mgr inż. Marek Śnieżek
Katedra Automatyki i Informatyki
Politechnika Rzeszowska

REALIZACJA *MASTER* - PROCESORA DLA STEROWNIKA LOGICZNEGO O PODWYŻSZONYCH WYMAGANIACH BEZPIECZEŃSTWA

Streszczenie: W pracy przedstawiono realizację procesora master wchodzącego w skład sterownika logicznego przeznaczonego dla procesów, gdzie jest wymagana gwarancja bezpiecznej pracy i sygnalizacja wystąpienia awarii. Do obliczeń zastosowano dwuprocesorową konfigurację master-slave. Wykrywanie błędów sprzętowych jest możliwe dzięki zastosowaniu dwóch takich par, wykonujących ten sam algorytm i prowadzeniu kontroli zgodności wszystkich danych wymienianych pomiędzy procesorami master i slave. Potencjalne błędy oprogramowania eliminuje formalna weryfikacja.

Abstract: This paper presents realisation of the master processor that is a part of a logic controller for process applications, where fail-safe operation and signalling of failure are expected. Dual-processor - master-slave - configuration is chosen. Hardware faults are detected by applying two identical pairs of master-slave, which execute the same program and compare every data being exchanged. Software errors are eliminated by formal proving.

1. ARCHITEKTURA STEROWNIKA

Sterowanie procesem jest wykonywane programowo, gdzie algorytm opisano schematem połączeń bloków funkcyjnych. Oprogramowanie sterownika bezpiecznego podlega kosztownej weryfikacji, dlatego zostało podzielone na dwie części:

- zmienną - obejmującą strukturę połączeń bloków
- niezmienną - zawierającą bibliotekę bloków.

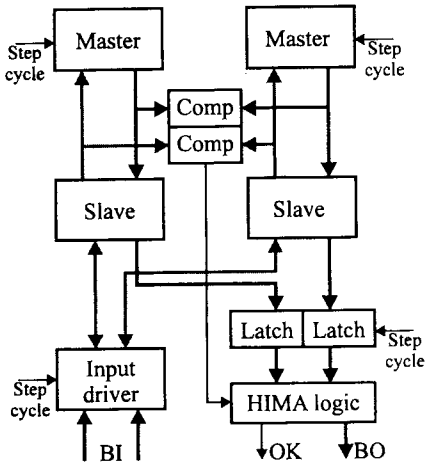
Do wykonywania programów wykorzystano procesory:

- *master* - wykonujący program obsługi procesu, poprzez wywołania bloków funkcyjnych
- *slave* - obliczający bloki funkcyjne schematu (logiczne, arytmetyczne itd.).

Program *slave'a* nie zmienia się, dzięki czemu podlega jednorazowej weryfikacji. Program *mastera* zależy od aplikacji. Jest zapisany w specjalnie opracowanym prostym języku, co ma ułatwić weryfikację. Schemat blokowy sterownika przedstawia rysunek 1, opis zawarto w [1].

Procesory są fizycznie rozdzielone i korzystają tylko z własnych pamięci programu i danych. Do wymiany danych wykorzystują dwukanałową magistralę buforowaną układami FIFO (ang. *First-In-First-Out*). Zastosowanie tych układów przenosi problemy dostępu i ochrony na poziom sprzętowy.

Master i *slave* wykonują programy w sposób zsynchronizowany i wzajemnie zależny. *Master* inicjuje *slave'a* do obliczenia bloku funkcyjnego. Odbywa się to przez przesłanie do niego identyfikatora funkcji obliczającej blok i parametrów: wejść bloku oraz stanów wewnętrznych

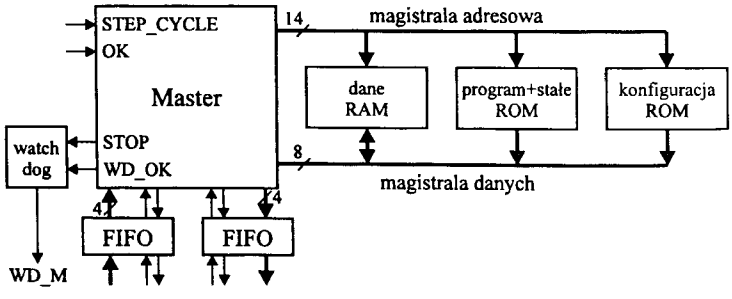


Rys. 1. Architektura sterownika

(jeśli blok takie posiada). Po otrzymaniu identyfikatora *slave* rozpoznaje funkcję, kompletuje parametry, po czym oblicza wyjścia i nowe stany wewnętrzne. Wartości te zostają odesłane do *mastera*, który je przechowuje. Pamięć *slave'a* jest podręczna i obejmuje tylko czas wykonywania funkcji. Dlatego *slave* może być utożsamiany z koprocesorem funkcyjnym lub specjalizowanym kalkulatorem.

Wykrywanie uszkodzeń sprzętowych jest możliwe dzięki zastosowaniu drugiej pary procesorów *master-slave*, wykonującej ten sam program i ciągłej kontroli zgodności wszystkich danych wymienianych w parach. Prowadzi się zatem monitorowanie poprawności wykonania, na poziomie bloków funkcyjnych. Porównania są realizowane przez szybkie komparatory *comp*. Ponieważ od poprawności ich pracy zależy pewność porównań, zostały one wykonane w *bezpiecznej* technologii [4].

Cykl pracy sterownika jest wyznaczany zmianami sygnału *step_cycle*, synchronizującego pracę *masterów*. Sterownik komunikuje się z otoczeniem poprzez porty wejścia i wyjścia dołączone do procesorów *slave*. Wszystkie sygnały są binarne. Stan wejść obiektowych jest zapamiętywany na początku cyklu i przechowywany w układzie *input_driver*. Wyjścia obiektowe są zatrzaskiwane na końcu cyklu w rejestrach *latch* i porównywane przez komparator. Równocześnie stają się one dostępne na zewnątrz sterownika. Porównania i obsługa wyjść są prowadzone w *bezpiecznym* układzie *HIMA logic* [5]. W nim także, na podstawie sygnałów poprawności ze wszystkich modułów, jest tworzony sygnał poprawności pracy sterownika



Rys. 2. Otoczenie procesora *master*

OK, dostępny na zewnątrz. W razie wystąpienia awarii sterownika zapewnia on ustawienie wyjść w stan wyłączenia [2]. Jest to *bezpieczny* stan sterownika.

Bezpośrednie otoczenie *mastera* pokazano na rys. 2. Procesor jest zaimplementowany w strukturze FPGA (ang. *Field Programmable Gate Array*), firmy Xilinx [6]. Korzysta on z pamięci programu i stałych ROM oraz danych RAM. FPGA wymaga pamiętania konfiguracji w zewnętrznej pamięci ROM. *Master* współpracuje z dwoma układami FIFO, przez które wysyła i odbiera dane. Wytwarza sygnały sterujące pracą *watch-doga*: odświeżania - WD_OK i zerowania - STOP. Procesor może zostać zatrzymany przez sygnał poprawności OK.

Slave jest typowym uniwersalnym procesorem i nie będzie tutaj omawiany [1].

2. JĘZYK PROGRAMOWANIA PROCESORA *MASTER*

Master nie wykonuje żadnych obliczeń, a jedynie operacje przesłania danych. Obliczenie bloku funkcyjnego wiąże się po stronie *mastera*, z wykonaniem następującego ciągu instrukcji, nazywanego makroinstrukcją:

1. wysłanie do *slave'a* identyfikatora funkcji
2. wysłanie do *slave'a* danej wejściowej funkcji
3. odbiór ze *slave'a* danej wyjściowej.

Instrukcja 1 zawsze jest na początku. Instrukcje 2 występują po kolei, a ich liczba i kolejność zależy od funkcji. Podobnie 3. Dla niektórych funkcji instrukcje 2 lub 3 mogą nie wystąpić. Segment programu to ciąg makroinstrukcji zakończony instrukcją `STEP`. Segmenty programu, zgodnie z ideą SFC (ang. *Sequential Function Charts*) tworzą kroki, a instrukcje `STEP` przejścia. `STEP` służy więc wykonaniu skoku w programie. Nie jest to jednak zwykły skok, bo kolejne przejścia są zdeterminowane czasowo przez zmieniający się sygnał taktujący *step_cycle*. Instrukcja `STEP` wykonuje jeszcze dodatkowe czekanie. Pojawienie się *step_cycle* w czasie wykonywania instrukcji innej niż `STEP`, jest błędem i powoduje zerowanie *watch-doga*, a w konsekwencji zatrzymanie sterownika. Wszystkie segmenty tworzą program.

Argumentem `STEP` jest adres SIA (ang. *Step-Initial-Address*). Dodatkowo *master* posiada 2 rejestry specjalne ustawianę odpowiednimi instrukcjami przesłania:

- rejestr adresu - NSA (ang. *Next-Step-Address*)
- jednobitowy rejestr sterujący (warunku) - TCR (ang. *Transition-Condition-Register*).

Skok odbywa się w chwili zmiany *step_cycle*, zgodnie z algorytmem:

jeśli TCR=0, to skok według zawartości SIA, w przeciwnym razie według NSA.

Dla sterownika wprowadza się następujące typy danych:

- *bool* (1 bit)
- *integer* (16 bitów)
- *long* (32 bity)

Organizacja pamięci i rejestrów danych w procesorze jest 8-bitowa, dlatego dana typu *bool* zajmuje 1 komórkę pamięci (8-bitów), *integer* - dwie, *long* - cztery. Wprowadzenie typów danych jest umowne. W istocie chodzi o rozmiar zmiennej. Sposób interpretacji zależy od funkcji, a kontrola zgodności typów następuje w czasie weryfikacji.

Rozróżnia się cztery obszary danych:

- stałe *const* - wejścia funkcji nie zmieniające się w czasie pracy
- zmienne *var* - wyjścia funkcji
- stany wewnętrzne funkcji *isv* - wartości nie związane bezpośrednio z wyjściami, ale niezbędne do ich obliczenia (np. poprzedni stan wejścia, bieżąca wartość licznika itd.)
- początkowe stany wewnętrzne *isv_0* - na początku pracy przepisywane do obszaru *isv*.

Dla ochrony dane *const* i *isv_0* są umieszczone w pamięci stałej. Pozostałe dane (*var*, *isv*) przechowuje się w RAM, umożliwiającej zapis i odczyt. W każdym obszarze są utworzone 512-elementowe tablice dla wszystkich typów danych (*bool*, *integer*, *long*). Różnicą w stosunku do innych procesorów jest stały podział pamięci danych uwzględniający rozmiar i znaczenie danej w programie. Ułatwia to formułowanie specyfikacji i późniejszą weryfikację.

W oznaczeniach mnemonicznych przyjęto zasadę, że przesłania z *mastera* rozpoczynają się od `PUT_`, a ze *slave'a* - `GET_`. Niektóre instrukcje oznaczają:

`PUT_id(f)` - przesłanie identyfikatora funkcji `f`
`PUT_var(t,n)` - przesłanie danej typu `t` z obszaru *var* o numerze `n`
`GET_isv(t,n)` - odebranie danej typu `t` i zapisanie w obszarze *isv* (pozycja `n`)

oraz

`STEP(adr)` - wykonaj instrukcję `STEP`. `adr` stanowi SIA.

Wszystkie instrukcje języka *mastera* są bezpośrednio jego rozkazami. Rozkazy są 16-bitowe (2 bajty), a ich liczba w programie jest ograniczona do 4096.

Najstarsze 4 bity określają kod rozkazu. Pozostałe bity, w różnych rozkazach, zawierają identyfikator funkcji, typ danej, numer danej w obszarze, adres skoku. Kod w rozkazach przesłania uwzględnia:

- miejsce, z/do którego są przesyłane dane: *const*, *var*, *isv*
- kierunek: *master* → *slave*, *slave* → *master*, *master* → TCR/NSA, *slave* → TCR/NSA.

3. BUDOWA PROCESORA *MASTER*

Wszystkie instrukcje języka programowania *mastera* są bezpośrednio wykonywane przez procesor. Stanowią zarazem jego listę rozkazów. Aby umożliwić wykonanie rozkazów listy zbudowano procesor, którego schemat blokowy pokazano na rys. 3. Wyróżniono w nim jednostkę sterującą *Control Unit* i część operacyjną, w skład której wchodzą rejestry, multipleksery, bufory, licznik rozkazów i dekoder adresowy. Funkcje poszczególnych elementów zostaną omówione w nawiązaniu do rozkazów, które mogą być wykonane.

Ponieważ rozkaz jest 2-bajtowy, a organizacja pamięci 1-bajtowa, konieczne jest 2-krotne czytanie zawartości dwóch kolejnych komórek pamięci programu. Rozkaz jest zapamiętywany w rejestrach procesora CRL i CRH. Ponadto, młodsza część rozkazu jest przechowywana w wyjściowym rejestrze danych ODR. Pozwala to uniknąć powtórnego czytania, gdyby rozkazem był `PUT_id`. Dla pozostałych rozkazów zapis w ODR nie jest potrzebny. Adres aktualnie pobieranego rozkazu jest pamiętany w liczniku rozkazów PC, którego zawartość zwiększa się za każdym razem po odczytaniu bajtu.

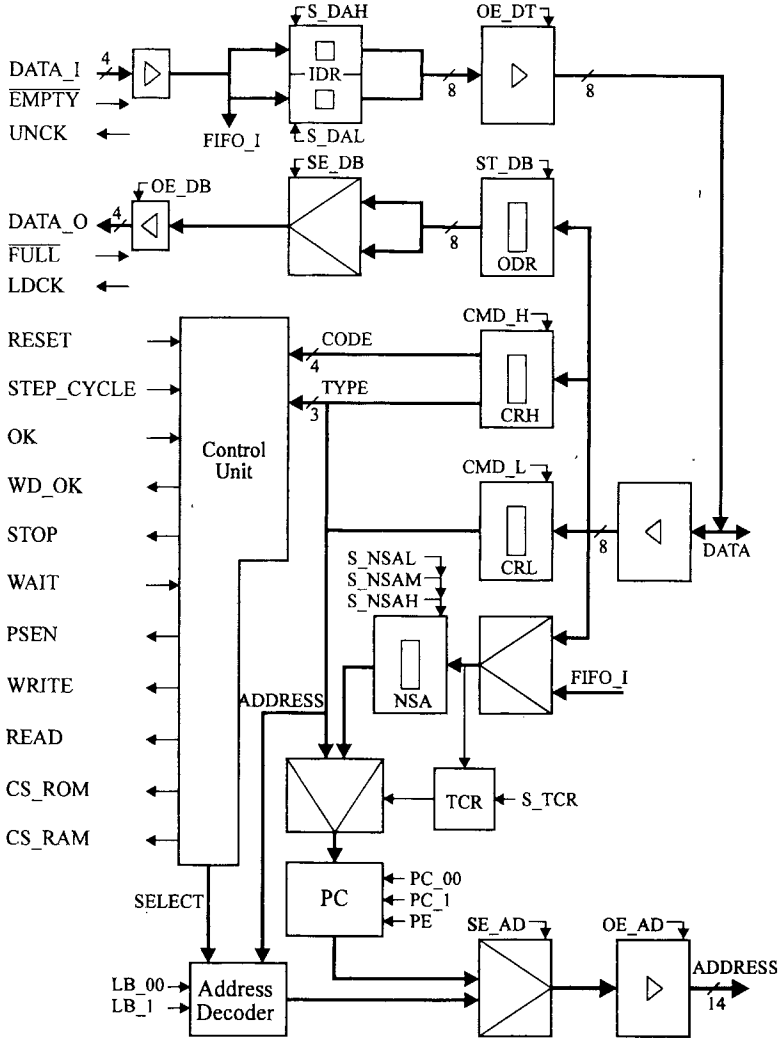
Jednostka sterująca dekoduje rozkaz i podejmuje akcję związaną z jego wykonaniem. Gdy rozkazem jest `PUT_id`, to identyfikator funkcji (pamiętany w ODR) jest przesyłany do *slave'a* 4-bitową magistralą DATA_O. Do wyboru właściwej czwórki danych służy multiplekser SE_DB. Identyfikator jest przesyłany dwukrotnie.

Grupa rozkazów `PUT_` obejmuje przesłania zarówno z pamięci procesora *master* do *slave'a* jak i do rejestrów specjalnych *mastera*. Adres danej jest wyznaczany w dekoderze adresowym na podstawie kodu obszaru pamięci *code* (*const*, *isv*, *var*), typu danej *type* (*bool*, *integer*, *long*) i numeru danej w obszarze. Tak wyznaczony adres jest podawany na magistralę adresową i następuje odczyt. Dana jest zapamiętywana w rejestrze ODR gdy przesłanie dotyczy *slave'a* lub w rejestrze specjalnym procesora *master* (NSA, TCR). Gdy przesłanie jest do *slave*, dana z rejestru ODR jest przepisywana do FIFO. Ponieważ rozkazy `PUT_` obejmują dane o różnej długości (1, 2, 4 bajty), wymaga to wprowadzenia licznika odczytanych bajtów. Stan licznika jest uwzględniany w dekoderze adresowym.

W przypadku rozkazów GET_ dwie dane 4-bitowe są kolejno odczytywane ze *slave'a* przez magistralę DATA_I. Jeśli przesłanie dotyczy pamięci są one zapamiętywane w 8-bitowym rejestrze danych wejściowych IDR, a następnie przepisywane do pamięci lub bezpośrednio zapamiętane w TCR/NSA. Liczba bajtów i ewentualny adres w pamięci są wyznaczane jak dla rozkazów PUT_.

Rozkaz STEP jest jedynym, który modyfikuje stan licznika rozkazów przez wpisanie równoległe. Pomiędzy STEP struktura programu jest liniowa, a licznik może być tylko zwiększany. W zależności od stanu TCR wpisuje się zawartość z NSA lub z rejestrów CRL i CRH, gdzie jest pamiętany SIA (jako argument rozkazu STEP).

Wszystkie sygnały sterujące układami wewnętrznymi procesora i zewnętrznymi (np. zapis/odczyt pamięci, obsługa FIFO itd.) są wytwarzane w jednostce sterującej.



Rys. 3. Schemat blokowy *mastera*

4. JEDNOSTKA STERUJĄCA PROCESORA *MASTER*

Zadanie jednostki sterującej rozdzielono na następujące zadania podrzędne:

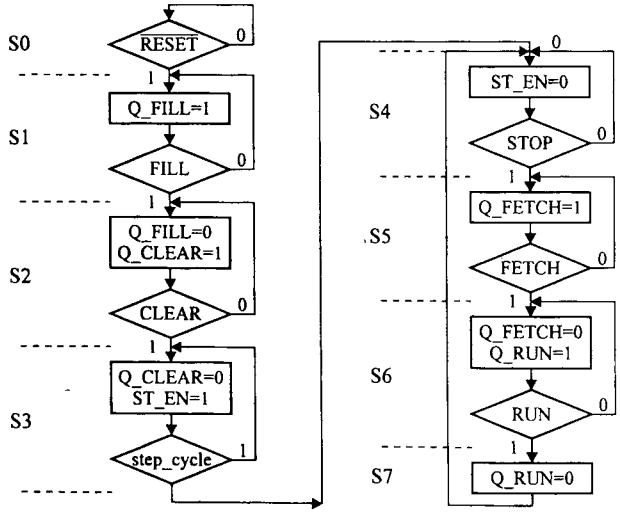
- FILL - przepisanie początkowych stanów wewnętrznych bloków z ROM do RAM
- CLEAR - zerowanie wyjść bloków w RAM
- FETCH - pobranie rozkazu z pamięci programu ROM do rejestrów
- PUT_ID - przesłanie identyfikatora bloku funkcyjnego z procesora *master* do *slave*
- PUT - przesłanie danej wejściowej z *mastera* do *slave'a*
- GET - odebranie danej wyjściowej ze *slave'a* przez *master*
- STEP - wykonanie rozkazu `STEP` kończącego cykl obliczeń.

Zadania FILL i CLEAR są wykonywane tylko na początku i wiążą się z przygotowaniem otoczenia procesora (inicjacja pamięci). FETCH pobiera kolejny rozkaz. Na podstawie jego kodu wykonywane jest alternatywnie jedno z zadań: PUT_ID, PUT, GET, STEP. Zarządzanie zadaniami prowadzi zadanie nadrzędne MAIN.

Każde zadanie opisano siecią działań. Sieć jest realizowana przez układ rozdzielacza warunkowego Moore'a [3]. W efekcie powstaje kilka rozdzielaczy o niewielkiej liczbie węzłów. Jako przykład rozważmy sieć działań dla układu MAIN, pokazaną na rys. 4. Dopóki sygnał $\overline{RESET} = 0$, MAIN znajduje się w stanie S0. $\overline{RESET}$ zeruje licznik w układzie rozdzielacza. Powoduje to, że układ z każdego stanu może przejść do S0 (nie pokazano tego na rys. 4).

Jeśli $\overline{RESET} = 1$, następuje przejście do stanu S1 i jest wytwarzany sygnał $Q\_FILL = 1$. Uaktywnia on układ FILL (wykonanie zadania FILL). Równocześnie MAIN bada sygnał FILL wytwarzany przez układ FILL, który informuje o statusie zadania (FILL = 0 - aktywne, FILL = 1 - koniec). Zakończenie zadania FILL powoduje przejście MAIN do S2. Sygnał Q_FILL jest zerowany i ustawiany Q_CLEAR, który z kolei uruchamia układ CLEAR. Proces przebiega analogicznie jak w S1.

W stanie S3 następuje oczekiwanie na pierwszą po $\overline{RESET} = 1$ zmianę sygnału *step_cycle*. Uaktywnia się sygnał ST_EN blokujący ustawienie przerzutnika błędu. Po zmianie *step_cycle* z 1 na 0, MAIN przechodzi do S4.



Rys. 4. Sieć działań układu MAIN

W S4 jest kontrolowany stan przerzutnika błędu i w przypadku jego ustawienia (STOP=0), przejście do S5 nie jest możliwe, bo przerzutnik może być wyzerowany tylko przez operatora ($\overline{RESET} = 0$).

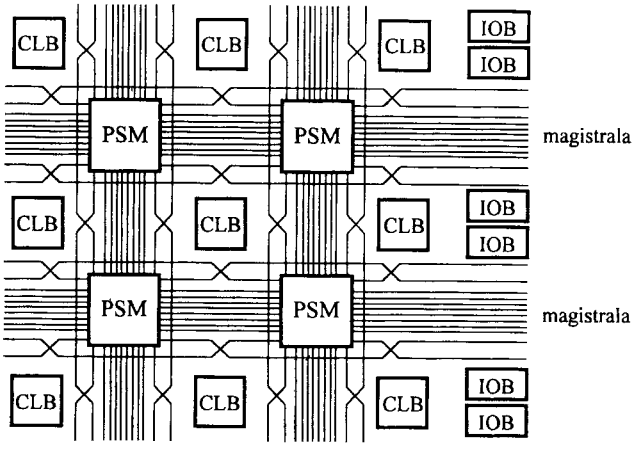
W stanie S5 staje się aktywny układ FETCH, który pobiera rozkaz. Wykonanie następuje w S6. Sygnał Q_RUN = 1 uruchamia jeden z układów podrzędnych: PUT_ID, PUT, GET, STEP. Wybór jest dokonywany na podstawie kodu rozkazu.

Z węzła S7 następuje skok bezwarunkowy do S4. Jeśli układ nie zostanie zatrzymany w S4 i cykl się powtarza. Przejścia przez S5, S6 powodują wykonywanie kolejnych rozkazów.

W analogiczny sposób zostały opisane pozostałe zadania, a następnie zbudowane układy do ich wykonywania. W rezultacie otrzymano kompletny schemat procesora, który stał się podstawą praktycznej realizacji.

5. STRUKTURA FPGA DLA REALIZACJI PROCESORA

W pracy wykorzystano struktury firmy Xilinx serii XC4000 [6]. Takie cechy jak regularna budowa, pełna przewidywalność zależności funkcjonalnych i czasowych, są wspólną cechą wszystkich struktur FPGA. Użytkownik otrzymuje gotową strukturę logiczną i łączeniową, dla której przydziela funkcje.



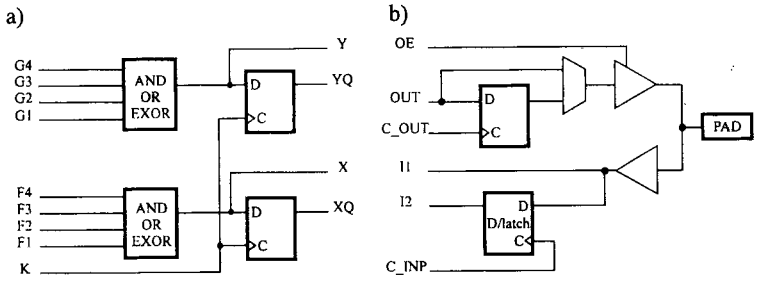
Rys. 5. Struktura układu FPGA

Do dyspozycji są 3 rodzaje elementów, pokazane na rys. 5:
- konfigurowalne bloki logiczne - CLB (ang. *Configurable Logic Block*)
- bloki wejścia/wyjścia - IOB (ang. *Input/Output Block*)
- magistrale i programowe matryce przełączające - PSM (*Programmable Switch Matrix*).

Bloki CLB zawierają elementy funkcjonalne do projektowania logiki. Blok IOB jest interfejsem pomiędzy wewnętrzną logiką, a wyprowadzeniem zewnętrznym (*pin*). Magistrale i PSM umożliwiają wykonanie połączeń między CLB oraz IOB.

CLB jest blokiem wielofunkcyjnym posiadającym:
- 2 niezależne 4-wejściowe generatory funkcyjne
- 2 przerzutniki typu D
- 2 wyjścia kombinacyjne i 2 przerzutnikowe (rys. 6a).

Blok CLB może być także użyty jako szybka pamięć RAM o pojemności 32*1 bit lub 16*2 bity. Posiada wbudowany mechanizm szybkiej generacji przeniesień arytmetycznych wykorzystywany w akumulatorach, licznikach i itp. Układy XC4000 posiadają od 64 do 1024 bloków. Dla orientacji można podać, że 24-bitowy akumulator zajmuje 13 CLB, a 16-bitowy licznik binarny z wpisem równoległym - 8 CLB (16 przerzutników).



Rys. 6. Elementy konfigurowalne: a) CLB, b) IOB

Wewnątrz bloku IOB (rys. 6b) znajdują się: przerzutnik wejściowy typu D lub zatrzask, przerzutnik wyjściowy D, bufor trójstanowy. Blok może pracować jako wejściowy, wyjściowy lub dwukierunkowy. Wyjście może być obciążone prądem do 12 mA. Układy XC4000 mają od 64 do 256 IOB.

Pakietem do programowania Xilinx jest XACT (*Xilinx Automated CAE Tools*) [7]. Współpracuje on ze znanymi edytorami schematów (np. *Viewlogic*, *OrCAD*, *Mentor V8*) i umożliwia: edycję schematu, symulację funkcjonalną i czasową, automatyczne przydzielanie elementów układu występujących na schemacie do CLB/IOB, tworzenie połączeń między nimi oraz generowanie konfiguracji. Obszerna biblioteka bloków funkcjonalnych: arytmetycznych, rejestrów, liczników itd., umożliwia relatywnie szybkie tworzenie nowych układów.

Regularna budowa struktur FPGA, polegająca na zastosowaniu jednakowych bloków CLB, IOB i PSM, ułatwia analizę funkcjonalną i czasową. Łatwiej też dokonywać weryfikacji.

Należy zwrócić uwagę, że dane konfiguracyjne w Xilinx są tracone po wyłączeniu zasilania. Ich odtworzenie wymaga zastosowania dodatkowej pamięci ROM, w której one rezydują. Po włączeniu, struktura FPGA powinna automatycznie odczytać konfigurację.

LITERATURA

1. HALANG W.A., JUNG S.-K., KRÄMER B.J., SCHEEPSTRA J.J.: *A Safety Licensable Computing Architecture*. World Scientific, Singapore 1993.
2. HALANG W.A., ŚNIEŻEK M., TRYBUS L.: *An Industrial Programmable Logic Controller with Fail Safe Behaviour*. 12 Internationale Mittweidaer Fachtagung. Innovative Technologien, Mittweida 1996, 33-42.
3. MOLSKI M.: *Modułowe i mikroprogramowalne układy cyfrowe*. WKŁ, Warszawa 1986.
4. SCHUCK H.: *Analoger Fensterkomparator in Fail-safe-Technik*. Technische Universität Braunschweig, Braunschweig 1987.
5. Paul Hildebrandt GmbH & Co. KG: *Fail-Safe Electronic Controls - The HIMA-Planar-System*. Brochure TI 92.08. Brühl, 1992.
6. *The Programmable Logic Data Book*. Xilinx, San Jose 1994.
7. XACT-Software. *User Guide*. Xilinx, 1994-95.