ICD Przewodnik

O przewodniku	2
Praca z debugger'em	3
Konfiguracja środowiska	3
Uruchomienie TRACE32-ICD	4
Główne okno TRACE32	5
O TRACE32	6
Online Help	7
Ustawienia środowiska Debug	9
Pliki wsadowe	14
Interfejs użytkownika	16
Podgląd i modyfikacja pamięci	18
Debugowanie programu	20
Jak ustawić punkty przerwań?	25
Programowe punkty przerwań	25
Punkty przerwań w pamięci ROM, Flash i EEPROM	30
Punkty przerwań na dostępie do danych	32
Punkty przerwań on-chip na różnych architekturach procesorów	34
Procesory RISC/CISC	35
Procesory DSP	39
Softcores	41
Rdzenie konfigurowalne	41
Podgląd i modyfikacja zmiennych HLL	42
Format zmiennych HLL	45
Kończenie pracy TRACE32	47

25.08.2009

O przewodniku

O czym jest ten przewodnik?

Przewodnik dotyczy wszystkich debugger'ów In-Circuit (TRACE32-ICD), które używają interfejsu typu on-chip (np. BDM, JTAG i ONCE).

Założenia

Przewodnik zakłada, iż oprogramowanie TRACE32 jest aktualnie zainstalowane. Pomocna jest podstawowa wiedza na temat oprogramowania debugującego oraz znajomość języka programowania C. Warunki te są konieczne w zrozumieniu przykładowych kodów źródłowych znajdujących się w tym dokumencie. Dodatkowo, zakładana jest elementarna znajomość systemu operacyjnego Windows. Wiedza na temat docelowych układów procesorowych oraz asemblerów i kompilatorów jest niezbędna do uruchomienia oraz pracy z oprogramowaniem TRACE32.

Przeznaczenie tego przewodnika

Celem niniejszego dokumentu jest opisanie podstawowych czynności konfigurujących środowisko pracy. Oznacza to, iż opisane są tu sposoby konfiguracji oprogramowania TRACE32 na komputerze typu host, uruchomienia debugger'a oraz tworzenia plików wsadowych w celu automatyzacji pracy. Dodatkowym aspektem przewodnika jest zaznajomienie Cię z podstawowymi cechami i właściwościami debugger'aa In-Circuit.

Jak używać przewodnika?

Przewodnik zawiera opis przykładowej sesji debugowania. Wykorzystuje on prosty program napisany w języku C w celu pokazania najważniejszych właściwości oprogramowania TRACE32. Wskazane jest, abyś przeprowadził kilka ćwiczeń czytając ten dokument. Rekomendowane jest również, dokładne i kompletne przeczytanie wszystkich zawartych tu rozdziałów, zarówno treści podstawowej (pisanej normalnym tekstem) jak i wskazówek (pisanych kursywą), które nie są powtarzane w innych rozdziałach.

Gdzie mogę znaleźć więcej informacji?

Interfejs użytkownika programu TRACE32 zawiera szczegółową pomoc (Online Help), która oferuje aktualne opisy wszystkich cech i właściwości przedstawionego tu środowiska. Rozdział 'Online Help' opisuje sposób pracy z systemem pomocy.

Ile czasu to zajmuje?

60 minut.

Konfiguracja środowiska

Po instalacji przeprowadzonej zgodnie z dokumentem 'ICD Szybka instalacja' pliki oprogramowania TRACE32 znajdują się w katalogu systemowych TRACE32. Dodatkowo, dostęp do programu możliwy jest przez menu Start. Proces instalacji ustawia wszystkie domyślne zmienne środowiskowe. Przedstawiona tu konfiguracja powinna być dostosowana do Twojego środowiska debugowania na komputerze PC typu host.

Uwaga: Skrót PC w kontekście oprogramowania TRACE32 zazwyczaj używany jest jako licznik programu (ang. Program Counter), dlatego też będziemy używać terminu host zamiast PC, oznaczającego urządzenie nadrzędne, które uruchamia środowisko TRACE32.

W celu dostosowania środowiska do swoich potrzeb, zapoznaj się z poniższymi terminami:

1. Definicja pliku konfiguracyjnego użytkownika.

Domyślnie używany jest plik konfiguracyjny config.t32 znajdujący się w folderze systemowym. Parametr –c umożliwia zdefiniowanie innej lokalizacji oraz nazwy dla tego pliku. Więcej informacji na temat plików konfiguracyjnych znajdziesz w dokumencie 'ICD Szybka instalacja'.

- Folder systemowy. Jest on podawany podczas procesu instalacyjnego. Zazwyczaj nie musisz dokonywać tu jakichkolwiek modyfikacji.
- Definicja twojego folderu roboczego. Zalecane jest wpisanie tu ścieżki folderu w którym będziesz przechowywał swoje projekty.
- 4. Rozmiar okna programu.
- 5. Przechowywane tu są tymczasowe pliki debugger'a. Zazwyczaj nie musisz dokonywać tu jakichkolwiek modyfikacji.



Uruchomienie TRACE32-ICD

Na samym wstępie uruchom system debugowania, a następnie swój układ docelowy.



Przy pracy z debugger'em In-Circuit (ICD) niezbędny jest uruchomiony system docelowy!

Zwróć uwagę na prawidłowy przebieg sekwencji włączenia / wyłączenia:

- Włączenie: debugger układ docelowy
- Wyłączenie układ docelowy debugger

W celu uruchomienia oprogramowania debugującego na komputerze host, otwórz folder TRA-CE32 w menu Start i uruchom interfejs użytkownika TRACE32. Jeśli utworzyłeś ikonę programu na pulpicie, kliknij ją dwa razy. Dla poniższego przykładu, zainstalowane zostało oprogramowanie dla procesorów z rodziny ARM.



Główne okno TRACE32

Po uruchomieniu oprogramowania TRACE32 zostanie wyświetlone główne okno debugger'a.



Program TRACE32 jest teraz uruchomiony i gotowy do pracy.

O TRACE32

Polecenie About TRACE32 w menu Help dostarcza informacji na temat wersji wszystkich modułów wchodzących w skład oprogramowania TRACE32-ICD.



Naciśnięcie przycisków Hardware lub Software znajdujących się na dole okna spowoduje wyświetlenie dalszych informacji odnośnie zainstalowanego sprzętu i oprogramowania, które są niezbędne przy kontakcie z naszym serwisem. Ponadto, można zauważyć, iż oprogramowanie zostało skonfigurowane zgodnie z rozdziałem 'Konfiguracja środowiska'. System Online Help zawiera kilkadziesiąt dokumentów, które są dostępne w formacie PDF z poziomu oprogramowania TRACE32 jak i z folderu Help. Specyficzne informacje dla konkretnych CPU umieszczone są w plikach postaci debugger_<cpu>.PDF. Istnieją trzy dostępne sposoby na uruchomienie systemu Online Help:

- Przycisk Help Topice na pasku narzędzi
- Pozycja Help Contents na pasku menu
- · Polecenie HELP wpisane w linii komend



System pomocy zorganizowany jest w postaci wielopoziomowej struktury. Ilustracja poniżej przedstawia sposób dostępu do niniejszego przewodnika.

2 HELP	
😵 Content 🎠 Index 🎢 Find 👫 Command Tree 🏾 🛃 Bookma	arks/ Print
Content	
close all open all 🗸 use filter: bdmarm:	
= TRACE32 Online Help = TRACE32 Directory	^
= TRACE32 Index	
■ TRACE32 Getting Started ■ ICD In-Circuit Debugger Getting Started	1
■ ICD Introduction ■ ICD Crick Testallation	_
= ICD Tutoria	
■ Tutorial kg = About the Tutorial	
= Start TRACE32-ICD	
= Setup the Application Properties = The Online Help	×
	<u>></u>

Oprócz pomocy składającej się z dokumentacji, dostępna jest również pomoc kontekstowa. Dostarcza ona informacji odnośnie specyficznych aspektu środowiska. Dostęp do pomocy kontekstowej składa się z dwóch etapów. Na samym początku należy ją aktywować poprzez naciśnięcie poniższego przycisku:



Kursor zmieni się na znak zapytania. Przesuń teraz kursor na interesujący Cię element, po czym otworzy się ramka z opisem wskazanego obiektu.

POWER	R TR	ACE ETHERN	ET @1	,	🔑 B::SYStem				-
Break	Ru	CPU Misc	Trace	Perf	Mode	MemAccess	Option	- Option	DisMode
break	1.000		more		ODown	O DAP	IMASKASM	DACR	OTUA ⊙
2 🕨	H	Step	F2	3	NoDebug	O TSMON3	IMASKHLL	MMU	O ACCES
101010105	ЪŔ.	Step Over Call	F3	and the second	060 2?	O PTMON3	TURB0	MPU	OARM
		-			O All Resets the	target and enables the	e debugger and start	the program execution	O THUM
	÷	Go Next	F4	0	O Sta See also: J	TAG Debugger for AR	M and XSDALE (deb	ugger_am.pdf)	
	5	Go Return	F5	hs:	O Up (StandBy)	 Denied 	INTDIS .		
	1.4	Colle	Ed	1993	OUp	- CpuAccess	DBGACK	- AMBA	MultiC
	6	Go up	PO	1000	· · · · · · · · · · · · · · · · · · ·	Enable	ShowError	NODATA	
	±	Go Til		2332	reset	 Denied 	EnReset	E DEC	
	Þ	Go	F7	200	RESetOut	 Nonstop 	WaitReset	SPLIT	
		Break	ES				TRST 🗹	- PC	
	11	or or one	.0	ENT	CPU	JtagElock	PWRDWN	0x0	
	122	Mode	F9	2250	ARM7TDMI	N.OMH: 🗸	TIDBGEN		
	200								

Like the command **Go** with a temporary breakpoint set to the next assembler command or next HLL line. This command can be used to overstep a subroutine call instruction or to leave a loop. See also the command **Step.Over**.

See also Go.Next Continue program

Ustawienia środowiska Debug

W celu skonfigurowania debugger'a musisz posiadać podstawową wiedzę na temat Twojego procesora oraz konfiguracji układu docelowego. Aby wgrywać do układu docelowego programy zawierające wszystkie symbole i informacje dla debugger'a, musisz także umieć posługiwać się kompilatorem.

Podstawowa procedura konfiguracji i opis specyficznych ustawień CPU dla debugger'a ICD opisana została w dokumencie 'ICD Target Manual'.

'ICD Target Manual' w szybki sposób daje Ci dostęp do opisu ustawień oraz dodatkowych możliwości Twojego procesora docelowego.

Oprócz tego, wszystkie komendy debugger'ów szczegółowo opisane są w dokumencie 'Reference ICE/FIRE/ICD'.

Jeśli używasz układu ewaluacyjnego, konfigurację jego peryferii możesz znaleźć w katalogu demo. Poniższa ilustracja pokazuje kilka przykładów dla procesorów PowerPC, ARM i TriCore.



Kolejne kroki przedstawiają typową procedurę konfiguracji debugger'a. W celu demonstracji niezbędnych kroków, przeprowadzimy ręczną konfigurację środowiska, a następnie zaprezentujemy znacznie szybszą metodą polegającą na wykorzystaniu plików wsadowych. Okno SYStem dostarcza wszystkich specyficznych ustawień dla konkretnego CPU. Otwórz okno System Settings w sposób pokazany poniżej:



Procesory są zgrupowane w rodziny (np. ARM, TriCore lub PowerPC). Okno System pokazuje wszystkie parametry, które są specyficzne dla konkretnej grupy procesorów. Każdy procesor posiada ponadto zestaw własnych parametrów i ustawień. Poniżej widać okno SYStem dla rodziny procesorów ARM.

🔑 B::SYStem				
Mode	MemAccess	Option	Option	DisMode
 Down 	ODAP	IMASKASM	DACR	OTUA ⊙
O NoDebug	O TSMON3	MASKHLL	MMU	ACCESS
Go	O PTMON3	TURBO	MPU	○ ARM
O Aitach	O UDMDN3	BigEndian	CFLUSH	O THUMB
◯ StandBy	ORealMON	ResBreak	CINV	
O Up (StandBy)	Denied	INTDIS		
OUp	- CpuAccess	DBGACK	AMBA	MultiCare
	OEnable	ShowError	NDDATA	
reset	 Denied 	EnReset	EXEC	
RESetOut	Nonstop	WaitReset	SPLIT	
		TRST	- PC	
CPU	JtagClock	PWRDWN	0x0	
	10.0MHz 💙	TIDBGEN	1	
2				

 Poinformuj program TRACE32 o używanym przez Ciebie typie procesora w układzie docelowym, jeśli automatyczne wykrycie CPU nie jest możliwe. Wybierz poprawny typ z listy CPU w oknie SYStem. (Alternatywnie możesz użyć komendy SYStem.CPU <CPU type>)



Standardowo, wszystkie dostępne opcje są ustawione w domyślnej konfiguracji. W katalogu demo, który jest dostarczony wraz z oprogramowanie TRACE32, możesz znaleźć przykłady konfiguracji parametrów różnych systemów.

👼 demo.cmm - Notepad		×
File Edit Format Yew Help		
 Example script file for ARM Integrator CM1136JF-S ; 07.01.04, PEG		>
; The following settings assume jumper J7 (CFGEN) is removed.	that	
 ; start-up debugger system.cpu arm1136jf System.option Res0reak OFF System.MultiCore IRPRE 4 System.MultiCore ORPRE 1 System.MultiCore GTBIRPOST 5 System.MultiCore GTBIRPOST 1 System.Up		
; load demo application Data.LOAD.AIF armla.axf Data.Assemble 0 b 2078 Register.Set r13 0x1000		
; open some windows WinCLEAR WinPOS 0% 0% 100% 50% Data.List WinPOS 0% 50% 50% 50% Var.Frame /Locals /Caller WinPOS 50% 50% 50% 50% Var.Watch Var.Addwatch ast flags		
ENDDO		<

2. Wykonaj inicjalizacje uruchomienia i podaj tryb debugowania.

W celu zresetowania procesora z włączonym trybem debug należy wybrać przycisk Up w sekcji Mode w oknie SYStem. (komenda: SYStem.UP)

W tej chwili TRACE32 nawiązuje połączenie z docelowym mikroprocesorem. Czynność ta powoduje reset docelowego CPU, uruchomienie go w trybie debugowania oraz zatrzymanie na wektorze resetu (ang. reset vector). Jeśli podczas tego procesu wystąpi jakiś błąd, zapoznaj się z dokumentacją 'ICD Target Manual'. W przypadku gdy wszystko przebiegnie pomyślnie, powinieneś mieć dostęp do układu docelowego, np. jego procesora lub pamięci.



3. Wykonaj specyficzną konfiguracje układu docelowego.

Procesor jest aktywny lecz zatrzymany. Rejestry są ustawione na wartości domyślne. W kolejnych krokach należy skonfigurować pamięć, poprzez zapisanie rejestrów o specjalnym przeznaczeniu (ang. special function registers) używając komendy PER.Set. Przykładowo, niektóre procesory potrzebują wybrania odpowiedniego chipu w celu dostępu do pamięci. Jeśli używasz płyty ewaluacyjnej, jej firmware może inicjalizować tego typu moduły.

4. Wgraj aplikacje do układu docelowego.

Następnym etapem jest załadowanie Twojej aplikacji do pamięci układu docelowego. Upewnij się koniecznie czy Twój system ma możliwość dostępu do pamięci, gdyż jest to czynność bardzo specyficzna dla każdego rodzaju CPU. W celu sprawdzenia, spróbuj zmienić wartość jakiejś komórki pamięci korzystając z komendy Data.Set.

Załaduj swoją aplikacje poprzez komendę Data.LOAD (Data.LOAD.<option> <file_ name>). Opcje, które są wymagane przez Twój kompilator mogą być znalezione w dokumentacji 'ICD Target Manual' w sekcji 'Support/Compilers'. Alternatywnie, możesz użyć komendy Data.LOAD * i wybrać plik z poniższego okna:



W celu zaprogramowania pamięci typu Flash skorzystaj z grupy komend FLASH oraz dokumentu 'TRACE32 Training / Training ICD In-Circuit Debugger / Training IDC Basics / Flash Programming'.

Aby wyświetlić kod źródłowy skompilowanego programu, musi on być dostarczony wraz z informacjami dla debugger'a (zazwyczaj: opcja kompilatora debug). W tym momencie TRA-CE32 będzie mógł bezpośrednio pracować z kodem aplikacji.

5. Zainicjuj licznik programu i wskaźnik stosu za pomocą komendy Register. Set.

Wiele kompilatorów dodaje automatycznie poniższe ustawienia kodu startowego do programów użytkownika.

🖪 demo.cmm - Notepad		\bowtie
File Edit Format View Help		
Example script file for ARM Integrator CM1136DF-S ; 07.01.04, PEG		<
; The following settings assume jumper J7 (CFGEN) is removed.	that	
; start-up debugger SYStem.CPU arm136jf SYStem.Option ResBreak OFF SYStem.MultiCore IRPRE 4 SYStem.MultiCore DRPRE 1 SYStem.MultiCore ETBIRPOST 5 SYStem.MultiCore ETBIRPOST 1 SYStem.Up		
 ; load demo application Data.LOAD.AIF armla.axf Data.Assemble 0 b 2078 Register.Set r13 0x1000		
; open some windows wincLEAR winPOS 0% 0% 100% 50% Data.List winPOS 0% 50% 50% 50% Var.Frame /Locals /Caller winPOS 50% 50% 50% 50% Var.Watch Var.Addwatch ast flags		
ENDDO		
		Y

Powyższa sekwencja demonstruje ręczny sposób konfiguracji środowiska debugger'a. Zdecydowanie szybszą i praktyczniejszą metodą jest wykorzystanie plików wsadowych, które wykonują te czynności w sposób automatyczny. W celu zagwarantowania poprawnej procedury konfiguracyjnej, zalecane jest napisanie tego typu pliku. W następnym rozdziale pokażemy jak można tego dokonać. Utwórz nowy plik wsadowy start.cmm w katalogu roboczym, poprzez użycie komendy PEDIT start.cmm.

TRACE32 posiada własny język skryptowy do wykonywania zadań wsadowych. Nazywa się on PRACTICE i posiada bardzo duże możliwości (zobacz 'PRACTICE User's Guide' i 'PRACTICE Reference'). Dozwolone są wszystkie polecenia oprogramowania TRACE32, komendy sterujące wykonywaniem aplikacji, polecenia warunkowe oraz komendy I/O. Domyślnym rozszerzeniem plików wsadowych jest '.cmm'.

Dodatkowo, istnieje możliwość debugowania programów napisanych w języku PRACTICE. W celu zasięgnięcia obszerniejszych informacji, przeczytaj dokumenty 'PRACTICE User's Guide' oraz 'PRACTICE Reference' (komendy: PLIST, PEDIT, PBREAK).



Wpisz niezbędne komendy i zakończ skrypt poprzez wpisanie polecenia ENDO i kliknięcie przycisku Save. Rysunek powyżej pokazuje przykładową procedurę startową procesora PowerPC505.

Rozpoczęcie wykonywania skryptu dokonuje się poprzez wybranie pozycji Batchfile... w menu File, paska menu.

TRACE32		_ 🗆 🗙
<u>File</u> <u>E</u> dit <u>V</u> iew	v Var <u>B</u> reak <u>R</u> un <u>C</u> PU <u>D</u> evices <u>Trigger</u> <u>A</u> nalyzer <u>P</u> erf <u>C</u> ov MPC8xx <u>W</u> indow <u>I</u>	<u>H</u> elp
<u>O</u> pen		
Batchfile		
Load		
Type	RRCD DO *	
<u>D</u> ump	0000.00	
Stop Comma	Suchen in: 🔄 Diab	
Choose Color	Cacheen.cmm	
<u>P</u> rint	2 Demo.cmm	
	Diabc.cmm	
ežor	Prefast.cmm	
	Presiow.cmm	
	Dateiname: Cffnen	
	Dateityp: Current (*.cmm)	
B::		
J		
emulate	Data Var trigger devices Analyzer other previous	
P:00080004 \\di	tiabp5\Global_start stopped MIX	EI //.

W celu kontynuowania naszego przewodnika, wybierz jeden przykładowy plik, który możesz znaleźć w katalogu systemowym TRACE32 pod folderem \demo\<processsor_family>\compiler\... np. \demo\powerpc\compiler\Diab\Diabc.cmm lub wykorzystaj swój własny plik skryptowy, jeśli już taki przygotowałeś.

Interfejs użytkownika



Otwórz okno w celu wyświetlenia rejestrów CPU. Alternatywnie, możesz wybrać Register z menu View, nacisnąć przycisk Register lub wpisać Register.view za znakiem zachęty B:: w linii poleceń.

Większość właściwości oraz narzędzi dostępna jest w menu kontekstowym, w głównym pasku narzędzi lub za pomocą wiersza poleceń. Pamiętaj o tych metodach, nawet jeśli w dalszej części tego przewodnika będziemy używać tylko jednej z nich.

Wiersz poleceń TRACE32 nie jest wrażliwy na wielkość znaków. W dostarczonej dokumentacji używamy dużych liter dla znaków, które są znaczące dla wprowadzanych poleceń np. Register.view może być skrócony do formy r. Następnym przykładem, pokazującym typową strukturę poleceń TRACE32 <command_family>. <subcommand> jest Data.List, który może być skrócony do d.l. Bardzo poręczną funkcją jest możliwość zastosowania przycisków skrótów (ang. softkeys). Dostarczają one wskazówek dla wpisywanych komend w linii poleceń, prezentując wszystkie możliwe komendy oraz ich parametry. Zamiast wpisywać całe polecenie, możesz zbudować je za pomocą klikania na odpowiednie przyciski.

Przykład: Konstrukcja komendy Data.dump za pomocą przycisków skrótów.

Softkova
B::Data. Sonkeys
[ok] dume View Print P:000811B8 \\diabp5\diabp5\sieve+4
B::DATA.DUMP
[ok] (range) (address) options
P:00080004 \\diabp5\Global_start
B:: Data.DUMP 10002000
[ok] options
P:000811B8 \\diabp5\diabp5\sieve+4

Więcej informacji odnośnie interfejsu użytkownika można znaleźć w dokumencie 'Operating System User's Guide'.

Nagłówek okna zawiera nazwę komendy, która wywołała aktualne okno.

🛄 B::Re	gister						_ 🗆 🗵
RØ	81190	R8	Ø	R16	ØFFFFFFFF	R24	ØDFFFFFFF 🔺
R1	8FF68	R9	ØFFFFFFFF	R17	ØFEFFFFFF	R25	ØFFFFFFFF 📃
R2	8C070	R10	90000	R18	ØFFFFFFF7	R26	ØFFFFFFF
R3	ØB	R11	0	R19	ØFFFFFFFF	R27	ØFFBFFBFF
R4	2C	R12	80468	R20	ØFFFFFBFF	R28	0
R5	37	R13	94308	R21	ØFFFFFFDF	R29	ØF
R6	8C35Ø	R14	ØFFFFFBFF	R22	ØFFFFFFFF	R30	3
R7	156	R15	0	R23	ØFFFFFFFF	R31	0
							•
•							► <i>[li</i> ,

Podgląd i modyfikacja pamięci

W celu sprawdzenia pamięci o zadanym adresie, należy użyć okna Data.dump w sposób jaki został pokazany w poprzednim rozdziale. Wpisz polecenie data.dump <zakres adresu> lub wybierz:

- Select Dump... z menu View.
- Naciśnij ikonę na pasku narzędzi.
- Wpisz polecenie Data.dump.

Okno podglądu pamięci zostanie otwarte. Wypełnij pozycje danych w poniższym oknie. Naciśnięcie przycisku Browse umożliwia przeglądanie danych za pomocą symboli. Wybierz etykietkę poprzez podwójne kliknięcie oraz potwierdź czynność przyciskiem OK. Jeśli używasz linii poleceń, możesz bezpośrednio wprowadzić interesujący Cię adres lub symbol.



Dane zostaną wyświetlone w oddzielnym oknie.

Poniższe zdjęcie prezentuje sposób wyświetlenia danych w pamięci za pomocą linii poleceń.

B::Data.dump	flags	/byte
[ok] optic	n:	
ST:000019F8 \Whum	ble\am\n	nain

🗱 B::Data.dump flags /Byte		
address 0 1 2 3 SD:00006770 47 62 57 E4 SD:00006780 B7 0C FF 00 SD:00006790 BF 60 FB 00 SD:00006740 67 86 B8 30 SD:00006740 FD 82 FF 00 SD:00006700 FF 24 7F 00 SD:00006700 BE 00 FF 00 SD:00006700 BE 00 FF 00 SD:00006720 2E 14 18 54 SD:000067F0 9B 33 D9 80	4 5 6 7 8 9 A B C A0 70 99 82 6D 09 3D 90 → D 3D 10 FF 40 FF 10 F7 04 E EF A8 FF 80 FF 04 FB 00 F E9 81 DF 73 BD 58 FD E2 F FE 68 57 00 97 44 FB 52 3 FF 10 BD 00 FE 00 DF 00 E FB 00 FF 00 FD 01 EF 00 F BB 00 BF 32 F6 E0 E3 80 51 BB 20 7F 10 96 9A 8F 31 2	D E F 0123456789ABCDEF 5 38 2F 38 GbWE8p38m 4=898/8 5 38 2F 38 GbWE8p38m 4=898/8 8 10 FF 40 9 FF N=85 0F 15 FF 16 15 N 0 11 FF 01 9 S805 19 S8X554 FF N615 N > A 02 C7 18 9 S805 19 S8X554 S6 16 N > > F BC EC 4 S54 NEA NANDERPSES < > > > > > > > > > > > > > >
<		2
Klasa namieci + adres	∳ Watoćć box	

Istnieją różne sposoby wyświetlenia danych i definiowania zakresu adresów:

- <adres początku>--<adres końca>
- <adres początku>++<offset>

Wartość pamięci spod zadanego adresu może być zmodyfikowana poprzez podwójne jej kliknięcie. Komenda Data.Set dla wybranego adresu wyświetlana jest w wierszu poleceń. Wprowadź nową wartość i potwierdź ją enterem.

4	A TRACE32 ARM [POWER TRACE ETHERNET @]
E4	le Edit View Var Break Run CPU Misc Trace Perf Cov Window Help ▶ ▶ ≠ ≠ ≠ ⊄ ▶ Ⅱ 🕺 ?? № 📰 👹 ன 🐼 🚱 🧕
	199 IB-Data dump flags /byte]
	address 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456769ABCDEF 5D:000067270 47 62 57 E4 A0 70 99 82 60 09 30 90+D5 38 2F 38 GDW\$293914-388/8 ▲ SD:00006780 83,0C FF 00 3D 10 FF 40 FF 10 F7 04 EB 10 FF 40 \$FF 4-\$\$6\$55\$5\$5\$
	SD:00006740 BF %0 FB 00 EF A8 FF 80 FF 04 FB 00 FF 01 FF 01 \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$ \$
ACCURCURATION OF	SD:000067F0 2E 14 18 54 BB 00 BF 32 F6 E0 E3 80 58 EC 9F 04 51 T6 F6 E0 E3 80 58 EC 9F 04 51 T6 F6 E0 F3 2F6 E0 E3 80 58 EC 9F 04 51 77 50 50 50 50 50 F6 E7 05 F1 10 F6 9F 00 F7 05 F5 74 F6 F0 67 75 24 17 F6 00 F6 00 F6 55 74 W W W W 10 SD:00006800 F7 05 F4 17 F7 00 F7 00 F7 00 F7 00 F7 00 F7 00 F6 00 F6 00 F6 00 F6 00
	SD:00006820 7A 19 DC A2 44 0D SE C2 3F 02 CD B8 91 14 BF 77 25200%257555555 ₩ M
B	::D.S SD:0x6770
SE	[ok] formats catalog options previous D:00006770 WthumbleVGlobal/sinewave+0x13 system ready MDX UP

Otwórz okno Data.List poprzez wybranie pozycji List Source w menu View. Listing kodu źródłowego w obrębie licznika programu zostanie wyświetlony.

🐥 TRACE32 ARM [POWER TRACE ETHERNET @]						
File Edit View Var Break Run CPU Misc Trace Perf Cov Window Help						
🕨 🖬 🖬 Registers 🔰 😵 N? 💿 🔄 🏭 📕 🍪 🍪 🍪 🤰						
2382 Dump						
B:Da List Source						
N Ster 🐼 Watch						
addr/1 Referenced Var label mnemonic comment						
SR:000 dcd 0x5999999A						
And brocce						
artiframe						
ST:000 Periharak main: push {r4,r7,r14}						
ST:000 sub sp,#0x8						
Source i						
d particular and a second of the last						
ST:000 How ro, How ro, How ro, How ro, How ro						
ST:000 Trace List 1dr r1.0x1850						
Message Area						
ST:0001402 2002 nov r0.H0x2						
51:000104/7308 vtripplearray[0][1][0] = 3:						
ST:00001A06 2003 Hov r0,#0x3						
51:0001408 /108 strp r0,[1,H084]						
ST:0001A0A 2004 Hov r0,#0x4						
ST:00001A0C 7048 STD F0.[F1.H0X1]						
595 func2();						
ST:0000140E F823F7FF DI 0x1058 ; FURC2						
597 func2a();						
N. MILLERY						
fok] (cardea) (cardeave) cotions						
CT.00001000 With white size and a size and						

Wykonaj pojedynczy krok, klikając na jeden z następujących elementów:

- pozycja Step w menu Run
- <F2>
- przycisk Step na pasku narzędzi
- przycisk Step w oknie Data.List
- komenda Step wprowadzona w wierszu poleceń

Run	CPU Misc	Trace
H s	itep N	F2
F 3	Rep Over දිබ්	P3
4	io Next	F4
4 5 (io Return	P5
¢ <	io Up	P6
± <	io Till	
•	50	P7
11 8	ireak	F8
32 •	1ode	P9



🖃 B::Data.List						
📕 Ştep 🛛 뵭	Over	↓ Next				
addt	code					
SR:000019F4	00005;	3CC				
586 ST:000019F8 ST:000019FA	nain((8590 8082)				

Teraz spojrzyj na pasek statusu. Adres aktualnej pozycji kursora (szary pasek w aktywnym oknie) jest wyświetlony.

Następne pole prezentuje obecny stan debugger'a: stopped oznacza, że Twój program jest zatrzymany. W tym momencie możesz np. podejrzeć lub zmienić jego pamięć.

A TRACE32 ARM [POWER TRACE ETHERNET @]	
File Edit View Var Break Run CPU Misc Trace Perf Cov Window Help	
N M I I I 20 8 N? ◎ I II II II 20 60 60 2	
📰 B::Data.List	
附 Step 📑 Over 📜 Next 🖋 Return 🙋 Up 🕒 Go 🔢 Break) 🎉 Mode 🛛 Find:	arm.
addr/line code label mnemonic comment	
SR:000019EC 3FB99999 dcd 0x3FB99999	
SR:000019F0 3555555A dcd 0x53555A	
586 main()	
ST:000019F8 8590 main: push {r4,r7,r14}	
ST:000019FA 8082 sub sp.#0×8	
int j; char * p;	
590 vtripplearray[0][0][0] = 1;	
ST:000019FC 2001 nov r0.#0x1	
ST:00001400 7008 strb r0.[r1]	
B::Sten	
a. ocep	
' Camalata baasa daxiaan baxa Data Yar DEPE atam Camaing	
emulare trigger devices trace Daka Var PENP other previous	
ST:000019FA \\thumble\arm\main+0x2 ST:000019FA \\thumble\arm\main+0x2 MIX	UP
(symboliczny) adres pozycji kursora 🛛 stan debugger'a 👘 tryb debug	

W zależności od użytego skryptu startowego, wyświetlanie kodu źródłowego będzie w postaci HLL (High Level Language) lub mieszanej (HLL i odpowiadające jemu mnemoniki asemblera). Pole trybu debug na pasku stanu sygnalizuje aktualnie używany tryb. Naciśnięcie przycisku u góry okna Data.List zmienia sposób wyświetlania kodu z HLL na mieszany i odwrotnie. Pasek stanu zawsze pokazuje tryb debug.

A TRACE32 ARM [POWER	TRACE ETHERNET @]	
File Edit View Var Break	Run CPU Misc Trace Perf Cov Window Help	
N# + 4 4 +	II 🖄 ? M? 💿 🗄 🖽 📕 🌌 🗟 🚳 🧯 🛓	
B::Data.List		
🕨 🚺 Step 🚺 Over 🗸	Next 🖌 Return 🖉 Up 🚺 🕨 Go 🛛 🔢 Break 🌋 Mode) Find:	arm.
addr/line code	label mnemonic comment	
SR:000019F0 9999999A	ARCESSER DX999999A	
SR:000019F4 000053CC	dcd 0x53CC	
() anim		
- Inditi (7		
Przyciski lokalne dla okna	Data List Przełącznik trybu debug	

Podgląd w trybie HLL i mieszanym:

📰 B::Data.List		🔡 B::Data.Lis	st				×
📕 Step 📑 Over	🕹 Next 🖉 Return 🙋 Up 🕒 Go 🛛 🖬 Break 🚺	📕 Step 🛛 🖬	Over	🕹 Next 🛛 🖋 Return 🚺	C Up	🕞 🕨 Go 🚺 Break	
addr/line source		addr/line	code	label	Inemon	ic	
580 581 582 583 }	<pre>index = 0; for (x = 0.0 ; x < 62.8 ; x += 0.1)</pre>	SR:000019EC SR:000019F0 SR:000019F4	3FB9999 9999999 000053C	9 A C	ded ded ded	0x3FB99999 0x9999999A 0x53CC	<
586 (asin()		586 ST:000019F8 ST:000019FA	8590 8082	main:	push sub	{r4,r7,r14} sp,#0x8	
	int j; char * p;			<pre>int j; char * p;</pre>			
590	<pre>vtripplearray[0][0][0] = 1;</pre>	- 590		Vtripplearray[0][0][0] =	1:	
591	vtripplearray[1][0][0] = 2;	ST:000019FC	2001		BOV	r0,#0x1	
592	vtripplearray[0][1][0] = 3;	ST:000019FE	4954		ldr	r1.0x1850	
593	vtrippiearray[0][0][1] = 4;	51:00001A00	17008	at visual excess [1][stro	r0.Lr11	
595	func2();	ST:00001A02	2002	A CLAPPINGE LOY CE DE	BOY	r0,#0x2	
597	func2a();	ST:00001A04 592	7308	vtripplearray[0][strb 1][0] -	r0,[r1,H0xOC] 3;	-
500	func(h():	ST:00001A06	2003		strb	r0,#0x3 r0.fr1.#0x41	
		593	1.00	vtripplearray[0][0][1] =	45	
601	func2c();	ST:00001404	2004		nov	r0.#0x4	
603	func2d();	31.0000 INOC	-		SCID	103111340811	
605	funcptr = func3;	ST:0000170E	EB23F7F	Func2();	Ь1	0×1058	
607	ast.count = 12345;	597		func2a();			
608	ast.left = *	ST:00001A12	FB53F7F	F	61	0×108C	
609	ast.field1 = 1;						-
610	ast.field2 = 2;	ST:00001416	SPCD575	func2b();	61	0×1054	
612	ast = func4(ast);	31.00001H10	recorri		01	081014	
614	<pre>j = (*funcptr)();</pre>	601 ST:00001A1A	FB7FF7F	func2c(); F	ь1	0x111C	
616 start: 617	j = func5((int) j, (char) 2, (long) 3); 💌	603 ST:00001A1E	FBD9F7F	func2d();	61	0x11D4	~
<			<	П		>	14

Przełącz tryb debug na HLL i wykonaj następny krok. Krok który właśnie wykonałeś został przeprowadzony w języku wysokiego poziomu, aż do następnej jego instrukcji. Jeśli przełączysz się znowu na tryb mieszany i naciśniesz przycisk Step to wykona się jedna instrukcja asemblera.

A TRACE32 AN	W [POWI	ER TRACE ET	HERINET @]		
File Edit View	Var Brea	k Run CPU	Misc Trace Perf	Cov Window Help	
	5 + 2	ь II (%)	2 M2 00 1	1 351 m 🐼 🛵	😹 💼 🏅
			9 .4. W 12		·····
🔡 B::Data.Lis	at				
N Step	Over	🕹 Next 🖌	Return 🕑 Up	► Go II Break	k 💹 Mode Find:
addr/line	code	18	bel mnemon	lc .	comment
607		ast.count	- 12345;		
ST:00001A28	4840		lar	r0,0x185C	
ST:00001A2A	60.49		tur	r1,0x1860 r0_[r1_00x4]	
608	0040	ast.left =	Sast:	107011780341	
ST:00001A2E	484C		ldr	r0,0x1860	
ST:00001A30	6088		str	r0,[r1,#0x8]	
609		ast.field1	= 1;		
ST:00001A32	6900		lar	r0,[r0,H0x10]	
ST:00001439	0080		161	r0,r0,#0x2	
ST:00001A38	2301		TOV	r3.00x1	
ST:00001A3A	4318		011	r0,r3	
ST:00001A3C	6108		str	r0,[r1,#0x10]	
610	10.10	ast.field2	- 2;		Program Address
ST:00001A3E	4848		lar	r0,0x1860	💼 Go Til
ST:00001A40	2310		TOP	r0,[r0,#0x10]	Breakpoint
ST:00001A44	4398		hic	r0.r3	G Development A
ST:00001A46	2308		EOV	r3,00x8	Breakpoints P
ST:00001A48	4318		orr	r0,r3	Display Memory
ST:00001A4A	6108		ştr	r0.[r1.#0x10]	Toggle Bookmark
ST:00001A4C	0500		IST	r0,r0,M0x18	As Cat OC Have
612		ant - form	4(ast):		Sec PC Here
ST:00001A4E	4558	0.00 - 10000	BOV	r0.r13	Edit Source
ST:00001A50	310C		add	r1,#0x0C	🗯 View Info
ST:00001A52	C990		ldmia.	r11,{r4,r7}	
ST:00001A54	C090		stmia	r01, {r4, r7}	🚽 Go Till There
ST:00001A56	4042		lar	r3,0x1860	There is there
ST:00001458	CBDE		ldeia	r31 (r1-r2)	300 000 0000
51.0000 1101	10000		100110	1013011163	Assemble here
B::					Modify here
1					
errolate t	iyyer 🗌	devices	trace Data	Var PEF	RF other previous
ST:00001A3C \\#	humble/\arm/	\main+0x44	stopped at brea	kpoint	MK UP

Wybierz linię kodu i naciśnij prawy przycisk myszy. Jeśli wybierzesz pozycję Go Till, wykonywanie programu zostanie wznowione, aż do momentu, kiedy program osiągnie wskazaną przez Ciebi linię kodu.

Praca krokowa jest jedną z podstawowych metod analizy i debugowania kodu. W celu zapoznania się z innymi metodami uruchamiania programu, przyjrzyj się pozycjom w menu Run, przyciskom w oknie Data.List oraz głównemu pasku narzędzi.





Pojedynczy krok Krok ponad wywołanie funkcji Idź do następnej linii kodu w listingu programu. Użyteczne np. do opuszczania pętli

Przerwij / Zatrzymaj wykonywanie w czasie rzeczywistym Idź / Rozpocznij wykonywanie w czasie rzeczywistym Idź w górę / wróć do funkcji wywołującej Wróć / Idź do ostatniej instrukcji Komendy Go next, Go Return i Go Up są dostępne tylko jeśli program jest uruchomiony w pamięci RAM lub jeśli procesor udostępnia punkty przerwań typu on-chip.

Na potrzeby poniższego przykładu, załóżmy, że posiadamy zagnieżdżenie funkcji gdzie funkcja main wywołuje func2(), a ta wywołuje funkcję func1().



Okno Var.Frame wyświetla zagnieżdżone funkcje w programie. Wykorzystując opcję LOCAL, lokalne zmienne każdej z funkcji będą możliwe do podejrzenia. Kiedy opcja CALLER jest ustawiona, kilka linii kodu w języku C będzie wyświetlone w celu zaznaczenia gdzie dana funkcja została wywołana. Poniższy rysunek prezentuje zagnieżdżenie i sekwencję wywołania funkcji opisanych w tym akapicie.



TRACE32-ICD dostarcza bardziej złożonych poleceń służących do kontrolowania pracy programu. Możliwa jest praca lub wykonywanie krokowe, dopóki jakiś z warunków nie zostanie spełniony. Przykładowo: Var.Step.Till j>9 wykona poszczególne kroki programu dopóki zmienna j nie będzie większa niż 9. Więcej szczegółowych informacji można znaleźć w dokumencie 'Reference ICE/FIRE/ICD' w opisie takich funkcji jak Step.Change, Step.Till, Go.Change, Go.Till, Var. Step.Change, Var.Step.Till, Var.Go.Change oraz Var.Go.Till.

Programowe punkty przerwań

Debugger ICD domyślnie używa programowych punktów przerwań (ang. breakpoint). W momencie kiedy programowy punkt przerwania jest ustawiony na jakimś wyrażeniu, kod w tym miejscu zastępowany jest specjalną instrukcją np. TRAP, która zatrzymuje wykonywany w czasie rzeczywistym program i przekazuje kontrolę do systemu debugującego on-chip. Metoda ta wymaga wolnej pamięci RAM w miejscu zatrzymania!. Jeśli uruchamiasz program w pamięci RAM, ilość programowych punktów przerwań jest nieograniczona.

W przypadku gdy Twoja aplikacja nie jest uruchamiana w pamięci RAM, zapoznaj się z rozdziałem 'Punkty przerwań w pamięci ROM, Flash, EEPROM'.

Wróćmy do naszego przykładowego programu. Kliknij dwukrotnie na linii kodu w której chcesz ustawić punkt przerwań. Zwróć uwagę, aby nie zaznaczyć pustej linii. Wszystkie instrukcje, w których są ustawione punkty przerwań oznaczone są małym, czerwonym paskiem.



Użyj pozycji List z menu Breakpoint w celu wyświetlenia informacji odnośnie wszystkich używanych punktów przerwań.



Format listy punktów przerwań jest następujący:

- Kolumna 1: Adres hex punktu przerwania
- Kolumna 2: Typ punktu przerwania
- Kolumna 3: Sygnalizuje sposób realizacji punktu przerwania programowy (SOFT), ONCHIP, wyłączony (DISABLED).
- Kolumna 4: Przerywana instrukcja. Np. func2\6 oznacza linie 6 kodu HLL w funkcji func2; func2\13+0x8 oznacza linie 13 kodu HLL w funkcji func2 plus 8 bajtów (użyteczne tylko w trybie mieszanym (MIX)).

A TRACE32 ARM [POWER TRACE ETHERNET @]	
File Edit View Var Break Run CPU Misc Trace Perf Cov	Window Help
H H + ✓ Ć ▶ II 図 💈 № 💿 📰 🕮	i 📰 🚳 🚳 🕸 😰 主
	🔄 B::Data.List 📃 🗖 🔀
	🔰 Step 📑 Over 🚽 Next 🗸 Return 🔮 Up 🕨 Go 🔢 Break
	addr/line source
	156 (*intptr)++;
B::Break.List	
Cheker All O Dirable Al S Enable All O Init 2 Select.	160 (
T:00001058 Program SOFT func2	int autovar; register int regvar;
T:0000105A Program DISABLE Func2\6 T:00001060 Program SOFT func2\7	static int fstatic = 44; /* init
T:00001066 Program DISABLE func2\9 T:0000106C Program S0FT func2\11	static int istatic2; /* not
Talez T	→66 autovar = regvar = fstatic; →67 autovar++;
	169 fime1/ Sautoway): /* to farme aut
X	171 funcl(&fstatic); /* to force fst
	173 for (regvar = 0; regvar < 5; regvar++
	175 Citation and that
	1/b Istatic += nstatic1;

TRACE32 ARM [POWER TRACE ETHERNET @]		
File Edit View Var Break Run CPU Misc Trace Perf Cov Window	Help	
▶ ▶ ↓ ↓ ↓ ↓	X 6d 63 😰 🔬	
	🔚 [B::Data.List]	
	Note Next of Return (addr/line code label	±Up ►Go 🖬 Break 👸 nnenton ic
B::Break.List	171 func1(&fstatic); ST:0000106C 4811 ST:0000106E FFEFF7FF	/* to force fstati 1dr r0,0x1084 b1 0x1050
address types 1mp1 T:00001072[Program SOFT func2\13 T:0000107A Program SOFT func2\13+0x8	5T:00001072 2700 ST:00001072 2700 ST:00001076 D802 ST:00001076 B00A ST:00001076 A3701 ST:0000107C E7FA	regvar < 5 ; regvar++) mov r7.m0x0 cmp r7.m0x0 blt 0x107E blt 0x107E b 0x1090 add r7.m0x1 b 0x1074
	174 wstatic1 + ST:0000107E 1C38 ST:00001080 9900 ST:00001082 4348	- regvartautovar; mov r0,77 Tdr r1,[r13] mul r0,r1 ⊻

Rozpocznij wykonywanie programu wybierając funkcję Go. Jeśli program nie osiągnie ustawionego przez Ciebie punktu przerwania, zawsze możesz zatrzymać program używając funkcji Break.

Aby usunąć punkt przerwania należy kliknąć dwukrotnie na oznaczonej linii kodu, lub wyłączyć go w oknie Break.List.

Celete All O Disable Al Enable Al O Ir address types	it 🖉 Select impl	Store	Loa	d 🔞 Sel			
T:00001BAA Program T:00001BBE Program	SOFT	sieve\11 sieve\19	©× ∽	Breakpoint Change Delete Enable			
				here 🔸	Address Go Til Breakpoint Breakpoints Display Memory Toggle Bookmark	•	Program Spot
					Edit Source		Read Write Alpha
							Charly Delta Echo

W celu ustawienia punktu przerwania na linii kodu, która nie jest aktualnie wyświetlona, można posłużyć się pozycją Show Function... w menu Var. Podwójne kliknięcie na nazwie funkcji spowoduje skok do jej kodu źródłowego, co umożliwia ustawienie punktu przerwania w interesującym nas miejscu.

TRACE32 ARM [POWER TRA	ACE ETHERNET @]			
Edit View Var Break Run	CPU Misc Trace Perf	Cov Window Help		
Wetch	8 N? .		i 🙂 🛓	
Q Data View				
😫 Breakpoint		🛓 B::sYmbol.Brow	se.Function * /Click "Data.List *" /D	Delete 🔲 🗖 🔀
Show Function	<u></u>	11444	t. Type:	Functions 🖌 🗌 Source
Show Locals		func40	(vold ())	address T:0000195C000019F7
Show Stack		func6	(float ())	T:0000123000001243 T:0000124400001293
Show Current	Vars	func8	(void ())	T:00001294000012FF T:000013000000146F
Normat		main	(int ())	T:00001970000019408 T:000019F800001878
		Dou	ble Click	1:0000187000001801
B:DataList sieve	A Mart C of Datum	Alla I A Ca I H	Prost Witteds End	
addr/line source	e Next Pressing		bleak (permode) Pint	
	Classe [CT22.4].		<u>^</u>	
char :	tiegstolc6+11;	10 -	and a for an effective set	
678 (Int 8	Level /	/* 81	leve of erathostenes "/	
	int anzahl;	JTIME, K)		
682	anzahl = 0;	Double Click		
684	for (i = 0 ; i <	(= SIZE ; flags[i++] - TRUE) ;	
686	for (i = 0 ; i <	(= SIZE ; i++)		
688	if (flag	gs[i])		
	-			
B::Break.List		had a let an let		
addre	ss types in	p]	Load] 💕 Set	
T:0000	187E Program SC	OFT sieve\4	<u>A</u>	
	<		5	

Drugim rodzajem typu punktów przerwań, które są dostępne w wersji programowej to punkty typu spot. Punkty przerwań spot są swoistego rodzaju punktami obserwacyjnymi, które powodują zatrzymanie programu w konkretnym miejscu na krótki odcinek czasu, w celu zaktualizowania wszystkich wyświetlanych informacji. Po odczytaniu danych wykonywanie programu zostaje wznowione.

Aby ustawić przerwanie typu spot, należy zaznaczyć instrukcję w pliku źródłowym, na której chcemy odświeżyć podglądane informacje oraz z menu kontekstowego wybrać pozycję Spotpoint.



W celu podejrzenia wszystkich zmian zmiennej primz, należy zaznaczyć ją, nacisnąć prawy przycisk myszy i wybrać pozycję Add to Watch Window z menu kontekstowego.



Punkty przerwań w pamięci ROM, Flash i EEPROM

Większość typów procesorów (oprócz 6833x i 6834x) dostarcza kilku sprzętowych punktów przerwań on-chip. Używane są one przez oprogramowanie TRACE32 jako zwykłe punkty przerwań lub typu spot, w momencie gdy aplikacja nie jest uruchomiona w pamięci RAM. Więcej informacji na ten temat znajdziesz w podrozdziale 'Punkty przerwań on-chip'.



Debugger domyślnie używa programowych punktów przerwań, dlatego musisz zaznaczyć, że chciałbyś wykorzystać ich sprzętową wersję!

MAP.BOnchip <zakres adresów>

Komenda MAP.BOnchip sygnalizuje, iż wszystkie ustawione punkty przerwań znajdujące się w podanej przestrzeni adresowej powinny być typu *on-chip*.

W poniższym przykładzie zademonstrujemy sposób ustawienia dwóch punktów przerwań - jednego typu on-chip, a drugiego programowego. Załóżmy, że funkcja func2a znajduje się w nieulotnej pamięci typu ROM, Flash lub EEPROM, a funkcja func2b rezyduje w pamięci RAM. Zgodnie z przedstawioną powyżej zależnością, func2a potrzebuje punktów przerwań typu on-chip, podczas gdy func2b może korzystać z przerwań programowych:

- 6. Pobierz przestrzeń adresową funkcji func2a i func2b (Show Function w menu Var)
- 7. Przypisz funkcji func2a nieulotną pamięć (MAP.BOnchip). Aby sprawdzić mapę pamięci użyj komendy MAP.List.
- 8. Ustaw punkty przerwań w funkcjach func2a i func2b.
- 9. Sprawdź punkty przerwań wywołując polecenie Break.List.



W momencie przekroczenia dopuszczalnej liczby punktów przerwań typu on-chip, zostanie wyświetlony poniższy komunikat. Aby kontynuować pracę, usuń nadmiarowe przerwania.

B::			
too many	onchip	breakpoin	ts set
emulate	trigger	devices	trace
ST:000019F8	\\thumble\	arm\main	

Punkty przerwań na dostępie do danych

Dla większości procesorów, możliwe jest zastosowanie punktów przerwań typu on-chip do zatrzymywania programu w momencie wystąpienia żądania zapisu lub odczytu danych spod określonego adresu. Więcej informacji odnośnie punktów przerwań on-chip Twojego procesora, znajdziesz w podrozdziale 'Punkty przerwań on-chip (zestawienie)'.

W celu zatrzymania wykonywania programu na odczycie wybranej zmiennej, należy zaznaczyć ją kursorem, kliknąć prawy przycisk myszy i wybrać pozycję Read z menu Breakpoint.



Rozpocznij działanie programu wybierając Go. Jeśli program nie osiągnie Twojego punktu przerwania, możesz go zatrzymać za pomocą funkcji Break.

W celu zatrzymania pracy programu na żądaniu zapisu zmiennej, możesz użyć pozycji Breakpoints... z menu Var.

- Klikając przycisk Browse będziesz miał możliwość wybrania interesującej Cię zmiennej spośród wszystkich występujących w aplikacji. Swój wybór potwierdź podwójnym kliknięciem i zaznaczeniem pola Write w oknie Variable Breakpoint Set. Zakończ klikając przycisk OK.
- Alternatywnie, możesz podać nazwę zmiennej lub wyrażenie zapisane w języku typu HLL w polu Expression znajdującym się w oknie Variable Breakpoint Set. Następnie wybierz Write i naciśnij przycisk OK.



Rozpocznij działanie programu wybierając Go. Jeśli program nie osiągnie Twojego punktu przerwania, możesz go zatrzymać za pomocą funkcji Break.

Większość dostępnych procesorów dostarcza bardziej złożonych mechanizmów przerywania i wyzwalania pracy programu. Obsługa tych możliwości przez oprogramowanie TRACE32-ICD zależna jest od typu konkretnego CPU. Więcej informacji na ten temat znajdziesz w dokumentacji 'ICD Target Manual'.

Punkty przerwań on-chip na różnych architekturach procesorów

Lista znajdująca się na następnej stronie stanowi zestawienie możliwości wykorzystania punktów przerwań *on-chip* dla poszczególnych rodzin procesorów.

Używana jest następująca notacja:

- Punkty przerwań on-chip: całkowita dopuszczalna liczba.
- Przerwania instrukcji: liczba punktów przerwań on-chip możliwa do wykorzystania jako punkty przerywające pracę programu w pamięci Flash/EEPROM/ROM.
- Przerwania dostępu do danych: liczba punktów przerwań on-chip możliwa do wykorzystania jako punkty przerwań w momencie czytania lub zapisu z/do poszczególnego adresu w pamięci.
- Przerwania danych: Liczba przerwań danych on-chip, które powodują przerwanie pracy programu w momencie zapisu lub odczytania konkretnej wartości spod ustalonego adresu pamięci.

Pojedynczy adres/zakres adresów/maski bitowe

W niektórych architekturach procesora przerwania typu on-chip mogą obejmować tylko pojedyncze adresy (np. ARM11).

Większość rodzin procesorów pozwala na użycie punktów przerwań typu on-chip dla poszczególnego zakresu adresowego. W zależności od technologii użytej w module debugującym, dostępne są następujące możliwości:

- Jedno przerwania on-chip może obejmować pojedynczy adres lub zakres adresów (np. Tri-Core).
- Dwa przerwania on-chip używane są w celu oznaczenia zakresu adresowego. Jedno przerwania ustawiane jest na adresie początkowym, a drugie na adresie końcowym pożądanego zakresu (np. MPC55xx).

Znaczna ilość architektur dostarcza jedynie maski bitowej w celu założenia przerwania on-chip na zadanym zakresie adresowym. W wypadku tym, zakres adresowy jest zawsze powiększony do najmniejszej maski bitowej zawierającej pożądany zakres. W celu kontrolowania aktualnie przerywanego adresu, zalecane jest użycie komendy Data.View <address>.

	Q. [B::Data.View	/ flags]				×
	breakpoint	address	data	value	synbol	1.1
	H	SD:00007E79	A5	. 8 .	\\arm\Global\sinewave+0x13AD	
	H	SD:00007E7A	3D	'='	\\arm\Global\sinewave+0x13AE	
	H	SD:00007E7B	98	181	\\arm\Global\sinewave+0x13AF	
	н	SD:00007E7C	39	'9'	\\arm\Global\flags	
Punkt nrzenwania zanisu	L H	SD:00007E7D	09	1 1 1	\\arm\Global\flags+0x1	
r dinte precimania Eapiod	H H	SD:00007E7E	44	'D'	\\arm\Global\flags+0x2	
został ustawiony na	L L	SD:00007E7F	01	151	\\arm\Global\flags+0x3	
rmiannai hII a nàrmia Mara'	i ii	SD:00007E80	B1	191	\\arm\Global\flags+0x4	
zmiennej nii o nazwie liags.	H H	SD:00007E81	33	' 3'	\\arm\Global\flags+0x5	
Liżyta architektura procesora	i ii	SD:00007E82	D3	181	\\arm\Global\flags+8x6	
office and interaction a procession	i ii	SD:00007E83	03	161	\\arm\Global\flags+0x7	
umożliwia użycie jedynie	H	SD:00007E84	14	11	\\arm\Global\flags+0x8	
maalii hitawai tatat wiakaza	L L	SD:00007E85	43	'C'	\\arm\Global\flags+Bx9	
maski bilowej, lotez większa	i ii	SD:00007E86	68	111	\\arm\Global\flags+0x0A	=
przestrzeń adresowa	H	SD:00007E87	BØ	181	\\arm\Global\flags+0x0B	
przeotrzen adresena	L L	SD:00007E88	01	161	\\arm\Global\flags+0x0C	
oznaczona jest punktem	H H	SD:00007E89	C7	5	\\arm\Global\flags+0x0D	
przonuonio zoniou	H H	SD:00007E80	99	121	\\arm\Global\flags+0x0E	
przerwania zapisu.	ü	SD:00007E8B	31	'i'	\\arm\Global\flags+0x0F	
	Ĥ Ĥ	SD:00007E8C	75	'u'	\\arm\Global\flags+0x10	
	L L	SD:00007E8D	41	' A'	\\arm\Global\flags+0x11	
	i ii	SD:00007E8E	20	<u></u>	\\arm\Global\flags+0x12	
	H	SD:00007E8F	ØC			
	Ü Ü	SD:00007E90	88	181	\\arm\Global_signalvector	
	Î Î	SD:00007E91	47	'Ğ'	\\arm\Global_signalvector+0x	1
	Î Î	SD:00007E92	03	181	\\arm\Global_signalvector+0x	2 -
			*	-		

Procesory RISC/CISC

Rodzina CPU	Punkty prze- rwań on-chip	Przerwania in- strukcji	Przerwania do- stępu do danych	Przerwania da- nych
68k				
6833x	-	-	-	-
6834x	-	-	-	-
68360	1	1	1	-
68HC12	do 2	do 2 pojedyn-	do 2 pojedyn-	1
68HC12A		czych adresow	czych adresow	
68HC16	-	-	-	-
Andes	0 8	do 8	do 8 zakresów jako maska bito- wa	do 8
ARM7	2 lub 1 (jeśli prze-	do 2 zakresów	do 2 zakresów	2
ARM9	rwania progra- mowe są użyte)	jako maska bi- towa	jako maska bito- wa	
Janus				
ARM10 /	2 16 instrukcji	2 16 pojedyn-	2 16 pojedyn-	-
ARM11	2 16 odczyt/ zapis	czych adresow	czych adresow	

C166CBC	4	do 4	do 4 zapis	do 4 zapis
C166SV2			do 1 odczyt	do 1 odczyt
Cortex-A	2 16 instrukcji 1 16 odczyt/ zapis	2 16 zakre- sów jako maska bitowa	1 16 zakresów jako maska bito- wa	-
Cortex-M	6 instrukcji 4 asynchroniczne (asynchroniczny punkt przerwań zatrzymuje pro- gram po wystą- pieniu przerwa- nia)	6 pojedynczych adresów i do 4 zakresów jako maska bitowa wykorzystująca asynchronicz- ność	do 4 asynchro- nicznych zakre- sów jako maska bitowa	1 (wykorzystuje 2 przerwania asyn- chroniczne)
Cortex-R	2 8 instrukcji 1 8 odczyt/za- pis	2 8 zakresów jako maska bito- wa	1 8 zakresów jako maska bito- wa	-
ColdFire	4 instrukcje, 2 odczyt/zapis	3 pojedyncze adresy, 1 maska bitowa	2 pojedyncze ad- resy lub 2 zakresy	2
eTPU	2	do 2 pojedyn- czych adresów	do 2 zakresów od- czytu/zapisu jako maska bitowa	2 (tylko z przerwa- niami zapisu)
H8S	2	do 2	do 2 zakresów jako maska bito- wa	2
H8SX	4	do 4	do 4 zakresów jako maska bito- wa	1
M32R	4 instrukcje 2 odczyt/zapis	4 pojedyncze adresy	2 pojedyncze ad- resy lub 2 zakresy	2
MCORE	2	2 pojedyncze adresy lub 1 za- kres jako maska bitowa	2 zakresy jako maska bitowa	-
MCS8	2	do 2 pojedyn- czych adresów	do 2 pojedyn- czych adresów (zredukowane do 1 jeśli wykorzysta- ne jest z daną)	1
MCS12 MCS12C	do 3	do 3 pojedyn- czych adresów	do 3 pojedyn- czych adresów	1

MCS12X	4	do 4 pojedyn- czych adresów lub 2 zakresy adresowe	do 4 pojedyn- czych adresów lub 2 zakresy ad- resowe	1
MGT5100	1 instrukcja (brak przerwań on- chip, jeśli użyte są przerwania programowe)	1/0 pojedynczy adres	1 pojedynczy ad- res	-
MIDS32	do 15 instrukcij	do 15 zakrosów	do 15 zakrosów	do 15
MIPS64	do 15 odczyt/za- pis	jako maska bi- towa	jako maska bito- wa	0013
MPC500	4 instrukcje,	4 pojedyncze	2 pojedyncze ad-	2
MPC800	2 odczyt/zapis	adresy lub 2 za- kresy	resy lub 1 zakres	
MPC5200	2 instrukcje (zre- dukowane do 1 jeśli użyte są przerwania pro- gramowe) 2 odczyt/zapis	2/1 2 pojedyncze adresy lub 1 za- kres	2 2 pojedyncze ad- resy lub jeden za- kres	-
MPC55xx	4 instrukcje 2 odczyt/zapis	4 pojedyncze adresy lub 2 za- kresy	2 pojedyncze ad- resy lub 1 zakres	-
MPC74xx MPC86xx	1 instrukcja (brak przerwań <i>on- chip</i> , jeśli użyte są przerwania programowe) 1 odczyt/zapis	1/0 pojedynczy adres	1 pojedynczy ad- res	-
MPC8240 MPC8245 MPC825x MPC826x (PQ2)	1 instrukcja (brak przerwań <i>on- chip</i> , jeśli użyte są przerwania programowe)	1/0 pojedynczy adres	-	-

MPC8247 MPC8248 MPC827x MPC8280 (PQ27) MPC83xx (PQ2 Pro)	2 instrukcje (zre- dukowane do 1 jeśli użyte są przerwania pro- gramowe) 2 odczyt/zapis	2/1 2 pojedyncze adresy lub 1 za- kres	2 2 pojedyncze ad- resy lub 1 zakres	-
MPC85xx (PQ3)	2 instrukcje (zre- dukowane do 1 jeśli użyte są przerwania pro- gramowe) 2 odczyt/zapis	2/1 2 pojedyncze adresy lub 1 za- kres	2 2 pojedyncze ad- resy lub 1 zakres	-
PPC401 PPC403	2 instrukcje 2 odczyt/zapis	2 pojedyncze adresy lub 2 za- kresy	2 pojedyncze ad- resy lub 2 zakresy	-
PPC405 PPC44x	4 instrukcje 2 odczyt/zapis	4 pojedyncze adresy lub 2 za- kresy	2 pojedyncze ad- resy lub 1 zakres	2
PPC600	1 instrukcja (brak przerwań <i>on- chip</i> , jeśli użyte są przerwania programowe)	1/0 pojedynczy adres	-	-
PPC740 PPC750	1 instrukcja (brak przerwań <i>on- chip</i> , jeśli użyte są przerwania programowe) 1 odczyt/zapis	1/0 pojedynczy zakres	1 pojedynczy ad- res	-
PWR-ficient	2 instrukcje 2 odczyt/zapis	2 pojedyncze adresy lub 1 za- kres	2 pojedyncze ad- resy lub jeden za- kres	-
SH2A ST4A	10	do 10	do 10 zakresów jako maski bitowe	2
SH3	2	do 2	do 2 zakresów jako maski bitowe	-
SH4 ST40	6	do 6	do 6 zakresów jako maski bitowe	2

SH7047	1	do 1	do 1	-
SH7144/45				
SH7058	12	do 12	do 12 zakresów jako maski bitowe	do 12
Super10	do 8	do 8	do 8	8
TriCore	do 8 instrukcji do 4 odczyt/zapis	do 8 pojedyn- czych adresów lub/i do 4 zakre- sów	do 4 pojedyn- czych adresów lub zakresów	-
XC800	4	do 4 do 1 zakresu (potrzebne 2 po- jedyncze)	do 1 pojedyncze- go adresu do od- czytu do 1 pojedyncze- go adresy do od- czytu lub zakresu	-
XSCALE	2 instrukcje 2 odczyt/zapis	2 pojedyncze adresy	2 pojedyncze ad- resy lub 1 zakres jako maska bito- wa	-

Procesory DSP

Rodzina CPU	Punkty prze- rwań on-chip	Przerwania in- strukcji	Przerwania do- stępu do danych	Przerwania da- nych
Blackfin	6 instrukcji	6 pojedynczych	2 pojedyncze ad-	-
	2 odczyt/zapis	zakresy	resy lub i zakres	
CEVA-X	4 instrukcje	4 pojedyncze	4 pojedyncze ad-	2
	4 odczyt/zapis	adresy	resy lub zakresy	
DSP56K				-
56k/56300/	2	2	2	
56800	1	1	1	
56100				

DSP	2	do 2 pojedyn-	do 1 pojedyncze-	-
56300		czych adresow	go adresu	
56800E				
MMDSP	2 instrukcje	2 pojedyncze	1 pojedynczy ad-	1
	1 odczyt/zapis	adresy	res	
OAK	3 instrukcje	3 pojedyncze	1 pojedynczy ad-	1
TeakLite	1 odczyt/zapis	adresy	jako maska bito-	
TeakLite II			wa	
Teak				
StarCore	12	do 12 pojedyn- czych adresów lub do 6 zakre- sów	do 6 pojedyn- czych adresów lub do 3 zakresów	1
STN8810	2	do 2	do 2	1
STN8815				
STN8820				
TeakLite III	2 instrukcje	2 pojedyncze	2 pojedyncze ad-	1
	1 odczyt/zapis	auresy	Tesy lub T Zakies	
TMS320	2	2 pojedyncze	-	-
C28x				
TMS320	2	2 pojedyncze adresy	-	-
C54x				
TMS320 C55x	4	do 4 pojedyn- czych adresów	do 3 danych, 1 zakres i 2 maski bitowe	do 3
TMS320	1	1 pojedynczy	-	-
C62x		adres		
TMS320	do 4	do 4 pojedyn-	-	-
C64x		czych adresów		
TMS320	1	1 pojedynczy	-	-
C67x		adres		
ZSP400	-	-	-	-
ZSP500	4	do 4 pojedyn- czych adresów	do 1 zakresu jako maska bitowa	1

Softcores

Rodzina CPU	Punkty prze- rwań on-chip	Przerwania in- strukcji	Przerwania do- stępu do danych	Przerwania da- nych
Micro-Blaze	0 4 instrukcje 0 4 odczyt/za- pis	0 4 zakresy jako maski bito- we	0 4 zakresy jako maski bitowe	-
NIOS2	0/4/8 (konfiguro- walne)	do 4	do 4 pojedyn- czych adresów lub 2 zakresy	do 4

Rdzenie konfigurowalne

Rodzina CPU	Punkty prze- rwań on-chip	Przerwania in- strukcji	Przerwania do- stępu do danych	Przerwania da- nych
ARC600 ARC700	0/2/4/8 (konfigu- rowalne)	do 0/2/4/8 za- kresów jako ma- ski bitowe	do 0/2/4/8 zakre- sów jako maski bitowe	do 0/1/2/4 (tylko za- pisy, tylko w pełnym trybie)
Diamond Co- res	2	do 2 zakresów jako maski bito- we	do 2 zakresów jako maski bitowe	2

Aby wyświetlić zawartość zmiennych typu HLL, należy wybrać opcję Watch z menu Var.

🐣 TRACE 32	
<u>File Edit View Var Break Run C</u> PU	,
N N L <u>Watch</u>	
View	Wybierz zmienną poprzez podwójne
<u>D</u> ump	kliknięcie jej na liście wszystkich symboli
Breakpoint	/
Delete Breakpoint	
Chau Eurotian	
📑 B::v.aw %m *	
Up Down Sym	nbols Fonctions Variables

ast	(struct struct1) D:0008C4380008C44B
aun	(struct union1) D:0008C7600008C777-
cstr1	(unsigned char [17 D:000812D4000812F4
def	(struct abc) D:0008C3BC0008C3C3
enumvar	(int) D:0008C3CC0008C3CF
flags	(unsigned ເຈົ້າar [19 D:0008C4680008C47A
funcØ	<u>الح</u>

😸 B::v.aw %m `\\diabp5\diabp5\enumvar`	
<pre>`\\diabp5\diabp5\enunvar` = 0</pre>	1 1 1
•	▶ <i>[li</i> ,





Za każdym razem kiedy dodasz nową zmienną do obserwowania, będzie się ona pojawiała na górze okna. Rozszerz okno, aby zobaczyć wszystkie wpisy.

Szybszą metodą podglądu zmiennej jest zaznaczenie jej kursorem w oknie Data.List i naciśnięcie prawego przycisku myszy. Z menu podręcznego Var należy wybrać Add to Watch Window.



Jeśli chcesz wyświetlić bardziej złożona strukturę lub tablicę w oddzielnym oknie, zawsze możesz skorzystać z pozycji View... w menu Var.

W wypadku kiedy potrzebujesz śledzić wartości wszystkich zmiennych występujących w aktualnym kontekście programu, powinieneś użyć pozycji Show Current Vars w menu Var i wykonać kilka pojedynczych kroków.





W większości okien, dostęp do menu podręcznego realizowany jest poprzez kliknięcie prawym przyciskiem myszy. Jeśli zaznaczysz zmienną, otrzymasz dostęp do menu Var, które umożliwi Ci jej podgląd oraz modyfikacje.

W przypadku kiedy chcesz podejrzeć zmienną, lecz nie jesteś dokładnie pewien jej nazwy, zawsze możesz skorzystać z przeglądarki symboli zawierającej bazę wszystkich używanych obiektów.

dow <u>H</u> elp			
List Symbols	B∷sYmbo	::sYmbol.Browse	
	[ok]	name	
	P:00081218	\\diabp5\diab	

Jeśli chcesz zmodyfikować jakąś zmienną, kliknij dwa razy na jej wartość. Odpowiednia komenda Var.Set zostanie automatycznie wpisana w wierszu poleceń. Wprowadź nową wartość i zatwierdź ją enterem.

😹 B::v.v %m `\\diabp5\diabp5\ast`	- 🗆 ×		
<pre>`\\diabp5\diabp5\ast` = (word = 0x0,</pre>			
count = 12345,			
right = 0×0 ,			
field1 = 1,			
field2 = 2)	-		
	<u> </u>		
B::V X (`\\diabp5\diabp5\ast`).count = (`\\diabp5\diabp5\ast`).count = 12345			
[ok] formats (Par>			

Format zmiennych HLL

W celu dostosowania sposobu wyświetlania zmiennych do Twoich potrzeb, należy zaznaczyć nazwę zmiennej, nacisnąć prawy przycisk myszy i wybrać Format... z menu Var.

	<pre>`\\diabp5\diabp5\ast` = (word)</pre>	0.0
		= ИхИ.
Change Variable Format radix radix Decimal Hex BINary Ascii DUMP display	Add to V View in V String SYmbol PDUMP Recursiv DFF Zaznacz pole 'Typ'	Vatch Window Vindow Jalue Memory
Index INherited Type Type Hidden Name Ok Apply	SPaces wyświetlenia zmien E kompletną informac jej typu	nej z :ją odnośnie
<mark>‱ B∷v.v ≵m `\\diabp5\diabp5\diabp5\ast`</mark> (static struct struct1) `\\diabp	5\diabp5\ast' = ((unsigned char *) word = (int) count = 12346, (struct struct1 *) left (struct struct1 *) right (unsigned int:2) field1 (unsigned int:3) field2	= 0x0, = 0x8C438, t = 0x0, = 1, = 2)

Jeśli wyświetlasz bardziej złożone struktury HLL, wybierz pole TREE w ramce Format w oknie Change Variable Format. Czynność ta pozwoli na sformatowanie osobnego sposobu wyświetlania każdego pola struktury poprzez kliknięcie + lub



Kończenie pracy TRACE32

W celu zapisania konfiguracji okien Twojego środowiska TRACE32-ICD, użyj pozycji Store Windows to... w menu Window. Funkcja ta wygeneruje plik skryptowy w języku PRACTICE zawierający wszystkie niezbędne polecenia do przywrócenia zapisanego środowiska. Podaj nazwę pliku w polu File name w oknie Store i naciśnij przycisk Save, aby zapisać ustawienia.



Zapisane ustawienia okien mogą być przywrócone z powrotem, w następnej sesji poprzez wybranie pozycji Load Windows from... w menu Window.

Dodatkowo, język PRACTICE obsługuje modułową strukturę programu, toteż możesz użyć wywołania funkcji w Twoim pliku startowym dokonującej automatycznego przywrócenia konfiguracji.



Aby zakończyć pracę programu wybierz pozycję Exit z menu File.

Zwróć szczególną uwagę na sekwencją włączania / wyłączania systemu:

- Włączanie: debugger układ docelowy
- Wyłączanie: układ docelowy debugger