MINUT 2022 (4), s. 26–36
ISSN 2719-3063

Dariusz CABAN

Wydział Automatyki, Elektroniki i Informatyki, Politechnika Śląska,
Katedra Systemów Rozproszonych i Urządzeń Informatyki, ul. Akademicka 16, 44-100 Gliwice

Prosty system wbudowany z układem FPGA

Streszczenie. W artykule przedstawiono ćwiczenie laboratoryjne, w trakcie którego studenci poznają jeden ze sposobów realizacji systemu komputerowego w oparciu o układ logiki programowalnej nie zawierający procesora. Sposób ten wymaga przygotowania specyfikacji systemu np. w języku opisu sprzętu, można w niej wykorzystywać opisy innych układów. Wspomniane na początku rozwiązanie jest możliwe, gdyż producenci układów programowalnych oferują także opisy procesorów.

Słowa kluczowe: system wbudowany, układ FPGA, język VHDL, procesor Picoblaze.

1. Wstęp

Układy FPGA (ang. *Field Programmable Gate Array*) to złożone układy programowalne, składające się z dużej liczby identycznych komórek logicznych (CLB, ang. *Configurable Logic Block*), rozmieszczonych regularnie, zazwyczaj w formie matrycy, łączonych w większe zespoły logiczne poprzez odpowiednie trakty połączeniowe. W uproszczeniu, komórka logiczna zawiera tablicę LUT (ang. *LookUp Table*), która stanowi generator funkcji kombinacyjnych, przerzutnik oraz multiplekser, przez który na wyjście komórki przekazywany jest sygnał z LUT lub przerzutnika. Strukturę układu cyfrowego realizowanego w układzie FPGA określa zawartość pamięci konfiguracyjnej. Odpowiedni plik konfiguracyjny jest tworzony na podstawie specyfikacji układu, przygotowanej przez projektanta w postaci schematu lub opisu w języku opisu sprzętu (HDL, ang. *Hardware Description Language*) [3].

System wbudowany (ang. *embedded system*) to system komputerowy specjalnego przeznaczenia, który jest integralną częścią obsługiwanego przez siebie sprzętu. Zawiera on procesor, zaprogramowany do wykonywania ograniczonej liczby zadań, w skrajnym przypadku tylko jednego. System może być oparty na mikroprocesorze lub mikrokontrolerze, można do jego realizacji użyć także układu FPGA, przy czym nie musi to być układ z procesorem w środku, bo i takie są produkowane. Istnieją bowiem opisy procesorów w językach opisu sprzętu (ang. *soft processors*), np. 8-bitowego Picoblaze oraz 32-bitowego Microblaze, opracowane w firmie Xilinx [1], [4].

Studenci kierunku Informatyka na Wydziale Automatyki, Elektroniki i Informatyki Politechniki Śląskiej w ramach przedmiotu Budowa Komputerów wykonują dwa ćwiczenia laboratoryjne, w trakcie któ-

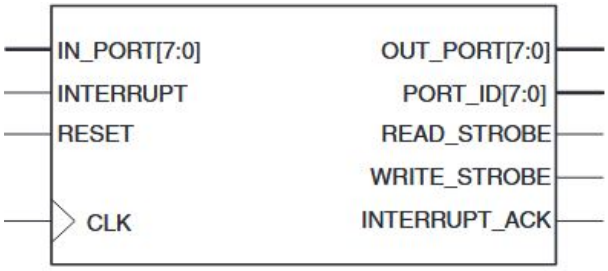
Autor korespondencyjny: D. Caban (Dariusz.Caban@polsl.pl).
Data wpłynięcia: 04.01.2022.

rych używają jednego z języków opisu sprzętu – VHDL (ang. *VHSIC Hardware Description Language*). Celem pierwszego ćwiczenia jest opanowanie podstaw języka, studenci przygotowują opisy prostych układów cyfrowych W trakcie drugiego studenci uzupełniają dany im przez prowadzącego wstępny opis systemu wbudowanego z procesorem Picoblaze. Działanie opisanych układów i systemu jest sprawdzane po ich zrealizowaniu w FPGA (symulacja jest tematem osobnego ćwiczenia).

2. Procesor Picoblaze

Picoblaze jest prostym, 8-bitowym procesorem o architekturze RISC (ang. *Reduced Instruction Set Computer*), autorem jego opisów w językach: VHDL oraz Verilog jest Ken Chapman. Można go użyć tylko do realizacji systemów w układach FPGA firmy Xilinx, gdyż w opisach procesora zostały wykorzystane ich wewnętrzne bloki. Zajmuje stosunkowo małą liczbę komórek logicznych, np. w układach rodziny Spartan-3 tylko 96. Znany jest również pod nazwą KCPSM (ang. *Constant(K) Coded Programmable State Machine*). Opisy procesora można pobrać bezpłatnie ze strony www.xilinx.com.

Procesor Picoblaze występuje w kilku wersjach, przewidzianych dla układów FPGA z różnych rodzin. W trakcie ćwiczeń laboratoryjnych są używane zestawy uruchomieniowe z układem z rodziny Virtex-5, dla których opracowano wersję procesora oznaczoną numerem 3 [2], [5]. Na rysunku 1 przedstawiono jego wyprowadzenia, w języku VHDL nazywane portami lub sygnałami zewnętrznymi.



Rysunek 1. Porty procesora Picoblaze [5]

Po lewej stronie rysunku pokazane są porty wejściowe. Port `clk` jest wejściem dla impulsów zegarowych, ich maksymalna częstotliwość zależy m. in. od układu FPGA. Na port `reset` jest podawany sygnał zerowania, a na port `interrupt` sygnał żądania przerwania, oba aktywne stanem wysokim i synchronizowane sygnałem zegarowym. Przez linie portu `in_port` są przekazywane dane z układów wejściowych, a przez linie portu `instruction` rozkazy z pamięci programu.

Adres komórki pamięci programu procesor wystawia na linie 10-bitowego portu `address`, zatem rozmiar przestrzeni adresowej tej pamięci wynosi 1024. Przez port `out_port` są przekazywane dane do układów wyjściowych. Adres układu wejściowego bądź wyjściowego jest podawany na liniach portu `port_id`, w systemie może być do 256 układów każdego rodzaju. Sygnały z pozostałych portów wyjściowych są aktywne stanem wysokim i są synchronizowane sygnałem zegarowym. Są to: `read_strobe` – sygnał odczytu danej z układu wejściowego, `write_strobe` – sygnał zapisu danej do układu wyjściowego, `interrupt_ack` – sygnał potwierdzenia przyjęcia żądania przerwania.

Procesor dysponuje zestawem 16 rejestrów roboczych, oznaczonych symbolami s0 – sF. Rejestry robocze są uniwersalne, można w nich przechowywać zarówno dane jak i adresy. W dwóch znacznikach: CARRY oraz ZERO jest zapisywana informacja o wyniku wykonanej przez jednostkę arytmetyczno-logiczną operacji arytmetycznej, logicznej, porównania lub testu. Stos w procesorze Picoblaze służy wyłącznie do

przechowywania adresów powrotu przy wywołaniach podprogramów oraz obsłudze przerwań. Pojemność stosu wynosi 31 słów 10-bitowych. Dodatkowym miejscem na dane programu jest 64-bajtowy blok pamięci RAM.

Procesor potrafi wykonywać 30 rozkazów, podzielonych na grupy obejmujące rozkazy:

- przesyłania danych (`load`, `fetch`, `store`, `input`, `output`),
- arytmetyczne (`add`, `addcy`, `sub`, `subcy`),
- logiczne (`and`, `or`, `xor`),
- porównań i testów (`compare`, `test`),
- przesunięć i przesunięć cyklicznych (`sl0`, `sl1`, `slx`, `sla`, `rl`, `sr0`, `sr1`, `srx`, `sra`, `rr`),
- sterowania przebiegiem wykonania programu (`jump`, `call`, `return`, wszystkie w wersji bezwarunkowej i warunkowej),
- związane z obsługą przerwań (`enable interrupt`, `disable interrupt`, `returni enable/disable`).

Każdy z rozkazów zajmuje jedno 18-bitowe słowo i jest wykonywany w dwóch cyklach zegara. Stosowane są trzy tryby adresowania: natychmiastowe, rejestrowe i pośrednie rejestrowe.

Sygnał żądania przerwania musi być aktywny przez co najmniej 2 cykle zegarowe. Jeżeli żądanie może zostać przyjęte to: blokowany jest układ przerwań, na stosie zapisywany jest adres powrotu, stan znaczników jest zapamiętywany w pomocniczych przerzutnikach (nie należy zatem w trakcie obsługi przerwania odblokowywać układu przerwań!), przyjęcie żądania zostaje potwierdzone sygnałem `interrupt_ack` i wykonywany jest skok pod adres 3FFh. Jest to adres ostatniej komórki pamięci programu i powinien być w niej zapisany rozkaz skoku do podprogramu obsługi przerwania. Obsługa przerwania powinna się zakończyć wykonaniem rozkazu `returni`, który odtwarza zawartość licznika rozkazów oraz stany znaczników.

Programy dla procesora Picoblaze są pisane w języku asemblera.

3. Wstępny opis systemu

Studenci otrzymują do uzupełnienia wstępny opis systemu, którego porty są następujące:

- `clk` – wejście dla sygnału zegarowego,
- `rst` – wejście dla sygnału zerowania,
- `switch` – 8-bitowy port wejściowy, do którego będą dołączone przełączniki,
- `led` – 8-bitowy port wyjściowy, do którego będą dołączone diody świecące.

Jest to wariant minimum. Opis zawiera:

- dwie deklaracje `component`, które służą do wskazania, że w opisie systemu są wykorzystywane opisy w języku VHDL innych układów, w tym przypadku procesora Picoblaze oraz pamięci programu,

- deklaracje potrzebnych sygnałów wewnętrznych, z których część będzie odpowiadać połączeniom między elementami systemu,
- dwie instrukcje łączenia komponentów `port map`, dla procesora oraz pamięci programu,
- instrukcje opisujące: dekoder adresów, wytwarzający sygnały wyboru dla układów wyjściowych, oraz układ do wprowadzania informacji wejściowej,
- niekompletne instrukcje opisujące: układy wytwarzające sygnał taktujący i wewnętrzny sygnał zerowania oraz proste układy wyjściowe.

W dalszej części rozdziału zostaną omówione te układy dołączane do procesora, dla których są podane kompletne opisy w języku VHDL.

3.1. Pamięć programu

Opis bloku pamięci programu w języku VHDL jest tworzony przez program narzędziowy `kcpsm3.exe` na podstawie programu źródłowego napisanego w języku asemblera. Wraz z opisem systemu studenci otrzymują zestaw programów testowych, w postaci zarówno źródłowej, jak i opisu bloku pamięci[1]. Pierwszy program, który procesor będzie wykonywać, pozwoli sprawdzić, czy m. in. poprawnie została uzupełniona instrukcja opisująca rejestr sterujący diodami świecącymi. Program ten cyklicznie, w 1-sekundowych odstępach czasu, zapala i gasi diody wskazane za pomocą przełączników. Odmierzanie czasu jest w nim wykonywane przez podprogramy opóźniające, które napisano przy założeniu, że częstotliwość sygnału taktującego dla procesora wynosi 50 MHz. Pisanie tego rodzaju podprogramów ułatwia fakt, że każdy rozkaz procesor wykonuje w ciągu 2 cykli zegara.

3.2. Dekoder adresów układów wyjściowych

Przyjęto założenie, że w systemie będzie maks. 8 układów wyjściowych, wybieranych sygnałami wyboru wytwarzanymi przez dekoder adresów, opisany instrukcją:

```
ADDR_DEC: process(port_id)
begin
   en <= port_id;
end process;
```

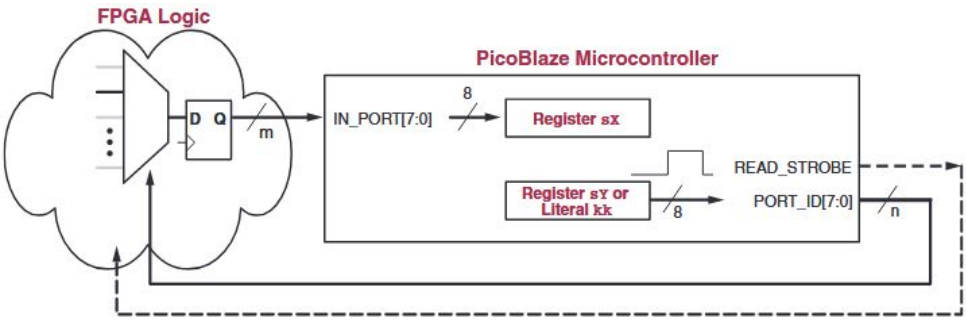
gdzie `en` jest wektorem 8 sygnałów wyboru. Opis dekodera nie jest skomplikowany, gdyż przy maks. 8 układach wyjściowych można zastosować tzw. adresowanie liniowe. Pojedynczy układ jest wybierany sygnałem pochodzącym z jednego bitu portu `port_id` procesora. Jeżeli stanem aktywnym sygnału wyboru będzie stan wysoki to układom wyjściowym można przydzielić adresy: 1, 2, ..., 128. Użycie adresowania liniowego przynosi, oprócz uproszczenia struktury systemu, jeszcze jedną korzyść – będzie możliwy zapis danej jednocześnie do kilku układów wyjściowych o różnych adresach. Oczywiście można byłoby się obejść bez tak opisanego dekodera i w opisach układów wyjściowych wykorzystywać sygnały `port_id(i)`, gdzie `i` = 0, 1, ..., 7. Rozwiązanie z dekoderem ma jednak tę zaletę, że gdy okaże się, że liczba układów wyjściowych przekroczy 8 to wystarczy zwiększyć liczbę bitów wektora `en` i w instrukcji procesu opisać

[1]Sami nie piszą programów testowych z dwóch przyczyn. Po pierwsze, na wykonanie ćwiczenia mają stosunkowo mało czasu (4 godziny lekcyjne). Po drugie, program `kcpsm3.exe` jest programem DOS-owym, którego nie da się uruchomić w okienku systemu DOS w systemie Windows 7 lub nowszym.

dekoder, który będzie wytwarzał sygnały wyboru na podstawie wartości części bitów adresowych (tzw. dekoder niepełny). W tym przypadku nie będzie możliwy jednoczesny zapis danej do układów o różnych adresach.

3.3. Układ do wprowadzania informacji wejściowej

Struktura tego układu jest przedstawiona po lewej stronie rysunku 2.



Rysunek 2. Struktura układu wprowadzania informacji do procesora [5]

Jest on złożony z multipleksera oraz rejestru, do którego zapis odbywa się przy każdym narastającym zboczu sygnału zegarowego. Sygnał `read_strobe` sygnalizuje odczytanie danej, ale w praktyce jest rzadko używany (wtedy, gdy układ wejściowy ma kolejkę FIFO). Przyjęto, że liczba układów wejściowych nie przekroczy 8, a rejestr będzie miał jeszcze wejście zerowania. Instrukcja opisująca układ do wprowadzania informacji wejściowej ma postać:

```
INPUTS: process(clk50MHz, reset)
begin
   if reset = '1' then
     in_port <= (others => '0');
   elsif rising_edge(clk50MHz) then
     case port_id(2 downto 0) is
        when "000" => in_port <= switch;
        when others => in_port <= (others => '0');
     end case;
   end if;
end process;
```

Zespołowi ośmiu przełączników przypisano adres 0, przy odczycie spod innych adresów otrzymuje się wartość 0. Przy dołączaniu kolejnego układu wejściowego trzeba dopisać człon `when` z odpowiednim adresem.

4. Zadania do wykonania w czasie ćwiczenia

W dostarczanym studentom opisie systemu są trzy niekompletne instrukcje. Układy, które powstaną w wyniku ich syntezy mają m.in zapewnić to, że procesor będzie działał. Instrukcje te należy uzupełnić w pierwszej kolejności, dopiero potem można przystąpić do rozbudowy systemu o inne układy, realizujące różnorodne funkcje.

4.1. Układ wytwarzający sygnał zerowania

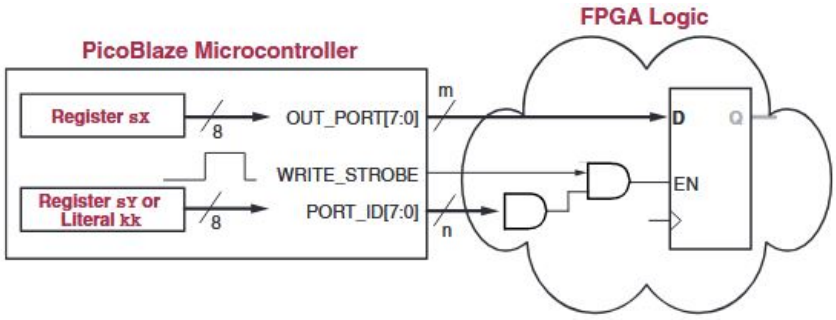
Sygnał podawany na wejście `rst` systemu będzie pochodził z przycisku niestabilizowanego. W zależności od sposobu podłączenia przycisku do FPGA sygnał ten można bezpośrednio wykorzystać do zerowania układów w systemie bądź nie. Dlatego do wejść zerowania układów w systemie doprowadzony jest wewnętrzny sygnał `reset`, odpowiednio ustawiany sygnałem `rst`, za co odpowiada pierwsza instrukcja w ciele architektury.

4.2. Układ wytwarzający sygnał taktujący

Na wejście `clk` będzie podawany sygnał zegarowy o częstotliwości 100 MHz. Przyjęto, że sygnałem taktującym procesor oraz inne układy, które wymagają taktowania, będzie sygnał `clk50MHz`. Ćwiczący muszą tak uzupełnić jedną instrukcję procesu, aby wynikiem jej syntezy był podzielnik częstotliwości przez 2. Pomocny w tym może być rysunek przedstawiający przebiegi na wejściu i wyjściu podzielnika.

4.3. Prosty układ wyjściowy

Ostatnią instrukcją wymagającą uzupełnienia jest instrukcja procesu opisująca prosty układ wyjściowy - 8-bitowy rejestr. Do wyjść rejestru zostaną dołączone diody świecące. Jak wynika z rysunku 3 zapis do rejestru ma nastąpić przy narastającym zboczu sygnału zegarowego i wysokim stanie sygnałów: zapisu do układu wyjściowego oraz wyjściowego z komparatora adresów. W danym systemie komparator ma nie wystąpić, do dyspozycji są sygnały wytwarzane przez dekoder adresów.



Rysunek 3. Prosty układ wyjściowy [5]

4.4. Układ wytwarzający cykliczne sygnały żądania przerwania

Pierwszy wykonywany przez procesor program w 1-sekundowych odstępach czasu zapala i gasi diody świecące, wybrane za pomocą przełączników. Odmierzanie odcinków czasu między zmianami stanów diod jest wykonywane przez podprogramy opóźniające, co jest rozwiązaniem prostym, ale nieefektywnym. Innym możliwym rozwiązaniem jest zastosowanie licznika programowego, zliczającego przerwania, którego żądania są zgłaszane cyklicznie przez układ czasowy np. co 1 ms. Przy 16-bitowym liczniku lepiej żądania zliczać w dół, gdyż porównanie jego zawartości z wartością 0 wymaga wykonania mniejszej liczby rozkazów niż porównanie z wartością niezerową. Taki licznik wykorzystuje kolejny program testowy. Studenci muszą opisać w języku VHDL układ wytwarzający przebieg prostokątny o częstotliwości 1 kHz, w którym stan wysoki będzie trwać tylko przez 1 cykl wejściowego sygnału zegarowego, którym taktowany jest także procesor. Układ ma być opisany jako osobna jednostka projektowa, w sposób sparametryzowany,

parametrem będzie współczynnik podziału częstotliwości wejściowej. Tak opisany układ będzie można wykorzystać przy dalszej rozbudowie systemu – nadajnik i odbiornik asynchronicznej transmisji szeregowej wymagają sygnału podstawy czasu, o takim przebiegu i częstotliwości zależnej od przyjętej prędkości transmisji.

Sygnał wyjściowy tego układu nie może być jednak sygnałem żądania przerwania, gdyż ten musi być w stanie wysokim przez co najmniej 2 cykle zegarowe. Dlatego potrzebny jest blok pośredniczący, który wytworzy sygnał żądania o wymaganym minimalnym czasie trwania. Jego działanie w sposób słowny można opisać następująco:

jeżeli sygnał `reset` jest w stanie wysokim to
    ustaw sygnał `interrupt` w stan niski
w przeciwnym razie jeżeli wystąpiło narastające zbocze sygnału `clk50MHz` to
    jeżeli sygnał `clk1kHz` jest w stanie wysokim to
        ustaw sygnał `interrupt` w stan wysoki
    w przeciwnym razie jeżeli sygnał `interrupt_ack` jest w stanie wysokim to
        ustaw sygnał `interrupt` w stan niski

Na jego podstawie trzeba przygotowywać odpowiednią instrukcję procesu. Taki układ może działać jeszcze według innej zasady.

4.5. Dołączenie nadajnika i odbiornika asynchronicznej transmisji szeregowej

Układy te są opisane w osobnych jednostkach projektowych i dostarczane wraz z opisem procesora Picoblaze. W pierwszej kolejności powinien zostać dołączony nadajnik. Odpowiedni program testowy m. in. co 2 sekundy będzie wysyłał przez łącze szeregowe tekst o długości 32 znaków, zakończony znakami: powrotu na początek wiersza (potocznie nazywany znakiem powrotu karetki, kod ASCII 13) oraz przejścia do nowego wiersza (kod ASCII 10). Nadajnik ma własną kolejkę FIFO o długości 16 znaków, ma również sygnały wyjściowe informujące o tym, czy kolejka jest zapełniona w całości i co najmniej w połowie. Sygnały te mają być częścią bajtu statusowego, który procesor będzie mógł odczytać. Informację zwrotną z nadajnika wykorzystuje podprogram zapisu znaku do wysyłki, aby mieć pewność, że w kolejce jest miejsce na nowy znak.

Odbiornik dołącza się jako drugi z tego powodu, że odpowiedni program testowy wykorzystuje również nadajnik – odsyła odbierane znaki do nadawcy (program „echo"). Odbiornik również ma kolejkę FIFO o długości 16 znaków oraz sygnały wyjściowe informujące o tym, że: 1) w kolejce jest co najmniej 1 znak, 2) kolejka jest zapełniona w całości, 3) kolejka jest zapełniona co najmniej w połowie. Wymienione sygnały wyjściowe też mają być częścią bajtu statusowego.

4.6. Układ eliminacji wpływu drgań zestyku przycisku

Do procesora można też doprowadzić sygnały z przycisków niestabilizowanych, podłączonych bezpośrednio do wejść układu FPGA. Przy naciskaniu i zwalnianiu przycisku występuje zjawisko drgań zestyku, trwające od kilku do kilkudziesięciu milisekund, a jego skutkiem jest seria impulsów. Wpływ drgań na pracę systemu daje się wyeliminować programowo, ale w prosty sposób można to zrobić sprzętowo, odciążając tym samym procesor. Dogodnie jest opisać w osobnej jednostce projektowej układ eliminacji wpływu drgań zestyku jednego przycisku, a potem, używając instrukcji `generate`, powielić wymaganą liczbę razy odpowiednią instrukcję łączenia komponentów. Układ taki może w regularnych odstępach

czasu próbkować stan przycisku. Stan wyjścia układu się zmieni, gdy wartości kilku kolejnych próbek będą takie same i przeciwne do bieżącego stanu wyjścia. Program testowy zlicza naciśnięcia jednego z przycisków i na diodach świecących wyświetla wynik w naturalnym kodzie dwójkowym.

5. Rozwiązania wybranych zadań

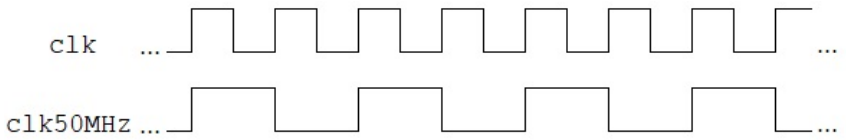
5.1. Układ wytwarzający sygnał zerowania

Na stronie [2] jest odsyłacz do pliku ze schematem układu na płycie zestawu. Ze schematu wynika, że gdy przycisk nie jest naciśnięty to na końcówce układu FPGA, do której dołączone jest wejście `rst` opisywanego systemu wbudowanego, jest stan wysoki. Stan wysoki jest stanem aktywnym sygnału zerowania procesora Picoblaze. Wewnętrzny sygnał `reset` powinien być zatem negacją sygnału `rst`, będzie go wytwarzał układ opisany instrukcją:

```
reset <= not rst;
```

5.2. Układ wytwarzający sygnał taktujący

Uzupełnienie instrukcji opisującej ten układ ułatwi rysunek 4.



Rysunek 4. Przebiegi czasowe sygnałów: `clk` i `clk50MHz`

Jak łatwo zauważyć, przy każdym zboczu narastającym sygnału `clk` następuje zmiana stanu sygnału `clk50MHz` na przeciwny do bieżącego. Układ wytwarzający sygnał o takim przebiegu można opisać instrukcją:

```
SYS_CLK: process(clk)
begin
   if rising_edge(clk) then
      clk50MHz <= not clk50MHz;
   end if;
end process;
```

Na liście wrażliwości procesu brak sygnału `reset`, gdyż sygnał taktujący musi być wytwarzany bez przerwy, sam sygnał zerowania jest nim synchronizowany. W procesie syntezy opis układu w języku VHDL jest przekształcany do opisu logicznego, w którym występują typowe elementy logiczne. Na poziomie tego opisu układ wytwarzający sygnał taktujący będzie się składał z synchronicznego przerzutnika D oraz bramki negacji. Na wejście wyzwalające przerzutnika będzie podany sygnał `clk`, a na wejście D zanegowany sygnał z jego wyjścia Q. Sygnałem `clk50MHz` będzie sygnał z wyjścia Q przerzutnika.

5.3. Prosty układ wyjściowy

Rejestrowi, któremu w opisie systemu w języku VHDL odpowiada zewnętrzny sygnał `led`, został przyporządkowany adres 1. Sygnałem wyboru rejestru będzie sygnał `en(0)` z dekodera adresów układów wyjściowych. Kompletna instrukcja opisująca rejestr będzie następująca:

```
OUT_REG: process(clk50MHz, reset)
begin
   if reset = '1' then
      led <= (others => '0');
   elsif rising_edge(clk50MHz) then
      if write_strobe = '1' and en(0) = '1' then
         led <= out_port;
      end if;
   end if;
end process;
```

Rejestr będzie złożony z ośmiu synchronicznych przerzutników D z wejściami wybierającymi CE, na które będzie podany sygnał będący iloczynem sygnałów: `write_strobe` i `port_id(0)` (sygnał ten odpowiada sygnałowi `en(0)`, jeżeli zostało zastosowane adresowanie liniowe).

5.4. Układ wytwarzający cykliczne sygnały żądania przerwania

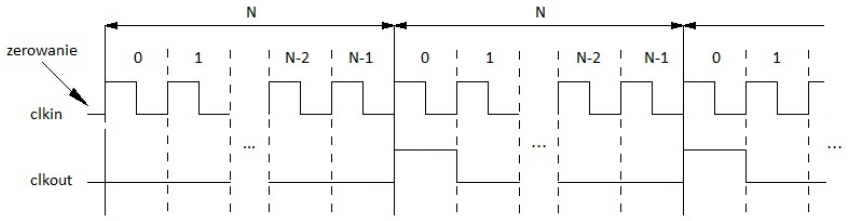
Będzie się on składał z dwóch bloków. Na wejście pierwszego będzie wchodził sygnał zegarowy o częstotliwości 50 MHz, na jego wyjściu ma być wytwarzany przebieg prostokątny o częstotliwości 1 kHz i czasie trwania stanu wysokiego równym czasowi trwania jednego cyklu sygnału wejściowego. Sygnał ten będzie sygnałem wejściowym dla drugiego bloku, który będzie wytwarzać sygnały żądania przerwania o wymaganym minimalnym czasie trwania, kasowane po potwierdzeniu przez procesor przyjęcia żądania.

Pierwszy z bloków ma być opisany jako osobna, sparametryzowana jednostka projektowa. Pozwoli to na jej wielokrotne wykorzystanie w projektowanym systemie. Deklaracja jednostki, w której określa się jej wejścia i wyjścia ma być następująca:

```
entity freqdiv is
   Generic (N : positive);
   Port (  clkin : in  STD_LOGIC;
             rst : in  STD_LOGIC;
          clkout : out  STD_LOGIC);
end freqdiv;
```

Będzie to zwykły dzielnik częstotliwości przez `N`. Wymagany przebieg jego sygnału wyjściowego przedstawiono na rysunku 5.

Dzielnik będzie zawierał licznik zliczający sygnały wejściowe, N sygnałów przypada na jeden okres sygnału wyjściowego. Po zerowaniu dzielnika przez pierwszy okres na wyjściu ma być stan niski. Stan wysoki ma być ustawiany na początku każdego kolejnego okresu, na czas trwania jednego sygnału wejściowego. Numer sygnału wejściowego jest równy zawartości licznika w chwili wystąpienia narastającego zbocza sygnału. Deklaracja i ciało architektury dzielnika mogą być następujące:



Rysunek 5. Przebieg sygnału na wyjściu clkout dzielnika częstotliwości przez N

```
architecture Behavioral of freqdiv is
   signal cntreg: natural range 0 to N - 1;
begin
   process(clkin, rst)
   begin
      if rst = '1' then
         cntreg <= 0;
         clkout <= '0';
      elsif rising_edge(clkin) then
         if cntreg < N - 1 then
            cntreg <= cntreg + 1;
            clkout <= '0';
         else
            cntreg <= 0;
            clkout <= '1';
         end if;
      end if;
   end process;
```

W powyższym opisie licznik jest reprezentowany przez sygnał wewnętrzny `cntreg`. Podanie w deklaracji zakresu wartości, które można zapisywać w sygnale służy systemowi projektowemu do określenia liczby przerzutników potrzebnych do syntezy licznika, bez tego zostałaby użyta ich liczba domyślna dla podtypu `natural`, tj. 31.

Określenie częstotliwości sygnału wyjściowego następuje w instrukcji łączenia komponentów dla jednostki `freqdiv`, przez przyporządkowanie odpowiedniej wartości parametrowi N:

```
   TIMER_1MS: freqdiv
   generic map (N => 50000)
   port map (  clkin => clk50MHz,
                 rst => reset,
              clkout => clk1kHz);
```

W podrozdziale 4.4. podano słowny opis działania drugiego z bloków, przygotowanie na jego podstawie opisu bloku w języku VHDL jest proste:

```
   TIMER_1MS_INT_FLAG: process(clk50MHz, reset)
   begin
      if reset = '1' then
```

```
      interrupt <= '0';
   elsif rising_edge(clk50MHz) then
      if clk1kHz = '1' then
         interrupt <= '1';
      elsif interrupt_ack = '1' then
         interrupt <= '0';
      end if;
   end if;
end process;
```

6. Zakończenie

W trakcie ćwiczenia studenci poznają jeszcze jeden sposób realizacji systemu wbudowanego. W projektach systemów wbudowanych, wykonywanych wcześniej w ramach innego przedmiotu, stosowali mikroprocesory lub mikrokontrolery. W obu przypadkach korzystali z gotowych już układów wejścia-wyjścia, zewnętrznych albo tych, w które wyposażony był mikrokontroler. Gdy system jest realizowany w oparciu o układ FPGA wówczas można tworzyć własne rozwiązania takich układów.

Dodatkową korzyścią z wykonania tego ćwiczenia będzie utrwalenie oraz poszerzenie umiejętności używania języka VHDL, jednego z najszerzej stosowanych języków opisu sprzętu.

Podziękowania

Autor pragnie podziękować recenzentom za trud włożony w recenzje.

Literatura

1. P. P. Chu, *FPGA prototyping by VHDL examples*, John Wiley and Sons, Inc., 2008.
2. Digilent, *Genesys Virtex-5 FPGA Development Board*, `https://digilent.com/reference/programmable-logic/genesys/` [dostęp 10.12.2021].
3. J. Majewski, P. Zbysiński, *Układy FPGA w przykładach*, Wydawnictwo BTC, Warszawa 2007.
4. M. Nowakowski, *Picoblaze. Mikroprocesor w FPGA*, Wydawnictwo BTC, Legionowo 2010.
5. Xilinx, *Picoblaze 8-bit Embedded Microcontroller User Guide*, `https://www.xilinx.com/support/documentation/ip_documentation/ug129.pdf` [dostęp 23.11.2021].