Adam MILIK, Andrzej PUŁKA
POLITECHNIKA ŚLĄSKA, INSTYTUT ELEKTRONIKI

Wielordzeniowa jednostka centralna sterownika logicznego z czasowo-deterministycznym oprogramowaniem

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 mieszanych analogowo-cyfrowych w językach opisu sprzętu, metody projektowania współbieżnego systemów wbudowanych z podziałem na sprzęt i oprogramowanie.

e-mail: andrzej.pulka@polsl.pl

Streszczenie

Maszyna deterministyczna czasowo, w odróżnieniu od typowej realizacji programowej pozwala na bardzo precyzyjną realizację zadania w czasie. Problem kolejności przetwarzania i dostępu do danych wspólnych, występujący we współbieżnej realizacji wielu zadań jest łatwy do opanowania. Artykuł przedstawia próbę implementacji wieloprocesorowej jednostki centralnej, wykorzystującej mechanizmy zapewniające determinizm czasowy. Obok implementacji przedstawiono również metodykę generacji wielowątkowego programu sterowania.

Słowa kluczowe: Sterownik Programowalny, Maszyna deterministyczna czasowo, FPGA, Układy wieloprocesorowe.

A PLC Multi-Core Precision Timed CPU

Abstract

Modern processors are optimized to execute instructions as fast as it is possible. A program is written in timeless domain. Problems of data integrity arise when facing a problem of concurrent multithread execution. The shared variables that are used by different threads must be processed in proper order, otherwise race conditions may occur, leading to incorrect results. A precision timed CPU helps to execute tasks in the precisely defined period of time. Time dependencies between properly scheduled tasks at compile time allow preserving the proper order of data processing. The proposed multi core CPU (Fig. 2) consists of 4 CPUs equipped with: local memory (MEM), time control units (TC – Fig. 3) and shared memory (SH_MEM). Time control unit allows controlling the execution time of a current task. The CPU loads to the TC required period of time and starts task execution. When the task is completed, CPU notifies TC which disables the instruction execution until passing the given period of time. The shared memory is constructed of dual port memory. It is equipped with arbitration unit with priority rotation that is able to properly split access requests. The control program is compiled to intermediate form of a directed acyclic graph (DAG – Fig. 1) which is then used to optimize the given problem and for scheduling purposes (Fig. 5).

Keywords: PLC, Precision Timed CPU, FPGA, Multiprocessor system.

1. Wstęp

Obecne systemy procesorowe nie pozwalają arbitralnie określić czasu realizacji zadania. Program powstaje w domenie bezczasowej z bliżej nieokreślonymi zależnościami czasowymi. Z kolei układy sprzętowe posiadają bardzo dokładnie określone własności i zależności czasowe. Realizacja operacji w systemach sprzętowych odbywa się w sposób deterministyczny, dzięki temu można bardzo dokładne określić czas odpowiedzi układu sprzętowego na zdarzenie. W przypadku systemu mikroprocesorowego, precyzyjne oszacowanie czasu realizacji zadania programowego zawierającego rozgałęzienia warunkowe jest niezwykle trudne. W przypadku bardziej złożonych systemów, określenie czasu realizacji staje się praktycznie niemożliwe. W systemach czasu rzeczywistego, czas potrzebny na wykonanie danego zadania oblicza się zwykle przyjmując założenie najgorszego przypadku. W niniejszej pracy, zaproponowano modyfikację architektury jednostki centralnej, a uzyskany dzięki temu system wieloprocesorowy pozwala na łatwiejsze i efektywniejsze wykorzystanie mocy obliczeniowej. Rozwiązanie to polega na sprzętowym wsparciu realizacji zadań oraz określenia czasu ich realizacji. Przedstawione badania zostały zainspirowane przez projekt badawczy o nazwie PRET prowadzony przez grupę na uniwersytecie Berkeley [1].

Jednostka centralna została zaprojektowana pod kątem wykorzystania jej w systemach sterownikowych. Wcześniejsze doświadczenia w tym zakresie okazały się bardzo pomocne [4]. W projektowanym układzie wykorzystano wiele rozwiązań i pomysłów opracowanych uprzednio zarówno dla układów sterowania programowego, jak i sprzętowego.

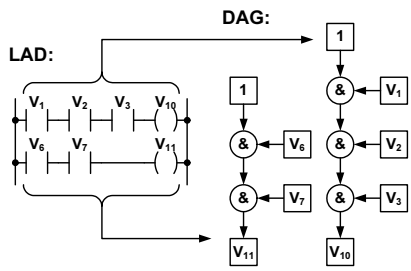
2. Koncepcja

Deterministyczno czasowy układ, pozwala na realizację zadań w określonym czasie. Przewidywalność czasowa umożliwia planowanie w czasie realizacji poszczególnych elementów programu sterowania. Zadania, które są niezależne od siebie mogą być wykonywane jednocześnie przez różne jednostki obliczeniowe. Uzyskuje się w ten sposób wielowątkowe przetwarzanie całego programu. Kluczowym zagadnieniem zapewniającym poprawne funkcjonowanie systemów wielowątkowych, wykonujących operacje na współdzielonym zbiorze danych wejściowych oraz wyników pośrednich jest odpowiednia kontrola dostępu w szczególności do wyników pośrednich [4]. Kontroli takiej dokonuje się poprzez umieszczenie znaczników informujących o stanie przetworzenia argumentów. Prowadzi to do znacznego wzrostu złożoności oprogramowania i wymaga dodatkowych zasobów pamięciowych oraz mocy obliczeniowej związanych z kontrolą procesu przetwarzania. W sytuacji, gdy niewłaściwie zostaje sygnalizowany stan zmiennej lub błędnie uwzględniono jej zależności obliczeniowe, dochodzi do wyścigu krytycznego, prowadzącego do uzyskania niepoprawnych wyników obliczeń. Procesy, których argumenty i wyniki są parami wzajemnie niezależne, mogą być obliczane jednocześnie. Współbieżna realizacja zadań jest możliwa, gdy zbiory argumentów zadań x_k jak i zbiory wyników zadań y_k spełniają założenia:

$$\begin{cases} y_1 = f_1(x_1); (x_2 \cup x_3) \cap y_1 = \varnothing \\ y_2 = f_2(x_2); (x_1 \cup x_3) \cap y_2 = \varnothing \\ y_3 = f_3(x_3); (x_1 \cup x_2) \cap y_3 = \varnothing \end{cases} \quad (1)$$

Ocenę własności zadań uzyskuje się po kompilacji programu. W wyniku kompilacji programu sterowania wyrażonego diagramem stykowym, listą instrukcji lub tekstem strukturalnym, uzyskuje się formę pośrednią, w postaci acyklicznego grafu skierowanego (DAG) [5]. Węzły grafu wyrażają operacje realizowane w programie sterowania. Podobne grafy wykorzystuje się w procesie syntezy logicznej [2]. Ze względu na różnicę w sposobie

przetwarzania w podejściu programowym i sprzętowym, występują nieznaczne różnice w kolejności węzłów. Pierwotna forma algorytmu sterowania w postaci grafu może zostać poddana przekształceniom takim jak obliczenie wyrażeń stałych, grupowanie wyrażeń wspólnych, uproszczenie wyrażeń logicznych. Etapy optymalizujące graf, pozwalają na wychwycenie potencjalnych błędów występujących w czasie tworzenia oprogramowania. Przekształcenia grafu zgodnie z regułami działań logicznych i arytmetycznych, pozwalają na zachowanie równoważności działania przy jego optymalizacji i dostosowaniu struktury do własności układowej. Przykładem takiego grafu może być transformacja szeregowej operacji dodawania akumulacyjnego na postać równoległą. Graf w postaci winorośli może zostać przekształcony do postaci drzewa zrównoważonego.



Rys. 1. Przekształcenie programu sterowania na postać pośrednia diagramu skierowanego
Fig. 1. Conversion of control program into intermediate DAG form

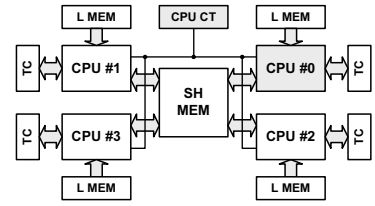
Podsumowując, podstawowymi elementami koncepcji są: jednostka centralna z mechanizmami determinizmu czasowego oraz kompilator języka pozwalający na odwzorowanie zadania sterowania dla danej jednostki obliczeniowej.

3. Deterministyczna czasowo jednostka obliczeniowa

Zwykle od typowego mikroprocesora wymaga się, by możliwie szybko wykonywał instrukcje. Stosuje się wiele rozwiązań pozwalających na zredukowanie średniego czasu wykonania instrukcji lub programu. Podejście to stawia sobie za cel, uzyskanie najlepszej wydajności obliczeniowej. W pewnych zagadnieniach istotnym staje się przewidywalność reakcji systemu w czasie. Przykładem takich zagadnień są systemy sterowania, gdzie odpowiedź systemu musi zostać wypracowana w ściśle określonym przedziale czasu. W celu zwiększenia wydajności obliczeniowej łączy się wiele jednostek przetwarzających, lecz takie rozwiązanie wprowadza dodatkowe ograniczenia, związane z wzajemną realizacją zadań oraz sposobem wymiany informacji wspólnych.

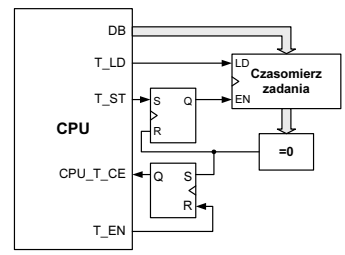
Do prowadzenia badań, wybrany został rdzeń mikroprocesora zgodny na poziomie instrukcji z mikroprocesorem Z80 opisany w języku VerilogHDL. Cechuje się on bogatą listą instrukcji oraz ugruntowaną pozycją w systemach wbudowanych i sterowania. Powyższy wybór podyktowany została relatywną prostotą pozwalającą na skoncentrowanie się nad implementacją cech funkcjonalnych. Istotną zaletą rdzenia jest możliwość jego modyfikacji i rozbudowy o niezbędne elementy dodatkowe.

Na rysunku (rys. 2) przedstawiono schemat blokowy zaprojektowanej jednostki centralnej. System składa się z 4 procesorów CPU #0..#3. Każdy z nich posiada własną pamięć lokalną programu i danych (L_MEM). Zapewnia to szybki i bezkonfliktowy dostęp do wykonywanego programu i danych lokalnych. Do przechowywania i wymiany informacji wspólnej, zaimplementowano pamięć współdzieloną z arbitrem dostępu (SH_MEM). Każda jednostka wyposażona jest w czasowo-licznikowy układ (TC) służący do nadzorowania czasu realizacji zadania. Jednostka o identyfikatorze #0 została wyróżniona jako jednostka koordynująca pracę całego systemu. Do synchronizacji pracy pozostałych jednostek wykorzystywany jest układ CPU CT. W dalszej części zostaną omówione poszczególne układy.



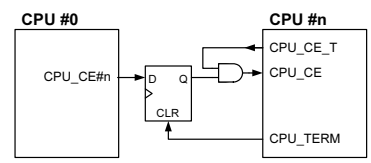
Rys. 2. Schemat blokowy jednostki centralnej z mechanizmami determinizmu czasowego
Fig. 2. Block diagram of the precision timed multi-core CPU

Jednym z najważniejszych elementów sterownika jest układ zapewniający wykonanie zadania w ściśle określonym czasie (TC). Poglądowy schemat funkcjonalny układu przedstawiono na rysunku (rys. 3). Układ został osadzony w przestrzeni pamięci każdego z mikroprocesorów tak aby uzyskać do niego łatwy dostęp. Do odmierzania czasu wykorzystano rewersyjny licznik taktowany systemowym sygnałem zegarowym. Obok kontroli czasu umożliwia on zatrzymanie działania mikroprocesora po wykonaniu zadania, aż do ściśle określonego momentu czasu, w którym zadanie ma zostać ukończone. W prologu zadania układ zostaje zainicjalizowany liczbą cykli zarezerwowanych na wykonanie zadania (T_LD). Po zapisaniu wartości początkowej, procesor uruchamia odliczania czasu poprzez wzbudzenie linii (T_ST) a następnie przechodzi do wykonania zadania. Bardzo ważne jest, by wszystkie procesory miały identycznie zbudowany prolog zadania realizowany w ten sam sposób, co zapewni pełny synchronizm czasowy między procesorami. Następnie rozpoczyna się wykonanie zadania obliczeniowego. W epilogu zadania, procesor sygnalizuje zakończenie zadania (T_EN). Działanie procesora zostaje wstrzymane przez sygnał CPU_CE_T aż do momentu upłynięcia zadanego interwału czasowego. System odmierzania czasu spełnia rolę koordynatora upływającego czasu.



Rys. 3. Schemat blokowy układu kontroli czasu zadania TC
Fig. 3. Block diagram of the time control unit

Obok mechanizmów kontroli czasu wykonania zadania, należy mieć możliwość uruchamiania i wstrzymywania procesorów jednostki centralnej. W układzie wyróżniono procesor #0, który obok funkcji obliczeniowych spełnia rolę koordynatora systemu. Do sterowania aktywnością poszczególnych procesorów wykorzystywany jest układ CPU CT (rys. 4).



Rys. 4. Schemat blokowy 1 z 3 układów sterowania jednostką w CPU CT
Fig. 4. Block diagram of one of three units in CPU control system

Jest to prosty układ umożliwiający sterowanie aktywnością wybranego procesora przez jednostkę nadrzędna. Przewidziano również możliwość samodzielnego przejścia do stanu wstrzymania (np. po wykonaniu zadania nie objętego zależnościami czasowymi) poprzez wysterowanie sygnału CPU_TERM. Podobnie jak poprzedni układ został on osadzony w przestrzeni pamięci umożliwiając łatwy dostęp. Układ CT umożliwia dokładne zsynchronizowanie działania procesorów w fazie inicjalizującej obliczenia

jak i dla zadań w których nie określono zależności czasowych. Szerzej zagadnienie synchronizacji zostanie omówione w części dotyczącej generacji programu.

Kolejnym istotnym elementem systemu jest pamięć współdzielona (SH_MEM). Zgodnie z zasadą budowy systemów wieloprocesorowych, każdy z układów posiada lokalną pamięć danych. Zapewnia ona bezkonfliktowy dostęp do danych bezpośrednio związanych z działaniem jednostki. Znaczna część danych związanych z procesem sterowania ma charakter wspólny dla wszystkich układów. Musi ona zostać umieszczona we wspólnej przestrzeni adresowej. Dostęp do pamięci obrazu procesu wymaga arbitrażu. Istotną zaletą systemu, jest wzajemna rozłączność zbioru zmiennych wynikowych wszystkich aktywnych procesów. Jak wspomniano wcześniej, podejście to w znacznym stopniu upraszcza dostęp do zmiennych. Należy tutaj zaznaczyć, że nie tylko fizyczny dostęp do zmiennej ale także jej aktualna wartość maja istotne znaczenie dla poprawności obliczeń. Zagwarantowanie rozłączności zbioru wyników, pozwala na swobodny dostęp do danych bez konieczności nadzorowania ich stanu przetwarzania.

W układzie arbitrażu zastosowano rozwiązanie polegające na rotacji priorytetów oraz dostępie rywalizacyjnym. Jednostki posiadają identyfikatory, którym odpowiadają spoczynkowe priorytety dostępu do pamięci. Elementem bazowym jest pamięć dwubramowa. Zadaniem komutatora jest łączenie 4 jednostek obliczeniowych z dwoma kanałami dostępu do pamięci. Znaczne uproszczenie układu komutacyjnego, zostało osiągnięte dzięki rozłączności obszarów zapisu dla poszczególnych procesów. W konsekwencji wpływa to na uproszczenie układu arbitrażu. Arbiter rozstrzyga prawo do dostępu w przypadku jednoczesnego zgłoszenia żądań od co najmniej trzech jednostek. Po zrealizowaniu żądania jednostka otrzymuje najniższy priorytet.

W tym miejscu warto nadmienić, że system może zostać wyposażony w kilka pamięci z oddzielnymi systemami arbitrażu, a w konsekwencji częstość występowania kolizji w dostępie do pamięci zostanie zredukowana.

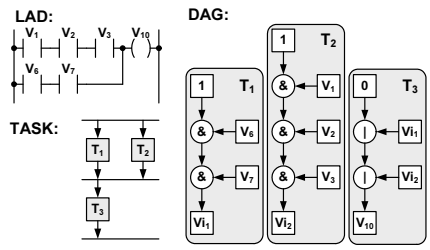
Opisana tutaj jednostka została zaimplementowana w układzie FPGA Virtex 5 [6]. Dla porównania w tabeli zebrano złożoność układową poszczególnych elementów oraz finalną złożoność jednostki centralnej wraz z pamięciami.

Tab. 1. Zasoby logiczne wykorzystywane przez układ i wybrane elementy
Tab. 1. Logic resources allocated by circuit and selected components

Moduł	LUT	Przerzutniki	BRAM
CPU	945	220	0
SH_MEM	113	10	1
Całość	4584	1127	5

4. Generacja programu sterowania

Aby w pełni wykorzystać własności wielordzeniowej jednostki obliczeniowej, należy odpowiednio przygotować program sterowania. Wszystkie zagadnienia związane z określaniem zależności czasowych, zostaną wyznaczone automatycznie na podstawie dostarczonego opisu źródłowego. Program źródłowy dostarczony w postaci standardowej listy instrukcji, podlega kompilacji do wewnętrznych struktur danych. Jak wspomniano we wstępie, program źródłowy zostaje przekształcony w acykliczny graf skierowany (rys. 1). Węzły grafu reprezentują operacje elementarne, opisane językiem programowania. Każdy z węzłów jest odwzorowywany w postaci odpowiedniego ciągu instrukcji jednostki centralnej a zarazem można mu przypisać czas realizacji. Istotną zaletą reprezentacji za pomocą grafów jest możliwość prowadzenia optymalizacji działań. Optymalizacje mają na celu wyliczenie wartości stałych, grupowanie podwyrażeń wspólnych czy też redukcję wyrażeń logicznych. Wymienione przekształcenia gwarantują zachowanie równoważności funkcjonalnej, pozwalając na uproszczenie programu wykonywalnego. Dodatkową zaletą przedstawionego podejścia jest możliwość informowania użytkownika o efektach implementacji. Przykładem może być niewykorzystanie zadeklarowanego wejścia lub sterowanie wyjścia wartością stałą. Są to informacje, które pozwalają sprawdzić czy w czasie tworzenia programu nie popełniono elementarnych błędów logicznych.



Rys. 5. Zastosowanie grafów acyklicznych w podziale zadań
Fig. 5. Application of program DAG to task decomposition

Najistotniejszym aspektem użycia acyklicznych diagramów skierowanych, jest wydzielenie zadań współbieżnych. Na rysunku (rys. 5) pokazano przykładowy diagram stykowy oraz powstałe grafy acykliczne. Jak można zauważyć, podziału dokonano na trzy funkcje logiczne wyrażone trzema grafami T_1-T_3. Argumenty T_1 i T_2 nie zależą od siebie wzajemnie, zatem spełniają warunek (1). Zadania T_1 i T_2 mogą zostać wykonane niezależnie od siebie. Zadanie T_3 wymaga uprzedniego wyznaczenia wyników z zadań T_1 i T_2. Jego wykonanie musi zostać odłożone do momentu ukończenia obu zadań. W wyniku przeprowadzonych rozważań powstał diagram obrazujący rozkład zadań oraz ich kolejność realizacji. Linia pozioma oznacza moment synchronizacji zadania w czasie realizacji. Ponieważ zadanie T_2 wymaga więcej czasu, to ono będzie determinowało długość kroku czasowego. Aby zachować synchronizm, zadaniu T_1 zostanie przydzielony dłuższy czas na realizację, aniżeli wynika to z jego potrzeb obliczeniowych.

5. Podsumowanie

Prace nad jednostką z deterministycznym mechanizmem czasowym wskazały na możliwość budowania wieloprocesorowych jednostek systemów sterowania. Kontrola nad poprawnością realizacji zadań jest zapewniona poprzez ich przydział i ocenę maksymalnego czasu realizacji na etapie kompilacji programu. W wyniku przeprowadzonej kompilacji i przypisania zadań, parametry czasowe realizacji programu sterowania są dokładnie określone. Jak pokazano, istotnym elementem jest wzajemne dopasowanie architektury oraz narzędzi programistycznych.

Badania nad powyższą jednostką są dalej prowadzone. W dalszych etapach przewiduje się zbudowanie jednostki złożonej z procesorów do przetwarzania informacji bitowej i bajtowej. Przewiduje się również próbę zastosowania mechanizmów mieszanych w realizacji zadań. Prowadzone będą w dalszym ciągu prace nad rozwojem kompilatora oraz algorytmu przydzielania zadań.

6. Literatura

[1] S. Forbes, H. D. Patel, E. A. Lee, H. A. Andrade: An Automated Mapping of Timed Functional Specification to A Precision Timed Architecture, 12-th IEEE International Symposium on Distributed Simulation and Real Time Applications, Vancouver, October 2008.
[2] Srinivas Devadas, Abhiji Ghosh, Kurt Keutzer: Logic Synthesis, McGraw-Hill, Inc. 1994.
[3] B. Fort, D. Capalija, Z.G. Vranesic and S.D. Brown: A Multithreaded Soft Processor for SoPC Area Reduction, Proceedings of the 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'06)-Volume 00, pages 131–142, 2006.
[4] M. Chmiel, E. Hrynkiewicz, A. Milik: Remarks on Improving of Operation Speed of The PLCs, 16th IFAC World Congress, Prague, 4-8 July 2005.
[5] N. Wirth: Algorytmy + Struktury Danych = Programy, WNT, Warszawa 1989.
[6] Xilinx, Virtex-5 User Guide, Xilinx, 2007.

Artykuł recenzowany