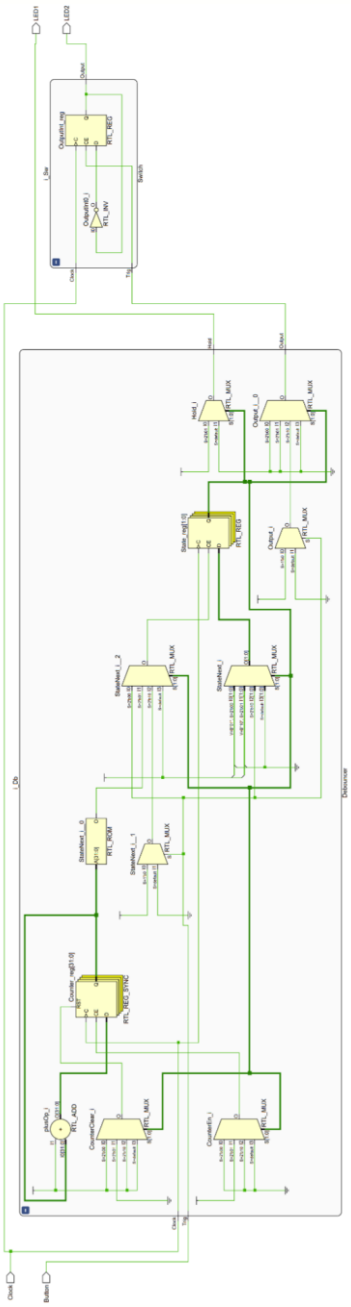
Dobrą praktyką opisu układów w języku VHDL jest dzielenie poszczególnych funkcji układu na mniejsze składowe, z których każda stanowi osobną strukturę (plik .vhd, komponent w rozumieniu *języka* VHDL). Mniejsze układy są łatwiejsze w projektowaniu i w symulacji. Przykład podziału na komponenty układu z zadania nr 2 przedstawiono na poniższym schemacie.



Na schemacie widoczne są elementy:

- Debouncer – odpowiedzialny za eliminację drgań przycisku,
- Switch – układ przełączający stan wyjścia,
- sygnały łączące w/w komponenty z nazwami wskazującymi na ich rolę.

W programie Vivado możliwe jest automatyczne wygenerowanie podobnych schematów na podstawie napisanego kodu (sekcja ***RTL Analysis -> Schematic***). Każdy z komponentów można rozszerzyć i wyodrębnić jego poszczególne składowe (rysunek poniżej). W przypadku Switch są to tylko 1-bitowy rejestr (przerzutnik typu D) oraz negator, natomiast Debouncer składa się z maszyny stanów. Na schemacie widoczne są m.in. rejestr stanu (State_reg[1:0]), rejestr licznika odmierzającego czas (Counter_reg[31:0]) oraz szereg multiplekserów będących odwzorowaniem logiki kombinacyjnej stanu następnego i stanu wyjść.



4. „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.