Grzegorz ANDRZEJEWSKI, Wojciech ZAJĄC
UNIWERSYTET ZIELONOGÓRSKI, INSTYTUT INFORMATYKI I ELEKTRONIKI

Implementacja sprzętowa przykładowego systemu przetwarzania danych w kontekście specyfikacji formalnej

dr inż. Grzegorz ANDRZEJEWSKI

Dr inż. Grzegorz Andrzejewski - absolwent Wydziału Elektrycznego Politechniki Poznańskiej, dyplom w specjalności elektronicznej aparatury i systemów pomiarowych uzyskał w roku 1995. Stopień doktora z zakresu informatyki uzyskał na Wydziale Informatyki Politechniki Szczecińskiej w roku 2002. Jego zainteresowania naukowe ukierunkowane są na zagadnienia modelowania i syntezy systemów sterowania cyfrowego.

e-mail: G.Andrzejewski@iie.uz.zgora.pl

dr inż. Wojciech ZAJĄC

Autor ukończył studia na kierunku Informatyka i Metrologia na Wydziale Elektrycznym Wyższej Szkoły Inżynierskiej w Zielonej Górze. W roku 2001 uzyskał stopień doktora nauk technicznych na Wydziale Elektroniki Politechniki Wrocławskiej. Zainteresowania naukowe autora obejmują szereg zagadnień cyfrowego przetwarzania danych wizyjnych, w szczególności problematykę zwalczania zakłóceń z wykorzystaniem technik maskowania błędów.

e-mail: W.Zajac@iie.uz.zgora.pl

Streszczenie

W artykule przedstawiono proces implementacji sprzętowej przykładu systemu przetwarzania danych w odniesieniu do specyfikacji formalnej takiego systemu. Przedstawiono implementowany system przetwarzania danych, omówiono i przedyskutowano kolejne kroki przygotowania systemu do implementacji oraz przedstawiono i przedyskutowano uzyskane wyniki.

Abstract

In the paper there is described process of software implementation of an example of data processing system in context of formal specification of such a system. The data processing system is described, implementation steps are described and commented and results are presented and discussed.

Słowa kluczowe: Algorytm HECA, implementacja sprzętowa, specyfikacja formalna

Keywords: HECA algorithm, software implementation, formal specification

1. Wstęp

Systemy przetwarzania danych bardzo często znajdują realizację w postaci implementacji sprzętowej. Dzięki wykorzystaniu metod modelowania i specyfikacji formalnej projektant uzyskuje możliwość dostosowania zamierzonego wyniku implementacji do docelowych warunków systemu.

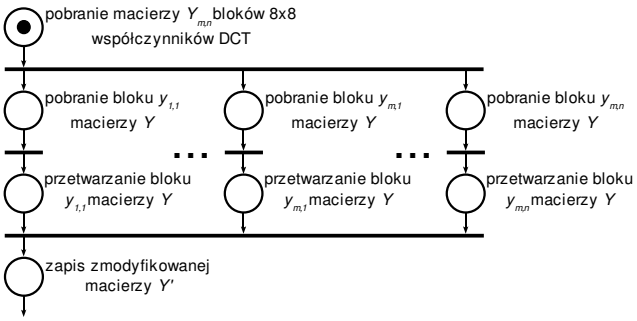
Realizacja współbieżnych zadań obliczeniowych jest problemem złożonym, mającym istotny wpływ na jakość implementacji praktycznych systemów przetwarzania danych. Popularne modele specyfikacji algorytmów przetwarzania danych (np. ASM – *Algorithmic State Machine*) opisują je w sposób, który daje dobre rezultaty na programowej platformie implementacyjnej.

Korzyścią stosowania tych metod jest możliwość stosunkowo prostej ich implementacji na platformie sprzętowej, jednaj wyniki tego procesu mogą okazać się niezadowalające.

W artykule przedstawiono omówienie implementacji sprzętowej przykładowego systemu przetwarzania danych, jakim jest blok stopnia filtrującego hybrydowego algorytmu maskowania błędów HECA, opisanego szczegółowo w [4, 5, 6]. Specyfikację ogólną działania filtru wejściowego można przedstawić z wykorzystaniem modelu formalnego sieci Petriego, jako szereg procesów współbieżnych, w których każdy operuje na niezależnych blokach pamięci rys. 1.

2. Dostęp do danych

W nowoczesnych strukturach reprogramowalnych dostępne są zasoby sprzętowe równoważne milionom bramek logicznych. Zasoby te są uporządkowane w sposób, który do pewnego stopnia determinuje możliwości konstrukcji architektury projektowanych systemów [7, 8]. W szczególności dotyczy to projektowania współbieżnych systemów przetwarzania danych.



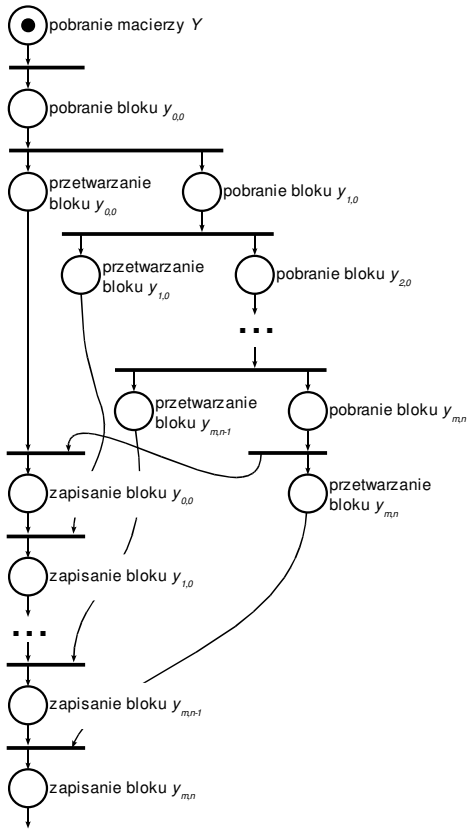
Rys. 1. Specyfikacja działania filtru wejściowego
Fig. 1. Input filter specification

Algorytm HECA jest przewidziany do detekcji i eliminacji uszkodzeń powstałych w czasie transmisji obrazów wizyjnych. Obrazy takie mogą być różnego rozmiaru. Popularne obrazy testowe mają rozmiar odpowiadający matrycy bloków danych o rozmiarze 32×32. Każdy blok zbudowany jest jako matryca o rozmiarze 8×8 liczb ośmiobitowych. Łącznie do zapisu takiego obrazu potrzebna jest pamięć o pojemności 65.536 słów ośmiobitowych. Rozmiar ten implikuje realizację pamięci jako układu o swobodnym dostępie sekwencyjnym.

Ze względu na ograniczone zasoby programowalnych matryc połączeń wewnątrzukładowych w systemach FPGA lub CPLD nie jest możliwe (lub nie jest opłacalne) takie zorganizowanie pamięci, aby zapewnić jednoczesny dostęp do wszystkich komórek pamięci. Wobec powyższego nie jest również możliwa pełna współbieżna realizacja algorytmu przedstawionego na rys. 1.

Kolejne procesy składają się z dwóch modułów funkcyjnych: modułu pobrania bloku danych (8×8) i modułu przetwarzania pobranego bloku. Przyjmując założenie, że czas przetwarzania bloku danych jest większy od czasu transmisji bloku danych z/do pamięci obrazu, można wprowadzić modyfikację algorytmu, uwzględniającą sekwencyjny dostęp do pamięci (rys. 2).

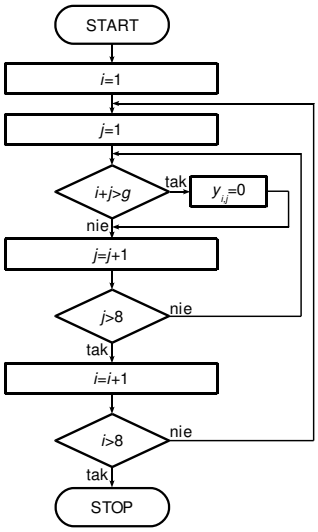
W modyfikacji tej po zakończeniu pobierania bloku danych ($y_{1,1}$) uruchamiane są dwa procesy: przetwarzania już pobranego bloku ($y_{1,1}$) i pobierania kolejnego ($y_{2,1}$). Po pobraniu ostatniego bloku danych ($y_{m,n}$) i po zakończeniu przetwarzania pierwszego bloku ($y_{1,1}$) uruchamiana jest procedura zapisu pierwszego bloku ($y_{1,1}$) do pamięci. Ostatni blok danych ($y_{m,m}$) może być zapisany po zakończeniu jego przetwarzania i zakończeniu zapisywania bloku poprzedniego ($y_{m,n-1}$).



Rys. 2. Modyfikacja specyfikacji filtru wejściowego
Fig. 2. Modification of input filter specification

3. Algorytm przetwarzania

Analizowany moduł przetwarzania działa wg algorytmu przedstawionego na rys. 3, gdzie *i* oraz *j* są zmiennymi indeksującymi wiersze i kolumny matrycy bloku danych, a *g* jest parametrem algorytmu HECA uzyskanym w wyniku badań przedstawionych w [4, 5, 6].



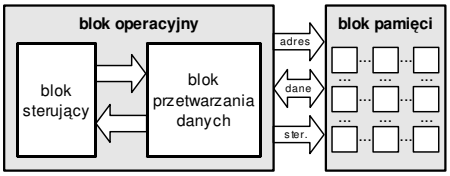
Rys. 3. Algorytm działania bloku przetwarzania
Fig. 3. Processing block algorithm

Naturalną konsewencją takiego przedstawienia algorytmu jest prosta realizacja, zakładająca sekwencyjne wyliczanie tych komórek macierzy, które wymagają wyzerowania (spełnienie warunku *i+j>g*). Przykład takiej realizacji wykorzystującej właściwości specyfikacji behawioralnej języka VHLD przedstawia następujący fragment kodu (rys. 4).

```
for i in 0 to 7 loop
  for j in 0 to 7 loop
    if (i+j)>g then
      resetCell(i,j);
    end if;
  end loop;
end loop;
```

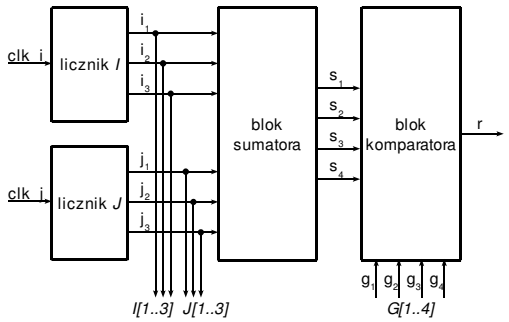
Rys. 4. Realizacja bloku przetwarzania na poziomie behawioralnym
Fig. 4. Processing block implementation on behavioral level

Zwykle lepsze rezultaty implementacyjne osiąga się wykorzystując do opisu systemu nie poziom behawioralny ale RTL (ang. *Register Transfer Level*). Uproszczony schemat blokowy na poziomie RTL systemu odpowiadającemu pojedynczemu blokowi funkcyjnemu przedstawiono na rys. 5.



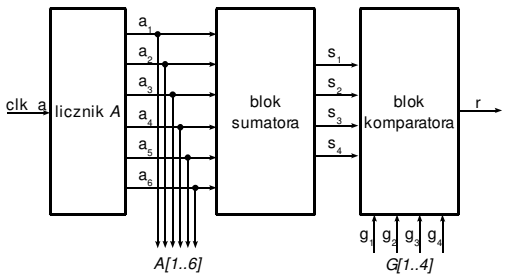
Rys. 5. Schemat ogólny pojedynczego bloku funkcyjnego
Fig. 5. General scheme of single function block

W systemie tym wyróżniono pamięć bloku obrazu (8×8) o dostępie sekwencyjnym oraz blok operacyjny. Zgodnie z przyjętymi zasadami projektowania systemów przetwarzania danych część operacyjna podzielona została na dwa moduły: blok sterujący oraz blok przetwarzania danych. Na rys. 6 przedstawiono schemat układu, który realizuje przetwarzanie danych zgodnie z algorytmem z rys. 3. Układ ten złożony jest z dwóch liczników adresujących matrycę pamięci danych, sumatora wewnątrzblokowych współrzędnych (*i*, *j*) oraz komparatora progu granicznego *g*. Po wykryciu przekroczenia progu g układ generuje sygnał r, wykorzystywany przez blok sterujący do zapisu odpowiedniej (określonej współrzędnymi *i* oraz *j*) komórki pamięci.



Rys. 6. Realizacja podstawowa bloku przetwarzania danych
Fig. 6. Basic implementation of data processing block

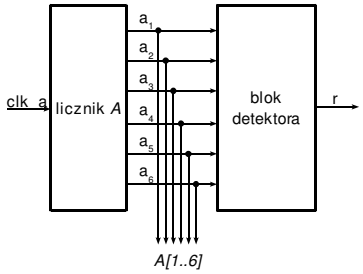
Analizując strategię sterowania licznikami dochodzi się do wniosku, że możliwe jest zastąpienie dwóch 3-bitowych liczników jednym licznikiem 6-bitowym (rys. 7) co nieco upraszcza konstrukcję bloku operacyjnego.



Rys. 7. Realizacja bloku przetwarzania danych z pojedynczym licznikiem
Fig. 7. Implementation of data processing block with single counter using

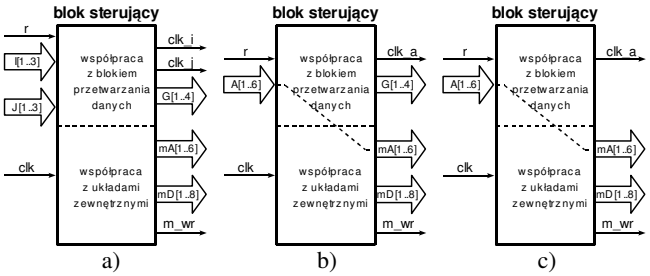
Kolejne uproszczenie wynika z analizy działania filtra wejściowego. W pracy [5] wykazano, że wartość progu granicznego *g* jest

stała i wynosi g=11. Wobec tego można zastąpić kombinacyjne bloki sumatora i komparatora jednym blokiem kombinacyjnym detektora adresu (rys. 8). Zmniejszy to zarówno rozmiar układu po syntezie, jak również przyspieszy działanie układu.



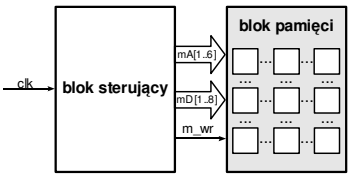
Rys. 8. Realizacja bloku przetwarzania danych z detektorem
Fig. 8. Implementation of data processing block with detector

Na rys. 9 przedstawiono bloki sterujące odpowiadające blokom przetwarzania danych z rys. 6, 7 i 8.



Rys. 9. Realizacje bloków sterujących odpowiednich do a) rys. 6, b) rys. 7, c) rys. 8
Fig. 9. Implementation of control blocks respective to a) fig. 6, b) fig 7, c) fig. 8

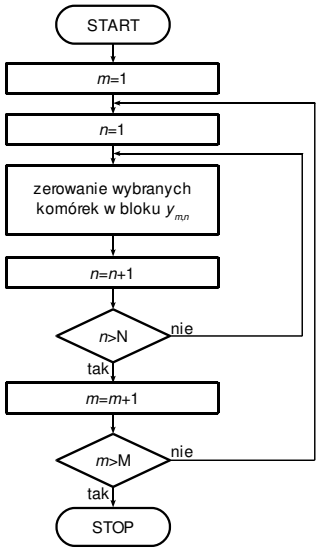
Analizując dalej działanie układu można zauważyć, że układ ma za zadanie wyzerowanie określonych komórek pamięci. Można zrobić to szybciej, jeśli zwykły licznik zastąpi się generatorem zadanej sekwencji wyjściowej, w tym przypadku adresami żądanych komórek pamięci. Zbędny będzie wówczas również blok detektora, a sam generator wygodnie jest zintegrować w jednym układzie z blokiem sterującym, który bezpośrednio współpracuje z blokiem pamięci danych (rys. 10).



Rys. 10. Realizacja wynikowa bloku funkcyjnego
Fig. 10. Final implementation of function block

Idąc dalej w rozważaniach dochodzi się do wniosku, że zbędne są również operacje kopiowania bloków pamięci z matrycy Y do lokalnych bloków pamięciowych. Zerowanie komórek matrycy w poszczególnych blokach matrycy Y należy wykonać na pamięci głównej. Oszczędzi to zarówno czas niezbędny do kopiowania z/do pamięci, jak również niebagatelne zasoby sprzętowe (524.288 przerzutników!). Ponieważ dostęp do pamięci obrazu jest sekwencyjny, pociągnie to za sobą również zmiany algorytmu głównego działania filtru wejściowego (rys. 11).

W algorytmie tym wyeliminowano wszelkie zbędne operacje, jak np. kopiowanie danych, zakładając, że blok przetwarzania będzie bezpośrednio operował na fragmentach pamięci głównej (matrycach 8×8), a wybór kolejnych matryc będzie realizowany za pomocą prostego generatora adresu.



Rys. 11. Wynikowy algorytm filtru wejściowego
Fig. 11. Final algorithm of input filter

3. Podsumowanie

Przedstawiony w pracy przykład analizy możliwości sprzętowej implementacji przykładowego systemu przetwarzania danych pozwala na wyciągnięcie wniosku, że proste zastosowanie specyfikacji formalnej w odniesieniu do algorytmu zorientowanego na realizację programową daje co prawda zysk w postaci przyspieszenia czasu jego realizacji, jednak kosztem powstania znaczących wymagań na zasoby sprzętowe. Praktyka pokazuje, że niektóre algorytmy przetwarzania danych można skutecznie implementować przy niskich nakładach sprzętowych i krótkich czasach obliczeń, jednak wymagane jest do tego duże doświadczenie inżynierskie.

4. Literatura

[1] G. Andrzejewski: *Programowy model interpretowanej sieci Petriego dla potrzeb projektowania mikrosystemów cyfrowych*, Zielona Góra, Oficyna Wydawnicza Uniwersytetu Zielonogórskiego, 2003
[2] G. Andrzejewski: *Hierarchical Petri nets for digital controller design*, in Design of embedded control systems, New York, Springer, 2005 pp. 27-36
[3] G. Andrzejewski, Z. Skowroński: *Zrównoleglanie algorytmów sterowania w systemach klasy PLC*, Pomiary Automatyka Kontrola, 2006, nr 6, wyd. spec., s. 62-64
[4] Zając W.: *Korekcja błędów transmisji cyfrowego obrazu monochromatycznego z wykorzystaniem algorytmu ukrywania błędów*", rozprawa doktorska. Politechnika Wrocławska, 2001.
[5] Zając W.: *An Error Concealment Algorithm for Digital Image Transmission*. Proceedings of XIV International Symposium on Computer and Information Sciences, Kusadasi, Turkey, October 1999
[6] Zając W.: *A Hybrid Method of Error Elimination in Digital Image Transmission*, Proceedings of the 5TH International Conference MIXDES'98, Łódź 1998
[7] http://www.altera.com
[8] http://www.xilinx.com

Title: Software implementation of an example data processing system in context of formal specification.

Artykuł recenzowany