ICD Przewodnik

Główne funkcje

14.09.2009

Wprowadzenie

Opisane w niniejszym dokumencie funkcje posiadają dwa główne zastosowania:

- pobieranie informacji odnośnie statusu urządzenia docelowego , narzędzi TRACE32 i oprogramowania TRACE32
- konwertowanie specyficznych formatów lub danych w inne reprezentacje

Funkcji zazwyczaj używa się wewnątrz skryptów napisanych w języku PRACTICE, dzięki czemu możliwe jest zgrupowanie poszczególnych wywołań w logiczne zbiory. Alternatywą dla tego rozwiązania jest bezpośrednie wprowadzenie wywołania funkcji w linii poleceń. W tym wypadku niezbędne jest wykorzystanie dodatkowych poleceń wyjściowych, takich jak *PRINT* czy *Data.Print* zależnych od typu zwracanej wartości funkcji.

UWAGA: Adres wymagany jako parametr funkcji może być tylko i wyłącznie typem przeznaczonym do przechowywania adresów komórek. Czysta wartość typu integer nie jest poprawnym adresem pamięci.

Np.:

```
PRINT Data.Long(P:0x1234abcd) // ok hex
PRINT Data.Long(P:1024.)      // ok decimal address due to postfix "."'
PRINT Data.Long(0x1234ffff)   // fails
```

Funkcje

A.CONFIG()	Zwraca TRUE jeśli podłączony jest analizator sprzętu.
A.CONFIG.ECC8() **A.CONFIG.FEC()** **A.CONFIG.HA120()** **A.CONFIG.POWERTRACE()** **A.CONFIG.POWERTRACE2()** **A.CONFIG.RISCTRACE()** **A.CONFIG.SA120()** **A.CONFIG.STU()** **A.CONFIG.TSU()**	**Zwraca TRUE jeśli odpowiedni analizator sprzętu jest podłączony.** Dla ECU32, analizatora HA120, analizatora SA120, TRACE32-ICE: analizatora ECC8, jednostki wyzwalającej HAC, TRACE32-PowerTrace, TRACE32-PowerTrace-II, TRACE32-RiscTrace (ICR), STU (jednostka wyzwalająca) dla SA120, TSU (analizator wydajności) dla SA120, TRACE32-FIRE: analizator FEC.
A.COUNTER.EVENT(<*nazwa_licznika*>**)**	Odczytuje licznik zdarzenia analizatora (jako zmienną *integer*)
A.COUNTER.EXTERN(<*nazwa_licznika*>**)**	Odczytuje zewnętrzny licznik analizatora (jako zmienną *integer*).
A.COUNTER.TIME(<*nazwa_licznika*>**)**	Odczytuje licznik czasu analizatora (jako zmienną *integer*).
A.DSEL()	Zarezerwowana do użytku wewnętrznego. Zwraca łańcuch zawierający wszystkie dozwolone dane wybranego rozszerzenia aktualnie używanego analizatora lub typu CPU.
A.FLOW.ERRORS()	Liczba błędów podczas *flowtrace*.
A.FLOW.FIFOFULL()	Liczba przepełnień kolejki FIFO podczas *flowtrace*.
A.FTRECORDS()	Liczba rekordów w analizatorze *flowtrace*.
A.MODE()	Zwraca aktualny tryb analizatora jako wartość *integer*. **Zwracana wartość Typ wyrażenia** **0 tryb fifo** **1 tryb stosu** **2 tryb leash**
A.MODE.FLOW()	Informuje za pomocą zmiennej *boolean* czy analizator jest w trybie *flowtrace*.

A.PC()	Zwraca tą samą wartość co ***Analyzer.PC***.
A.RECORD.ADDRESS(<*recordno*>**)**	Zwraca adres (offset i klasę dostępu) spod zadanego rekordu analizatora.
A.RECORD.DATA(<*recordno*>**)**	Zwraca daną spod zadanego rekordu analizatora.
A.RECORD.OFFSET(<*recordno*>**)**	Zwraca liczbę porządkową offsetu adresu z zadanego rekordu analizatora.
A.RECORDS()	Liczba rekordów w analizatorze.
A.REF()	Liczba zaznaczonych rekordów odniesienia w analizatorze.
A.SIZE()	Zwraca rozmiar jeśli analizator buforuje.
A.STATE()	Stan analizatora jako liczba *integer*: 0 = wyłączony 1 = uzbrojony 2 = przerwany 3 = wyzwolony 4 = nieaktywny
A.THRESHOLD()	Zwraca napięcie progowe analizatora.
A.TRIGGER.A()	Zwraca bieżącą wartość sondy wejściowej podłączonej do złącza TRIGGER_A modułu analizatora (tylko ICE) lub TRIGGER dla ECC8.
A.TRIGGER.B()	Zwraca bieżącą wartość sondy wejściowej podłączonej do złącza TRIGGER_B modułu analizatora (tylko ICE).
ADDRESS.ACCESS(<*adres*>**)**	Pobiera klasę dostępu jako porządkową liczbę z zadanego adresu (to samo co ***ADDRESS.SPACE***).
ADDRESS.DATA(<*adres*>**)**	Zwraca TRUE jeśli klasa pamięci podanego adresu odnosi się do danych.
ADDRESS.OFFSET(<*adres*>**)**	Obiekt <*adres*>, który zazwyczaj zwracany jest przez inne funkcje zawiera klasę oraz numeryczną wartość specyficznego adresu w pamięci. Funkcja ***ADDRESS.OFFSET*** zwraca numeryczną wartość spod podanego adresu.

```
PRINT ADDRESS.OFFSET(TRACK.ADDRESS())
PRINT ADDRESS.OFFSET(sieve)
```

ADDRESS.ONCHIP(<*adres*>**)**	Zwraca TRUE jeśli adres odnosi się do przestrzeni adresowej *on-chip*.
ADDRESS.PROGRAM(<*adres*>**)**	Zwraca TRUE jeśli klasa pamięci podanego adresu odnosi się do programu.
ADDRESS.SPACE(<*adres*>**)**	Pobiera klasę dostępu jako liczbę porządkową z podanego adresu.
ANALYZER()	Zwraca TRUE jeśli analizator sprzętu jest podłączony.
BDM()	Zwraca TRUE jeśli debugger sprzętowy BDM/ JTAG jest podłączony.
CACHE.DC.DIRTY(<*set*>, <*way*>**)**	Zwraca TRUE jeśli zadane parametry *set/way* danych podręcznych są poprawne. Użyj *CACHE.DC.DIRTY* dla systemów ze skonsolidowaną pamięcią podręczną *cache* L1.
CACHE.DC.DIRTYMASK(<*set*>, <*way*>**)**	Zwraca wartość zawierającą wszystkie flagi *dirty* podanych parametrów danych *cache*. Użyj *CACHE.DC.DIRTYMASK* dla systemów ze skonsolidowaną pamięcią podręczną *cache* L1.
CACHE.DC.TAG(<*set*>, <*way*>**)**	Zwraca *tag* adresu podanych parametrów danych *cache*. Użyj *CACHE.DC.TAG* dla systemów ze skonsolidowaną pamięcią podręczną *cache* L1.
CACHE.DC.VALID(<*set*>, <*way*>**)**	Zwraca TRUE jeśli zadane parametry *set/way* danych podręcznych są poprawne. Użyj *CACHE.DC.VALID* dla systemów ze skonsolidowaną pamięcią podręczną *cache* L1.
CACHE.DC.VALIDMASK(<*set*>, <*way*>**)**	Zwraca wartość zawierającą wszystkie flagi *valid* podanych parametrów danych *cache*. Użyj *CACHE.DC.VALIDMASK* dla systemów ze skonsolidowaną pamięcią podręczną *cache* L1.
CACHE.IC.DIRTY(<*set*>, <*way*>**)**	Zwraca TRUE jeśli podane parametry *set/way* skonsolidowanej pamięci podręcznej L1 są DIRTY. (dla czystych instrukcji pamięci *cache* wynikiem jest zawsze FALSE).
CACHE.IC.DIRTYMASK(<*set*>, <*way*>**)**	Zwraca wartość zawierającą wszystkie flagi *dirty* podanych parametrów skonsolidowanej pamięci podręcznej L1, i 0 dla czystych instrukcji.

CACHE.IC.TAG(<*set*>, <*way*>**)**	Zwraca *tag* adresu podanych parametrów *set/way* instrukcji (lub skonsolidowanej) pamięci *cache*.
CACHE.IC.VALID(<*set*>, <*way*>**)**	Zwraca TRUE jeśli podane *set/way* instrukcji (lub skonsolidowanej) pamięci *cache* są poprawne.
CACHE.IC.VALIDMASK(<*set*>, <*way*>**)**	Zwraca wartość zawierającą wszsytkie flagi *valid* podanych *set/way* instrukcji (lub skonsolidowanej) pamięci *cache*.
CACHE.L2.DIRTY(<*set*>, <*way*>**)**	Zwraca TRUE jeśli podane *set/way* pamięci podręcznej L2 posiadają ustawioną flagę DIRTY. Jeśli *way* pamięci *cache* podzielone są w sektorach, wynikiem jest TRUE jeśli co najmniej jedna flaga DIRTY jest ustawiona.
CACHE.L2.DIRTYMASK(<*set*>, <*way*>**)**	Zwraca wartość zawierającą wszystkie flagi DIRTY podanych parametrów *set/way* pamięci podręcznej L2.
CACHE.L2.TAG(<*set*>, <*way*>**)**	Zwraca *tag* adresu podanych parametrów *set/way* pamięci podręcznej L2.
CACHE.L2.VALID(<*set*>, <*way*>**)**	Zwraca TRUE jeśli podane *set/way* pamięci podręcznej L2 posiadają ustawioną flagę VALID. Jeśli *way* pamięci *cache* podzielone są w sektorach, wynikiem jest TRUE jeśli co najmniej jedna flaga DIRTY jest ustawiona.
CACHE.L2.VALIDMASK(<*set*>, <*way*>**)**	Zwraca wartość zawierającą wszystkie flagi VALID podanych parametrów *set/way* pamięci podręcznej L2.
COUNT.LEVEL()	Bieżący poziom logiczny wejściowego licznika częstotliwości.
COUNT.VALUE()	Wynik ostatniej akcji licznika częstotliwości (uruchomiony przez ***COUNT.GO***).
CPU()	Zwraca nazwę procesora jako łańcuch (to samo co ***STATE.PROCESSOR***). Jest to nazwa wybrana w *SYStem.CPU*.
CPUBONDOUT()	Zwraca nazwę procesora *bondout* jako łańcuch.
CPUDERIVATE()	Zwraca główną cześć nazwy procesora.
CPUFAMILY()	Zwraca nazwę rodziny procesora jako łańcuch.
CPUHELP()	Zarezerwowane do użytku wewnętrznego.

D.AL.ERRORS()	Zwraca liczbę błędów wykrytych przez funkcję *Data.AllocList*.
DAP.USER0() **DAP.USER1()**	Zwraca status pinu DEP *USER0* lub *USER1*. Poziom niski to FALSE, poziom wysoki TRUE. Dostępne tylko dla ICD-TriCore i ICD-C166.
DATA.BYTE(<*adres*>**)** **lub** **D.B(**<*adres*>**)**	Zwraca zawartość pamięci w postaci bajta. Podany adres musi być sklasyfikowany, np. D:0x200, gdzie każdy symbol implikuje konkretną klasę pamięci. Dostępne klasy pamięci zależne są od docelowego procesora. Odpowiednią listę można znaleźć w rozdziale *„memory classes”* w dokumencie *„ICD Target Guide”*.

```
PRINT Data.Byte(D:0x200)            // nazwa funkcji w formie długiej
PRINT D.B(D:0x200)                  // nazwa funkcji w formie krótkiej
PRINT Data.B(D:0x200)               // niepoprawna nazwa funkcji
PRINT D.Byte(D:0x200)
PRINT Data.Byte(a+17.)    // wyświetli zawartość pamięci spod adresu
                          // wskazanego przez symbol "a" plus offset 17
```

DATA.FLOAT(<*format*>, <*adres*>**)** **lub** **D.F(**<*format*>, <*adres*>**)**	Odczytuje liczbę zmiennoprzecinkową z pamięci. Format musi być podany jako łańcuch.

```
PRINT Data.Float("IEEE",D:0x200)
```

DATA.LONG(<*adres*>**)** **lub** **D.L(**<*adres*>**)**	Zwraca zawartość pamięci w postaci danej typu *long*. Adres musi być sklasyfikowany, np. D:0x200, gdzie każdy symbol odpowiada konkretnej klasie pamięci. Proszę nie mylić ***funkcji*** D.L („data.long”) z ***poleceniem*** D.L („data.list”).

```
PRINT Data.Long(D:0x200)
PRINT Data.Long(0x200)          // błąd – parametr jest liczbą całkowitą
                                // i nie zawiera informacji
                                // o sposobie dostępu np. „D:” lub „P:”
```

DATA.LONG.BIGENDIAN(<*adres*>**)** **lub** **D.L.BE(**<*adres*>**)**	Układ bajtów interpretowany jako BigEndian.
DATA.LONG.LITTLEENDIAN(<*adres*>**)** **lub** **D.L.LE(**<*adres*>**)**	Układ bajtów interpretowany jako LittleEndian.
DATA.QUAD(<*adres*>**)** **lub** **D.Q(**<*adres*>**)**	Zawartość pamięci w postaci 8 bajtów.

DATA.QUAD.BIGENDIAN(<*adres*>**)** **lub** **D.Q.BE(**<*adres*>**)**	Układ bajtów interpretowany jako BigEndian.
DATA.QUAD.LITTLEENDIAN(<*adres*>**)** **lub** **D.Q.LE(**<*adres*>**)**	Układ bajtów interpretowany jako LittleEndian.
DATA.SLONG(<*adres*>**)** **lub** **D.SL(**<*adres*>**)**	Odczytuje z pamięci wartość *long* ze znakiem – znak rozszerza lokalnie wartość do 64 bitów.
DATA.STRING(<*adres*>**)** **lub** **D.S(**<*adres*>**)**	Odczytuje z pamięci łańcuch zakończony znakiem zero. Wynikiem jest łańcuch. Adres musi być sklasyfikowany, np. D:0x200, gdzie każdy symbol odpowiada konkretnej klasie pamięci. Proszę nie mylić ***funkcji*** D.S („data.string”) z ***poleceniem*** D.S („data.set”).

```
PRINT Data.String(cstr1)
```

DATA.SUM() **lub** **D.SUM()**	Pobiera sumę kontrolną z ostatnio uruchomionego polecenia *Data.SUM*.

```
Data.Set P:0x00--0xff %Byte 1
Data.SUM P:0x00--0xff
PRINT DATA.SUM()                                ; wyświetli wartość
0x100
```

DATA.TBYTE(<*adres*>**)** **lub** **D.TB(**<*adres*>**)**	Pobiera zawartość pamięci w postaci potrójnego bajta. Adres musi być sklasyfikowany, np. D:0x200, gdzie każdy symbol odpowiada konkretnej klasie pamięci.

```
PRINT Data.TByte(D:0x200)
```

DATA.WORD.BIGENDIAN(<*adres*>**)** **lub** **D.W.BE(**<*adres*>**)**	Układ bajtów interpretowany jako BigEndian.
DATA.WORD.LITTLEENDIAN(<*adres*>**)** **lub** **D.W.LE(**<*adres*>**)**	Układ bajtów interpretowany jako LittleEndian.
DEBUG()	Zwraca TRUE jeśli debugger jest uruchomiony na interfejsie debugującym (JTAG/BDM) urządzenia docelowego.
DEBUGMODE()	Zwraca w postaci łańcucha bieżący tryb debugowania. Zwracane wartości to: “ASM”, “MIX”, “HLL”

```
IF DEBUGMODE!="HLL"
Mode.Hll
```

DONGLEID()	Zwraca numer seryjny debuggera jako liczbę całkowitą.
DPP(<*rejestr*>**)**	Zwraca zawartość podanego rejestru DPPn (tylko C166/ST10)
EPOC.DATAADDRESS()	Tylko dla debuggera EPOC! Zwraca adres początkowy obszaru danych należących do aktualnego, aktywnego zadania.
EPOC.ENTRYPOINT()	Tylko dla debuggera EPOC! Zwraca adres dostępu aktualnie aktywnego zadania.
EPOC.TEXTADDRESS()	Tylko dla debuggera EPOC! Zwraca adres początkowy obszaru kodu należącego do aktualnie aktywnego zadania.
ERROR.ADDRESS()	Zwraca adres błędu z ostatniej wywołanej komendy TRACE32.
ESI()	Zwraca TRUE jeśli debugger uruchomiony jest za pomocą ESI (Symulator EEPROM).
ETK()	Zwraca TRUE jeśli *SYStem.Option ETK* jest włączony. Komenda jest użyteczna w skryptach PRACTICE w celu sprawdzenia wyniki polecenia *SYStem.Option ETKS AUTO*. Dostępna tylko dla *ICD-TriCore* i *ICD-PCP*.
ETM()	Zwraca TRUE jest ETM jest dostępne.
ETM.ADDRCOMP()	Zwraca numer indeksu ostatniego przypisanego komparatora adresu ETM (tylko do użytku wewnętrznego).
ETM.ADDRCOMPTOTAL()	
ETM.COUNTERS()	Zwraca liczbę dostępnych liczników ETM.
ETM.DATACOMP()	Zwraca liczbę dostępnych komparatorów danych ETM.
ETM.EXTIN()	Zwraca liczbę zewnętrznych wejść ETM.
ETM.EXTOUT()	Zwraca liczbę zewnętrznych wyjść ETM.
ETM.FIFOFULL()	Zwraca 1 jeśli *fifofull logic* jest dostępny.

ETM.MAP()	Zwraca liczbę rejestratorów mapy pamięci ETM.
ETM.PROTOCOL()	Zwraca numer wersji protokołu ETM.
ETM.SEQUENCER()	Zwraca liczbę dostępnych urządzeń sekwencyjnych ETM.
EVAL()	Zwraca wartość parametru wyrażenia z ostatniej komendy *Eval*. Wartość końcowa będzie zwrócona tylko dla wyrażeń typu *boolean*, *binary, hex, integer,* stałych ASCII i adresów. We wszystkich innych przypadkach zwracaną wartością jest 0.
EVAL.TYPE()	Zwraca typ parametru wyrażenia z ostatniej komendy *Eval*. **Zwracana wartość Typ wyrażenia** 0x0001 boolean 0x0002 binary 0x0004 hex 0x0008 integer 0x0010 float 0x0020 ASCII constant 0x0040 string 0x0080 numeric range 0x0100 address 0x0200 address range 0x0400 time 0x0800 time range 0x4000 bitmask 0x8000 empty

```
entry &delayvalue
eval  &delayvalue                    ; wylicz podaną wartość
if eval.type()!=0x400                ; wprowadzono wartość czasu?
gosub err_no_timevalue
```

EXTENDED()	Zwraca TRUE jeśli rejestr CBAR jest > 0 (tylko Z80).
FASBASE.ADDRESS()	Zwraca adres bazowy FASRAM'u (tylko FIREC32X).
FDX.INSTRING(<*adres*>**)**	Zwraca zawartość podanego adresu pamięci FDX w postaci łańcucha.
FIRE()	Zwraca TRUE jeśli debugger jest uruchomiony z wykorzystaniem TRACE32-FIRE.

FLAG(<*zakres_adresu*>**)**	Zwraca TRUE jeśli systemowa flaga sprzętu jest dostępna.
FLAG.READ(<*zakres_adresu*>**)**	Zwraca liczbę bajtów z ustawieniem bitu dostępu READ w pamięci flag.
FLAG.WRITE(<*zakres_adresu*>**)**	Zwraca liczbę bajtów z ustawieniem bitu dostępu WRITE w pamięci flag.
FLASH.CFI.SIZE(<*adres*>,<*szerokość*>**)**	Zwraca rozmiar w postaci liczby hex pojedynczego lub równoległego CFI urządzenia Flash. Zwraca 0 jeśli TRACE32 nie może odczytać informacji CFI.

```
PRINT FLASH.CFI.SIZE(P:0x0,Word)
PRINT FLASH.CFI.SIZE(D:0x40000000,Long)
```

FPU(<*nazwa*>**)**	Pobiera zawartość rejestru FPU (jako liczbę zmiennoprzecinkową).
FPUCR(<*nazwa*>**)**	Pobiera wartość rejestru kontrolnego FPU (jako liczbę całkowitą).
I.ANALOG()	Zwraca wartość różną od 0 jeśli analogowa sonda jest podłączona do PowerIntegrator. Dla złączy A..F, J..O wartościami są 0x0001..0x0020, 0x0100..0x2000.
ICD()	Zwraca TRUE jeśli debugger jest uruchomiony z wykorzystaniem TRACE32-ICD (JTAG/BDM).
ICE()	Zwraca TRUE jeśli debugger jest uruchomiony z wykorzystaniem TRACE32-ICE.
ICEFAMILY()	Zwraca nazwę rodziny modułu bazowego ICE jako łańcuch.
ID.CABLE()	Zwraca ID przewodu debugującego (nie numer seryjny).
ID.PREPRO()	Zwraca ID preprocesora (nie numer seryjny).
INTEGRATOR() – *przestarzałe, użyj POWERINTEGRATOR()*	Zwraca TRUE jeśli PowerIntegrator jest podłączony.
IOBASE()	Pobiera adres bazowy wewnętrznych portów I/O jako uporządkowaną liczbę bez klasy dostępu.
IOBASE.ADDRESS()	Pobiera adres bazowy wewnętrznych portów I/O jako wartość adresu wraz z klasą dostępu.

IOBASE2()	Pobiera drugi adres bazowy wewnętrznych portów I/O jako uporządkowaną liczbę bez klasy dostępu. Zobacz *System.Option MOBAR*.
IPROBE	Zwraca TRUE jeśli debugger jest uruchomiony za pomocą TRACE32-IPROBE.
IPROBE.ANALOG	Zwraca TRUE jeśli sonda analogowa podłączona jest do TRACE32-IPROBE.
JTAG.PIN(<*nazwa_sygnału*>**)**	Odczytuje poziom podanego sygnału JTAG. Zobacz *JTAG.PIN*.
JTAG.SHIFT()	Odczytuje wyjście TDO przesunięcia JTAG. Najpierw LSB. Ograniczony do 64 bitów. Zobacz *JTAG.SHIFTREG* i *JTAG.SHIFTTDI*.
L.RECORDS()	Liczba rekordów wykorzystywana przez *Logger'a*.
L.REF()	Liczba wybranych rekordów odniesienia w *Loggerze*.
L.STATE()	Stan *Logger'a* jako liczba całkowita. 0 = off 1 = armed 2 = breaked 3 = triggered 4 = disabled
MAP.CONFIG.FDPRAM()	Zwraca TRUE jeśli pełny dwuportowy moduł RAM podłączony jest wewnątrz TRACE32-ICE.
MAP.RAMSIZE()	Zwraca całkowity rozmiar pamięci emulowania dla TRACE32-ICE oraz TRACE32-FIRE.
MAP.ROMSIZE()	Zwraca całkowity rozmiar zdefiniowanej pamięci ROM.
MMU(<*nazwa_rejestru_mmu*>**)**	Pobiera wartość rejestru MMU.
MMU.LOGICAL(<*adres*>**)**	Konwertuje fizyczny adres na logiczny adres.
MMU.PHYSICAL(<*adres*>**)**	Konwertuje logiczny adres na fizyczny adres.
MONITOR()	Zwraca TRUE jeśli debugger uruchomiony jest jako monitor.
O.RECORDS()	Liczba rekordów wykorzystanych przez śledzenie *Onchip*.
O.REF()	Liczba wybranych rekordów odniesienia w śledzeniu *Onchip*.

O.STATE()	Stan śledzenia *Onchip* jako liczba całkowita. 0 = off 1 = armed 2 = breaked 3 = triggered 4 = disabled
O.TRACK.RECORD()	Aktualny rekord w śledzeniu *Onchip*, np. po zakończonej sukcesem operacji szukania.
P.GET(<*nazwa_kanału*>**)**	Zwraca aktualną wartość portu podanego kanału.
P.RECORDS()	Liczba rekordów w analizatorze portów.
P.REF( )	Liczba wybranych rekordów odniesienia w analizatorze portów.
P.STATE( )	Stan analizatora jako liczba całkowita (Slave/Off/Armed/Triggered, Breaked).
PER.ARG(<*integer*>**)**	Zwraca (opcjonalnie) argument polecenia *Per. View*. Parametr aktualnie nie jest wykorzystywany. Funkcja użyteczna tylko wewnątrz plików definiujących peryferia.
PER.EVAL(<*integer*>**)**	Zwraca wartość wartość dynamicznego wyrażenia znajdującego się wewnątrz pliku definiującego peryferia. Dynamiczne wyrażenia są wykorzystywane przez polecenia ***IF*** oraz ***BASE***. Parametr definiuje, które wyrażenie jest zwracane (0 = pierwsze).
PERF.WATCHTIME(<*index*>**)**	Pobiera pole ***WatchTime*** z jednej linii analizatora osiągów.
PORTANALYZER()	Zwraca TRUE jeśli analizator portów jest podłączony do TRACE32-ICE lub TRACE32-FIRE.
POWERDEBUG()	Zwraca TRUE jeśli debugger uruchomiony jest na PowerDebug lub PowerTrace.
POWERINTEGRATOR()	Zwraca TRUE jeśli PowerIntegrator jest podłączony.
POWERNEXUS()	Zwraca TRUE jeśli debugger jest uruchomiony na PowerNexus.
POWERPROBE()	Zwraca TRUE jeśli PowerProbe jest dostępny.
POWERTRACE()	Zwraca TRUE jeśli debugger uruchomiony jest na PowerTrace lub PowerTrace-II oraz preprocesor lub adapter Nexus jest podłączony.

POWERTRACE2()	Zwraca TRUE jeśli debugger uruchomiony jest na PowerTrace-II oraz preprocesor lub adapter Nexus jest podłączony.
PROBE() - *przestarzałe, użyj funkcji POWERPROBE()*	Zwraca TRUE jeśli PowerProbe jest dostępny.
Register(<*nazwa*>**)**	Pobiera zawartość podanego rejestru.
RUN()	Zwraca stan flagi run (CPU uruchomione na układzie docelowym) jako *boolean*.
RUNTIME.ACTUAL() **RUNTIME.LAST()** **RUNTIME.REFA()** **RUNTIME.REFB()**	Zwraca wartości wyświetlone w oknie RunTime (jako czas od zera).
RUNTIME.INACCURACY()	Zwraca błąd pomiaru licznika RunTime w sekundach.
SIMULATOR()	Zwraca TRUE jeśli debugger jest uruchomiony na symulatorze instrukcji.
SNOOP.RECORDS()	Liczba rekordów używanych przez śledzenie SNOOPer.
SNOOP.REF()	Liczba wybranych rekordów odniesienia w śledzeniu SNOOPer.
SNOOP.STATE()	Stan śledzenia SNOOPer jako liczba całkowita (Off/ Armed/Triggered/Breaked /Disabled). 0 = off 1 = armed 2 = breaked 3 = triggered 4 = disabled
STATE.HALT()	Zwraca stan wyświetlacza „halt” (np. Brak cykli CPU) jako wartość *boolean*.
STATE.POWER()	Zwraca stan linii zasilającej układu decelowego jak wartość *boolean*.
STATE.PROCESSOR()	Zwraca nazwę procesora jako łańcuch.
STATE.RESET()	Zwraca stan linii resetu układu docelowego jako *boolean*.
STATE.RUN()	Zwraca stan flagi run (CPU uruchomione na układzie docelowym) jako *boolean*.

STG()	Zwraca TRUE jeśli generator Stimuli jest dostępny.
SYSTEM.ACCESS.DENIED()	Zwraca informację jako *boolean*, czy dostęp do pamięci jest możliwy podczas emulacji w czasie rzeczywistym.
SYSTEM.AMBA()	Zwraca TRUE jeśli *SYStem.Option AMBA* jest aktywne.
SYSTEM.BIGENDIAN()	Zwraca TRUE jeśli rdzeń układu docelowego uruchomiony jest w trybie *big endian*.
SYSTEM.HOOK()	Zwraca adres funkcji przechwytującej zdefiniowanej za pomocą *SYStem.Option HOOK*.
SYSTEM.LITTLEENDIAN()	Zwraca TRUE jeśli rdzeń układu docelowego pracuje w trybie *little endian*.
SYSTEM.MODE()	Zwraca aktualny tryb sondy (jako liczbę całkowitą). Wartości większe od 5 reprezentują stan „Up”, mniejsze natomiast oznaczają stan „Down”. 0..5 = stany *down* 6 = AloneInt (tylko ICE/FIRE) 7 = AloneExt (tylko ICE/FIRE) 8 = EmulInt (tylko ICE/FIRE) 9 = EmulExt (tylko ICE/FIRE) 11 = Up (tylko ICD)
SYSTEM.NOTRAP()	Zwraca 1 jeśli *SYStem.Option NOTRAP* jest aktywny.
SYSTEM.TRACEEXT()	Podaje informacje jako *boolean* czy zewnętrzny *trace* jest aktywny.
SYSTEM.TRACEINT()	Podaje informacje jako *boolean* czy wewnętrzny *trace* jest aktywny.
SYSTEM.UP()	Zwraca informacje jako *boolean* czy aktualnym trybem sondy jest *up*.
T.METHOD.ANALYZER()	Zwraca TRUE jeśli metoda śledzenia używa sprzętowego analizatora.
T.METHOD.ART()	Zwraca TRUE jeśli metoda śledzenia to ART.
T.METHOD.FDX( )	Zwraca TRUE jeśli metoda śledzenia to FDX.
T.METHOD.HYPER()	Zwraca TRUE jeśli metoda śledzenia to HYPER.
T.METHOD.INTEGRATOR()	Zwraca TRUE jeśli metoda śledzenia wykorzystuje sprzętowy analizator Integrator.

T.METHOD.LOGGER()	Zwraca TRUE jeśli metoda śledzenia to LOGGER.
T.METHOD.ONCHIP()	Zwraca TRUE jeśli metoda śledzenia to ONCHIP.
T.METHOD.PROBE( )	Zwraca TRUE jeśli metoda śledzenia wykorzystuje sprzętowy analizator PowerProbe.
T.METHOD.SNOOPER()	Zwraca TRUE jeśli metoda śledzenia to SNOOPER.
TA32( )	Zwraca TRUE jeśli TA32 jest dostępne.
TASK.BACK()	Numer zadnia w tle.
TASK.F1( ) **TASK.F2( )** **TASK.F3( )** **TASK.F4( )**	Wyłącznie do użytku wewnętrznego.
TASK.FORE()	Numer zadania pierwszoplanowego.
TERM.LINE(<*adres*>**,** <*linia*>**)**	Zwraca zawartość specjalnej linii aktywnego okna terminala. ***<adres>*** jest portem komunikacyjnym wirtualnego terminala.
TPUBASE.ADDRESS()	Adres w którym TPU jest zalokowany.
TRACK.ADDRESS() **TRACK.ADDRESS.DATA()** **TRACK.ADDRESS.PROG()**	Aktualny adres śledzenia np. komendach przeszukiwania lub testowania pamięci.

```
Data.Find flags++0xFF 0x01
IF FOUND()
(
   Data.Set TRACK.ADDRESS() 0xAA
   Data.Print TRACK.ADDRESS()
)
```

TRACK.RECORD()	Aktualnie śledzony rekord analizatora np. po zkaończonej sukcesem operacji szukania.
TRACK.TIME()	Wynik odejmowania wartości czasu aktualnie śledzonego rekord i czasu zero.
TRIGGER.ACCESS()	Pobiera klasę dostępu jako uporządkowaną liczbę ostatnio wyzwolonego zdarzenia.
TRIGGER.ADDRESS()	Adres ostatnio wyzwolonego zdarzenia (nie ECC8).
TRIGGER.BYTES()	Starsze cztery bity zawierają *byte-enables* na 32-bitowej magistrali (nie ECC8).

TRIGGER.COUNT.ALPHA() **TRIGGER.COUNT.BETA()** **TRIGGER.COUNT.CHARLY()**	Zwraca wartości licznika wyzwalacza.
TRIGGER.CYCLE()	Typ cyklu wyzwalacza, bit 0 to odczyt/zapis (nie ECC8).
TRIGGER.DELAY.CYCLE() **TRIGGER.DELAY.TIME()** **TRIGGER.DELAY.TRACE()**	Zwraca wartości liczników wyzwalacza (jako liczba całkowita lub czas).
TRIGGER.OFFSET()	Adres ostatnio wyzwolonego zdarzenia jako liczba całkowita (nie ECC87).
TRIGGER.SOURCE()	Źródło ostatnio wyzwolonego zdarzenia jako kod hex (nie ECC8).
TRIGGER.STATE()	Zwraca stan wyzalacza (jako liczba całkowita). 1 = armed 2 = trigger 3 = break
TRIN.VALUE()	Zwraca bieżącą wartość sondy wejściowej wyzwalacza podłączonej do złącza EXTERNAL (nie ECC8).
TSS()	Zwraca adres bazowy TSS ostatnio załadowanego pliku obiektowego (tylko 80386).
V.ADDRESS(<*wyrażenie*>**)**	Zwraca adres wyrażenia hll.
`Data.Print V.ADDRESS(flags)` `Data.Print V.ADDRESS(flags[3])`	
V.BITPOS(<*wyrażenie*>**)**	Zwraca pozycję początkowego bitu elementu wewnątrz pola bitowego.
`PRINT V.BITPOS(vbfield.d)`	
V.BITSIZE(<*wyrażenie*>**)**	Zwraca rozmiar w najtach elementu pola bitowego.
`PRINT %D V.BITSIZE(vbfield.f)`	
V.END(<*wyrażenie*>**)**	Zwraca ostatni adres zajęty przez wyrażenie hll.
`Data.Print V.END(vbfield)`	
V.FVALUE(<*wyrażenie*>**)**	Zwraca zawartość wyrażenie hll jako liczbę zmiennoprzecinkową.

```
PRINT V.FVALUE(ast.float)
PRINT V.FVALUE(i)
PRINT V.FVALUE(ast->left.x)
```

V.ISBIT(<*wyrażenie*>**)** — Zwraca informację jako *boolean* czy wyrażenie hll jest polem bitowym czy nie.

```
PRINT V.ISBIT(vbfield.f)
PRINT V.ISBIT(vbfield)
```

V.RANGE(<*wyrażenie*>**)** — Zwraca zakres adresowy zajęty przez wyrażenie hll w pamięci.

```
Data.Print V.RANGE(flags)
Data.Find V.RANGE(flags) 0x00
IF FOUND()
(
   Data.Set TRACK.ADDRESS() 0xBB
   Data.Print TRACK.ADDRESS()
)
enddo
```

V.SIZEOF(<*wyrażenie*>**)** — Zwraca rozmiar zajęty przez wyrażenie hll w pamięci.

```
Data.Print flags++V.SIZEOF(flags)
Data.Find flags++V.SIZEOF(flags) 0x00
IF FOUND()
(
   Data.Set TRACK.ADDRESS() 0xDD
   Data.Print TRACK.ADDRESS()
)
Enddo
```

V.STRING(<*wyrażenie*>**)** — Zwraca zawartość wyrażenia hll jako łańcuch.

```
&enumvalue=V.STRING(ptr->member)
PRINT „&enumvalue”
```

V.VALUE(<*wyrażenie*>**)** — Zwraca zawartość wyrażenia hll jako hex.

```
PRINT V.VALUE(ast.field1)
PRINT V.VALUE(i)
PRINT V.VALUE((int)ast->left.count)
```

VCO() — Zwraca ustawioną częstotliwość generatora VCO.

VERSION.BUILD() — Zwraca numer kompilacji oprogramowania.

VERSION.CABLE() — Zwraca numer przewodu debugującego.

VERSION.SERIAL.DEBUG() — Zwraca numer seryjny modułu Debug.

VERSION.SERIAL.PREPRO()	Zwraca numer seryjny Preprocesora.
VERSION.SERIAL.TRACE()	Zwraca numer seryjny modułu Trace.
Y.END(<*symbol*>**)**	Zwraca ostatni adres zajęty przez symbol

```
Data.Print Y.END(vbfield)
```

Y.EXIST(<*symbol*>**)**	Zwraca TRUE jeśli symbol istnieje lub FALSE w przeciwnym wypadku.

```
PRINT Y.EXIST(vbfield)
PRINT Y.EXIST(qwertzuiop)
```

Y.EXIT(<*symbol*>**)**	Zwraca adres wyjścia zadanej funkcji.

```
PRINT Y.EXIT(func40)
```

Y.FUNCTION(<*adres*>**)**	Zwraca jako łańcuch ścieżkę i nazwę funkcji, która zawiera podany adres. Adres musi być zklasyfikowany, np. P:0x200.

```
PRINT Y.FUNCTION(P:0x40001000)
```

Y.IMPORT()	Zwraca nazwę następnego, zaimportowanego i jeszcze nie wykonanego pliku. Funkcja może być użyta do poruszania się po liście zaimportowanych nazw (przeznaczona do pracy z programami *.exe, np. pod WindowsCE).
Y.MATCHES()	Zwraca liczbę wystąpień przetwarzanej komendy takiej jak *Browse* lub *ForEach*.

```
sYmbol.Browse func*0
PRINT Y.MATCHES()
ENDDO
```

Y.NAME(<*adres*>**)**	Zwraca ścieżkę symboli i nazwę podanego adresu. Adres musi być sklasyfikowany, np. D:0x200. Jeśli adres jest wewnątrz ciała funkcji, to wynik jest taki sam jak ten zwrócony przez funkcję *Y.FUNCTION*.

```
PRINT Y.NAME(D:0x40005EB8)
```

Y.PRANGE(<*symbol*>**)**	Zwraca przestrzeń adresową zajętą przez podany symbol (np. funkcję lub zmienną) bez wstępnego pobierania obszaru na który może zachodzić inny symbol. Tak więc pobiera ona zakres np. w celu monitorowania analizy osiągów. Uzyskany zakres jest identyczny lub mniejszy od aktualnego (który jest dostarczony przez *Y.RANGE*).

```
Data.Print Y.PRANGE(sieve)
```

Y.RANGE(<*symbol*>**)**	Zwraca zakres adresowy zajęty przez podany symbol.

```
Data.Print Y.RANGE(flags)
```

Y.SEARCHFILE(<*nazwa_pliku*>**)**	Porusza się po ścieżce poszukiwań w celu znalezienia pliku źródłowego i zwraca absolutną ścieżkę dostępu do pliku, który został znaleziony jako pierwszy. Jeśli nie znaleziono żadnych plików, funkcja zwraca pusty łańcuch.

```
PRINT Y.SEARCHFILE(demo.c)
```

Y.SECADDRESS(<*sekcja*>**)**	Zwraca logiczny adres początku podanej sekcji.
Y.SECEND(<*sekcja*>**)**	Zwraca logiczny adres końca podanej sekcji.
Y.SECPRANGE(<*sekcja*>**)**	Zwraca fizyczną zakres adresowy zajęty przez podaną sekcję.
Y.SECRANGE(<*sekcja*>**)**	Zwraca logiczny zakres adresowy zajęty przez podaną sekcję.
Y.SIZEOF(<*symbol*>**)**	Zwraca rozmiar zajęty przez zamienną w pamięci.

```
Data.Print flags++Y.SIZEOF(flags)
Data.Find flags++Y.SIZEOF(flags) 0x00
IF FOUND()
(
   Data.Set TRACK.ADDRESS() 0xEE
   Data.Print TRACK.ADDRESS()
)
Enddo
```

Y.SOURCEFILE(<*klasa_pamieci*>:<*adres*> | <*symbol*>*)*	Zwraca nazwę pliku źródłowego jako łańcuch dla podanego adresu lub symbolu. Adres musi być sklasyfikowany, np. D:0x200.

```
PRINT Y.SOURCEFILE(main)
```

Y.SOURCELINE(<*klasa_pamieci*>:<*adres*>*)*	Zwraca numer linii kodu HLL spod określonego adresu (jako liczbę całkowitą). Adres musi być sklasyfikowany, np. P:0x200.

```
PRINT Y.SOURCELINE(P:0x40005EB8)
; command extension to call external editor (~~/demo/practice/edit.
cmm)
; adapt path and parameter syntax to your own editor
; (e.g. uedit32.exe filename/line)
ON CMD EDT GOSUB
(
   LOCAL &file &line &cmdline
   ENTRY &file
   IF „&file"==""
   (
      &file=Y.SOURCEFILE(P:Register(pc))
      &line=Y.SOURCELINE(P:Register(pc))
   )
   &cmdline="os uedit32.exe &file/"+STRING.CUT(„&line",-1)
   &cmdline
   RETURN
)
STOP
ENDDO
```

Y.SOURCEPATH(<*katalog*>**)**

Zwraca TRUE jeśli podany catalog jest już zdefiniowany jako catalog poszukiwań (poprzez *sYmbol.SourcePATH* lub opcję */PATH*).

```
&my_path=Y.SOURCEFILE(main)
&my_path=OS.FILE.PATH(&my_path)
PRINT „&my_path"
sYmbol.SourcePATH.Set &my_path
PRINT Y.SOURCEPATH(&my_path)
sYmbol.SourcePATH.List
ENDDO
```

Y.TYPE(<*symbol*>**)**

Zwraca podstawowy typ symboplu jako wartość numeryczną.
0 = symbol nie istnieje
1 = zwykła etykietka bez informacji o typie
2= funkcja HLL
3 = zmienna HLL
Inne wartości mogą być zdefiniowane w przyszłości.

```
PRINT Y.TYPE(qwertzuiop)
PRINT Y.TYPE(_main)
PRINT Y.TYPE(sieve)
PRINT Y.TYPE(flags)
```

Y.VARNAME(<*adres*>**)**

Zwraca nazwę zmiennej lub elementu struktury spod podanego adresu (jako łańcuch).