tISC-16 - jak sobie zrobić mikroprocesor

Przedstawiamy projekt, do którego nie będziemy oferować płytki drukowanej ani podzespołów. Nie będzie w sprzedaży kitu, a wszystko co jest niezbędne do jego wykonania można sobie bezpłatnie pobrać z Internetu lub po prostu skopiować z naszej płyty. Chociaż może być przygotowany bezpłatnie, prezentowany projekt należy do awangardowych - jest to bowiem mikrokontroler RISC, opisany za pomocą języka VHDL. Na świecie takie wirtualne scalaki są coraz częściej stosowane.

Mikroprocesory są coraz częściej spotykanymi elementami systemów cyfrowych. Ich szerokie zastosowanie wynika przede wszystkim z ich elastyczności. Nie bez znaczenia jest również możliwość zmiany funkcji realizowanych przez układ już po jego wykonaniu. Obecnie dostępnych jest wiele mikroprocesorów, których możliwości można dostosować do wymagań aplikacji - począwszy od 4-bitowych mikrokontrolerów stosowanych w zabawkach i prostych systemach sterowania, aż po 32-bitowe procesory sieciowe takie jak np. 79RC32V334 firmy IDT.

Jednak nie każde zadanie może być wykonane przez uniwersalny mikrokontroler lub procesor, a w każdym razie nie zawsze można go wykonać opłacalnie. Przykładowo rozważmy tani oscyloskop o częstotliwości próbkowania 100 MHz. Jeżeli ma on cztery kanały o rozdzielczości 8 bitów, to obróbka generowanych danych wymaga utworzenia kanału dla odbioru danych o przepływności 3,2 Gb/s, następnie należy je przesłać do pamięci, co daje kolejne 3,2 Gb/s. Rzadko który procesor jest w stanie zapewnić taki transfer.

Dlatego istotną rolę w tego typu aplikacjach pełnią układy programowalne. Układ FPGA, który jest w stanie pełnić funkcję kontrolera oscyloskopu, nie jest zbyt drogi, a projekt takiego układu nie jest szczególnie skomplikowany. Z drugiej strony, takie zadania, jak późniejsze przetworzenie zebranych w ten sposób danych, są oczywiście łatwiejsze do implementacji w postaci programu dla procesora. Oznacza to, że bliskie optymalnemu wykonanie projektu będzie składało się z dwóch elementów: FPGA i kontrolera, co jednak nie zawsze jest możliwe do zaakceptowania, jeżeli miałoby się wiązać np. z zastosowaniem dwóch układów scalonych zamiast jednego. Takie problemy rozwiązano w uniwersalnych modułach *IP (Intellectual Property)*, czyli gotowych blokach funkcjonalnych, zawierających opisy w językach HDL (*Hardware Description Language*) na przykład kompletnej jednostki centralnej, UART-a, interfejs USB itp. Bloki te są obecnie przygotowane przez projektantów wyspecjalizowanych firm, którzy tworzą wirtualne moduły peryferyjne opisane za pomocą (najczęściej) języka VHDL lub Verilog. Zastosowanie *IP cores* znacznie upraszcza implementację rozbudowanych projektów w układach PLD, jak np. integrację procesora i bloków specjalizowanych w jednym układzie programowalnym.

Wybór architektury

Architektury spotykane powszechnie dzielą się na dwie kategorie:
- von Neumanna - taka, która ma wspólną pamięć danych i programu, a zatem wspólną szynę,
- harwardzka - opracowana przez zespół Howarda Aikena, o rozdzielonej pamięci programu i danych.

Architektura von Neumanna ma wiele zalet, dlatego jest stosowana w kontrolerach Texas Instruments MSP430. Pozwala optymalnie wykorzystać daną pamięć, a ponadto wymaga mniejszej liczby nóżek z obudowy.

Architektura harwardzka ma przewagę przy korzystaniu z FPGA: prostotę projektu. Istotnie, układ kontrolera dla procesora opartego na architekturze harwardzkiej jest prostszy, ponieważ wiadomo, że to, co przychodzi szyną programu, jest programem, a to, co pobierane jest szyną danych, należy do danych. Dodatkową zaletą jest ochrona przed

Blok IP (w postaci kodu VHDL) z opisem mikrokontrolera tISC-16 oraz narzędzia do jego symulacji i syntezy, a także makroasembler opublikowaliśmy na płycie CD-EP5/2002B.

skutkami błędnego skoku: w architekturze von Neumanna kończy się on niekiedy traktowaniem danych jako kodów instrukcji - rezultat jest opłakany.

Podstawową zaletą architektury harwardzkiej jest jednak szybkość. Możliwość równoczesnego dokonania pobrania kodu i danych likwiduje tak zwane „wąskie gardło von Neumanna“ *(von Neumann bottleneck*). Proponuję zatem skorzystanie z architektury harwardzkiej.

Zestaw instrukcji

Tak jak w innych dziedzinach życia, także w projektowaniu procesorów ogromną rolę odgrywa moda. W latach 70. i 80. prym wiodły niepodzielnie układy CISC, czyli układy o liście rozkazów zawierającej także instrukcje złożone (*Complex Instruction Set Computer*), wymagające wielu taktów zegarowych. Ukoronowaniem tej linii rozwojowej był iAPX432 - procesor, który był wspaniałym osiągnięciem zarówno pod względem układowym, jak i technologicznym. Ponieważ jego budowę zoptymalizowano pod kątem bezpośredniego wykonywania programów napisanych w języku ADA, procesor ten szybko zniknął z rynku.

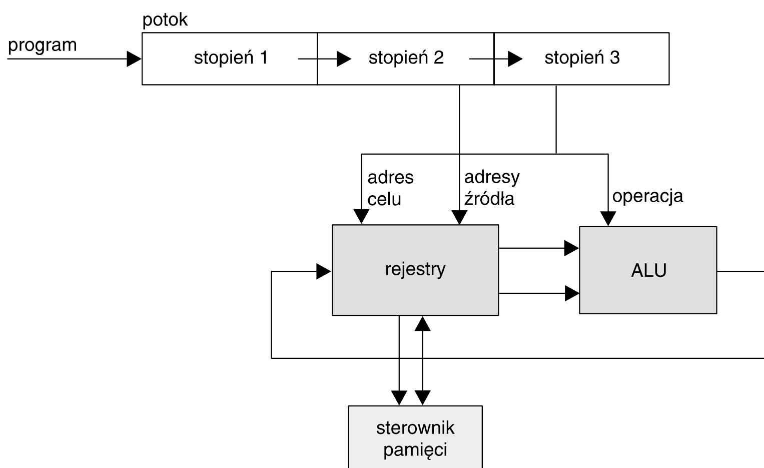
Do grupy procesorów CISC należą układy serii x86, aczkolwiek nowsze projekty oparte są na wewnętrznej architekturze RISC i jest ona jedynie „tłumaczona“ sprzętowo na CISC. Innymi przedstawicielami tej grupy układów są układy rodziny Motorola 68000.

Przeciwieństwem rdzeni CISC są rdzenie typu RISC (*Reduced Instruction Set Computer*), które charakteryzują się zredukowaną liczbą instrukcji. Dzięki temu budowa rdzenia procesora jest znacznie prostsza (a więc tańsza), możliwe jest szybsze wykonywanie programu (do wykonania większości instrukcji niezbędny jest tylko jeden takt zegara), łatwiejsza jest także optymalizacja kodu wynikowego.

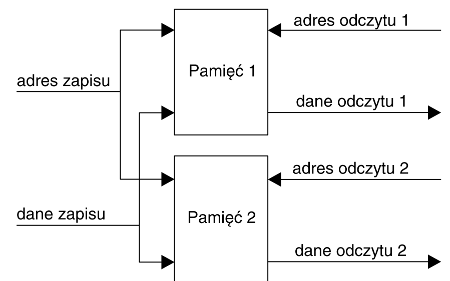
Ponieważ cena układów PLD jest zależna przede wszystkim od ich zasobów logicznych, podczas projektowania własnego procesora warto ograniczyć listę instrukcji do zestawu niezbędnego dla realizacji zamiarów użytkownika. Opracowany przeze mnie procesor realizuje tylko dwa rozkazy. Tak znaczne uproszczenie było możliwe dzięki wykorzystaniu faktu, że za ich pomocą można wykonać praktycznie dowolne działanie arytmetyczne i logiczne - łatwo dowieść, że do realizacji każdej funkcji przełączającej wystarczy bramka NAND, łącznie z operacjami arytmetycznymi ADD (dodawanie). I choć jest to na pierwszy rzut oka niespodziewane, większość operacji typowych dla maszyn CISC nie zajmuje więcej niż 4 operacje w tej implementacji.

Zestaw rejestrów

Wiele dawniejszych procesorów miało architekturę akumulatorową - wyróżniony rejestr zwany akumulatorem, był jedynym miejscem, w którym można było poddawać dane obróbce i do którego wpisywane były rezultaty operacji. Innym możliwym, lecz rzadko stosowanym, rozwiązaniem jest zastosowanie architektury stosowej, w której parametry do rozkazu pobierane są z wierzchołka stosu, a wynik jest odkładany z powrotem na stos.



Rys. 1. Schemat blokowy procesora tISC-16



Rys. 2. Wyjaśnienie zasady mirroringu

Jednak zdecydowałem, że najbardziej celowe jest wykorzystanie w pełni symetrycznej architektury rejestrowej, w której parametry mogą być pobrane z dowolnie wyznaczonych rejestrów, a wynik może zostać zapisany w każdym rejestrze.

Dla prostoty zrezygnowałem z bezpośredniego adresowania pamięci danych. Szyna adresowa pamięci danych jest sterowana wyłącznie przez jeden z rejestrów (rejestr adresowy), a szyna pamięci danych odpowiada innemu rejestrowi. Odczyty z tego rejestru są tłumaczone jako odczyty z pamięci, a zapisy do rejestru jako zapisy do pamięci. Inny rejestr specjalny pełni funkcję rejestru stałej. Pobranie z tego rejestru dostarcza następnego słowa z pamięci kodu. Jest to ustępstwo na rzecz architektury von Neumanna.

Rejestr warunku określa czy należy wykonywać dodawanie. Jeżeli liczba zawarta w tym rejestrze jest niezerowa, dodawanie powinno być wykonane. W przeciwnym przypadku rezultat operacji dodawania nie jest zapisywany do rejestru docelowego. Instrukcje NAND są wykonywane zawsze, niezależnie od zawartości rejestru warunku.

Szerokość magistral

Ponieważ w systemie dostępne są tylko dwie instrukcje, do ich zakodowania wystarczy 1 bit. Dogodna w typowych zastosowaniach liczba rejestrów to 32, czyli na przechowanie numeru rejestru

Listing 1. Rejestry, zdefiniowane przez system

```
.equ SP @23  ; wskaźnik stosu
.equ #F @22  ; stała 65535
.equ #1 @21  ; stała 1
.equ TR @20  ; rejestr tymczasowy
```

Listing 2. Przykładowe operacje arytmetyczne
```
; kopiowanie danej
.macro MOV  mov.dest, mov.src
       ADD  mov.dest, mov.src, #0
.endm  MOV

; odejmowanie
.macro SUB  sub.dest, sub.src1, sub.src2
       NAND TR,       sub.src2, sub.src2
       ADD  TR,       TR,       #1
       ADD  sub.dest, sub.src1, TR
.endm  SUB
```

potrzeba 5 bitów. Każda instrukcja numeruje trzy rejestry: dwa źródłowe i docelowy. Zatem szerokość słowa kodu wynosi 16 bitów. Dla symetrii, szerokość ścieżki danych również jest ustalona na 16 bitów.

Implementacja

Prezentowany procesor można zrealizować jako automat mikroprogramowany, czyli zawierający dodatkową pamięć z opisem poszczególnych operacji wykonywanych przez procesor (aktywuj zatrzask, wyślij adres, wyprowadź impuls na nóżkę RD itp.) - jednak przy dwóch instrukcjach i architekturze harwardzkiej nie ma to zbyt wielkiego sensu.

Przyjęta przeze mnie architektura bardzo ułatwia realizację przetwarzania potokowego, w którym podczas wykonywania jednej instrukcji następuje już pobranie i wykonywanie kolejnej. Udało się osiągnąć dobry rezultat, czyli jedno pobranie słowa z pamięci programu w cyklu zegarowym.

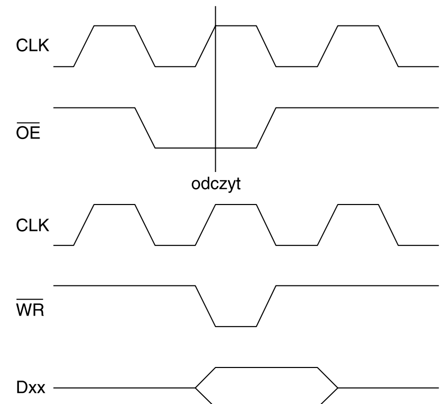
Specyfikacja

Na podstawie powyższych rozważań, można sformułować następujące założenia projektowe:
- architektura harwardzka, 16 bitów danych i adresu na obu szynach,
- 4-stopniowy potok, 1 pobranie z pamięci programu na cykl zegara,
- dostępne 2 operacje:

Kod	Operacja
0	NAND rd,rs1,rs2 zapisz do rejestru rd negację iloczynu logicznego rejestrów rs1 i rs2;
1	ADD rd,rs1,rs2 jeżeli rejestr CR (warunku) ma wartość niezerową, zapisz do rejestru rd wynik arytmetycznego dodawania rejestrów rs1 i rs2;

- 32 rejestry, z czego osiem specjalnych:

Kod	Rejestr	Opis
0..23		@0..@23 rejestry ogólnego przeznaczenia
24	PD	rejestr danych (odpowiada komórce pamięci zaadresowanej przez rejestr PA)
25	PA	adres pamięci danych
26	IP	wskaźnik instrukcji, inkrementowany co cykl
27	CR	rejestr warunku
28	imm	rejestr stałej - jeżeli jest on wybrany jako rejestr źródłowy, procesor pobiera kolejne słowo z pamięci programu i używa go jako liczby
29	XA	adres przestrzeni we/wy
30	XD	dane przestrzeni we/wy (na takiej zasadzie jak PD)
31	#0	stała zero



Rys. 3. Typowe przebiegi występujące podczas operacji dostępu do pamięci danych

Szczegóły wykonania

Na **rys. 1** przedstawiono schemat blokowy procesora tISC-16, zaprojektowanego przeze mnie w oparciu o powyższe założenia. Ponieważ niektóre układy FPGA,

Listing 3. Kontrola programu
```
; czyszczenie rejestru warunku
.macro CCR
       NAND CR,  #0,  #0
.endm  CCR

; skok
.macro JMP  jmp.addr
       ADD  IP,  #0,  jmp.addr
.endm  JMP

; wykonaj jeśli suma niezerowa
.macro IFSNZ  ifsnz.one,  ifsnz.two
       ADD    CR,         ifsnz.one,  ifsnz.two
.endm  IFSNZ

; wykonaj jeśli suma zerowa
.macro IFSZ   ifsz.one, ifsz.two
       ADD    TR,       #0,       #0
       IFSNZ  ifsz.one, ifsz.two
       ADD    TR,       #F,       #0
       NAND   CR,       TR,       #F
.endm  IFSZ

; wywołanie procedury
; zauważ, że wszystkie operacje tutaj przekładają się na ADD, a zatem
; można wywoływać podprogramy warunkowo
.macro CALL call.addr
       ADD  PA,  SP,  #0
       ADD  PD,  IP,  4
       ADD  SP,  SP,  #1
       ADD  IP,  call.addr,  #0
.endm  CALL

; powrót z procedury
; także i tutaj możliwe jest wykonanie warunkowe
.macro RET
       POP  IP
       NOP
.endm  RET
```

Listing 4. Przykładowy program

```
.INCLUDE TISC.AI

  MOV @0, 1
  MOV XA, 0
  MOV XD, @0
  INC XA
  MOV XD, 65535
  DEC XA
LOOP:
  ADD @0, @0, @0
  MOV XD, @0
  JMP LOOP
```

w szczególności układy produkowane przez firmę QuickLogic, zawierają specjalizowane bloki pamięci, koszt implementacji rejestrów jest niski. W związku z tym celowym wydaje się skorzystanie z mechanizmu mirroringu, którego zasadę wyjaśniono na **rys. 2**. Polega on na stworzeniu dwóch bliźniaczych bloków pamięci, do których zapis jest równoczesny (tak, że dane w nich zawarte są zawsze identyczne), natomiast odczyty są niezależne. Dzięki temu można w jednej chwili pobrać zawartość obydwu rejestrów źródłowych i zaoszczędzić czas.

Potok pracuje następująco:

CLK
- *IP-1* pobranie rejestrów źródłowych; obliczenia
  zapis do rejestru docelowego
- *IP* pobranie kodu operacji albo wyzerowanie przy korzystaniu z imm
  dekodowanie instrukcji; podanie adresów rejestrów, pobranie zawartości rejestrów źródłowych; obliczenia
  zapis do rejestru docelowego
- *IP+1* pobranie kodu operacji albo wyzerowanie przy korzystaniu z imm
  dekodowanie instrukcji; podanie adresów rejestrów

Szczególnego objaśnienia wymaga pobranie kodu operacji. Otóż niektóre instrukcje korzystają z rejestru 28 „imm", co oznacza, że następujące po nich słowo jest słowem danych, a nie kodu. W takim przypadku należy wstawić do potoku instrukcję `ADD @31,@31,@31`, co odpowiada brakowi operacji. Ta poprawka jest wykonywana właśnie podczas pobierania kolejnego słowa.

Interfejs pamięci danych jest jedną z bardziej złożonych części projektu. Konsekwencją dostosowania się do typowych pamięci SRAM jest długość trwania operacji zapisu i odczytu. Z tego powodu dwa kolejne rozkazy nie mogą być rozkazami dostępu do pamięci danych. Typowe przebiegi przedstawiono na **rys. 3**.

Procesor opisałem w języku VHDL. Dzięki temu, mimo przystosowania do architektury QuicLogic pASIC3 (np. korzystania z wbudowanych bloków RAM), można go przenieść także na wiele innych FPGA z bardzo małymi zmianami.

W układzie QL4036 można wykonać procesor, który jest w stanie przełączać się między dwoma maszynami wirtualnymi (zestawami rejestrów) bez opóźnienia. Sygnałem na jednej z nóżek układu wybierana jest aktywna maszyna. Dzięki temu można realizować przerwania, i to w czasie zerowym. Podczas gdy ostatnia instrukcja działająca w środowisku starej maszyny opuszcza potok, już jest wczytywana instrukcja dla nowej maszyny. Pozwala to na przyspieszenie przetwarzania w czasie rzeczywistym oraz odpowiadanie na sytuacje krytyczne w bardzo krótkim czasie.

Programowanie

Ponieważ dostępne są tylko dwie instrukcje, pisanie programu jest dosyć uciążliwe - dlatego dla procesora tISC istnieje makroasembler. Przykłady makr znajdują się na **list. 1**, **list. 2** i **list. 3**.

Jak się okazuje, nawet tak złożona operacja jak CALL zajmuje zaledwie 4 instrukcje i typowo 5 cykli zegara. Jest to dobry wynik, zważywszy, że w układzie 8051 LCALL zajmuje 24 cykle. Mnożenie zajmuje w tym układzie 64 cykle (jest to mnożenie 16-bitowe). Mnożenie 8-bitowe zajmuje 32 cykle, a więc również mniej niż w 8051 (który wykonuje je w 48 cyklach). Podczas wykonywania większej liczby mnożeń, po kolei można zmniejszyć czas ich trwania o 25% magazynując wykorzystywane stałe w rejestrach. Możliwości tego procesora dowodzą słuszności zastosowania architektury RISC.

Do kompilacji modułu IP procesora tISC-16 niezbędny jest system projektowy z obsługą języka VHDL. Może to być jeden z bezpłatnie udostępnianych pakietów, jak np.: Quartus II (Altera), WebPack ISE (Xilinx), Warp (Cypress) czy też QuickWorks (QuickLogic). Na płycie CD-EP5/2002B publikujemy m.in. 30-dniową, najnowszą wersję pakietu QuickWorks, za pomocą którego opracowano mikrokontroler prezentowany w artykule.

Jeżeli potrzebne jest szybsze mnożenie, można dołączyć układ sprzętowego mnożenia do szyn XA i XD, otrzymując wynik mnożenia w 5-ciu cyklach zegara, koniecznych do zapisania danych i odczytania wyniku.

Osiągi procesora

Zgodnie z symulacjami, układ może pracować z zegarem 25 MHz, co daje ok. 22 MIPS dla typowego programu. Adresowana pamięć danych wynosi 128 kB, a pamięć programu - 65536 instrukcji. Wąskim gardłem jest szybkość pracy pamięci programu, ponieważ pamięci stałe typu EPROM nie mogą zapewnić czasu odczytu 40 ns. Realistyczna szybkość pracy nie przekracza więc 12 MHz dla pamięci Flash typu Fast Boot, Block firmy Intel (80 ns).

List. 5. Procesor tISC16 w akcji

```
   k:4.0*  @0000=6FFF  ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
   k:5.0 @0001=5BFF  ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
 .....................................................
 .....................................................
 .....................................................
   k:15.0  @000B=FFFF  ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
   k:16.0  @000C=F7B6  ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
   k:17.0  @000D=8000  ZZZZZZZZZZZZZZZZ0000000000000001
   k:18.0  @000E=F81F  ZZZZZZZZZZZZZZZZ0000000000000001
   k:19.0  @000F=EBFC  ZZZZZZZZZZZZZZZZ0000000000000001
   k:20.0  @0010=000D  ZZZZZZZZZZZZZZZZ0000000000000001
   k:21.0  @000D=8000  ZZZZZZZZZZZZZZZZ0000000000000010
   k:22.0  @000E=F81F  ZZZZZZZZZZZZZZZZ0000000000000010
   k:23.0  @000F=EBFC  ZZZZZZZZZZZZZZZZ0000000000000010
   k:24.0  @0010=000D  ZZZZZZZZZZZZZZZZ0000000000000010
   k:25.0  @000D=8000  ZZZZZZZZZZZZZZZZ0000000000000100
   k:26.0  @000E=F81F  ZZZZZZZZZZZZZZZZ0000000000000100
   k:27.0  @000F=EBFC  ZZZZZZZZZZZZZZZZ0000000000000100
   k:28.0  @0010=000D  ZZZZZZZZZZZZZZZZ0000000000000100
   k:29.0  @000D=8000  ZZZZZZZZZZZZZZZZ0000000000001000
```

Wyniki symulacji

Aby zademonstrować procesor tISC w działaniu, przygotowałem krótki program (**list. 4**), wyprowadzający jedynkę logiczną na kolejne nóżki wejścia - wyjścia.

Na **list. 5** znajduje się plik pochodzący z kilku cykli symulacji.

Pierwsza kolumna powyższego listingu zawiera numer cyklu (gwiazdka oznacza aktywny sygnał RESET). Pierwsza liczba szesnastkowa (po znaku @) to adres pamięci programu, natomiast kolejna (po znaku równości) oznacza odczytany kod programu. Następnie wypisywane są stany nóżek we-wy. Jak widać, program jest realizowany poprawnie.

Na listingu można zauważyć wpływ przetwarzania potokowego. Na przykład prawe 16 wejść-wyjść jest ustawianych w tryb wyjścia w cyklu 17.0, mimo że instrukcja, która za to odpowiada, została pobrana w cyklach 14.0 i 15.0 (FB9F FFFF, czyli ADD XD, 65535, #0 - rozwinięte makro MOV XD, 65535).

Podsumowanie

Przedstawiony kontroler łączy dużą wydajność obliczeniową z małym zużyciem zasobów. Jego uniwersalna architektura pozwala zaimplementować każdy algorytm. Wiele zadań, które do tej pory wymagało użycia osobnych układów scalonych, może być zintegrowanych w jednym układzie. Ponadto, szyna X umożliwia łatwe rozszerzenie procesora o elementy peryferyjne, co ułatwia łączenie procesora z pozostałymi zasobami struktury programowalnej.

Procesor tISC może znaleźć zastosowanie także tam, gdzie wykorzystywane są układy FPGA programowalne w systemie. Do zamontowanego na płytce FPGA można załadować blok procesora i w ten sposób wykonać diagnostykę układów podłączonych do struktury programowalnej.

Architektura procesora tISC jest przejrzysta i ułatwia programowanie. Ważną jej cechą jest także łatwość rozszerzania. 32-bitowa wersja wspierająca pracę SIMD (jedna instrukcja - wiele danych) jest w przygotowaniu.

Stanisław Skowronek

Dodatkowe informacje

Dodatkowe informacje, opis źródłowy (VHDL) oraz oprogramowanie do symulacji i syntezy prezentowanego układu można znaleźć na płycie CD-EP5/2002B oraz w Internecie pod adresem:
- http://www.et.put.poznan.pl/~skowron/tisc/,
- http://www.symphonyeda.com/Downloads/VHDLSimili20b21.exe,
- http://www.symphonyeda.com/Downloads/Simili20b21b-linux-x86.tar,
- http://www.cmosexod.com/.