Adam ZIĘBIŃSKI, Stanisław ŚWIERC
POLITECHNIKA ŚLĄSKA W GLIWICACH

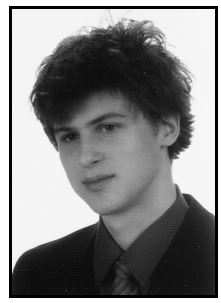
Implementacja parametryzowanego procesora MIPS w układach reprogramowalnych

Adam ZIĘBIŃSKI

Ukończył studia w 1996 r. w Instytucie Informatyki na Wydziale Automatyki Elektroniki i Informatyki Politechniki Śląskiej. Na tym samym wydziale rozpoczął pracę jako asystent w 1996 r. Pracę doktorską obronił w 2002 r. Obecnie pracuje na stanowisku adiunkta w Instytucie Informatyki w Zakładzie Urządzeń Informatyki. Jego zainteresowania to sprzętowo-programowe projektowanie koprocesorów problemowo zorientowanych z wykorzystaniem układów reprogramowalnych VLSI.

e-mail: adam.ziebinski@polsl.pl

Stanisław ŚWIERC



Student czwartego roku kierunku Informatyka na Wydziale Automatyki Elektroniki i Informatyki Politechniki Śląskiej. Kształcił się również w Technical University of Denmark Lyngby Dania jako stypendysta programu Erasmus w latach 2007/2008. Od 2 lat zajmuje się technologią FPGA.

e-mail: stanislaw.swierc@gmail.com

Streszczenie

W pracy przedstawiono projekt systemu wbudowanego zrealizowanego w układzie FPGA. Sercem systemu jest rdzeń procesora wzorowanego na procesorach architektury MIPS. Procesor ten został zaimplementowany w języku VHDL w taki sposób, by podczas syntezy jego lista rozkazów była ograniczona do rozkazów obecnych w pamięci programu. W efekcie wykonany procesor nie będzie posiadał logiki, która nie będzie wykorzystywana. Takie rozwiązanie pozwala zaprojektować system wbudowany, który ma mniejsze zapotrzebowanie na zasoby sprzętowe matrycy programowalnej, co dodatkowo powinno umożliwić zwiększenie szybkość jego działania.

Słowa kluczowe: systemy wbudowane, FPGA, MIPS, VHDL.

The VHDL implementation of a reconfigurable MIPS processor

Abstract

The paper presents a project of an embedded system realization on a FPGA array. The core element is a simplified MIPS processor [1, 2, 4] implemented in the VHDL in the way that its instruction set can be reduced to the set of instructions present in the program memory. After completing the processors datapath design, it is analyzed in order to determine which modules take part in execution of certain instructions. Knowing the dependencies between the instructions and the modules, it is possible to show how the processor should be built if it has to support a specific subset of instructions. Conditional synthesis is not what the common HDL languages offer [7]. Nevertheless, it was noticed that at the optimization stage of the synthesis all IF statements in which the condition value is known and it is false are omitted. This feature was used to regulate the hardware organization. Figure 3 presents how a single boolean parameter can regulate the XOR instruction support in the ALU. Initially, all parameters had to be set manually. It was error-prone. Therefore a new entity integrating the CPU and program memory was introduced. It can accept the byte-code, analyze it, and adjust the supported instruction set during the synthesis (Figs. 4 and 5). This solution yields a device that requires fewer system gates to be synthesized and has a potential to increase the maximal operational frequency.

Keywords: embedded systems, FPGA, MIPS, VHDL.

1. Wstęp

Mianem systemu wbudowanego określa się system informatyczny specjalnego przeznaczenia, stanowiący integralną częścią obsługiwanego przez niego sprzętu, wykonujący z góry określone zadanie.

Systemy wbudowane pomimo swojej różnorodności cechują się zapotrzebowaniem na moc obliczeniową [3]. Niezbędna jest obecność w systemie procesora lub mikroprocesora w zależności od skali urządzenia. W przypadku matryc programowalnych typu FPGA (Field Programmable Gate Array), można ten problem rozwiązać przynajmniej na 3 sposoby:

- wykorzystać jedną ze współcześnie produkowanych matryc zawierającą zintegrowany rdzeń procesora (Virtex firmy Xilinx - Power PC 405, FPSlic firmy Atmel – AVR),
- wykorzystać rdzeń w postaci gotowego komponentu wirtualnego (IP),
- samodzielnie zaimplementować procesor zdolny wykonać wymagane zadanie.

Ostatnie rozwiązanie choć wymaga największych nakładów pracy, może potencjalnie umożliwić uzyskanie najlepszych efektów, poprzez dostosowanie procesora do wyznaczonego zadania.

Jednym z możliwych sposobów osiągnięcia tego celu jest zredukowanie zasobów sprzętowych tak, by powstały produkt był w stanie wykonać tylko rozkazy zawarte w konkretnym algorytmie zamiast wszystkich instrukcji ze standardowej listy. Typowo takie podejście byłoby niedopuszczalne, ale w przypadku systemów wbudowanych jest jak najbardziej uzasadnione.

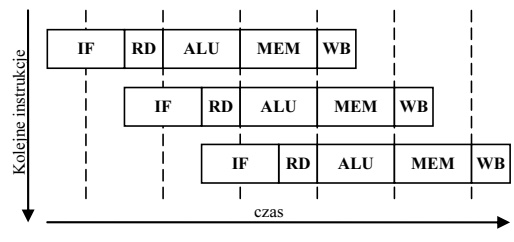
Ta myśl stała się inspiracją do przeprowadzenia badania nad możliwościami implementacji w VHDL (Very High Speed Integrated Circuits Hardware Description Language) [5, 9] rdzenia procesora o zmiennej liście rozkazów, którą będzie można określić za pomocą zestawu parametrów.

2. Projekt

Do badań wykorzystano istniejącą architekturę MIPS, która od powstania w latach osiemdziesiątych uważana jest za najbardziej eleganckie rozwiązanie spośród architektur RISC [1, 2]. Procesory te odniosły największy sukces w zastosowaniach wbudowanych. Powodzenie to zawdzięczane jest optymalizacji procesora pod kątem potokowego przetwarzania danych.

W podstawowej wersji, która została wybrana do realizacji badań, potok posiada pięć poziomów (rys. 1).

Poszczególne poziomy potoku realizują następujące funkcje:
IF - instruction fetch - pobranie instrukcji z pamięci programu,
RD - read register - odczytanie danych z banku rejestrów,
ALU - arithmetic/logic unit - wykonanie operacji arytmetyczne i logiczne na podanych argumentach,
MEM - memory - zapis i odczyt danych z pamięci danych,
WB - write back - zapisanie wyników do banku rejestrów.

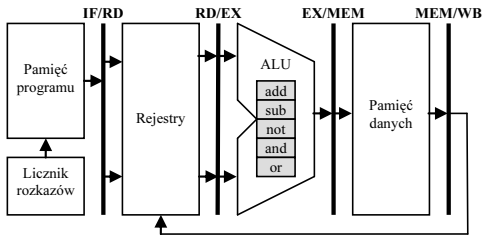


Rys. 1. Schemat potoku procesora MIPS
Fig. 1. MIPS pipeline schema

Projekt procesora wykonano na podstawie specyfikacji udostępnianej przez MIPS Technologies [4] oraz opisu architektury MIPS autorstwa Johna L. Hennessy [1].

Proponowane przez nas parametryzowanie procesora polega na ograniczeniu listy rozkazów do instrukcji sterujących wykonaniem programu oraz wykorzystywanych operacji arytmetycznych i logicznych. Takie podejście pozwoli na zmniejszenie zapotrzebowania procesora na zasoby sprzętowe matrycy. Dodatkowo umożliwi syntezę urządzenia w mniejszych układach FPGA.

Na etapie projektowania systemu wyróżniono moduły procesora, które są niezbędne do wykonania określonego rozkazu. Znając zależności między rozkazami i modułami można określić z jakich elementów powinien składać się procesor by był w stanie wykonywać zadaną listę rozkazów. Rysunek 2 przedstawia uproszczony schemat procesora (moduły dołączane warunkowo zacieniowane).



Rys. 2. Uproszczony schemat procesora MIPS
Fig. 2. Simplified MIPS datapath

MIPS implementuje architekturę harwardzką. Posiada dwie odrębne magistrale na instrukcje i dane. Aby móc się porozumieć z takim procesorem niezbędne jest albo dodanie pamięci podręcznej, która pośredniczyłaby pomiędzy pamięcią, albo dołączenie dwóch osobnych modułów pamięci.

Ponieważ typowo w systemach wbudowanych raz załadowany program z reguły nie ulega zmianie [3], zdecydowano się na drugie rozwiązanie. Ponadto moduł pamięci programu został zintegrowany z procesorem tworząc nową encję. Dzięki temu zabiegowi kod bajtowy programu jest dostępny podczas syntezy procesora i może zostać wykorzystany do jego konfiguracji.

Zadanie to realizowane jest w następujących krokach:
- pobranie kodu bajtowego z zewnątrz[1],
- przekazanie kodu programu do modułu pamięci,
- przeprowadzenie analizy programu,
- określenie listy wykorzystanych instrukcji,
- konfiguracja rdzenia procesora.

3. Implementacja

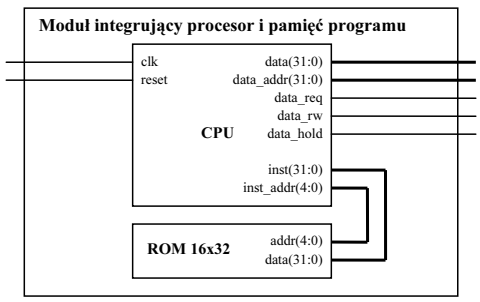
Uwzględniając założenia projektowe procesor opisano w VHDL tak, by mógł korzystać z dwóch osobnych modułu pamięci. Pamięć programu została zbudowana w oparciu o komórki ROM. Natomiast modułu pamięci danych to hybryda składająca się z komórek RAM oraz rejestrów kontrolerów urządzeń peryferyjnych zamapowanych do przestrzeni adresowej danych.

W VHDL nie ma mechanizmu znanego z języka programowania ”C ”jako kompilacja warunkowa. Wszystko co zostanie zapisane w pliku źródłowym zostanie zinterpretowane przez narzędzie syntezujące. Niezbędne było więc znalezienie rozwiązania na obejście tego problemu. Zauważono, że podczas syntezy pomijane są wyrażenia umieszczone w bloku warunkowym IF dla którego warunek jest znany na etapie syntezy i ma wartość fałszu.
Wprowadzono więc do procesora szereg parametrów typu boolowskiego określających czy dana instrukcja ma być obsługiwana przez powstały procesor (rys. 3).

Początkowo wszystkie parametry musiały zostać określone ręcznie jeszcze przed przystąpieniem do syntezy [10], co niestety sprzyjało powstawaniu błędów. Aby wyeliminować ten problem utworzono nową encję zawierającą w sobie procesor, która jest odpowiedzialna za analizę zadanego kodu programu na etapie syntezy. Pamiętając, że kolejnym modułem, do którego musi trafić kod źródłowy, jest pamięć programu, zdecydowano się na zintegrowanie go z procesorem w nowo powstałej encji.

```
ARCHITECTURE behaviour OF cpu IS
 --sygnał określający operacje do wykonania przez ALU
 SIGNAL func   : std_logic_vector(5 DOWNTO 0);
--argumenty oraz produkt ALU
 SIGNAL alu_arg1 : std_logic_vector(32 DOWNTO 0)
 SIGNAL alu_arg2 : std_logic_vector(32 DOWNTO 0);
 SIGNAL alu_prod : std_logic_vector(32 DOWNTO 0);
[...]
BEGIN
[...]
 --przetwarzanie instrukcji
 CASE func IS
  [...]
  WHEN func_xor =>
   --czy instrukcja XOR ma być obsługiwana
   IF inst_xor THEN
        alu_prod <= alu_arg1 XOR alu_arg2;
   ELSE
        alu_prod <= (OTHERS=>'-');
   END IF;
  WHEN OTHERS =>
   alu_prod <= (OTHERS=>'-');
 END CASE;
[...]
END behaviour;
```

Rys. 3. Synteza warunkowa
Fig. 3. Conditional synthesis



Rys. 4. Interfejs modułu integrującego procesor i pamięć programu
Fig. 4. MIPS datapath

Chcąc wykorzystać ten moduł w systemie wystarczy przekazać mu kod programu w zestawie parametrów zupełnie tak, jakby była to zwykła pamięć ROM.

Zmuszenie narzędzia do syntezy by przeprowadziło analizę kodu programu osiągnięto poprzez zdefiniowanie szeregu funkcji boolowskich określających obecność danej instrukcji (rys. 5). Mechanizm ten działa tylko w przypadku, kiedy wszystkie argumenty mają ustalone wartości na etapie syntezy.

```
ENTITY Cpu_mem IS
 GENERIC(
  --zawartość komórki pamięci pod adresem "0000"
  addr_00000 : word_t := word_dont_care;
  [...]
);
[...]
ARCHITECTURE structure OF Cpu_mem IS
[...]
 CPU_comonent: cpu
  GENERIC MAP(
   inst_xor =>
    --czy komórka "0000" zawiera instrukcję XOR
    (addr_00000(31 downto 26) = op_Rformat AND
     addr_00000( 5 downto 0) = funct_xor)
    --sprawdzenie pozostałych komórek pamięci
    OR [...];
[...]
```

Rys. 5. Analiza kodu maszynowego
Fig. 5. Machine code analysis

System wykonano na zestawie uruchomieniowym ZL6PLD zawierającym matrycę FPGA z rodziny Spartan 3 XC3S200 [8].

[1] w zaproponowanym rozwiązaniu jest to plik źródłowy VHDL definiujący architekturę encji systemu zawierającego moduł procesora

Ponadto zestaw ten zawiera urządzenia peryferyjne takie jak: klawiatura, 4-cyfrowy multipleksowany wyświetlacz LED. Dla tych urządzeń zaprojektowano osobne interfejsy komunikacyjne, co pozwoliło włączyć je do systemu.

Synteza oraz programowanie zostały przeprowadzone przy pomocy narzędzi wchodzących w skład pakietu WEB PACK firmy Xilinx [9].

4. Testowanie

Procesor został wykonany zgodnie ze specyfikacją architektury MIPS [4], potrafi więc wykonać program zapisany w kodzie bajtowym pod warunkiem, że nie zawiera on instrukcji, które pominięto przy określaniu podzbioru obsługiwanych rozkazów.

Do celów testowych zaimplementowano w asemblerze prostą aplikację wyliczającą 47 wyraz ciągu Fibonacci'ego. Jest to wyraz o najwyższym indeksie, który można zapisać w 32 bitowej liczbie bez znaku. Zdecydowano się na ten algorytm gdyż jego zapotrzebowanie na różnorodne operacje arytmetyczne i logiczne jest niewielkie. Wynik działania jest zwracany na 4 cyfrowym wyświetlaczu LED. Ponieważ wyświetlenie 32 bitowej liczby wymagałoby 8 cyfr kodu heksadecymalnego, wynik jest multipleksowany. Za pomocą określonego przycisku można wybrać, która połowa wyniku ma zostać wyświetlona.

5. Wyniki

Do asemblacji wykorzystano symulator procesora MIPS napisany przez James'a R. Larus'a - SPIM [7]. Poza możliwościami uruchamiania i debuggowania programów przygotowanych na procesory MIPS R2000/R300 pozwala on wygenerować plik logu, który zawiera kod bajtowy w formie ciągu słów w kodzie heksadecymalnym. Jest to idealna forma, gdyż tak zapisany kod można bezpośrednio umieścić w pliku źródłowym VHDL jako zawartość pamięci.

System zsyntezowano w pakiecie WEB PACK przy dwóch różnych opcjach optymalizacji: szybkości działania i wykorzystania zasobów. W każdym przypadku implementację przeprowadzano dwukrotnie. W pierwszym przypadku globalna flaga była ustawiana tak, by wymusić obecność wszystkich instrukcji w urządzeniu wynikowym. W drugim mechanizm parametryzowania zbioru instrukcji był odblokowywany.

Wyniki doświadczeń zestawiono w tabelach 1 i 2.

Tab. 1. Porównanie maksymalnych częstotliwości pracy[2]
Tab. 1. Comparison of maximal operational frequency[2]

Kryterium optymalizacji	Maksymalna częstotliwość pracy [MHz]	
	Pełna lista rozkazów	Zredukowana lista rozkazów
szybkość („speed")	34.342	37.812
wykorzystanie zasobów („area")	28.796	26.549

Dane zawarte w tabeli 1 wskazują na to, że zaproponowany mechanizm ma wpływ na maksymalną częstotliwość pracy urządzenia. Najwyższy wynik osiągnięto przy optymalizacji pod kątem szybkości działania i zredukowanym zbiorze rozkazów. Synteza procesora z pełnym zbiorem instrukcji dałaby urządzenie działające w przybliżeniu o 10% wolniej.

Tab. 2. Porównanie wykorzystania zasobów[2]
Tab. 2. Device utilization comparison[2]

Kryterium optymalizacji: zasoby		Wykorzystanie			
		Pełna lista Rozkazów		Zredukowana lista rozkazów	
Rodzaj zasobów	L. dostępn.				
rejestry	3 840	813	21%	802	20%
bloki LUT	3 840	2 764	71%	2 421	63%
bloki funkcyjne[3]	1 920	1 918	99%	1 761	91%

Tabela 2 przedstawia zasoby matrycy FPGA zajęte na potrzeby urządzenia. W tym przypadku zysk z wykorzystania zaproponowanego rozwiązania nie jest wielki. Warto jednak zauważyć, że wraz ze zbliżaniem się do całkowitego zajęcia dostępnych zasobów w matrycy, możliwość zmniejszenia zapotrzebowania urządzenia na bloki funkcyjne nabiera znaczenia. Zysk nawet kilku procent może przecież przesądzić o możliwości zrealizowania systemu w danej matrycy FPGA.

W pierwotnej wersji rozwiązania, gdy parametry procesora ustawiane były ręcznie uzyskane wyniki były znacznie lepsze [10]. Różnica jest największa w liczbie zajętych bloków LUT. Poprzednio udało się ją zredukować do 38%, czyli aż o 25% lepiej. Jest to zjawisko niespodziewane gdyż przeniesienie analizy programu do etapu syntezy nie wiąże się z zajęciem żadnych dodatkowych zasobów.

Ostatecznie udało się ustalić, że przyczyną jest zbyt dokładna analiza. Pseudoinstrukcja NOP (no operation) rozwijana była przez asembler w rozkaz, który wymuszał dołączenie zbędnych z punku widzenia algorytmu modułów.

6. Wnioski

Proponowane rozwiązanie, ujawnia potencjał generowania urządzenia wynikowego, które ma mniejsze zapotrzebowanie na zasoby sprzętowe matrycy programowalnej. Dodatkowo wpływa na maksymalną możliwą częstotliwości pracy projektu zaimplementowanego w danym układzie FPGA, a w efekcie zwiększenie szybkość działania systemu wbudowanego.

Rozwiązanie to nie jest jednak pozbawione wad. Największą z nich stanowią trudności modyfikacji systemu. Przy zmianach wykonywanego programu niezbędna jest pełna synteza rdzenia. Może to wpłynąć znacząco na parametry czasowe systemu.

Zaobserwowano również niepożądane zjawisko dołączania zbędnych modułów opisane w poprzedniej sekcji. Wyeliminowanie go wymagałoby wpłynięcia na proces asemblacji programu. Aktualnie jednak najlepsze efekty można osiągnąć tylko poprzez świadome kodowanie algorytmu pod kątem procesu przez który będzie musiał przejść.

Pomimo powyższych wad rozwiązanie wydaje się bardzo obiecujące. W kolejnym etapie prac badawczych planowane jest dodanie możliwości zredukowania liczby rejestrów obecnych w procesorze. Przewiduje się, że znacznie poprawi to wyniki w aplikacjach o niskim skomplikowaniu, które nie wykorzystują pełnego potencjału procesora.

7. Literatura

[1] David A. Patterson, John L. Hennessy: Computer Organization and Design. Morgan Kaufmann, 2005.
[2] Dominic Sweetman: See MIPS Run second edition, Morgan Kaufmann, 2007.
[3] Peter Marwedel: Embedded System Design. Springer, 2004.
[4] Charles Price: MIPS IV Instruction Set Revision 3.2. MIPS Technologies, 1995.
[5] Peter J. Ashenden: The Designer's Guide to VHDL. Morgan Kaufmann, 2002.
[6] James R. Larus: SPIM S20: A MIPS R2000 Simulator. Computer Sciences Department, University of Wisconsin–Madison, 1993.
[7] M. Zwoliński: Digital System Design with VHDL, Pearson Prentice Hall, 2004.
[8] System uruchomieniowy ZL6PLD, http://www.kamami.pl/?idplik=xilinxfpga
[9] Xilinx, http://www.xilinx.com/
[10] Adam Ziębiński, Stanisław Swierc: The VHDL implementation of reconfigurable MIPS procesor, International Conference on Man-Machine Interactions, 2009 (w druku).

Artykuł recenzowany

[2] Dane pozyskane z raportów syntezy
[3] Do bloków funkcyjnych zaliczono konfigurowalne bloki logiczne CLB oraz komórki we/wy