JTAG - światowy standard testowania i programowania układów cyfrowych, część 2

W poprzednim numerze EP omówiliśmy szczegóły standardu programowania i testowania układów cyfrowych - JTAG. Standard ten znalazł szerokie poparcie wśród producentów układów cyfrowych - praktycznie wszystkie obecnie produkowane układy programowalne oraz większość zaawansowanych kontrolerów jest wyposażona w złącze zgodne z JTAG. Bardzo ciekawe uzupełnienie tego standardu zaproponowała niedawno Altera. W pracowniach badawczych tej firmy powstał uniwersalny język obsługi interfejsu JTAG - nosi on nazwę JAM.

O tym, dlaczego powstanie takiego języka było konieczne, jak z niego korzystać, gdzie zdobyć dodatkowe materiały, oprogramowanie i specyfikację języka - dowiecie się z artykułu.

Co to jest *JAM*?

JAM jest interpretowanym językiem programowania wysokiego poziomu, który opracowano z myślą o programowaniu i testowaniu struktur programowalnych w systemie (ang. In System Programmable) poprzez interfejs JTAG. Składnia języka jest zbliżona do popularnego BASICa, przy czym zestaw instrukcji rozszerzono o polecenia obsługi interfejsu JTAG.

Na **list. 1** przedstawiono przykładowy program, napisany w języku *JAM*, obliczania silni kolejnych liczb, a wynik obliczeń jest zapisywany w pliku (**list. 2**). Jest to jeden z przykładów obrazujących uniwersalność tego języka. Nieco inaczej wygląda plik wygenerowany przez system projektowy zawierający informacje o konfiguracji i algorytmie programowania układu docelowego. Przykład (znacznie skrócony) takiego pliku przedstawiono na **list. 3**. Wszyscy Czytelnicy, którzy posługiwali się lub posługują BASICiem, zauważą na tym listingu szereg znanych rozkazów. Także struktura logiczna tego pliku źródłowego nie odbiega od typowych konstrukcji programu w tym języku. Fragment programu rozpoczynający się od etykiety *L0:* odpowiada za ustalenie w jakim stanie znajduje się programowany układ (czy np. należy go skasować przed programowaniem) i jakiej operacji wymaga użytkownik od programu.

Na **rys. 1** przedstawiono sposób wykorzystania języka *JAM* w typowej aplikacji. Program *JAM Composer* odpowiada za konwersję pliku wynikowego (np. w formacie JEDEC, HEX), który jest generowany przez program projektowy, do postaci tekstowej zgodnej ze specyfikacją *JAM*.

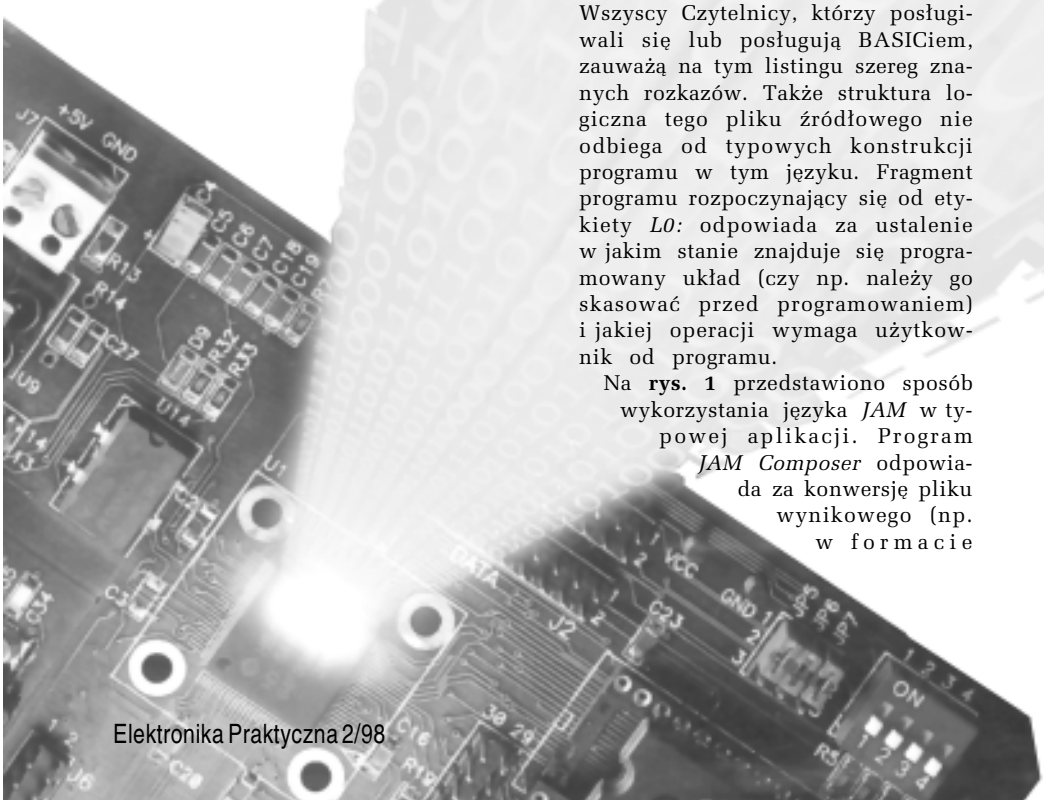
W przypadku, kiedy projektant korzysta z oprogramowania narzędziowego nowej generacji (np. systemu Max+Plus II firmy Altera, w wersjach od 8.0 począwszy) *JAM Composer* stanowi integralną część pakietu i nie ma konieczności korzystania z dodatkowych programów narzędziowych (**rys. 2**).

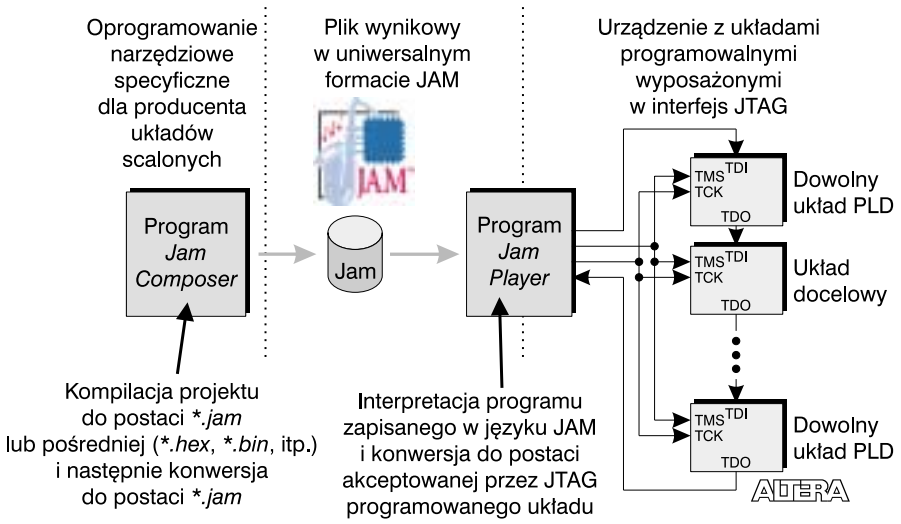
W przypadku, kiedy wykorzystywany przez konstruktora system projektowy nie potrafi bezpośrednio generować programu w standardzie *JAM*, możliwe jest zastosowanie zewnętrznego konwertera programowego, który zmieni postać pliku ze standardu JEDEC, HEX, POF lub dowolnego innego, na postać zgodną ze specyfikacją *JAM* (**rys. 3**).

Zadaniem programu-konwertera jest m.in. zapisanie w pliku wyjściowym programu opisującego sposób programowania lub testowania wybranej struktury PLD. Założeniem przyjętym przez twórców języka *JAM* było, że programy konwertujące będą dostarczane przez firmy tworzące oprogramowanie projektowe dla wybranych układów PLD lub bezpośrednio przez producentów struktur.

Profity wynikające ze stosowania języka *JAM*:

- możliwość uniezależnienia się od jednego dostawcy oprogramowania projektowego;
- oprogramowanie w formacie *JAM* może być stosowane na różnym sprzęcie, bez konieczności rekompilacji projektu;
- możliwość znacznego skrócenia czasu programowania układów;
- dzięki elastycznemu zestawowi instrukcji języka istnieje możliwość opisania przy jego pomocy dowolnego algorytmu programowania; tak więc obsługa układów, które dopiero pojawią się na rynku nie będzie stanowić żadnej trudności.

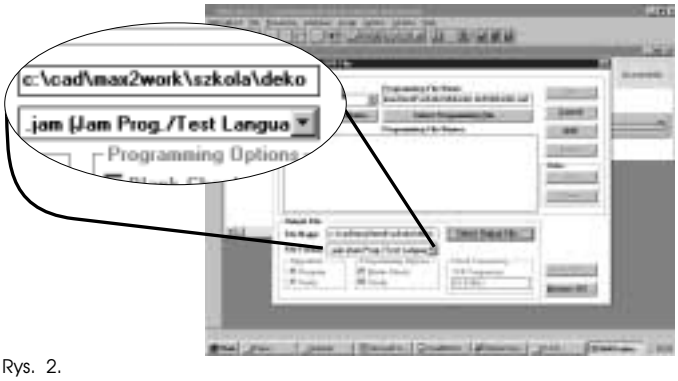




Rys.1.

Listing 2.

```
Factorial
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
Error on line 16: integer overflow.
Program terminated.
```



Rys. 2.

Niezbędnym elementem JAMowego „łańcucha" jest program noszący nazwę *JAM Player* (rys. 1, rys. 3). Zadaniem tego programu jest interpretacja poleceń języka JAM i realizacja zapisanego w pliku programu. Program ten może mieć postać pliku wykonywalnego np. dla komputera PC. Można go także zaimplementować w dowolnym systemie mikroprocesorowym, dzięki czemu procesor sterujący pracą tego systemu będzie mógł samodzielnie modyfikować struktury wykorzystanych w nim układów programowalnych.

Jak pokazaliśmy na przykładzie z list. 1, programy w języku *JAM* mogą wykonywać czynności zupełnie nie związane z procedurami programowania lub testowania układów PLD. Niezależnie od implementacji, *JAM Player* musi zapewniać pełną obsługę interfejsu JTAG, a także poleceń nie związanych bezpośrednio z jego sterowaniem.

Chcąc ułatwić zbudowanie na bazie *JAMu* nowego standardu obsługi interfejsu JTAG, Altera udostępniła kody źródłowe, napisane w języku C, wszystkich modułów wykorzystywanych przez program *JAM Player*. Program ten został opracowany w taki sposób, aby zminimalizować trudności z jego przeniesieniem na dowolny komputer lub mikrokontroler.

JAM w praktyce

Jedną z najważniejszych zalet języka *JAM* jest łatwość jego przenoszenia pomiędzy różnymi urządzeniami końcowymi i pełna niezależność od platformy sprzętowej na jakiej jest stosowany.

Na **rys. 4** przedstawiono schematycznie cztery przykładowe aplikacje tego języka. Program w języku *JAM*, wygenerowany przez dowolne narzędzie projektowe, można zastosować do konfiguracji układów programowalnych zamontowanych w systemie. W przypadku *Systemu 1* (rys. 4) program *JAM Player 1* znajduje się w pamięci procesora sterującego systemem.

Nieco inaczej wygląda sytuacja w przypadku *Systemu 2*. Tym razem *JAM Player 2* spełnia rolę programu obsługującego laboratoryjny programator układów programowalnych. Można go tutaj porównać z driverem dedykowanym pewnej grupie układów.

System 3 jest specjalizowanym testerem układów programowalnych. *JAM Player 3* spełnia w nim rolę podobną jak w przypadku *Systemu 2*, czyli drivera dedykowanym pewnej grupie układów. W przypadku korzystania ze standardowego testera lub programatora możliwe jest także wykorzystanie formatu *JAM*. Wymaga to jednak zastosowania konwertera programowego, który zmieni format zapisu programu na postać zrozumiałą dla programu obsługującego urządzenie (jak w przypadku *Systemu 4*).

Tak więc dzięki zastosowaniu języka *JAM* możliwe jest ograniczenie liczby stosowanych formatów zapisu

Listing 1.

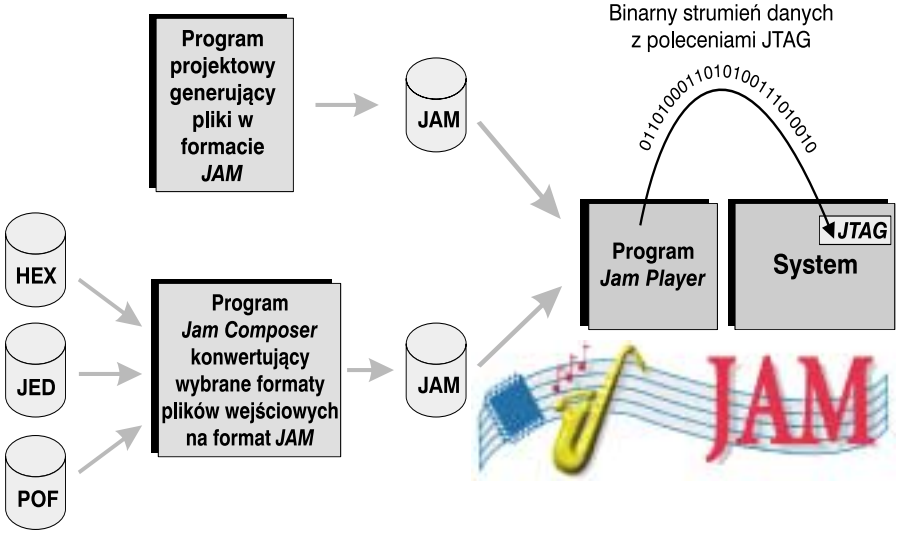
```
' Factorial program
'
' Copyright (C) Altera Corporation 1997
'

PRINT "Factorial";

INTEGER num;
INTEGER fact;
INTEGER count;

FOR num = 1 TO 20;
            LET fact = 1;
            FOR count = 1 TO num;
                        LET fact = fact * count;
            NEXT count;
            PRINT num, "! = ", fact;
NEXT num;

EXIT 0;
```



Rys. 3.

Listing 3.

```
NOTE CREATOR "POF to JAM converter
Version 8.0 6/12/97";

NOTE DEVICE "EPM7128S";

'Device #1: EPM7128S -
c:\cad\max2work\praca\demo.pof Mon
Mar 19 13:44:04 1997

BOOLEAN A21[104320] = ACA
mB300uAlQrh_g5rQjMMhrQQjMhTrthwLV@t
Vzt1@rVDz@@@qUjtLB@QvhwBd_Njc
MhrQQj_hwrQjiMhroth_sglQDlMhLrV@Nsh@w
zhMhR8N8nL8@gV@@AVrwhyLhr25
3QzMbyNfzq0P@7Q8Uzth253grQfeKBbYwh
ysAloRGJm@Q1dthrVQzN@w@@heMgro
XuzaGK0m8EJCeIhqASD6eqAz9KP6OjIhsTVbd_h_NQ
j_yUzdhq@hVw@hMJfLQL0j
C8s4641732BX1ni910_@@0;

BOOLEAN A22[65920] = ACA
mO200u@@@t@@@l@lzV@@@@_@@@yPMzxVf_t@@kD@@@F
I@@Yko@I100jwUvr4blW2
0R9OLZVxXDeX1_@@@zVf0OWMlcyOIj0G0Gj0Kqmjr@z@@
@x@@@cx@@Vj6Gh@@dK@
@lqVt@GPWyUIUePur@@FKC0G038KCik4i@@QvV41m
@@@l@fPN80u@1DlUIV01jb6
dGGDm@@@VGzGeOZ@t@V04w@XI2Ql1h2g3zd@@7o
CuVVpd@@@6f@@FJvVf1F32PPu
@t@@;

NOTE JAM_VERSION "1.0";
NOTE ALG_VERSION "1.0";
INTEGER A0[12] =
791, 237, 253, 261, 273, 281, 293,
317, 791, 791, 791, 791;
INTEGER A1[12] =
640, 80, 160, 240, 320, 400, 480,
640, 160, 200, 240, 280;
INTEGER A2[12] =
165, 0, 0, 0, 0, 0, 0, 0, 165, 165,
165, 165;
INTEGER A3[12] =
112, 0, 0, 0, 0, 0, 0, 0, 64, 80, 96, 112;
NTEGER V53 = 3;
INTEGER V54 = 3;
INTEGER A11[V53 * V54 * 11] =
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
96, 1, 0, 97, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
96, 0, 0, 97, 0, 1, 96, 1, 2, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
96, 0, 0, 0, 0, 0, 0, 0, 0, 0,
96, 2, 0, 97, 0, 1, 96, 1, 2, 94,
1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
95, 0, 0, 0, 0, 0, 0, 0, 0, 0,
95, 0, 0, 0, 0, 0, 0, 0, 0, 0,
94, 1, 0, 0, 0, 0, 0, 0, 0, 0;

' VARIABLES
INTEGER V0 = 1;
INTEGER A12[V0] = 4;
INTEGER A13[V0] = 1 | 4;
INTEGER A25[V0] = 10;
INTEGER V1 = V0 - 1;
INTEGER V2 = 0;
INTEGER V3 = 0;
INTEGER V4 = 1;
INTEGER A17[V0];
BOOLEAN b;
INTEGER i;
INTEGER j;
INTEGER k;
INTEGER l;
INTEGER V10;
INTEGER V11;
BOOLEAN A18[10];
BOOLEAN A19[10];
BOOLEAN A20[10];
INTEGER V14;
INTEGER V15;
INTEGER V42 = 0;
BOOLEAN V43 = 0;
BOOLEAN V44 = 0;
BOOLEAN V45 = 0;
INTEGER V46;
INTEGER V47;
INTEGER V48;
BOOLEAN V49 = 0;
BOOLEAN V50 = 0;
BOOLEAN DO_ERASE = 0;
BOOLEAN DO_BLANKCHECK = 0;
BOOLEAN DO_PROGRAM = 0;
BOOLEAN DO_VERIFY = 0;
BOOLEAN DO_SECURE = 0;
BOOLEAN DO_SECURE_ALL = 0;

L0:
CALL L3;
LET V4 = 1;
CALL L16;
IF (V2 == 0) THEN GOTO L1;
IF (V42 == 0) THEN CALL L33;
IF ((V42 == 0) && DO_ERASE) THEN CALL
L23;
IF ((V42 == 0) && DO_BLANKCHECK) THEN
CALL L32;
IF ((V42 == 0) && DO_PROGRAM) THEN
CALL L21;
IF ((V42 == 0) && DO_VERIFY) THEN
CALL L28;
IF ((V42 == 0) && DO_SECURE) THEN
CALL L131;
L1:
CALL L14;
EXIT V42;
L3:
INTEGER V64 = 0;
IF (DO_PROGRAM) THEN LET DO_ERASE =
1;
IF (DO_SECURE_ALL) THEN LET DO_SECURE
= 1;
IF ((DO_ERASE || DO_BLANKCHECK) &&
!DO_PROGRAM &&
(DO_VERIFY || DO_SECURE)) THEN LET
V42 = 1;
IF (V42 != 0) THEN GOTO L13;
LET V21 = 0;
FOR i = 0 TO V1;
LET j = 1;
LET k = 1;
IF (((A13[i] & 1) == 0) ||
((A13[i] & 4) == 0)) THEN GOTO L4;
LET V2 = A12[i];
LET j = A1[V2];
LET k = A5[V2];
L4:
LET V21 = V21 + j;
LET V64 = V64 + k;
NEXT i;
BOOLEAN A26[V21];
LET k = 0;
FOR i = 0 TO V1;
IF (((A13[i] & 1) == 0) ||
((A13[i] & 4) == 0)) THEN GOTO L5;
LET V2 = A12[i];
FOR j = 0 TO (A1[V2] - 1);
IF ((j % 5) == 4) THEN LET A26[k] = 1;
LET k = k + 1;
NEXT j;
GOTO L6;
L5:
LET k = k + 1;
L6:
NEXT i;
LET V16 = 0;
FOR i = 0 TO V1;
LET j = 1;
IF (((A13[i] & 1) == 0) ||
((A13[i] & 4) == 0)) THEN GOTO L10;
LET V2 = A12[i];
LET j = A0[V2];
L10:
LET V16 = V16 + j;
NEXT i;
LET V18 = V16;
BOOLEAN A28[V18];
LET V19 = V18;
IF (V21 > V19) THEN LET V19 = V21;
BOOLEAN A29[V19];
BOOLEAN A30[V19];
FOR i = 0 TO (V19 - 1);
LET A30[i] = 1;
NEXT i;
LET V20 = 2 * 5 * V64;
BOOLEAN A31[V20];
LET V12 = 0;
LET V13 = 0;
FOR i = 0 TO V1;
LET V12 = V12 + A25[i];
IF (A25[i] > V13) THEN LET V13 = A25[i];
NEXT i;
BOOLEAN A32[V12 + 50];
BOOLEAN A39[V13];
BOOLEAN A33[V13];
FOR i = 0 TO (V13 - 1);
LET A39[i] = 1;
NEXT i;
CALL L19;
IRSTOP IRPAUSE;
DRSTOP IDLE;
PADDING 0, 0, 0, 0;
STATE RESET;
STATE IDLE;
LET A18[0..9] = 071;
LET V4 = 1;
CALL L143;
WAIT 10000 USEC;
L13:
RETURN;
L15:
IF (V42 == 0) THEN
PRINT "DONE";
IF (V42 == 1) THEN
PRINT CHR$(7), CHR$(7), CHR$(7),
"Invalid option combination
specified";
IF (V42 == 2) THEN
PRINT CHR$(7), CHR$(7), CHR$(7),
"Unrecognized device";
IF (V42 == 3) THEN
PRINT CHR$(7), CHR$(7), CHR$(7),
"Device revision is not supported";
IF (V42 == 4) THEN
PRINT CHR$(7), CHR$(7), CHR$(7),
"Device programming failure";
IF (V42 == 5) THEN
PRINT CHR$(7), CHR$(7), CHR$(7),
"Device is not blank";
IF (V42 == 6) THEN
PRINT CHR$(7), CHR$(7), CHR$(7),
"Device verify failure";
RETURN;
L37:
LET V27 = V27 + V54;
NEXT j;
PRINT "Device #", V0 - i ," Silicon ID is ",
CHR$(A16[0]), CHR$(A16[1]),
CHR$(A16[2]), CHR$(A16[3]),
CHR$(A16[4]), CHR$(A16[5]),
CHR$(A16[6]), CHR$(A16[7]),
"(", CHR$(((V36 >> 4) & 1) + 48),
CHR$((V36 & 15) + 48), ")"
;
IF (!V40) THEN LET V42 = 3;
L38:
NEXT i;
RETURN;
L54:
LET A18[0..9] = A19[0..9];
CALL L143;
WAIT 15 USEC;
LET A18[0..9] = A20[0..9];
CALL L143;
IF (V34 == 0) THEN GOTO L55;
DRSCAN V24, A29[0..(V24 - 1)],
CAPTURE A31[V30..V31];
GOTO L57;
L60:
LET V30 = 0;
LET V31 = V24 - 1;
L61:
LET V28 = 4;
IF (V10 > 88) THEN LET V28 = 0;
FOR i = 0 TO V28;
PUSH i;
LET A20[0..9] = 022;
CALL L54;
IF (V29 != 0) THEN GOTO L62;
LET A19[0..9] = 032;
CALL L54;
LET A19[0..9] = 036;
L62:
LET V30 = V30 + V24;
LET V31 = V31 + V24;
IF (!V45) THEN GOTO L63;
LET b = 0;
CALL L169;
GOTO L64;
L63:
IF (V10 != V51) THEN GOTO L64;
LET V44 = 1;
LET b = 0;
CALL L163;
L64:
LET A20[0..9] = 026;
CALL L54;
IF (V29 != 0) THEN GOTO L65;
LET A19[0..9] = 032;
CALL L54;
LET A19[0..9] = 036;
L65:
LET V30 = V30 + V24;
LET V31 = V31 + V24;
IF (!V45) THEN GOTO L66;
LET b = 1;
CALL L169;
GOTO L67;
L66:
IF (V10 != V51) THEN GOTO L67;
LET b = 1;
CALL L163;
LET V44 = 0;
CALL L167;
L67:
IF (V28 == 0) THEN GOTO L68;
LET A18[0..9] = 016;
CALL L143;
STATE DRSHIFT;
STATE IDLE;
LET A18[0..9] = 01A;
CALL L143;
STATE DRSHIFT;
STATE IDLE;
L68:
POP i;
NEXT i;
LET V15 = V30;
IF (!V39) THEN GOTO L69;
IF (V72) THEN LET V42 = 5;
IF (!V72) THEN LET V42 = 6;
L69:
RETURN;
L94:
IF ((V10 != 0) && (V10 != 107) && (V11
!= V52)) THEN
GOTO L98;
LET V30 = 0;
FOR j = 0 TO V1;
LET V31 = V30;
IF ((A17[j] & 8) != 0) THEN GOTO L96;
LET V2 = A12[j];
IF (V11 >= (A0[V2] - 108)) THEN GOTO
L95;
IF (((A17[j] & 1) == 0) ||
((A17[j] & 4) == 0)) THEN GOTO L96;
LET V31 = V31 + (A0[V2] - 1);
LET A28[V31 - V11] = 1;
IF (V11 < 108) THEN LET A28[(107 +
V30) - V11] = 1;
GOTO L96;
L95:
IF ((A17[j] & 8) == 0) THEN LET V49 =
1;
LET A17[j] = A17[j] | 8;
L96:
LET V30 = V31 + 1;
NEXT j;
LET A18[0..9] = 01E;
CALL L143;
DRSCAN V30, A28[0..(V30 - 1)];
LET V30 = 0;
FOR j = 0 TO V1;
LET V31 = V30;
LET V2 = A12[j];
IF (((A17[j] & 1) == 0) ||
((A17[j] & 4) == 0) ||
((A17[j] & 8) != 0)) THEN GOTO L97;
LET V31 = V31 + (A0[V2] - 1);
LET A28[V31 - V11] = 0;
IF (V11 < 108) THEN LET A28[(107 +
V30) - V11] = 0;
L97:
LET V30 = V31 + 1;
NEXT j;
IF (V49 && (V11 == V52)) THEN
CALL L171;
GOTO L99;
L98:
LET A18[0..9] = 01E;
CALL L143;
STATE DRSHIFT;
STATE IDLE;
L99:
LET V10 = V11;
RETURN;
L118:
LET V30 = V14;
IF (V49) THEN CALL L149;
LET V31 = V30 + (V23 - 1);
IF (V73) THEN GOTO L119;
LET A18[0..9] = 016;
CALL L143;
DRSCAN V23, A21[V30..V31];
LET V30 = V30 + V23;
LET V31 = V31 + V23;
LET A18[0..9] = 01A;
CALL L143;
DRSCAN V23, A21[V30..V31];
LET V30 = V30 + V23;
LET V31 = V31 + V23;
GOTO L120;
L119:
LET A18[0..9] = 016;
CALL L143;
DRSCAN V23, A29[0..(V23 - 1)];
LET A18[0..9] = 01A;
CALL L143;
DRSCAN V23, A29[0..(V23 - 1)];
L165:
IF (V43) THEN LET A29[V27 + j] = b;
IF (V44) THEN LET A30[V27 + j] = b;
L166:
LET j = k + 1;
NEXT i;
RETURN;
L167:
LET j = 317;
IF (V51 == -1) THEN LET V51 = 0;
FOR i = 0 TO V1;
IF (((A17[i] & 1) == 0) ||
((A17[i] & 4) == 0)) THEN GOTO L168;
LET V2 = A12[i];
IF (V51 >= A9[3 * (V2 - 1)]) THEN
GOTO L168;
IF (j > (A9[3 * (V2 - 1)] - V51)) THEN
LET j = A9[3 * (V2 - 1)] - V51;
L168:
NEXT i;
IF (j != 317) THEN LET V51 = V51 + j;
IF (j == 317) THEN LET V51 = -1;
RETURN;
L169:
LET j = 0;
FOR i = 0 TO V1;
LET k = j;
IF (((A17[i] & 1) == 0) ||
((A17[i] & 4) == 0)) THEN GOTO L170;
LET V2 = A12[i];
LET k = k + (A5[V2] - 1);
LET V27 = (16 * (A6[V2] -
A10[(2 * (V2 - 1)) + 1])) -
A10[2 * (V2 - 1)] - 1;
LET A30[V27 + j] = b;
L170:
LET j = k + 1;
NEXT i;
RETURN;
L171:
LET j = 317;
FOR i = 0 TO V1;
IF (((A17[i] & 1) == 0) ||
((A17[i] & 4) == 0)) THEN GOTO L172;
LET V2 = A12[i];
IF (V52 >= (A0[V2] - 108)) THEN
GOTO L172;
IF (j > A0[V2]) THEN LET j = A0[V2];
L172:
NEXT i;
IF (j != 317) THEN LET V52 = j - 108;
IF (j == 317) THEN LET V52 = -1;
RETURN;
' END OF FILE
CRC B71B;
```

danych do jednego formatu, niezależnego od stosowanych narzędzi projektowych i sprzętu, przejrzystego w zapisie i łatwego w stosowaniu w samodzielnie tworzonych konstrukcjach.

O *JAMie* niemal wszystko

Ponieważ Altera jest prekursorem nowej idei, na jej barkach spoczął obowiązek opracowania ogólnodostępnej specyfikacji języka *JAM* oraz podstawowego oprogramowania źródłowego.

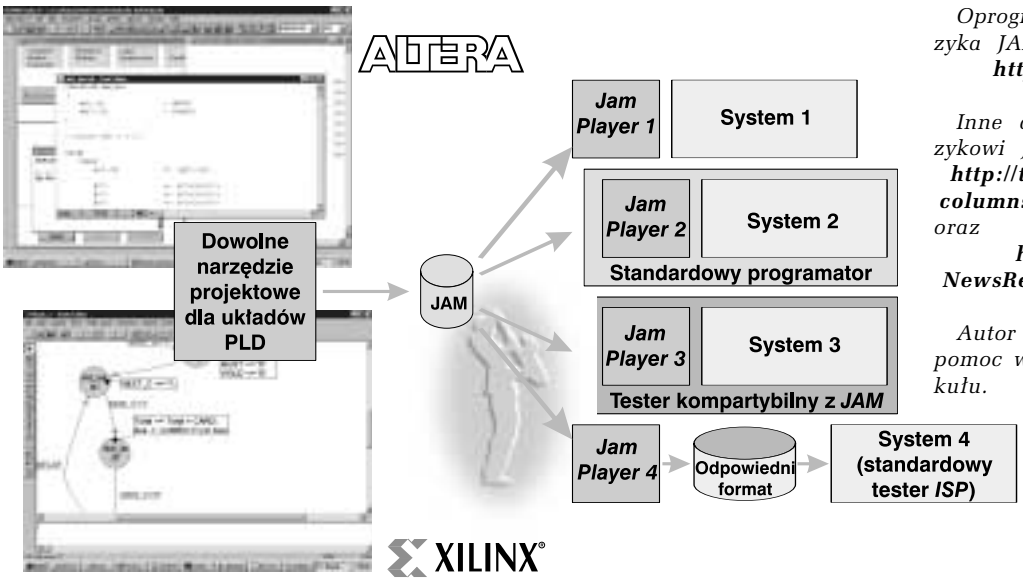
Internetowa strona *JAMu* znajduje się pod adresem *www.altera.com/jam/index.html*. Znajduje się tam opis języka (w postaci plików PDF), jego kompletna dokumentacja oraz oprogramowanie (z bogatą dokumentacją, w której opisano źródłową postać programu) opracowane przez firmę Altera.

W skład zestawu wchodzą dwa programy:

1. *jam.exe*, który spełnia rolę *JAM Playera* współpracującego z programatorem układów ISP Byte Blaster firmy Altera (jego polski odpowiednik kosztuje ok. 200zł). Program *jam.exe* jest kompletnym interpreterem *JAM*, można więc wykorzystać go do pisania programów bez konieczności stosowania programatora.
2. *jamdata.exe*, który jest konwerterem jednego z czterech formatów binarnych (BIN, HEX, RLC, ACA) na postać binarnej tablicy zgodnej ze specyfikacją *JAM*. Dzięki temu programowi możliwe jest umieszczanie plików binarnych jako fragmentu programu *JAM*.

Udostępnione zostały także źródła (napisane w języku C) obydwu programów, przy czym źródło *JAM Playera* jest podzielone na szereg modułów funkcjonalnych, dzięki czemu jest możliwe łatwe przeniesienie ich do dowolnego kompilatora C (także dla mikrokontrolerów). Tak więc konstruktorzy i programiści otrzymują, dzięki uprzejmości Altery, komplet materiałów niezbędnych do popularyzacji nowego standardu.

Idea *JAMu* nie jest nowa. Na początku lat '90 firmy Cypress



Rys. 4.

oraz ACT próbowały bezskutecznie wypromować podobny standard, noszący nazwę *EasyPLD*. Dzięki przejrzystej polityce promocyjnej i dużej rynkowej sile przebicia Altery, *JAM* ma duże szanse stać się prawdziwym standardem, uznanym także przez firmy konkurencyjne.

Piotr Zbysiński, AVT

Więcej informacji na temat standardu JTAG można znaleźć pod adresem:
http://www.jtag.com.
Główna strona poświęcona językowi JAM znajduje się pod adresem:
http://www.altera.com/jam/index.html.

Oprogramowanie i dokumentację języka JAM znajduje się pod adresem:
http://www.altera.com/jam/jam_download.html.
Inne ciekawe strony poświęcone językowi JAM:
http://techweb.cmp.com/edtn/news/columns/Rohleder/rohleder_7_16.htm
oraz
http://www.eedesign.com/NewsReleases/Archives/071097.html.

Autor dziękuje firmie JAWI ASIC za pomoc w zdobyciu materiałów do artykułu.

Oficjalne logo standardu *JAM*.