Adam MILIK, Andrzej PUŁKA
POLITECHNIKA ŚLĄSKA, INSTYTUT ELEKTRONIKI
ul. Akademicka 16, 44-100 Gliwice

Implementacja wyrażeń arytmetycznych w rekonfigurowalnych sterownikach logicznych

Dr inż. Adam MILIK

Ukończył studia na Wydziale Automatyki, Elektroniki i Informatyki Politechniki Śląskiej. Pracę doktorską obronił w 2003 r. Jest adiunktem w Instytucie Elektroniki Politechniki Śląskiej. Jego zainteresowania naukowe to układy logiki programowalnej, sterowniki programowalne, modelowanie i synteza złożonych układów sprzętowo-programowych.

e-mail: adam.milik@polsl.pl

Dr inż. Andrzej PUŁKA

Ukończył studia na Wydziale Automatyki, Elektroniki i Informatyki Politechniki Śląskiej. Pracę doktorską obronił w 1997 r. Jest adiunktem w Instytucie Elektroniki Politechniki Śląskiej. Jego zainteresowania naukowe to zastosowanie metod sztucznej inteligencji w elektronice, projektowanie, modelowanie i symulacja układów cyfrowych i analogowo-cyfrowych w językach opisu sprzętu, metody projektowania współbieżnego systemów wbudowanych z podziałem na sprzęt i oprogramowanie.

e-mail: andrzej.pulka@polsl.pl

Streszczenie

W artykule przedstawiono metodę odwzorowania operacji arytmetycznych przeznaczoną dla rekonfigurowalnych sterowników logicznych. Istotą opracowanej metody jest wykorzystanie własności układów sprzętowych oraz architektury FPGA. W procesie implementacji brane są pod uwagę czas realizacji obliczeń oraz ograniczone zasoby logiczne. W oparciu o metodę szacowania czasu propagacji zrealizowano metodę łańcuchowego łączenia operacji kombinacyjnych pozwalającą na wykonanie wielu operacji w cyklu obliczeniowym.

Słowa kluczowe: sterownik programowalny, FPGA, synteza logiczna wysokiego poziomu, arytmetyka, układy rekonfigurowalne.

On arithmetic operation implementation in a reconfigurable logic controller

Abstract

The paper presents a package for arithmetic operation synthesis dedicated for reconfigurable logic controllers. Different representations (graphical or textual) commonly used are handled. The synthesis process starts from transforming algorithm representation into a data flow graph. The constant reduction and the tree height reduction optimization method are applied to the flow graph (Fig. 2). The developed method combines the ALAP and list allocation strategies with original elements. The main constraint is put to the number of available logic resources that can be allocated. The procedure attempts to allocate resources assuring it proper utilization in a calculation process. Together with resource allocation the operation scheduling is performed. During operation assignment the propagation time based concept of operation scheduling is used. The proposed method allows using sequential and combinatorial units. Operations are chained inside one state until total combinatorial propagation time does not exceed the assumed cycle time. This allows reducing the required number of calculation cycles by introducing combinatorial chains of operations (Figs. 3 and 4). Finally, an example of PID controller implementation is considered and compared with previous manual implementations (Fig. 5). Introducing the automatic implementation method allows reducing radically the calculation time (2.18 times) with little increase in hardware resources (+18%) (see Tab. 1).

Keywords: PLC, FPGA, high level logic synthesis, arithmetic, reconfigurable hardware.

1. Wstęp

Implementacja programu sterowania z wykorzystaniem reprogramowalnych układów logicznych pozwala osiągnąć szybkość przetwarzania bez porównania większą niż oferowane rozwiązania programowe. Równoległe bezpośrednie wykonywanie operacji w sprzęcie jest własnością stanowiącą istotną zaletę tego rozwiązania w odróżnieniu od szeregowego podejścia programowego. Należy również zauważyć bardzo istotną cechę układów sprzętowych, jaką jest niezwykle precyzyjne odmierzanie interwałów czasowych i właściwa dla nich przewidywalność działania w czasie. Prędkość reakcji na zdarzenia zewnętrzne jest również znacznie większa aniżeli w przypadku układów programowych.

Poważnym ograniczeniem napotykanym we wdrażaniu układów reprogramowalnych jest bardzo złożony i wymagający doświadczenia proces projektowania i implementacji algorytmu sterowania. Powstaje wiele narzędzi komercyjnych i akademickich wspierających różne aspekty projektowania systemowego z wykorzystaniem układów programowalnych [1, 6, 9]. Przedstawione w niniejszym artykule narzędzia stanowią kolejny element rozwijanej koncepcji układów sterowania sprzętowego [2]. Do tej pory koncentrowano się nad różnymi aspektami implementacji sterowania binarnego z wybranymi elementami arytmetyki w postaci modułów bibliotecznych (układy licznikowe, czasowe, regulator PID).

2. Koncepcja implementacji wyrażeń arytmetycznych

Wyrażenia arytmetyczne mogą być przedstawione przez projektanta w postaci graficznej powiązanych ze sobą bloków funkcjonalnych (diagram drabinkowy – LAD, diagram blokowy – FBD) lub w formie zdań (tekst strukturalny – ang. ST). Ponadto, stosuje się również formę mieszaną, która łączy w sobie elementy graficzne i tekstowe w postaci diagramów działań (np. SFC). W oparciu o standardowe formy reprezentacji algorytmu należy zbudować układ sprzętowy pozwalający na efektywne wykonanie algorytmu oraz racjonalne gospodarowanie zasobami sprzętowymi.

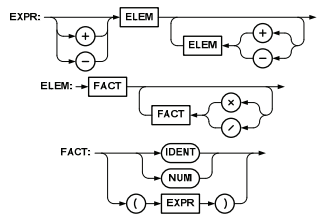
Wiele narzędzi stosuje bezpośrednie odwzorowanie zapisanych wyrażeń do postaci programowej lub sprzętowej [9]. Rozwiązania takie nie pozwalają na efektywne gospodarowanie czasem (podejście szeregowe) lub zasobami sprzętowymi.

Istotnym aspektem jest potrzeba zbudowania własnego narzędzia do implementacji wyrażeń arytmetycznych w strukturach sprzętowych. Pozwoli to na precyzyjne dostosowanie uzyskiwanych rozwiązań do potrzeb i koncepcji architektonicznych sterownika rekonfigurowanego. W pierwszym podejściu ograniczono rozważania do implementacji 4 podstawowych operacji arytmetycznych.

3. Reprezentacja pośrednia w procesie implementacji

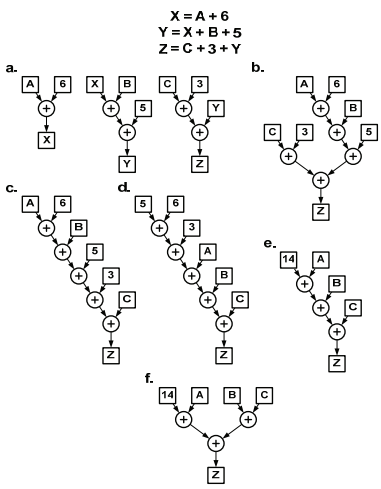
Wyrażenia arytmetyczne zapisane przez użytkownika w postaci graficznej lub w postaci wyrażeń muszą zostać przekształcone do wspólnej postaci pozwalającej na dokonanie transformacji a następnie procesu odwzorowania. Do opisu działań powszechnie stosuje się skierowane diagramy acykliczne reprezentujące przepływ informacji oraz wykonywane na nich działania [1, 4, 7, 8]. Konstrukcja diagramu w przypadku środowiska graficznego, będzie polegała na dokonaniu ciągu odwzorowań funkcjonalnych w odpowiednie węzły operacji lub poddrzewa reprezentujące funkcjonalność bloku. Kompilacja wyrażeń tekstowych wymaga dokonania rozbioru zdania. Do rozbioru zdania wykorzystano

procedury rozbioru, działające zgodnie z przedstawionymi diagramami składni (rys. 1) [4].



Rys. 1. Diagram rozbioru wyrażeń arytmetycznych
Fig. 1. The arithmetic expression syntax diagram

Wstępnie skonstruowany diagram przepływu informacji wymaga dokonania analizy oraz ewentualnych przekształceń mających na celu zredukowanie liczby operacji i ich zrównoleglenie. Proces ten umożliwia również poinformowanie użytkownika o redukcji operacji wynikających z ewentualnych błędów przy tworzeniu opisu. Przekształcenia odbywają się zgodnie z regułami operacji arytmetycznych (przemienność, łączność). Przykładową konstrukcję oraz przekształcenie diagramu pokazano na rysunku (rys. 2). Po dokonaniu przekształcenia schematu lub zapisu wyrażenia otrzymuje się zbiór grafów realizujących algorytm (rys. 2a). W celu wykonania przekształceń dokonuje się scalenia operacji, która została rozbita na wiele wyrażeń lub fragmentów graficznych (rys. 2b).



Rys. 2. Konstrukcja i przekształcanie diagramu operacji
Fig. 2. Construction and transformations of data flow diagram

W następnym etapie graf przepływu informacji w grupach operacji addytywnych i multiplikatywnych zostaje przekształcony w pojedynczą gałąź sekwencyjnie realizującą ciąg operacji (rys. 2c). Liniowa forma reprezentacji ułatwia przemieszczanie węzłów w pożądane miejsce poprzez zamianę wskaźników argumentów i wyników w parze węzłów podlegających zamianie. Forma liniowa ułatwia zgrupowanie wartości stałych poprzez sprowadzenie ich do wierzchołka fragmentu liniowego (rys. 2d). Następnie dokonuje się eliminacji węzłów, w których oba argumenty są wartościami stałymi. Węzeł operacji zastępuje się wyliczoną wartością stałą Procedurę eliminacji powtarza się iteracyjnie aż do momentu, kiedy warunek eliminacji nie może zostać spełniony. Gwarantuje to propagację wartości stałych poprzez wszystkie fragmenty liniowe grafu.

Po dokonaniu propagacji i redukcji wartości stałych można dokonać przekształcenia polegającego na redukcji wysokości drzewa (rys. 2f). Redukcja wysokości drzewa wprowadza możliwość równoległego wykonywania operacji, przez co możliwe jest skrócenie czasu realizacji obliczeń. Uzyskany graf ma istotny wpływ na kolejne etapy procesu implementacji.

4. Podział zadań

Otrzymany w poprzednim etapie graf (zbiór grafów) operacji, przedstawia bezpośrednią implementację układu. Należy jednak zauważyć, że bezpośrednie odwzorowanie grafu prowadzi do nadmiarowego rozwiązania nieefektywnie gospodarującego zasobami sprzętowymi układu a w szczególności, gdy należy brać pod uwagę ich skończoną liczbę. Zaletą bezpośredniej implementacji jest najszybsza realizacja obliczeń i łatwa implementacja.

W zaproponowanej procedurze odwzorowania, przyjęto ograniczoną liczbę zasobów obliczeniowych i logicznych, co w przypadku rozważanej rodziny układów typu Spartan 3 [10] jest w pełni uzasadnione. Układy tej serii mają ograniczoną liczbę układów mnożących oraz zasobów logicznych ogólnego przeznaczenia. Należy również zwrócić uwagę na fakt, że zespół funkcji arytmetycznych stanowi fragment układu sterownika logicznego i będzie on współdziałał z częścią logiczną. Przy definiowaniu dostępnych zasobów logicznych należy część z nich zarezerwować na implementację tej części sterownika, która odpowiada za przetwarzanie zmiennych dwustanowych.

W oparciu o metody przydziału zadań bazujących na koncepcjach ASAP, ALAP oraz listowej [1, 8] zbudowano specyficzną procedurę odwzorowania, pozwalającą na osiągniecie stawianych wymagań.

Metoda ASAP w kolejnych krokach umieszcza wszystkie Możliwe do zrealizowania operacje. Z kolei metoda ALAP przydziela operacje podążając od korzenia drzewa i ustalając realizację operacji od końca cyklu obliczeniowego. W przypadku tej metody operacje realizowane są w ostatnim możliwym cyklu. Obie metody maja charakter zachłanny i nie zakłada się w nich ograniczenia dostępnych zasobów. Przyjęto również przydzielenie identycznej ilości czasu na realizację każdej operacji. Metoda listowa bazuje z kolei na danym zbiorze elementów obliczeniowych, do których przypisywane są kolejne węzły diagramu.

W zaproponowanej metodzie starano się uwzględnić zarówno dostępne zasoby, jak i czas realizacji obliczeń oraz własności układu FPGA. Stanowi ona mieszaną strategię, która stara się rozdzielić dostępne zasoby pomiędzy operacjami biorąc pod uwagę czas ich realizacji. Do opisu czasowego operacji konieczne jest rozróżnienie elementów o charakterze kombinacyjnym oraz sekwencyjnym. Czas realizacji obliczeń na potrzeby syntezy zostanie zapisany w postaci dwóch liczb opisujących czas propagacji do wyjścia oraz liczbę cykli koniecznych do zrealizowania operacji (istotne w przypadku zastosowania sekwencyjnych układów mnożących i dzielących). Takie podejście pozwoli zbudować kombinacyjne łańcuchy obliczeniowe nie ograniczając się do wykonania pojedynczej operacji w cyklu.

W pierwszym etapie zostaje przeszukane drzewo operacji w celu wyodrębnienia zbioru realizowanych operacji. Pozwala to na dokonanie przypisania zasobów logicznych układu do minimalnego zbioru niezbędnych operacji. Jeżeli układ nie posiada minimalnej liczby zasobów, implementacja algorytmu jest niemożliwa (w przyjętym sposobie odwzorowania). Z racjonalnego punktu widzenia jest to przypadek, który można wykluczyć nawet dla niewielkich układów FPGA.

Kolejnym etapem jest indeksowanie węzłów operacji. Węzeł otrzymuje indeks wskazujący na cykl wykonania według metody ASAP. Indeks może zostać przypisany do węzła, jeżeli jego argumenty zostały wyznaczone (węzły poprzedzające otrzymały indeksy). Indeks węzła wyznaczany jest z następującej zależności:

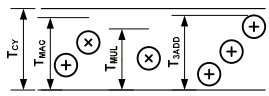
$$ID_n = \max\{ID_L, ID_R\} + 1 \tag{1}$$

Gdzie ID_L, ID_R oznacza indeksy węzłów argumentów koniecznych do wykonania węzła ID_n. Największy indeks wskazuje na najdłuższy łańcuch operacji.

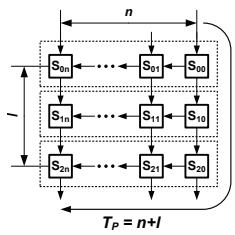
Operacja przydziału rozpoczyna się od wybrania węzła o największym indeksie. Następnie podążając od korzenia drzewa dokonuje się przypisania operacji. Warunkiem przypisania operacji jest dostępny zasób obliczeniowy w danym stanie. Jeżeli istnieją wolne zasoby logiczne zostaje z nich utworzony nowy element obliczeniowy i dodany do listy dostępnych elementów.

W bieżącym stanie dla przypisanej operacji podejmuje się próbę dołączenia kolejnych węzłów będących argumentami tej operacji w celu utworzenia kaskady obliczeniowej (rys. 3). Mogą to być węzły zarówno o charakterze kombinacyjnym jak i sekwencyjnym. Podejście takie pozwala na zredukowanie liczby zmiennych

przechowywanych w rejestrach. Warunkiem dołączenia węzła jest nieprzekroczenie czasu obliczeń dla założonej długości cyklu obliczeniowego. Próbę dołączenia podejmuje się dla obu argumentów przypisanej operacji. Kolejne przypisania w bieżącym stanie dokonuje się do momentu, gdy występuje wystarczający zapas czasu na wykonanie utworzonej operacji kaskadowej lub są dostępne zasoby logiczne. Jeżeli w danym stanie nie można dokonać dalszego przypisania operacji tworzony jest kolejny stan. Operacja alokacji jest powtarzana, jeżeli występują wolne zasoby obliczeniowe.



Rys. 3. Odwzorowanie kombinacyjne uwarunkowane czasem cyklu obliczeniowego
Fig. 3. Propagation time driven combinatorial path mapping



Rys. 4. Obliczanie czasu propagacji w operacjach łańcuchowych
Fig. 4. Delay calculation in chained combinatorial operation

Wyznaczanie czasu propagacji jest dokonywane dla poddrzewa stanu. Uwzględnia ono liczbę warstw logicznych układów arytmetycznych występujących na drodze sygnału. Sumatory, jako podstawowe elementy układów arytmetycznych mają budowę układów iteracyjnych. Kaskada złożona z kilku sumatorów identycznej długości ma czas propagacji, który nie jest wprost sumą czasu propagacji poszczególnych sumatorów, lecz wynika z najdłuższej ścieżki logicznej w układzie (rys. 4). Obok elementów bezpośrednio uczestniczących w procesie obliczeniowym, konieczne jest umieszczenie dodatkowych elementów związanych z dystrybucją sygnału pomiędzy elementami pamięciowymi i układami obliczeniowymi. Wnoszą one dodatkowe opóźnienia, które prowadzą do wydłużenia czasu realizacji cyklu obliczeniowego.

5. Odwzorowanie sprzętowe

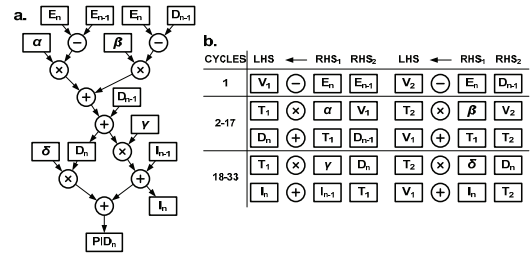
Na podstawie wyznaczonego przydziału operacji zostaje utworzony opis w języku HDL podlegający syntezie logicznej. Dokonano w nim ekstrakcji układu sterowania oraz układu obliczeniowego. Układ sterowania odpowiada za właściwe kierowanie przepływem informacji w systemie. Wyznaczenie liczby cykli koniecznych na dokonanie obliczeń pozwala na odpowiednie dopasowanie układu sterującego cyklem obliczeniowym sterownika [2]. W celu skrócenia reakcji układu, pobieranie informacji zewnętrznych rozpoczyna się przed ukończeniem obliczeń ze względu na dokładnie znaną długość tego procesu. Dystrybucja wyników procesu sterowania odbywa się z kolei w trakcie realizacji kolejnego cyklu obliczeniowego.

6. Podsumowanie

Proponowany sposób implementacji zostanie przedstawiony na podstawie implementacji dyskretnego regulatora PID danego za pomocą następującego zestawu równań [5]:

$$\begin{aligned} D_n &= D_{n-1} + \alpha\,(E_n - E_{n-1}) + \beta\,(E_n - D_{n-1}) \\ I_n &= I_{n-1} + \gamma D_n \\ P_n &= \delta D_n \\ V_n &= I_n + P_n \end{aligned} \tag{2}$$

Na podstawie równań został wygenerowany diagram operacji oraz dokonano podziału operacji (rys. 5). W celu porównania do istniejącej implementacji [3], w implementacji nie użyto kombinacyjnych układów mnożących. Korzystając ze scalenia operacji kombinacyjnych uzyskano 3 cykle obliczeniowe. Wyniki porównania zebrano w tabeli 1. Ostatecznie uzyskano ponad 2-krotne przyspieszenie obliczeń przy niewielkim wzroście zasobów logicznych (18.5%).



Rys. 5. Implementacja regulatora PID: a. Diagram operacji b. Przypisanie operacji
Fig. 5. PID controller implementation: a. data flow graph, b. operation allocation

Tab. 1. Porównanie efektów syntezy i implementacji
Tab. 1. Comparison of synthesis and implementation results

Moduł	LUT	Cykle
PID [3]	212	72
autoPID	251	33
Względny	118.4%	45.8%

Przeprowadzone eksperymenty wykazały znaczną użyteczność metody. Budowa własnych algorytmów syntezy, optymalizacji i odwzorowania operacji arytmetycznych pozwala na spełnienie specyficznych wymagań stawianych przez układy sterowania. Takie funkcje nie są dostępne w komercyjnych pakietach.

Prace nad rozwojem pakietu implementacji wyrażeń są prowadzone w dalszym ciągu. W najbliższym czasie chcemy rozszerzyć możliwości pakietu o wprowadzenie funkcji logicznych oraz elementów warunkowych. Pojawienie się ścieżek realizowanych warunkowo, stawia nowe wyzwania przed algorytmem odwzorowania. W kwestii odwzorowania technologicznego niezwykle ciekawym elementem jest układ DSP48, który jest dostępny w rodzinach układów FPGA począwszy od Virtex4. Należyte jego wykorzystanie wymaga stworzenia odpowiednich algorytmów w zakresie odwzorowania operacji.

Praca naukowa finansowana ze środków na naukę w latach 2010-2011 jako projekt badawczy 5391/B/T02/2010/38.

7. Literatura

[1] Gajski D., Dutt N., We A., Lin S.: High-Level Synthesis Introduction to Chip and System Design, Kluwer Academic Publishers, 1994.
[2] Milik A.: High Level Synthesis – Reconfigurable Hardware Implementation of Programmable Logic Controller. PDeS 2006 Programable Devices and Embedded Systems, Brno, 2006.
[3] Milik A., Hrynkiewicz E.: PID Module for Reconfigurable Logic Controller. PDS 2000, Ostrava, 2000.
[4] Wirth N.: Algorytmy + Struktury Danych = Programy. WNT, Warszawa 1989.
[5] Bibero R. J.: Microprocessors in Instruments and Control. John Willey & Sons 1990.
[6] Mishchenko A.: ABC: A system for sequential synthesis and verification. 2010, http://www.eecs.berkeley.edu/alanmi/abc/.
[7] Mishchenko A., Cho S., Chatterjee S. and Brayton R.: Combinational and sequential mapping with priority cuts. IEEE/ACM (ICCAD). Piscataway, NJ, USA: IEEE Press, 2007, pp. 354–361.
[8] Anderson J. and Wang Q.: Improving logic density through synthesis-inspired architecture. FPL 2009, pp. 105–111.
[9] MATLAB, Simulink HDL Coder 2.0, 2010.
[10] Xilinx, "DS-099 Spartan-3 FPGA Family" ver.2.1, Xilinx, 2006.

otrzymano / received: 15.03.2011
przyjęto do druku / accepted: 04.07.2011 *artykuł recenzowany*