ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1993

Seria: INFORMATYKA z. 22 Nr kol. 1195

Leszek PŁONKA
Instytut Informatyki Teoretycznej i Stosowanej PAN
ul. Bałtycka 5, 44–100 Gliwice

ANALIZA UKŁADÓW LOGICZNYCH I ANALIZA OPROGRAMOWANIA

Streszczenie. W artykule zostało omówione zagadnienie analizy układów logicznych i analizy oprogramowania oraz wybrane aspekty konstrukcji analizatorów układów logicznych i analizatorów oprogramowania. Przedstawione zostały również rozwiązania konstrukcyjne komparatora stanów i komparatora sekwencji stanów – będących istotnymi elementami analizatora – zastosowanych w zbudowanej w Instytucie Informatyki Teoretycznej i Stosowanej PAN rodzinie stanowisk wspomagających uruchamianie systemów cyfrowych i mikroprocesorowych czasu rzeczywistego.

LOGIC ANALYSIS AND SOFTWARE ANALYSIS

Summary. The paper discusses the techniques of logic analysis and software analysis as well as selected aspects of logic analyzer and software analyzer design. State comparators and sequence comparators – important logic analyzer elements – implemented in a family of real-time system development workstations built at Institute of Theoretical and Applied Computer Science, Polish Academy of Sciences, are also presented.

ANALYSE DE SYSTÈMES LOGIQUES ET DU LOGICIEL

Resumé. L'article discute les techniques d'analyse de systèmes logiques et d'analyse de logiciel ainsi que certains aspects de construction de ces systèmes. On présente les comparateurs d'états et implementés dans la famille de stations qui aident la mise en marche de systèmes numériques et microprocesseurs temps réel, elaborée dans l'Institut de l'Informatique Théorique et Appliquée de l'Academie Polonaise des Sciences.

1. Wstęp

Analiza układów logiczych i analiza oprogramowania odgrywają istotną rolę w pracy projektantów współczesnych systemów cyfrowych i mikroprocesorowych. Chociaż wstępne fazy projektów charakteryzują się zwykle szerokim stosowaniem metod teoretycznych oraz narzędzi typu CAD, jednak w końcu następuje etap fizycznej realizacji projektu i testowania w warunkach rzeczywistych. Do realizacji tego etapu niezbędne są narzędzia dające możliwość obserwacji stanów zbudowanego systemu i pomiaru jego parametrów. Rodzaj poszukiwanej informacji o badanym układzie zależy od zadań projektanta. Jeżeli zajmuje się on uruchamianiem części sprzętowej projektu, interesować go będą przebiegi w pewnych punktach układu, zależności czasowe między sygnałami itp. Projektant oprogramowania systemu czasu rzeczywistego chce zwykle znać przepływ sterowania w swoim programie, stany zmiennych (obszarów pamięci), czasy wykonania poszczególnych modułów programowych, szybkość i sposób reakcji na zdarzenia zewnętrzne itp. Metody analizy układów logicznych oraz analizy oprogramowania są bardzo przydatne w zbieraniu tych informacji, mogą więc być stosowane do uruchamiania i testowania sprzętu i oprogramowania, a także do integracji sprzętowo-programowej.

Niniejsze opracowanie stanowi propozycję przedstawienia całości zagadnienia analizy układów logicznych i analizy oprogramowania. Szczególny nacisk położony został na opis samej metody, a nie jej zastosowań czy też budowy przyrządów ją wykorzystujących. Algorytmy nie są złożone i mogłyby być przedstawiane różnymi metodami. Jednym z ciekawszych sposobów wydaje się być zastosowanie notacji CSP (Communicating Sequential Processes) [1, 2], która charakteryzuje się prostotą i zwięzłością oraz wprowadza bardzo istotne dla zagadnienia analizy układów logicznych pojęcie *śladu* procesu (dodatek A zawiera krótkie omówienie CSP).

2. Analiza układów logiczych

Analiza układów logicznych jest metodą uruchamiania systemów cyfrowych (mikroprocesorowych) polegającą na rejestracji w czasie rzeczywistym i późniejszym przetworzeniu/analizie przebiegów czasowych lub sekwencji stanów danego systemu. Urządzenie służące do tego celu nazywa się analizatorem układów logicznych.

Dane rejestrowane przez analizator tworzą *ślad* (ang. trace) pracy systemu, a proces ich rejestracji nazywany będzie *śladowaniem*.

Znajomość rzeczywistych przebiegów występujących w badanym układzie może stanowić nieocenioną pomoc przy jego uruchamianiu lub naprawie (zwłaszcza w pracach konstrukcyjnych, gdyż do naprawy sprzętu produkowanego seryjnie znacznie wygodniejsze i efektywniejsze są metody wykorzystujące analizę sygnatur).

Proces analizy układów logicznych można scharakteryzować w następujący sposób:

1. analizator jest fizycznie połączony z badanym układem cyfrowym;
2. analizator jest dla badanego systemu 'przezroczysty', tzn. nie zakłóca pracy badanego systemu pod względem elektrycznym ani logicznym;
3. czas w procesie analizy jest dyskretny, a kolejne chwile wyznaczane są przez sygnał cyfrowy, zwany sygnałem zegarowym;
4. stany badanego systemu w kolejnych chwilach czasu stanowią dla analizatora wektory wejściowe;
5. kolejne wektory wejściowe zapamiętywane są w czasie rzeczywistym w pamięci analizatora;
6. pamięć analizatora ma skończoną pojemność;
7. prezentacja zebranych danych odbywa się z szybkością umożliwiającą użytkownikowi ich zrozumienie i interpretację.

Należy tu podkreślić znaczenie punktów (2) i (5) mówiących o przezroczystości analizatora oraz jego pracy w czasie rzeczywistym, ponieważ analiza układów logicznych z założenia jest metodą dynamiczną, mającą na celu śledzenie funkcjonowania układu pracującego z pełną szybkością, w odróżnieniu od metod statycznych, wymagających wstrzymywania badanego układu w każdym cyklu pracy.

W rozumieniu notacji CSP ślad procesu jest sekwencją *zdarzeń*, w których ten proces uczestniczył. Na potrzeby opisu zagadnienia analizy układów logicznych śladem nazwiemy

sekwencję *stanów* systemu cyfrowego, przy czym zakładamy, że ślad ten obejmuje tylko te stany układu, które wystąpiły po uruchomieniu badanego układu. Ślad analizowanego układu T oznaczymy przez t ($t \in traces(T)$), natomiast ślad zdarzeń zarejestrowanych przez analizator oznaczymy przez a ($a \in T^*$). Ślad a tworzony jest w następujący sposób: początkowo ślad jest pusty ($a = \langle\rangle$), następnie, wraz z każdym taktem zegara analizatora, do śladu dodawany jest wektor binarny x_k, reprezentujący stan wejść analizatora:

$$\begin{aligned} &\text{if } \#a < n \text{ then} \\ &\quad (a = a^\frown\langle x_k\rangle) \\ &\text{else} \\ &\quad (a = a'^\frown\langle x_k\rangle) \end{aligned}$$

przy czym n jest maksymalną dopuszczalną długością zarejestrowanego śladu (zawsze $\#a \leq n$ – ograniczenie to spowodowane jest skończoną pojemnością pamięci śladu analizatora).

Kolejne chwile k wyznaczane są przez sygnał zegarowy synchronizujący pracę analizatora.

2.1. Śladowanie warunkowe

Śladowanie warunkowe ma na celu wyodrębnienie interesujących użytkownika sekwencji stanów badanego systemu cyfrowego. Za realizację śladowania warunkowego odpowiada *komparator sekwencji stanów*, który sygnalizuje blokowi śladowania analizatora konieczność zarejestrowania aktualnego stanu wejść. Komparator sekwencji stanów jest układem sekwencyjnym mającym za zadanie wyszukanie, w ciągu danych wejściowych analizatora, sekwencji wektorów spełniających zadeklarowane przez użytkownika warunki i odpowiednie sterowanie procesem rejestracji danych.

Warunkiem elementarnym będziemy nazywali formę zdaniową

$$XRY \tag{1}$$

gdzie X jest stanem wejścia analizatora, Y jest liczbą lub przedziałem liczbowym zadeklarowanym przez użytkownika, natomiast (zwykle) relacja $\mathcal{R} \in \{=, \neq, >, <, \geq, \leq, \in\}$.

Rejestracja danego stanu przez analizator następuje w momencie spełnienia warunku śladowania W, zadeklarowanego przez użytkownika, czyli

$$\text{if } W \text{ then } a = a^\frown\langle x_k\rangle$$

Warunek W może składać się z jednego lub kilku warunków elementarnych, połączonych za pomocą operatorów logicznych (np. AND, OR, NOT).

W przypadku relacji równości i różności konstrukcja analizatora powinna również umożliwiać maskowanie bitów wektora wejściowego. Ma to na celu wyeliminowanie ich wpływu na wartość logiczną wyrażenia (1).

Różnorodność praktycznie stosowanych metod deklaracji warunków śladowania (*kwalifikatorów* śladu warunkowego) jest znaczna [4, 7, 8, 9, 10], dlatego uproszczony opis przedstawiony powyżej jest jedynie pewnym uogólnieniem, przydatnym do rozważań teoretycznych.

2.2. Algorytm pracy analizatora układów logicznych

Algorytm pracy modułu śladowania można, w ogólnym przypadku, przedstawić następująco

$$A = (tr \rightarrow acq \rightarrow A \,\square\, br \rightarrow STOP)$$

Spełnienie warunku śladowania tr prowadzi do rejestracji acq stanu wejść. Proces ten zachodzi cyklicznie (co jest symbolizowane przez rekurencję $A \rightarrow \ldots A$) aż do momentu spełnienia warunku przerwania śladowania br.

Równocześnie z blokiem rejestracji danych (śladowania) A pracuje blok komparacji C, sygnalizujący blokowi A wystąpienie warunku śladowania

$$C = (trcond \rightarrow tr \rightarrow C \,\square\, brcond \rightarrow br \rightarrow STOP)$$

$trcond$ oznacza spełnienie warunku śladowania W, które jest sygnalizowane blokowi śladowania A jako zdarzenie tr; $brcond$ oznacza spełnienie warunku przerwania śladowania, sygnalizowane blokowi A jako zdarzenie br.

Przyczyna przerwania śladowania może być następująca:

- spełnienie warunku przerwania śladowania zadeklarowanego przez użytkownika, np. zapełnienie pamięci śladu;
- interwencja użytkownika, np. naciśnięcie klawisza 'break'.

Cały czas pracuje również obserwowany układ prototypowy T (realizujący zdarzenia ze swojego słownika αT)

$$T = (z : \alpha T \rightarrow T)$$

W notacji CSP równoległą pracę układów A, C i T zapisuje się $A \parallel C \parallel T$.

2.3. Fizyczna realizacja śladowania

Przy założeniu, że m oznacza pamięć śladu, natomiast x_k jest wektorem wejściowym w chwili k, realizację elementarnej operacji *acq* można przedstawić następująco

$$m(j) \leftarrow x_k$$

$$j \leftarrow j \oplus 1$$

przy czym j jest licznikiem adresów pamięci śladu; symbol $\leftarrow$ oznacza operację przypisania, natomiast symbol $\oplus$ oznacza operację dodawania modulo n.

Ponieważ czas zapisu do pamięci jest stosunkowo duży, w praktyce stosuje się rejestry buforujące. Rejestr zapamiętuje stan wejść, a następnie jest przepisywany do pamięci śladu

$$r \leftarrow x_k$$

$$m(j) \leftarrow r$$

Z uwagi na ograniczoną pojemność pamięci śladu nie jest możliwa rejestracja długich sekwencji (zawsze $\#a \leq n$). Możliwe jest jednak pewne zwiększenie długości śladu przez zastosowanie *kompresji* strumienia danych

$$\begin{aligned} &\text{if } x_k \neq m(j \ominus 1) \text{ then} \\ &\quad (\ m(j) \leftarrow x_k \\ &\quad\ j \leftarrow j \oplus 1 \\ &\quad\ t(j) \leftarrow k) \end{aligned}$$

przy czym symbol $\ominus$ oznacza operację odejmowania modulo n, natomiast t w tym przypadku oznacza pamięć *numeru próbki* k. Zamiast operacji $t(j) \leftarrow k$, analizator może również wykonywać operację $t(j) \leftarrow \Delta k$, gdzie $\Delta k = k - t(j \ominus 1)$.

Wektor x_k zapisywany jest do pamięci śladu tylko wtedy, gdy jest on różny od swojego poprzednika x_{k-1}. Aby możliwe było odtworzenie oryginalnego przebiegu, zapamiętane zostają także numery sąsiednich próbek (bądź też różnice ich numerów, czyli odległości między nimi). W przypadku stosowania kompresji strumienia danych ślad badanego systemu T jest sekwencją *par* (x_k, k) lub $(x_k, \Delta k)$.

Zapis do pamięci śladu jest synchronizowany przebiegiem zegarowym, wyznaczającym kolejne chwile zapisu. Przy założeniu, że $f_d = 1/T_d$ oznacza maksymalną częstotliwość zmian danych na wejściu analizatora, natomiast $f_c = 1/T_d$ określa częstotliwość zegara wyznaczającego chwile zapisu do pamięci śladu, w trybie analizy czasowej warunek

$$f_c > f_d \quad \text{lub} \quad T_c < T_d \tag{2}$$

(ograniczenie szybkości) określa zakres stosowalności analizatora układów logicznych.

Proces kompresji, dla trybu analizy czasowej, można poglądowo wyjaśnić w następujący sposób: częstotliwość wewnętrznego zegara powinna być proporcjonalna do maksymalnej częstotliwości zmian stanu wejść analizatora (tak aby był spełniony warunek (2)), przy czym musi ona być odpowiednio duża, by uzyskać zadowalającą dokładność pomiaru i równocześnie możliwie mała, aby efektywnie wykorzystać pamięć śladu. Najlepsze efekty można osiągnąć, gdy częstotliwość wewnętrznego zegara będzie dostosowywać się *dynamicznie* do zmieniającej się częstotliwości przebiegów wejściowych. Rozwiązaniem jest generowanie sygnału zegarowego w wyniku zmiany stanu wejść analizatora, oczywiście synchronicznie z wewnętrznym zegarem, określającym maksymalną szybkość jego pracy.

2.4. Analiza stanów logicznych

Podczas analizy stanów logicznych (ang. state analysis) chwile zapisu danych do pamięci analizatora są wyznaczane przez zegar badanego systemu (zegar zewnętrzny). Ten tryb pracy analizatora jest dogodny do analizy stanów występujących w systemie synchronizowanym przez jeden przebieg zegarowy (takich jak np. stany magistral systemu mikroprocesorowego lub też stany klasycznego synchronicznego układu sekwencyjnego). Można np. śledzić wykonywanie programu przez system mikrokomputerowy, gdyż istnieje możliwość zapamiętania wszystkich cykli dostępu do pamięci systemu, co pozwoli stwierdzić, jakie instrukcje zostały wykonane przez mikroprocesor. Jeżeli interesują nas zależności czasowe w badanym układzie, niezbędne jest rejestrowanie czasu rzeczywistego, jak poniżej

$$m(j) \leftarrow x_k$$

$$c(j) \leftarrow \mathrm{clk}(k)$$

$$j \leftarrow j \oplus 1$$

$\mathrm{clk}(k)$ oznacza stan zegara czasu rzeczywistego, w jaki wyposażony musi być analizator stanów logicznych.

2.5. Analiza przebiegów czasowych

Analiza przebiegów czasowych (ang. timing analysis) ma na celu śledzenie funkcjonowania sprzętu cyfrowego poprzez rejestrację przebiegów czasowych w nim występujących.

Wprowadzanie danych pomiarowych do pamięci analizatora jest asynchroniczne względem wszystkich sygnałów występujących w badanym układzie, chwile zapisu są bowiem wyznaczane przez wewnętrzny generator zegarowy analizatora ρ regulowanej częstotliwości. Wtedy

$$x_k = X(kT)$$

gdzie T jest okresem zegara analizatora, natomiast $X(kT)$ oznacza stan wejść analizatora w chwili kT. Operację śladowania stanu wejść można więc przedstawić następująco:

$$m(j) \leftarrow X(kT)$$

$$j \leftarrow j \oplus 1$$

Ten tryb pracy jest dogodny do analizy zależności czasowych występujących między sygnałami oraz zjawisk spowodowanych błędnym funkcjonowaniem sprzętu.

Często analizator układów logicznych zawiera oddzielne bloki służące do analizy czasowej i analizy stanów. Bloki te mogą pracować niezależnie lub razem, ułatwiając w ten sposób rozwiązywanie złożonych problemów sprzętowo-programowych. Prostsze analizatory są wyposażone w tylko jeden moduł śladowania, ale dają możliwość wyboru między zegarem zewnętrznym i wewnętrznym.

Tryb analizy czasowej zwany jest też trybem asynchronicznym (ang. asynchronous mode), natomiast tryb analizy stanów zwany jest też trybem synchronicznym (ang. synchronous mode).

2.6. Prezentacja danych pomiarowych

Sposób prezentacji danych pomiarowych może mieć istotny wpływ na komfort pracy użytkownika analizatora układów logicznych. Sposób najlepszy zależy od tego, co przedstawiają dane znajdujące się aktualnie w pamięci śladu oraz jakich informacji użytkownik poszukuje. Do prezentacji danych służy zwykle monitor ekranowy analizatora lub drukarka.

W przypadku analizy czasowej dane pomiarowe są zwykle przedstawiane w postaci wykresów czasowych (ang. waveform display).

W przypadku analizy stanów logicznych dane pomiarowe mogą być przedstawiane:

- w formie tablic stanów (ang. state table display), z możliwością wyboru żądanego formatu, np. dwójkowego, ósemkowego, dziesiętnego lub szesnastkowego;
- w postaci symbolicznej (ang. symbolic display), stosowanej w przypadku śladowania stanów magistral systemu mikroprocesorowego. Zawartość pamięci śladu jest w

tym przypadku poddawana przetworzeniu przez disasembler odpowiedni do typu mikroprocesora zastosowanego w badanym układzie;

- w formie grafów stanów (ang. state graph display). Zawartość kolejnych komórek pamięci śladu TM traktowana jest jako ciąg liczbowy TM(i) (gdzie $i = 0, \ldots, n-1$), który przedstawiany jest graficznie w układzie współrzędnych. Ten sposób prezentacji danych wygodny jest do obserwacji przepływu sterowania w programie, lokalizacji pętli programowych lub analizy pracy przetworników analogowo-cyfrowych.

2.7. Budowa analizatora układów logicznych

W każdym analizatorze układów logicznych można wyróżnić kilka podstawowych bloków funkcjonalnych:

- moduł śladowania realizujący funkcję śladowania;
- moduł komparatora sekwencji sterujący procesem śladowania;
- moduł komunikacji z użytkownikiem służący do prezentacji danych pomiarowych oraz przyjmowania zleceń.

Na poszczególne bloki składają się dodatkowe podukłady, takie jak:

- sondy wejściowe – służą do połączenia analizatora z badanym układem;
- rejestry wejściowe – buforują dane doprowadzone do pamięci śladu;
- pamięć śladu – przechowuje dane pomiarowe;
- układ tworzenia adresu pamięci śladu;
- komparator sekwencji stanów;
- komparator stanów – stanowi część komparatora sekwencji;
- generator zegarowy;
- układy sterownika monitora ekranowego oraz klawiatury.

Cechą analizatora, równie ważną jak możliwości funkcjonalne zdeterminowane przez jego konstrukcję, jest odpowiednie oprogramowanie, umożliwiające użytkownikowi korzystanie z urządzenia w sposób prosty i efektywny, pozwalające szybko przeprowadzić pomiary, a także przedstawić ich wyniki w najodpowiedniejszej w danym przypadku postaci.

Z całą pewnością nie jest to zadanie łatwe i do końca rozwiązane, o czym może świadczyć znaczna różnorodność rozwiązań spotykanych w produkowanych obecnie analizatorach [4, 7, 8, 9, 10].

Poniżej zostały przedstawione niektóre aspekty konstrukcji analizatora układów logicznych.

Układy wejściowe

Zwykle układy wejściowe analizatora układów logicznych składają się z sond wejściowych oraz rejestrów buforujących. Zadaniem sond jest połączenie badanego układu z pozostałymi modułami analizatora, dlatego muszą się one charakteryzować odpornością na znaczne napięcia wejściowe (zarówno dodatnie, jak i ujemne), bardzo małą pojemnością wejściową, dużą rezystancją wejściową i, przede wszystkim, bardzo małym czasem propagacji. Inną cechą sond musi być zdolność do dokonywania konwersji poziomów logicznych. Analizator musi służyć do śledzenia pracy urządzeń cyfrowych zbudowanych z dowolnych układów, powinna więc istnieć możliwość zmiany progu przełączania sond wejściowych, tak aby zawsze możliwe było dostosowanie ich do określonej rodziny układów scalonych. Sondy powinny również charakteryzować się małym ciężarem oraz wymiarami, odpowiednią długością przewodów łączących itp.

Rejestry służą do buforowania danych doprowadzonych do pamięci śladu, gdyż czas trwania zapisu do rejestru wyzwalanego zboczem jest znacznie mniejszy od czasu zapisu do pamięci. Na wyjściach rejestrów utrzymują się stabilne dane przez czas niezbędny do zakończenia cyklu zapisu do pamięci śladu.

Zwiększanie częstotliwości próbkowania

Czas dostępu do pamięci stanowi istotny czynnik ograniczający maksymalną szybkość pracy analizatora układów logicznych, możliwe jest jednak ominięcie tego ograniczenia przez zastosowanie tzw. przeplotu pamięci śladu, polegającego na zastosowaniu p banków pamięci, pracujących na przemian

$$m_i \leftarrow x_k$$

$$i \leftarrow i \oplus 1$$

przy czym symbol $\oplus$ oznacza dodawanie modulo p.

Metoda ta pozwala na p-krotne przyspieszenie pracy pamięci.

Maksymalną częstotliwość pracy można zwiększyć również przez zastosowanie *potoku* rejestrów przesuwających sr

$$sr_1 \leftarrow x_k$$

$$\forall 2 \leq i \leq p.\ sr_i \leftarrow sr_{i-1}$$

Po p taktach zegara zawartości wszystkich rejestrów zostają przepisane do rejestrów równoległych r

$$\forall 1 \leq i \leq p.\ r_i \leftarrow sr_i$$

które w ciągu czasu odpowiadającego p taktom zegara synchronizującego pracę analizatora mogą być przepisane do wszystkich banków pamięci

$$\forall 1 \leq i \leq p.\ m_i \leftarrow r_i$$

Analizator wymaga więc zastosowania p banków pamięci, podobnie jak w przypadku przeplotu.

Jednemu cyklowi pamięci odpowiada n cykli zapisu do potoku rejestrów, czyli minimalny czas przyjęcia próbki równy jest

$$T_a = \frac{T_p + T_m}{p}$$

gdzie T_p jest sumą czasów wyprzedzenia informacji na wejściu rejestru sr oraz jego czasu propagacji, a T_m to czas dostępu do pamięci.

Wadą tej metody jest konieczność spełnienia warunku $pf \leq f_s$, gdzie f_s jest maksymalną dopuszczalną częstotliwością pracy rejestru sr oraz $f = 1/T_a$ i konieczność użycia większej liczby rejestrów, natomiast zaletą jest możliwość zastosowania zegara jednofazowego oraz tylko jednego układu adresowania pamięci. W obu metodach zwraca również uwagę różna organizacja pamięci śladu oraz różna struktura danych w tej pamięci. Przy okazji omawiania układów wejściowych warto zwrócić uwagę na zjawisko metastabilności.

Metastabilność

Podczas pracy synchronicznego przerzutnika bistabilnego niezbędne jest zapewnienie wymaganych czasów wyprzedzenia oraz trzymania informacji (t_{setup} oraz t_{hold}) względem aktywnego zbocza przebiegu zegarowego przy zmianie poziomu logicznego na wejściu informacyjnym przerzutnika. Jeżeli jednak wymagania te zostaną naruszone, to istnieje pewne niezerowe prawdopodobieństwo, że wyjście przerzutnika przejdzie do stanu metastabilnego, tzn. poziom napięcia wyjściowego przez pewien czas będzie się znajdował w obszarze zabronionym, tj. między stanem niskim i wysokim. Chociaż w praktyce przerzutnik wychodzi ze stanu metastabilnego po czasie niewiele większym od czasu propagacji tego przerzutnika, to jednak teoretycznie stan ten może trwać nieskończenie długo. Niebezpieczeństwo wynikające z wystąpienia zjawiska metastabilności jest oczywiste: jeżeli

wystąpienie stanu metastabilnego spowoduje przekroczenie założonego czasu propagacji, to istnieje potencjalne niebezpieczeństwo błędnego zadziałania systemu.

W prawidłowo zaprojektowanym układzie synchronicznym praktycznie nie ma możliwości przejścia przerzutników do stanu metastabilnego. Niestety możliwość wystąpienia tego stanu pojawia się w miejscach asynchronicznego sprzęgania układów, tzn. w miejscach, gdzie wykorzystywane są przerzutniki (rejestry) do synchronizowania przebiegów asynchronicznych. Jeżeli stan wejść przerzutnika (rejestru) zmienia się asynchronicznie względem przebiegu zegarowego doprowadzonego do układu, to oczywiście nie ma możliwości zapewnienia wymaganych czasów t_{hold} i t_{setup}, co nieuchronnie musi doprowadzić do występowania stanów metastabilnych. W analizatorze układów logicznych sytuacja taka ma miejsce, gdy pracuje on w trybie analizy czasowej, wtedy bowiem dane wejściowe są próbkowane asynchronicznie przy wykorzystaniu wewnętrznego generatora zegarowego.

Jeżeli przez F oznaczymy częstość występowania zdarzenia polegającego na tym, że wyjście przerzutnika znajduje się w stanie metastabilnym po czasie t od aktywnego zbocza zegara, to zachodzi następująca zależność [3]:

$$F = k_1 f_c f_d e^{-k_2(t-t_0)} \tag{3}$$

gdzie:

f_c – częstotliwość sygnału zegarowego,

f_d – częstotliwość zmian sygnału na wejściu danych,

t – czas od aktywnego zbocza zegara.

Na przykład dla rejestrów typu 374 parametry k_1, k_2 i t_0 wynoszą [3]:

Producent	Typ układu	k_1[sek]	k_2[1/ns]	t_0[ns]
MMI	LS374	2×10^{-7}	1.8	27.5
FAIRCHILD	F374	2×10^{-7}	11.5	17.5

Projektując urządzenie, w którym mogą wystąpić stany metastabilne, zakłada się pewien czas, w którym przerzutnik powinien wyjść ze stanu metastabilnego. Przekroczenie tego czasu może spowodować błąd w pracy urządzenia. Najczęściej przyjmuje się za dopuszczalną możliwość wystąpienia jednego błędu na rok.

Po przekształceniu wzoru (3) otrzymamy wzór, który pozwoli obliczyć maksymalną częstotliwość przebiegu zegarowego, przy którym częstość występowania błędu (czyli przekroczenia przez czas trwania stanu metastabilnego wartości t) jest nie większa od założonej:

$$f_c = \frac{F e^{k_2(t-t_0)}}{k_1 f_d}$$

Jeżeli dodatkowo założymy, że użycie analizatora o czasie akwizycji próbki T_a ma sens w przypadku, gdy okres przebiegów wejściowych T jest większy od czasu T_a (czyli $f_c > f_d$), to możemy obliczyć maksymalną częstotliwość przebiegu zegarowego:

$$f_c = \sqrt{F e^{\frac{k_2(t-t_0)}{k_1}}}$$

oraz odpowiadający jej minimalny czas przyjęcia próbki $T_a = 1/f_c$.

Możemy również obliczyć czas t przy założeniu częstotliwości przebiegu zegarowego f_c oraz zadowalającej nas wartości F:

$$t = t_0 + \frac{1}{k_2} \ln(k_1 f_c^{k_2/F}) \tag{4}$$

Przy wykorzystaniu danych z zestawienia przedstawionego wyżej oraz przyjęciu przykładowych wartości:

$f_c = 10$ MHz
$F = 1/\text{rok} = 3.2 \times 10^{-8}$

możemy ze wzoru (4) obliczyć czas t dla dwóch rejestrów typu 374, jak poniżej:

Producent	Typ układu	t[ns]	t_p[ns]
MMI	LS374	46	19
FAIRCHILD	F374	20	6.1

Czas t_p to katalogowa wartość czasu propagacji sygnału od zegara do wyjścia dla tych typów układów. Z powyższego zestawienia widać, że czas trwania stanu metastabilnego jest ok. 2–3 razy dłuższy od czasu propagacji (dla założonych wartości $F = 1/\text{rok}$ oraz $f_c = 10\text{MHz}$) i nawet w przypadku szybkiego rejestru F374 wynosi aż 20 ns.

Z punktu widzenia poprawności wyników analizy istotna jest jedynie częstość występowania zjawiska metastabilności, nie jest natomiast istotny czas jego trwania. Każde wystąpienie zjawiska metastabilności oznacza potencjalną możliwość zapisania do pamięci śladu wartości błędnej, tzn. różnej od stanu wejścia w momencie wystąpienia aktywnego zbocza sygnału zegarowego. W celu uniknięcia błędów należy dążyć do tego, by szerokość strefy $t_{setup} + t_{hold}$ rejestrów wejściowych była jak najmniejsza.

W przypadku komparatorów stanów suma czasu trwania stanu metastabilnego oraz czasu komparacji powinna być mniejsza od okresu zegara synchronizującego pracę analizatora w celu niedopuszczenia do błędnego obliczenia warunku śladowania.

Wykrywanie krótkich impulsów

Za krótkie impulsy uznaje się impulsy, których czas trwania jest mniejszy od okresu próbkowania. Zwykle są one generowane przez wadliwie działające układy. Takie impulsy pozostaną niezauważone, jeżeli pojawią się między chwilami próbkowania, dlatego często stosuje się dodatkowe układy służące do ich wykrywania, a informacja o ich wystąpieniu zapisywana jest w dodatkowej pamięci analizatora (ang. glitch memory). Czasem, w celu wyeliminowania dodatkowej pamięci, stosuje się dwa tryby przyjmowania danych: *sample* oraz *latch*. W trybie *sample* krótkie impulsy nie są w ogóle wykrywane, natomiast w trybie *latch* zostają one zapamiętane w zwykłej pamięci śladu w następnym takcie zegara (po ich wystąpieniu), w postaci informacji o zmianie stanu wejść.

Pamięć odniesienia

Analizator układów logicznch jest zwykle wyposażony w pamięć odniesienia (ang. reference memory), do której użytkownik może wprowadzić dane wzorcowe. Zawartość tej pamięci może następnie zostać programowo porównana z zawartością pamięci śladu, a użytkownik zostanie poinformowany o znalezionych różnicach. Do testowania sprzętu cyfrowego przydatna jest możliwość automatycznego powtarzania procesu śladowania (ang. multiple-shot acquisition) do czasu, gdy zawartość pamięci śladu będzie taka sama (lub różna), jak zawartość pamięci odniesienia.

System sterujący

Nowoczesny analizator układów logicznych musi być wyposażony w efektywny mikroprocesorowy system sterujący, do którego zadań należy m.in.:

- sterowanie pracą analizatora;
- przyjmowanie poleceń od użytkownika;
- prezentacja danych pomiarowych;
- obsługa interfejsu komunikacyjnego;
- przeprowadzanie automatycznego testu analizatora.

3. Analiza oprogramowania

Analiza oprogramowania (ang. software analysis) ma na celu optymalizację tworzonego oprogramowania czasu rzeczywistego. System analizy oprogramowania (ang. software analysis workstation) pozwala na uzyskanie różnego rodzaju danych o wykonywanym przez mikrokomputer programie, a w szczególności:

1. prezentuje projektantowi, w postaci symbolicznej, rzeczywiście wykonany przez system program (ang. symbolic trace).

2. informuje projektanta, jakie fragmenty programu zostały wykonane podczas testowego wykonania tego programu (ang. code coverage);

3. informuje projektanta, ile czasu trwało wykonanie poszczególnych modułów programu (ang. module duration);

Aspekt (1) jest charakterystyczny dla analizy stanów magistral systemu mikroprocesorowego, natomiast aspekty (2) i (3) analizy oprogramowania są charakterystyczne dla analizy efektywności (ang. performance analysis), mającej na celu optymalizację oprogramowania oraz sprzętu pod względem czasowym.

3.1. Analiza stanów magistral

Analiza stanów logicznych w odniesieniu do systemu mikroprocesorowego to analiza stanów magistral. Polega ona na rejestracji kolejnych stanów magistral systemu mikroprocesorowego podczas wykonania programu. Zapamiętane stany stanowią tzw. ślad przebiegu programu, który może być następnie przedstawiony użytkownikowi – zwykle w postaci symbolicznej. Analiza stanów magistral daje możliwość śledzenia wykonania programu w czasie rzeczywistym (symboliczny ślad instrukcji oraz przesyłów danych stanowi obraz wykonania programu).

Użytkownik ma możliwość zadeklarowania warunków śladowania, dzięki czemu zarejestrowane będą tylko interesujące go fragmenty programu. Warunkiem śladowania może być wykonanie wyróżnionej instrukcji lub dostęp do określonej lokacji pamięci danych programu. Zapewnienie możliwości użycia warunków śladowania jest bardzo istotne, ponieważ z uwagi na ograniczoną pojemność pamięci śladu nie jest możliwe zarejestrowanie wszystkich wykonanych instrukcji, niezbędne jest więc wyodrębnienie stosunkowo niewielkich, interesujących użytkownika fragmentów programu.

Obok stanu magistrali zapamiętywany jest stan zegara czasu rzeczywistego, co umożliwia określenie czasu wykonania poszczególnych instrukcji (z uwzględnieniem stanów WAIT, HALT itp.).

Typowy system mikrokomputerowy zawiera trzy magistrale:

- danych;
- adresową;
- sygnałów sterujących.

Dla przykładu, strukturę magistrali mikroprocesora Intel 80386 można przedstawić następująco za pomocą notacji zbliżonej do języka Pascal:

```
type Bus = record
     D     : array [0..31] of bit;
     A     : array [2..31] of bit;
     BE0#  : bit;
     BE1#  : bit;
     BE2#  : bit;
     BE3#  : bit;
     BS16# : bit;
     WR#   : bit;
     DC#   : bit;
     MIO#  : bit;
     LOCK# : bit
end;
```

Tak przedstawiona struktura magistral określa strukturę pamięci śladu analizatora. W przypadku procesorów nie tworzących kolejki rozkazów do wykonania (ang. prefetching) pamięć śladu analizatora, po zakończeniu cyklu śladowania, zawiera sekwencję wszystkich cykli rozkazowych wykonywanych instrukcji i_k, we właściwej kolejności

$$a = i_1 \hat{} i_2 \hat{} i_3 \hat{} \ldots$$

Jeżeli procesor tworzy kolejkę rozkazów, to

$$a \ interleaves(i_1, i_2, i_3 \ldots)$$

przy czym przemieszane są również cykle pobrania rozkazu i wykonawcze. Aby określić właściwą kolejność kodów w pamięci śladu, potrzebne są dodatkowe informacje, np. ślad specjalnych sygnałów generowanych przez mikroprocesor i informujących o stanie wewnętrznej kolejki rozkazów; można też stosować metody heurystyczne [4]. Wynika stąd podział funkcji analizatora na funkcje sprzętowe, realizowane w czasie rzeczywistym, oraz funkcje programowe, polegające na odpowiednim przetworzeniu zebranych danych (ang. post-processing).

Użycie analizatora pozwala na zebranie danych o pracy systemu komputerowego bez konieczności jego spowalniania (eliminacja tzw. *probe effect*). Uzyskane dane mogą być wykorzystane do sterowania programami uruchomieniowymi (ang. debugger) wykorzystującymi tzw. historie zdarzeń (praca [5] przedstawia przegląd metod uruchamiania programów współbieżnych).

Często analizator stanów magistral (lub analizator oprogramowania) wchodzi w skład bardziej złożonych systemów uruchomieniowych, tzw. emulatorów układowych [6].

Pomiar rzeczywistego czasu wykonania modułów programowych jest istotny zwłaszcza w przypadku programów uzależnionych czasowo, gdyż błędy czasowe są w ich przypadku równie groźne jak błędy specyfikacji lub realizacji algorytmu, lecz często jeszcze trudniejsze do wykrycia.

3.2. Analiza efektywności

Celem analizy efektywności jest dostarczenie programiście maksymalnej ilości danych, umożliwiających optymalizację tworzonego przez niego oprogramowania. Najczęściej oprogramowanie optymalizuje się pod kątem objętości kodu oraz szybkości wykonania.

3.2.1. Pomiar czasu wykonania modułu

Pomiar czasu wykonania modułu (ang. module duration measurement) umożliwia użytkownikowi uzyskanie informacji o czasach wykonania instrukcji, grup instrukcji, procedur lub segmentów programu. Możliwy jest również pomiar odstępów czasowych rozdzielających poszczególne wykonania danego fragmentu programu. Dane nie są statystyczne, więc użytkownik otrzymuje informację o czasie każdego wykonania określonego modułu, nie tylko średnią z wielu wykonań. Jest to szczególnie ważne w przypadku oprogramowania czasu rzeczywistego, gdzie nawet bardzo rzadkie przekroczenie dopuszczalnego czasu obliczeń może być niedopuszczalne.

Możliwy jest również pomiar całkowitego czasu trwania wszystkich modułów pro-

gramu równocześnie, co pozwoli określić, które z nich wpływają w największym stopniu na czas wykonania całego programu. Zidentyfikowanie najbardziej czasochłonnych procedur jest pierwszym krokiem do czasowej optymalizacji programu.

Pomiar czasu wykonania modułu wymaga zarejestrowania czasu rozpoczęcia oraz zakończenia wykonania danego modułu programowego (procedury, funkcji). Analizator może tego dokonać rozpoznając pobranie kodu pierwszego i ostatniego rozkazu danego modułu.

Dostęp procesora do określonej lokacji pamięci, w celu pobrania kodu rozkazu do wykonania, stanowi określone *zdarzenie* zachodzące w systemie. Inne tego typu zdarzenia, również związane z dostępem do pamięci systemu, to np. odczyt danej, portu lub reakcja na przerwanie. Zwykle projektanta interesują zależności czasowe między wieloma zdarzeniami w systemie, analizator oprogramowania musi więc posiadać układy dokonujące równoczesnego porównywania stanu magistral z wieloma warunkami. Najlepsze do tego celu wydaje się być zastosowanie układów wzorowanych na pamięci asocjacyjnej, realizujących funkcję

$$\varphi : A \times D \times S \rightarrow e$$

A, D i S to stany magistral systemu (odpowiednio adresowej, danych i sterującej), natomiast e jest identyfikatorem zdarzenia, różnym dla każdego zdarzenia interesującego projektanta

$$e \in \{e_0, e_1 \ldots e_p\}$$

przy czym e_0 jest identyfikatorem wszystkich zdarzeń bez znaczenia dla użytkownika, tzn. tych, których wystąpienie nie musi zostać zarejestrowane.

Gdy komparator wykryje, że $e \neq e_0$, analizator zarejestruje w swojej pamięci zdarzeń parę (e, clk), gdzie clk jest stanem zegara czasu rzeczywistego. Zarejestrowane dane są przetwarzane programowo po zakończeniu cyklu pracy analizatora (zapełnieniu pamięci).

3.2.2. Zliczanie wywołań modułu

Istotną, z punktu widzenia programisty, informacją jest liczba wywołań danej procedury w czasie wykonania całego programu. Umożliwia ona stwierdzenie, czy na całkowity czas wykonania tej procedury w większym stopniu wpływa znaczna liczba jej wywołań, czy też mała szybkość jej wykonania. Jeżeli wywołań jest niewiele, przyspieszenie wykonania procedury może być uzyskane np. przez zmianę algorytmu lub użycie języka asemblera. Jeżeli wywołań jest dużo, to poprawę może przynieść uproszczenie sposobu przekazywania parametrów aktualnych do procedury lub zakodowanie jej w postaci makrodefinicji (ang. inline code).

3.2.3. Mapa wykonania programu

Mapa wykonania programu (ang. code coverage map) pozwala na określenie, które fragmenty programu (instrukcje, procedury) zostały wykonane podczas przebiegu programu, a także które komórki pamięci operacyjnej zostały zapisane lub odczytane. Jest to istotne zwłaszcza podczas testowania programu, gdyż pozwala na upewnienie się, czy wszystkie procedury zostają wykonane, czy program nie korzysta z niedozwolonych obszarów pamięci, czy nie nastąpiło przepełnienie stosu itp.

4. Przykłady realizacji

Niniejszy rozdział zawiera omówienie sposobów realizacji dwóch istotnych elementów analizatora układów logicznych – komparatora stanów i komparatora sekwencji stanów. Przykłady pochodzą z systemów zaprojektowanych i wykonanych w IITiS-PAN w Gliwicach.

4.1. Komparator sekwencji stanów

Często stosowanymi operatorami służącymi do łączenia warunków elementarnych w *sekwencje* są operatory *ne* oraz *ni* (będące skrótami angielskich określeń *next eventually* i *next immediately*).

Dla przykładu oznaczmy przez W_1, W_2, W_3 i W_4 zdarzenia polegające na spełnieniu, przez dane wejściowe, pewnych warunków. Wtedy

$$W_1 \ ne \ W_2$$

oznacza, że po wystąpieniu zdarzenia W_1, zdarzenie W_2 może wystąpić w dowolnej chwili, natomiast

$$W_1 \ ni \ W_2$$

oznacza, że zdarzenie W_2 musi wystąpić bezpośrednio po W_1.

W celu zilustrowania powyższych wyjaśnień można przykładową sekwencję

$$W_1 \ ne \ W_2 \ ni \ W_3 \ ne \ W_4$$

przedstawić za pomocą notacji zbliżonej do języka Pascal:

1: if W_1 then goto 2 else goto 1;
2: if W_2 then goto 3 else goto 2;
3: if W_3 then goto 4 else goto 2;
4: if W_4 then goto 5 else goto 4;
5: (* sekwencja znaleziona *)

Analizator układów logicznych, w którym do opisu sekwencji warunków zastosowano operatory *ni* i *ne*, działa zwykle wg następującego algorytmu:

$$A \parallel C \parallel CB$$

gdzie

$$CB = (trM \rightarrow seq \rightarrow trNM \rightarrow brcond \rightarrow STOP)$$

natomiast A i C zostały przedstawione w poprzednim rozdziale.

W wyniku pracy według tak zdefiniowanego algorytmu analizator:

1. zarejestruje m próbek (zdarzenie trM);
2. rozpocznie poszukiwanie sekwencji warunków zadeklarowanej przez uzytkownika (z użyciem operatorów *ni* i *ne*), równocześnie śladując dane wejściowe;
3. po znalezieniu sekwencji (zdarzenie *seq*) zarejestruje jeszcze $n-m$ próbek (zdarzenie $trNM$).

W wyżej przedstawionym algorytmie n jest pojemnością pamięci śladu, natomiast m jest liczbą deklarowaną przez użytkownika, określającą położenie zdarzenia wyzwalającego (czyli poszukiwanej sekwencji warunków) w tzw. oknie danych. Deklarując m, użytkownik może wpływać na liczbę zarejestrowanych próbek poprzedzających wystąpienie sekwencji warunków.

W analizatorach stanów oraz analizatorach oprogramowania często stosuje się złożone komparatory sekwencji, pozwalające na bardziej precyzyjne określenie sekwencji interesujących użytkownika. Przykładem może być komparator zastosowany w analizatorze systemów opartych na mikroprocesorze Intel 80386 – RTDS-32 [4]. Komparator ten został zaprojektowany jako układ mikroprogramowalny, dzięki czemu charakteryzuje się prostą konstrukcją i dużymi możliwościami funkcjonalnymi.

Użytkownik deklaruje trzy zbiory warunków elementarnych

- **A** – zbiór warunków, które muszą być spełnione przez stany magistrali adresowej mikroprocesora;

- D – zbiór warunków, które muszą być spełnione przez stany magistrali danych mikroprocesora;
- S – zbiór warunków, które muszą być spełnione przez stany magistrali sterującej mikroprocesora;

przy czym w skład każdego zbioru może wchodzić maksymalnie osiem warunków elementarnych

$$\mathrm{A} = \{A_1 \ldots A_8\}$$

$$\mathrm{D} = \{D_1 \ldots D_8\}$$

$$\mathrm{S} = \{S_1 \ldots S_8\}$$

Użytkownik może zadeklarować do szesnastu warunków postaci

Label$_l$: if $f_l(A_j, D_j, S_j)$ then trace
if $g_l(A_j, D_j, S_j)$ then break
goto $h_l(A_k, D_k, S_k)$

$f_l(A_j, D_j, S_j)$ jest warunkiem śladowania;
$g_l(A_j, D_j, S_j)$ jest warunkiem zatrzymania;
$h_l(A_k, D_k, S_k)$ określa etykietę skoku, czyli numer następnego warunku.

f_l, g_l, h_l zostają określone przez użytkownika, przy czym do ich definicji można stosować operatory logiczne AND, OR, NOT oraz relacje $>, <, \leq, \geq, \in$.

4.2. Komparator stanów

Jednym z możliwych rozwiązań konstrukcyjnych komparatora stanów może być struktura zbudowana z bramek logicznych. Metoda ta wydaje się być godna polecenia w przypadku posiadania możliwości wykonania całego komparatora w postaci jednego układu scalonego. Konstrukcja wykorzystująca elementy SSI, proste układy typu PLD lub też gotowe bloki funkcjonalne (np. komparatory 7485) charakteryzować się będzie małą prędkością działania oraz dużym stopniem komplikacji.

Inny sposób polega na użyciu pamięci statycznych o krótkim czasie dostępu. Funkcję realizowaną przez pamięć o organizacji $2^n \times 1$ bitów można przedstawić następującym wzorem:

$$\mathrm{mem}(a) = \bigvee_{i=0}^{2^n-1} m(i)\delta(a,i) \tag{5}$$

gdzie:

$\mathrm{mem}(a)$ oznacza stan wyjścia pamięci, gdy wejście adresowe przyjmie wartość a,

$m(i)$ jest wartością (0 lub 1) zapisaną pod adresem i,

$$\delta(a,i) = \begin{cases} 1, & \text{gdy } a = i \\ 0, & \text{w przeciwnym przypadku,} \end{cases}$$

znak $\bigvee$ oznacza wielokrotną sumę logiczną.

Odpowiednio wypełniona pamięć o pojemności 2^n bitów może służyć do sprawdzenia, czy dana n-bitowa liczba binarna x równa jest liczbie y, np. jeżeli pamięć zostanie wypełniona tak, aby

$$m(a) = \begin{cases} 1 & \text{dla } a = y \\ 0 & \text{dla } a \neq y \end{cases}$$

natomiast na wejścia adresowe pamięci podana zostanie liczba x, to łatwo stwierdzić, że

$$\mathrm{mem}(x) = \bigvee_{i=0}^{2^n-1} m(i)\delta(x,i) =$$

$$= \bigvee_{i=0}^{y-1} m(i)\delta(x,i) + m(y)\delta(x,y) + \bigvee_{i=y+1}^{2^n-1} m(i)\delta(x,i) =$$

$$= \delta(x,y) = \begin{cases} 1 & \text{gdy } x = y \\ 0 & \text{gdy } x \neq y \end{cases}$$

Komparator zbudowany z wykorzystaniem pamięci może wykryć również spełnienie sumy logicznej warunków $x = y_1 \vee x = y_2 \vee x = y_3 \ldots$

Niestety komparator przedstawiony wyżej ma również poważną wadę: wymagana pojemność pamięci rośnie wykładniczo wraz ze wzrostem n. Możliwe jest zmniejszenie pojemności pamięci kosztem pewnego zmniejszenia możliwości funkcjonalnych. Zmodyfikowany komparator, wykorzystujący pamięci typu RAM, został zastosowany w analizatorze systemów mikrokomputerowych RTDS-32 [4].

Załóżmy, że komparator stanów będzie wykrywać przynależność wektora wejściowego x, traktowanego jak liczba binarna, do przedziału $[a,b]$, a więc

$$\text{warunek spełniony} \Leftrightarrow x \in [a,b] \Leftrightarrow x \geq a \wedge x \leq b$$

$n \times d$-bitową liczbę binarną można podzielić na n grup po d bitów traktowanych również jak liczby binarne

$$x = \sum_{i=0}^{n-1} x_i B^i, \quad a = \sum_{i=0}^{n-1} a_i B^i, \quad b = \sum_{i=0}^{n-1} b_i B^i$$

gdzie $B = 2^d$. Warunki $x \geq a$ oraz $x \leq b$ można zdefiniować następująco:

$$x \geq a \Leftrightarrow \exists k.\ (x_k \geq a_k \wedge \forall i > k.\ x_i = a_i) \tag{6}$$

$$x \leq b \Leftrightarrow \exists j.\ (x_j \leq b_j \wedge \forall p > j.\ x_p = b_p) \tag{7}$$

Definiujemy

$$\begin{aligned} \mathrm{eq}(a_m) &\equiv x_m = a_m \\ \mathrm{ge}(a_m) &\equiv x_m \geq a_m \end{aligned} \tag{8}$$

oraz

$$\begin{aligned} \mathrm{eq}(b_m) &\equiv x_m = b_m \\ \mathrm{le}(b_m) &\equiv x_m \leq b_m \end{aligned} \tag{9}$$

Powyższe funkcje można zrealizować za pomocą pamięci, korzystając z (5).

Zgodnie z (6) i (8) możemy napisać, że

$$\begin{aligned} x \geq a \Leftrightarrow \mathrm{ge}(a_{n-1}) \vee \mathrm{ge}(a_{n-2})\mathrm{eq}(a_{n-1}) \vee \ldots \vee \mathrm{ge}(a_0)\mathrm{eq}(a_{n-1}) \ldots \mathrm{eq}(a_1) = \\ = \bigvee_{i=0}^{n-1} [\mathrm{ge}(a_i) \bigwedge_{j=i+1}^{n-1} \mathrm{eq}(a_j)] \end{aligned} \tag{10}$$

przy czym symbol $\bigwedge$ definiujemy następująco:

$$\bigwedge_{i=p}^{n} z_i = \begin{cases} z_p \cdot \ldots \cdot z_n & \text{jeżeli } p \leq n \\ 1 & \text{jeżeli } p > n \end{cases}$$

natomiast znak $\vee$ oznacza sumę logiczną.

$x \leq b$ określimy analogicznie:

$$x \leq b \Leftrightarrow \mathrm{le}(b_{n-1}) \vee \mathrm{le}(b_{n-2})\mathrm{eq}(b_{n-1}) \vee \ldots \vee \mathrm{le}(b_0)\mathrm{eq}(b_{n-1})\ldots\mathrm{eq}(b_1) = \bigvee_{i=0}^{n-1}[\mathrm{le}(b_i) \bigwedge_{j=i+1}^{n-1} \mathrm{eq}(b_j)] \qquad (11)$$

Ponieważ

$$x \in [a,b] \Leftrightarrow x \geq a \wedge x \leq b$$

sprzętowe sprawdzenie warunku przynależności do przedziału możliwe jest przez realizację iloczynu wyrażeń (10) i (11).

Komparator analizatora zbudowanego w IITiS-PAN (przeznaczony do komparacji stanów magistral mikroprocesora Intel 80386) wykorzystuje pamięci statyczne, programowane przez komputer sterujący analizatorem, do realizacji funkcji $\mathrm{eq}(a_i)$, $\mathrm{eq}(b_i)$, $\mathrm{ge}(a_i)$ i $\mathrm{le}(b_i)$. Magistrala mikroprocesora Intel 80386 jest magistralą 32-bitową ($n \times d = 32$), natomiast użyte pamięci komparatora mają 8 wejść adresowych ($d = 8$), czyli $n = 4$. Układ typu PAL realizuje funkcję iloczynu wyrażeń (10) i (11), doprowadzonego do postaci sumy iloczynów (możliwa jest więc realizacja przez dwupoziomową strukturę bramek AND-OR, co ma istotny wpływ na szybkość działania komparatora).

5. Podsumowanie

Analiza układów logicznych i analiza oprogramowania są jednymi z podstawowych metod stosowanych przez projektantów układów cyfrowych i mikroprocesorowych. Celem niniejszego artykułu było przedstawienie tych metod zarówno od strony podstaw teoretycznych, jak i zakresu stosowalności. Artykuł zawiera również dyskusję wybranych elementów konstrukcji analizatora układów logicznych. Przedstawione rozwiązania nie są jedynymi możliwymi, wręcz przeciwnie – istniejące konstrukcje cechują się znacznym zróżnicowaniem zarówno pod względem funkcjonalnym, jak i parametrów technicznych, co odzwierciedla poglądy ich autorów na metodę analizy układów logicznych i możliwości techniczne będące do ich dyspozycji. Jednak mimo mnogości konkretnych rozwiązań, wszystkie z nich posiadają pewne, przedstawione w niniejszym artykule, cechy wspólne.

A Notacja CSP

Notacja CSP może służyć do opisu różnego rodzaju systemów i zjawisk sekwencyjnych jak i równoległych. Podstawowymi pojęciami wprowadzanymi przez CSP są *proces* (oznaczany dużymi literami, np. P) i *zdarzenie* (oznaczany małymi literami, np. a). Proces o nazwie *STOP* nie bierze udziału w żadnych zdarzeniach. Słownik procesu P, oznaczany αP, jest zbiorem wszystkich zdarzeń, w których może brać udział proces P. Procesy mogą być wykonywane równolegle, przy czym równoczesne wykonanie procesów $P1$ i $P2$ oznaczane jest $P1 \parallel P2$. Symbol $\rightarrow$ służy do łączenia zdarzeń w definicji procesu wg kolejności ich występowania, np. $P = (a \rightarrow b \rightarrow P)$. Zapis rekurencyjny pomaga opisywać zjawiska zachodzące cyklicznie. Symbol $\square$ (czytany 'lub') pozwala na opis procesów mogących brać udział w alternatywnych 'ścieżkach' zdarzeń, np. $P = (a \rightarrow b \rightarrow P \,\square\, c \rightarrow P)$. W zdarzeniach należących do słowników większej liczby procesów muszą równocześnie brać udział wszystkie te procesy, w pozostałych zdarzeniach procesy biorą udział niezależnie.

Ślad procesu (oznaczany małą literą, np. t) jest zarejestrowaną sekwencją zdarzeń, w których dany proces brał udział, np. $\langle a, b, c, c \rangle$. Operacje, jakie można wykonać na śladach to, między innymi, łączenie, np. $t_1 = t_2 \hat{} t_3$ i obliczenie długości śladu $\#t$. Pierwszy element śladu t oznacza się przez t_0, natomiast przez t' oznacza się ślad t bez pierwszego elementu. P^* jest zbiorem wszystkich śladów możliwych do uzyskania z elementów αP, natomiast $traces(P)$ jest zbiorem wszystkich możliwych śladów P. Wyrażenie $t_1 interleaves(t_2, t_3)$ oznacza, że ślad t_1 składa się z przemieszanych fragmentów śladów t_2 i t_3.

LITERATURA

[1] Hoare C.A.R.: Communicating Sequential Processes, Communications of the ACM, vol. 21, no. 8, 1978.

[2] Hoare C.A.R.: Communicating Sequential Processes, Prentice-Hall International, UK, Ltd., 1985.

[3] PAL/PLE Programmable Logic Handbook, Monolithic Memories, Inc., 1985.

[4] Raport Zespołu Mikroprogramowania IITiS-PAN: Stanowisko wspomagające analizę systemów mikrokomputerowych bazujących na procesorze Intel 80386, Instytut Informatyki Teoretycznej i Stosowanej PAN, Gliwice 1990.

[5] McDowell C.E., Helmbold D.P.: Debugging Concurrent Programs, ACM Computing Surveys, December 1989.

[6] Zonenberg D.: Emulator układowy mikroprocesorów jako narzędzie uruchomieniowe w ujęciu programistycznym, Informatyka, Luty 1992.

[7] Tektronix Catalog, Tektronix, Inc., 1991.

[8] Electronic Measurement, Design, Computation Catalog, Hewlett-Packard Company, 1987.

[9] 318W/318W S1 Logic Analyzer Manual, Tektronix, Inc., 1986.

[10] HP 1652B/1653B Logic Analyzers Operation Reference, Hewlett-Packard Company, 1990.

Recenzent: Doc. dr inż. Henryk Małysiak

Wpłynęło do Redakcji 23 kwietnia 1992 r.

Abstract

Logic analysis and software analysis belong to the basic tools used by digital and microprocessor systems designers. The goal of this paper was to present both the theoretical foundations of those techniques and their areas of applicability. The paper also contains a discussion of several aspects of logic analyzer design. The solutions presented in the paper are not the only ones possible, on the contrary, typical implementations vary in functionality and performance, thus reflecting both the views of their designers on logic analysis and the technology at their disposal. Despite the proliferation of logic analysis tools, they all share certain features presented in the paper.