Tomasz WIERCIŃSKI
ZACHODNIOPOMORSKI UNIWERSYTET TECHNOLOGICZNY, WYDZIAŁ INFORMATYKI

Zastosowanie standardu OpenMP do projektowania systemów wbudowanych

Dr inż. Tomasz WIERCIŃSKI

Absolwent Politechniki Szczecińskiej. Adiunkt w Katedrze Inżynierii Oprogramowania Wydziału Informatyki ZUT. Zakres badań obejmuje zagadnienia budowy kompilatorów języków opisu sprzętu oraz automatycznej syntezy sprzętu ze źródeł w językach VHDL i SystemC. Prowadzone badania dotyczą również translacji programów równoległych OpenMP do kodu SystemC na poziomie behawioralnym i RTL.

e-mail: twiercinski@wi.ps.pl

Streszczenie

Artykuł prezentuje nowe podejście do projektowania systemów wbudowanych z użyciem języka C z dyrektywami OpenMP. Opisano w nim motywację użycia standardu OpenMP do syntezy sprzętowo-programowej. Przedstawiono proponowane rozwiązanie oraz porównano je z klasycznym projektowaniem systemów sprzętowych. Przedstawiono także konstrukcje równoległe standardu OpenMP, syntezowane do postaci współbieżnych układów cyfrowych. Pokazano przykładowy program w języku OpenMP wraz z jego przekładem do kodu SystemC oraz schemat RTL układu będącego wynikiem syntezy opisanego źródła.

Słowa kluczowe: systemy wbudowane, programowanie równoległe, OpenMP, języki opisu sprzętu, SystemC.

Use of OpenMP standard for embedded systems describing

Abstract

The embedded system is a special-purpose computer that performs one or a few dedicated tasks. It contains hardware and software parts [3]. The paper presents a new approach to embedded system design using C language with OpenMP directives. It is different from classic hardware design (Fig. 1a) because it allows describing both hardware and software using a common language (Fig. 1b). OpenMP is a standard that specifies parallel programs using a shared memory architecture. It is the collection of compiler directives and runtime library functions in C/C++ and Fortran languages [11]. Support for concurrency that corresponds to hardware performance is the main motivation of using OpenMP to embedded system design. OpenMP enables describing chips on high level of abstraction without knowledge about details of its structure. It improves flexibility of the software/hardware migration. OpenMP offers simulation, verification and estimation of the system performance. There is sufficient amount of legacy C libraries which facilitate the task of system modeling. Fig. 2a shows an example of OpenMP code that adds two matrixes A and B using a parallel loop. The systemC program being the results of behavioral synthesis of the example 2a is presented in Fig. 2b. Parallel regions in OpenMP have been transformed to SC_METHODS processes in SystemC. Fig. 3 shows the RTL schematic diagram of the chip synthesized from a code 2b. It contains three blocks *proc1*, *proc2*, *proc3* that are equivalent to threads in OpenMP program. A schematic diagram of the single block is presented in Fig. 4. The unit consists of an adder, a FDE flip-flop that realizes barrier synchronization and two FDR flip-flops representing signals S and R.

Keywords: embedded systems, parallel programming, OpenMP, hardware description languages, SystemC, VHDL.

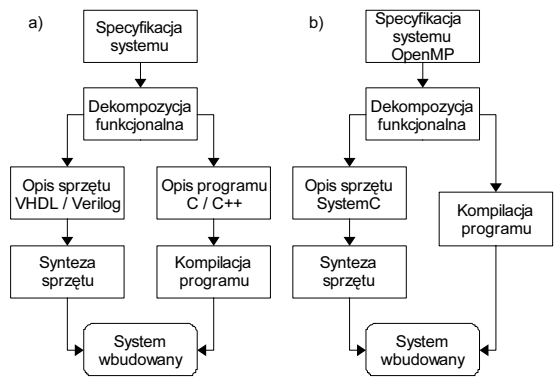
1. Wstęp oraz prace pokrewne

System wbudowany jest komputerem specjalnego przeznaczenia, który wykonuje jedno lub wiele dedykowanych zadań (programów). Jest on najczęściej składnikiem większego urządzenia takiego jak sprzęt RTV i AGD, telefon komórkowy czy pojazd [3]. Program jest uruchamiany na jednym lub więcej procesorach i może być wspomagany przez dedykowaną jednostkę sprzętową.

Rozwój technologii pozwolił na integrację całego systemu w pojedynczym układzie cyfrowym (ang. System On Chip). Projektowanie takich systemów stanowi dziś wyzwanie zarówno dla programistów komputerowych jak i projektantów sprzętu.

Można wyróżnić dwa główne podejścia w projektowaniu systemów sprzętowych. Proces klasyczny polega na rozłącznym projektowaniu sprzętu i oprogramowania przez dwa różne zespoły specjalistów, odpowiednio programistów oraz projektantów sprzętu. W tym podejściu system opisuje się przy pomocy dwóch różnych języków używając oddzielnych narzędzi do syntezy sprzętowej oraz kompilacji programu (rysunek 1a).

Różnice w projektowaniu sprzętu i oprogramowania wynikają z ich odrębnej natury. Układy cyfrowe są kompozycją połączonych ze sobą działających równolegle komponentów. Każdy z tych komponentów opisuje w sposób deterministyczny przepływ danych. W przeciwieństwie do sprzętu systemy programowe są zbudowane z sekwencyjnie działających komponentów, takich jak obiekty i wątki, których struktura często zmienia się w sposób dynamiczny (komponenty są tworzone, niszczone oraz mogą migrować) [3].



Rys. 1. Proces projektowania układów cyfrowych: klasyczny a), z wykorzystaniem OpenMP b)
Fig. 1. Chip design process: classical a), using OpenMP b)

W przeciwieństwie do klasycznych układów sprzętowych, które mogą być wytwarzane przez dwa zespoły niezależnie, systemy wbudowane wymagają projektowania całościowego przez jeden wyspecjalizowany zespół programistów-projektantów [4]. Naturalnym rozwiązaniem w tym przypadku jest użycie wspólnego języka oraz jednego środowiska pracy do opisu zarówno sprzętu jak i oprogramowania.

Najbardziej popularne rozwiązanie proponowane od 1980 roku polega na użyciu języków C/C++ z dodatkowymi rozszerzeniami pozwalającymi na opisywanie sprzętu. Głównym powodem wyboru tych języków jest ich popularność. C i C++ są językami dla których istnieje największa ilość narzędzi do pisania, debugowania oraz kompilowania programów. Istnieje również szeroka rzesza programistów posługujących się tymi językami, którzy mogą być przekwalifikowani w projektantów sprzętu.

Inną ważną motywacją użycia języków C/C++ jest fakt, że często w początkowych fazach projektów systemów sprzętowo-programowych, nie jest znany podział funkcjonalności pomiędzy sprzętem i oprogramowaniem. Użycie w tym przypadku wspólnego języka ułatwia ewentualny proces migracji [1].

Głównym rozszerzeniem niezbędnym do opisu sprzętu w językach pochodnych od C/C++ jest wsparcie do wyrażania równoległości dostępne na dwa sposoby [1]. Połowa z tych języków do-

starcza specjalnych konstrukcji równoległych. Do tej grupy należą HardwareC [7] i SystemC [8], które używają konstrukcji procesów taktowanych zboczem zegara. Handel-C [5] i SpecC [6] potrafią grupować konstrukcje współbieżne.

Innym podejściem, stosowanym w drugiej grupie języków, jest przeniesienie odpowiedzialności za identyfikację równoległości na kompilator (Transmogrifier C [9], C2Verilog [10]). Transmogrifier C, zamiast procesów, pozwala tworzyć wiele wątków, z których każdy jest niezależnym programem. Każdy wątek musi być kompilowany osobno a następnie grupa wątków jest łączona w pojedynczą listę połączeń.

2. Proponowane rozwiązanie

Do opisu obu części sprzętowej i programowej systemów wbudowanych autor proponuje użycie języka C rozszerzonego o dyrektywy OpenMP. OpenMP jest standardem specyfikacji programów równoległych przy użyciu modelu pamięci współdzielonej. Standard definiuje zbiór dyrektyw kompilatora oraz bibliotekę funkcji wspierające równoległość jako rozszerzenie języków C/C++ i Fortran [11].

OpenMP bazuje na modelu wykonawczym fork-and-join, w którym pojedynczy proces główny (ang. master thread) tworzy, po napotkaniu konstrukcji równoległej, zespół wątków. Każdy z tych wątków wykonuje współbieżnie instrukcje zawarte w rejonie równoległym. Dla każdego regionu równoległego tworzona jest niejawnie bariera synchronizacyjna kończąca wykonanie tego regionu, po której tylko wątek główny kontynuuje swoje działanie.

Poniższa lista zawiera motywacje użycia standardu OpenMP do opisu systemów wbudowanych:

1. Ponieważ OpenMP pozwala na pisanie programów działających współbieżnie, można go użyć zarówno do opisu programu jak i sprzętu we wspólnym środowisku programowym.
2. OpenMP pozwala opisywać system wbudowany na wysokim poziomie abstrakcji bez zagłębiania się w szczegóły dotyczące struktury układu cyfrowego. Na poziomie projektowania nie ma potrzeby specyfikowania, które części kodu źródłowego opisują sprzęt, a które zostaną zaimplementowane jako program. Zwiększa to elastyczność migracji między sprzętem a oprogramowaniem.
3. Język C oferuje szybką i łatwą symulację, estymację wydajności oraz weryfikację funkcjonalności.
4. Język C daje możliwość korzystania z dużej ilości istniejących źródeł oraz gotowych bibliotek co ułatwia i przyspiesza modelowanie systemu [14].
5. Istnieje szeroka grupa programistów języków C/C++, którzy mogą zostać przekwalifikowani w projektantów sprzętu.

Użycie języka C z dyrektywami OpenMP zmienia proces projektowania systemów wbudowanych zgodnie z rysunkiem 1b. Zarówno oprogramowanie jak i sprzęt stanowią pojedynczy program napisany przy użyciu jednego języka programowania. Program ten może być kompilowany w całości w celu symulacji oraz weryfikacji.

Po etapie podziału część sprzętowa systemu wbudowanego jest kompilowana do kodu HDL na poziomie behawioralnym lub RTL. Najlepszym językiem do tego celu z powodu pokrewieństwa z C/C++ wydaje się być język SystemC. Algorytmy translacji źródeł w języku C z dyrektywami OpenMP do kodu w języku SystemC na poziomie behawioralnym są opisane w [14, 15, 16]. Zostały one zaimplementowane w kompilatorze C2SystemC opisanym w [16]. Nowym wyzwaniem jest rozszerzenie kompilatora C2SystemC o możliwość generowania kodu na poziomie RTL. Autor jest w trakcie opracowywania odpowiednich algorytmów.

Synteza sprzętowa ze źródeł C wymaga określenia regionów równoległych. Można to osiągnąć poprzez użycie dyrektyw OpenMP. Dyrektywa, która tworzy region równoległy to *#pragma omp parallel.* Ilość wątków równoległych, odpowiadająca ilości współbieżnych elementów układu cyfrowego, może być określona w sposób statyczny lub dynamiczny. Statyczną ilość wątków ustawia się przy pomocy klauzuli *num_threads*, zmiennej środowiskowej *OMP_NUM_THREADS* lub funkcji *omp_set_num_threads*. Dynamiczna liczba wątków może być obliczona przy pomocy funkcji *omp_get_max_threads*. Algorytm obliczający ilość wątków równoległych został opisany w [14].

Sposób wykonania regionu równoległego przez poszczególne wątki zależy od dyrektywy podziału pracy [13]. OpenMP definiuje następujące dyrektywy podziału pracy:

1. *#pragma omp for* – określa pętlę równoległą, której iteracje są podzielone pomiędzy poszczególne wątki w zespole,
2. *#pragma omp section* – definiuje zbiór bloków strukturalnych, z których każdy jest wykonywany przez inny wątek,
3. *#pragma omp single* – specyfikuje blok strukturalny, który musi być wykonany tylko przez jeden wątek z zespołu.

Algorytmy syntezy behawioralnej wszystkich dyrektyw podziału pracy są zaprezentowane w [12]. Poza tym artykuł [13] opisuje szczegółowo algorytm syntezy behawioralnej konstrukcji pętli.

3. Przykład

Przykład przedstawiony na rysunku 2a jest fragmentem programu w języku C z dyrektywami OpenMP. Program sumuje macierze A i B w sposób równoległy, wynik zapisywany jest w macierzy C. Obliczenia odbywają się w trzech wątkach, co jest wymuszone przez klauzulę *num_threads* z wartością 3.

a)
```
#define rows 3
int A[rows], B[rows], C[rows];
void MatrixMultiply() {
  int i;
  #pragma omp parallel private(i)
  num_threads(3) {
    #pragma omp_for
      for( i = 0; i < rows; i++ ) {
        C[ i ] = A[ i ] + B[ i ];
      }
  }
}
```

b1)
```
SC_MODULE(add_matrix) {
public:
  sc_in<bool> clk, start;
  sc_in<char> a[3], b[3];
  sc_out<char> c[3];
  sc_out<bool> R;
  sc_signal<bool> S1,S2,S3;
  sc_signal<bool> R1,R2,R3;
private:
  void run();
  void proc1();
  void proc2();
  void proc3();
  void assign2r();
public:
  SC_CTOR(add_matrix) {
    SC_METHOD(run);
    sensitive << clk.pos();
    SC_METHOD(proc1);
    sensitive << clk.pos();
    SC_METHOD(proc2);
    sensitive << clk.pos();
    SC_METHOD(proc3);
    sensitive << clk.pos();
    SC_METHOD(assign2r);
    sensitive << R1 << R2 << R3;
  }
};
```

b2)
```
#include <systemc.h>
#include "add_m.h"
void add_matrix::run() {
  S1.write(0);
  S2.write(0);
  S3.write(0);
  if ((start.read() == 0) && (R.read() == 0)) {
    S1.write(1);
    S2.write(1);
    S3.write(1);
  }
}
void add_matrix::proc1() {
  R1.write(1);
  if (S1.read() == 1) {
    c[0].write(a[0].read() + b[0].read());
    R1.write(0);
  }
}

void add_matrix::proc2() {
  R2.write(1);
  if (S2.read() == 1) {
    c[1].write(a[1].read() + b[1].read());
    R2.write(0);
  }
}

void add_matrix::proc3() {
  R3.write(1);
  if (S3.read() == 1) {
    c[2].write(a[2].read() + b[2].read());
    R3.write(0);
  }
}

void add_matrix::assign2r() {
  R.write(R1.read() || R2.read() || R3.read());
}
```

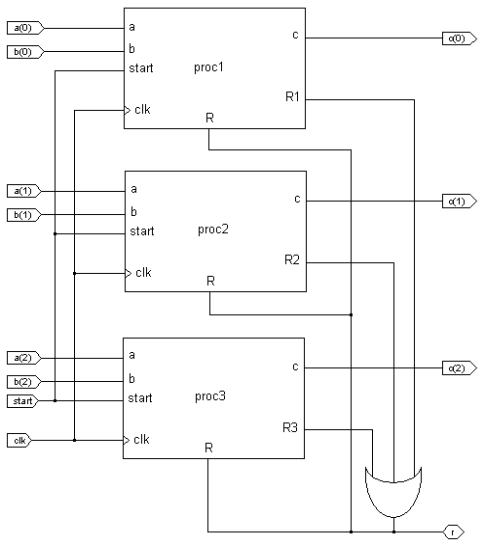
Rys. 2. Program w języku C z OpenMP a), wynik translacji w języku SystemC b)
Fig. 2. C program with OpenMP directives a), translation result in SystemC b)

Rysunek 2b zawiera program w języku SystemC będący wynikiem syntezy behawioralnej źródła z rysunku 2a. Regiony równoległe w OpenMP zostały przekształcone w równoważne im procesy *SC_METHOD* w języku SystemC. Dla podanego przykładu generowane są trzy procesy zgodnie z wartością klauzuli *num_threads* co odpowiada pełnemu rozwinięciu pętli równoległej

(ang. full loop unrolling). Każdy proces wykonuje swoją część iteracji: *proc1* wykonuje iteracje dla zmiennej *i* równej 0, *proc2* wykonuje iteracje dla zmiennej *i* równej 1, *proc3* wykonuje iteracje dla *i* równego 2.

Program rozpoczyna działanie w momencie pojawienia się wartości „1" sygnału wejściowego *start* oraz wartości „0" sygnału *R* tworzącego barierę synchronizacyjną. Generowane są wówczas wartości „1" dla sygnałów *S1*, *S2* i *S3*, które z kolei uruchamiają procesy odpowiednio *proc1*, *proc2*, *proc3*. Po wykonaniu operacji dodawania procesy zmieniają wartość sygnału *R* na „0" informując tym samym o zakończeniu wykonywania regionu równoległego. Procesy mogą być uruchomione ponownie gdy wartości sygnałów *R1, R2 i R3* są równe „0";

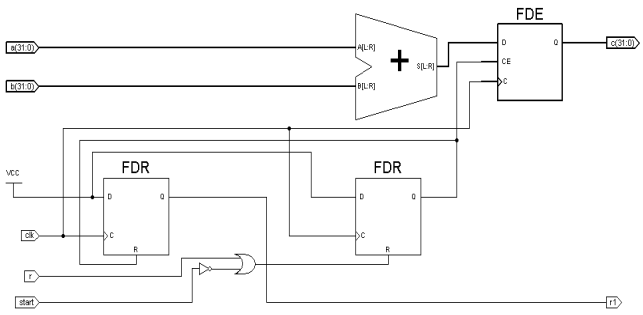
Na rysunku 3 został umieszczony schemat RTL układu będący wynikiem syntezy źródła z rysunku 2b, c.



Rys. 3. Schemat RTL układu z przykładu 2b
Fig. 3. RTL schematic diagram of the digital circuit from example 2b

Układ składa się z trzech bloków odpowiadających procesom *proc1*, *proc2* i *proc3*. Każdy blok zawiera wejścia sumujące *a* i *b*, wejście *start* oraz wejście *R* realizujące synchronizację barierową. Na wyjściu *c* zwracana jest wartość sumy, natomiast *Ri* jest wyjściem synchronizacyjnym pojedynczego bloku. Układ jest taktowany sygnałem zegarowym clk.

Szczegółowy schemat pojedynczego bloku układu został przedstawiony na rysunku 4.



Rys. 4. Schemat pojedynczego bloku układu z przykładu 2b
Fig. 4. Schematic diagram of a single unit of the digital circuit from example 2b

Blok składa się z sumatora wykonującego operacje dodawania, przerzutnika *FDE* zapewniającego synchronizację barierową oraz dwóch przerzutników FDR będących realizacją sygnałów *S* i *R*.

4. Podsumowanie

W artykule zaproponowano nowe podejście do projektowania systemów wbudowanych z użyciem języka C oraz dyrektyw standardu OpenMP. Główną motywacją tego rozwiązania jest wsparcie standardu OpenMP dla współbieżności, która jest podstawową cechą sprzętu. Istnieją również powody ekonomiczne jak redukcja kosztów osiągnięta przez wykorzystanie w procesie projektowania gotowych bibliotek oraz wykorzystanie programistów C w procesie projektowania układów.

5. Literatura

[1] Stephen A. Edwards: The Challenges of Hardware Synthesis from C-like Languages, Department of Computer Science, Columbia University, New York, 2005.

[2] Manuel Lang: SystemC for Embedded System Design, Seminar Embedded Systems (WS 06/07), Leopold-Franzens-University of Innsbruck, 2006.

[3] Thomas A. Henzinger, Joseph Sifakis: The Embedded Systems Design Challenge, Proceedings of the 14th International Symposium on Formal Methods (FM), Lecture Notes in Computer Science 4085, Springer, 2006.

[4] A. A. Jerraya: Long Term Trends for Embedded System Design, Proceedings of the Digital System Design, EUROMICRO Systems, IEEE Computer Society, Washington, DC, USA, 2004.

[5] Celoxica: Handel-C Language Reference Manual, 2003.

[6] Rainer Dömer, Andreas Gerstlauer, Daniel Gajski: SpecC Language Reference Manual. SpecC consortium, version 2.0 edition, March 2001.

[7] D. C. Ku and G. De Micheli: HardwareC: A language for hardware design. T.R. CSTL-TR-90-419, Stanford University, CA, Aug. 1990.

[8] Open SystemC Initiative: SystemC 2.0.1 Language Reference Manual, Revision 1.0, 2003.

[9] David Galloway: The Transmogrifier C hardware description language and compiler for FPGAs. In Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM), Napa, California, April 1995.

[10] Donald Soderman, Yuri Panchul: Implementing C algorithms in reconfigurable hardware using C2Verilog. In Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM), Los Alamitos, CA, April 1998.

[11] OpenMP Architecture Review Board: OpenMP Application Program Interface, Version 2.5 May 2005.

[12] Piotr Dziurzański, Włodzimierz Bielecki: Defining synthesizable OpenMP directives and clauses. In Proceedings of the International Conference on Computational Science 2004, Springer-Verlag 2004.

[13] Piotr Dziurzański, Włodzimierz Bielecki: Generating SystemC Specifications for Loops Statements in Ansi C Code with OpenMP Directives, Proceedings of the 11th IEEE International Conference on Mixed Design of Integrated Circuits and Systems - MIXDES 2004, Szczecin, Polska, 24-26 czerwca, 2004.

[14] Konrad Trifunović: Translacja ANSI C programów z dyrektywami OpenMP do źródeł w języku SystemC wraz z estymacją czasu wykonania, MS Thesis, Politechnika Szczecińska, Wydział Informatyki, Szczecin, 2006.

[15] Synopsys Inc.: Describing Synthesizable RTL in SystemC, Version 1.0, May 2001.

[16] Frank Vahid, Tony Givargis: Embedded System Design. A Unified Hardware/Software Introduction, John Wiley & Sons, Inc., 2002.

Artykuł recenzowany