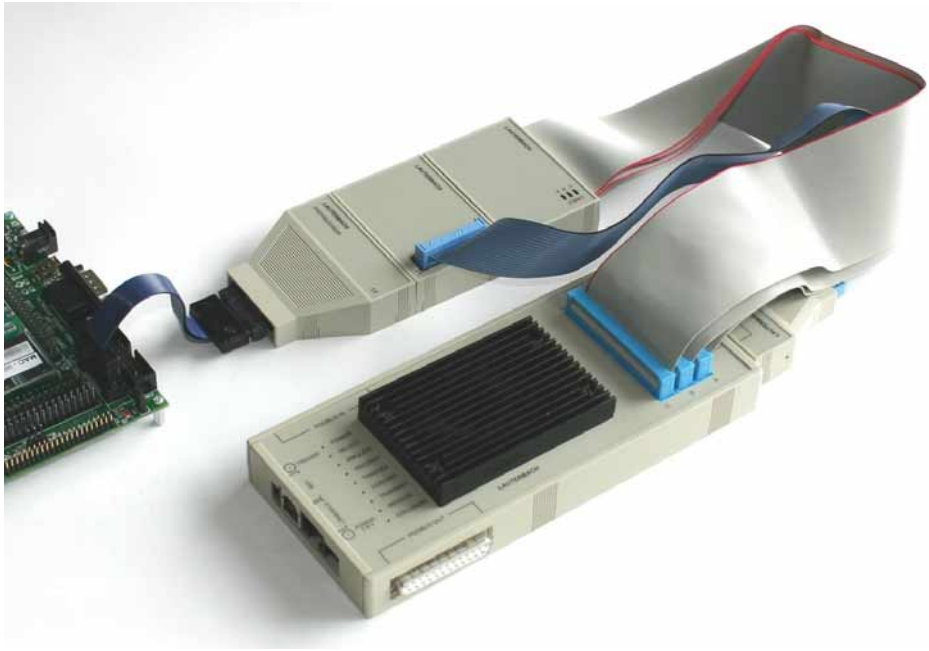
Trace di kernel e processi con MMU dinamica

L'articolo illustra come i debugger Lauterbach possano ricostruire il real-time trace di kernel e processi con gestione dinamica della MMU

Rudolf Dienstbeck
RTOS Integrations
Lauterbach Gmbh Datentechnik

Andrea Provasi
Supporto Tecnico
Lauterbach Italia



Per molti anni l'interfaccia JTAG è stata il mezzo attraverso il quale effettuare il debug di SoC basati su core ARM. In questa metodologia lo sviluppatore ha la possibilità di eseguire il debug del software ad alto livello: è possibile introdurre breakpoint, accedere a memoria e registri, e il processore può essere fermato e fatto ripartire a comando.
Gli stati del sistema sono diventati visibili staticamente; in questo scenario sembra mancare la visibilità dei cicli macchina eseguiti. Con l'introduzione della Embedded Trace Macrocell, comunemente abbreviato in ETM, ciò è stato reso possibile permettendo il tracciamento del programma attraverso una porta dedicata. Lo sviluppatore ha ora la possibilità di vedere una storia dei cicli eseguiti.
La ETM permette di catturare gli indirizzi logici dove il processore ha eseguito del codice, fornendo così la possibilità di tracciare il programma. Questo processo funziona basandosi su di una MMU (Memory Management Unit) statica, dove a ogni indirizzo logico corrisponde un solo indirizzo fisico nella memoria dell'hardware.
Molti dei sistemi operativi moderni però, come Linux o WindowsCE, utilizzano una MMU dinamica in cui la mappa degli indirizzi logici dipende dal processo eseguito in un dato momento; per renderne possibile il debug è quindi necessario implementare nuove funzioni nel debugger.
Questo articolo si propone di illustrare il metodo che permette ai debugger Lauterbach di analizzare il program trace, nonostante il meccanismo degli indirizzi logici dipendenti dal processo. Prima di procedere è necessario analizzare la tecnologia della ETM.

Tecnologia ETM

La cella ETM è un core che affianca la CPU. Il suo scopo principale è monitorare i cicli macchina eseguiti dalla CPU e rendere disponibile, attraverso una porta trace dedicata, la traccia del programma al debugger. Per far ciò la ETM fornisce in uscita solamente informazioni di discontinuità nel flusso di

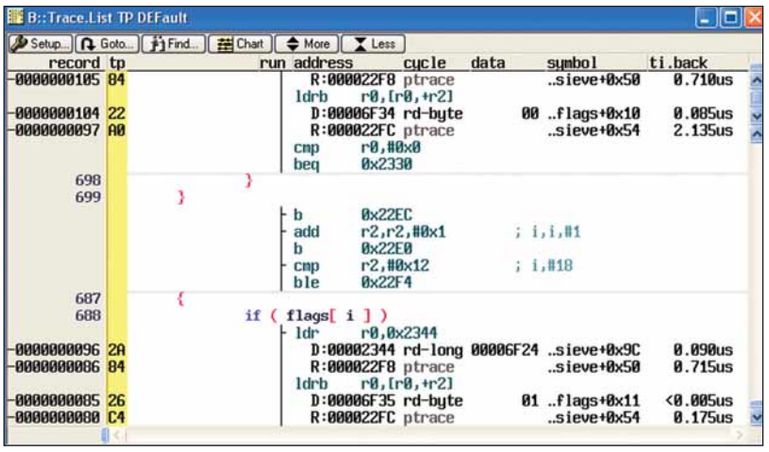


Fig. 1 - L'area evidenziata in giallo sulla sinistra mostra i dati grezzi tracciati così come sono stati catturati dal debugger. Il reale flusso di programma è mostrato sulla destra, così ricostruito dal codice trovato nella memoria del target

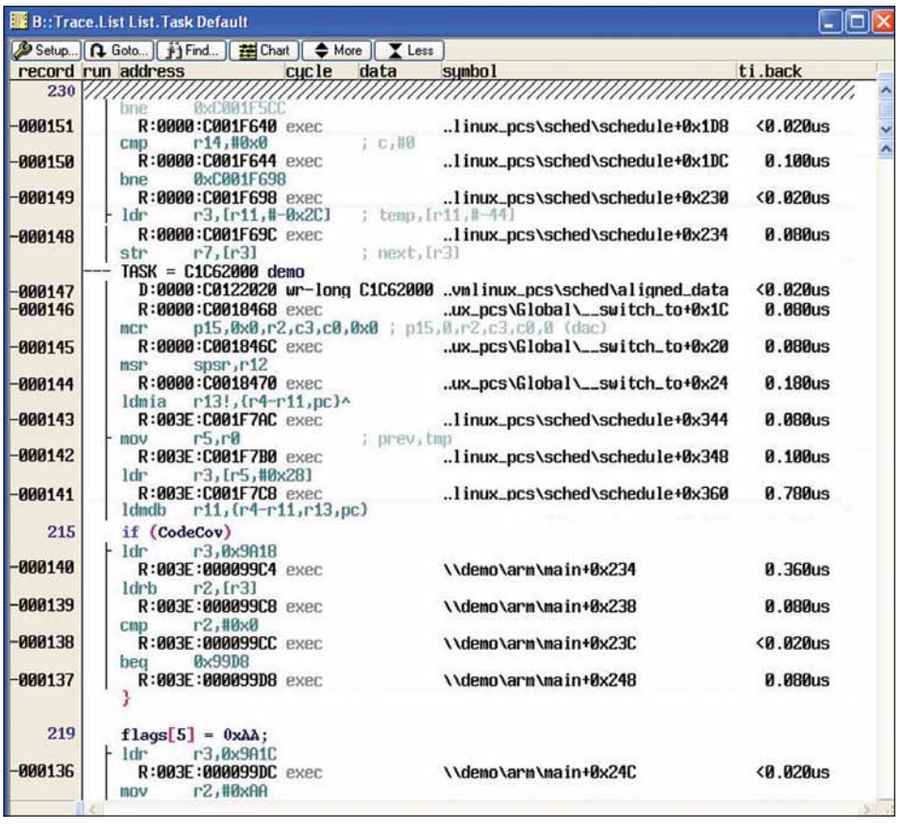


Fig. 2 - In questa immagine si può vedere un tracciamento durante il quale è avvenuto un cambio di processo

programma, come branch o interrupt, che eccedono dall'esecuzione lineare del codice della CPU. Con queste informazioni e la conoscenza di quale codice fosse presente in memoria al momento dell'esecuzione, una traccia completa del programma può essere ricostruita. Per mostrare il flusso completo di istruzioni, il debugger legge le corrispondenti istruzioni nella memoria del target per riempire i gap tra le discontinuità registrate dalla trace port (Fig. 1).

Oltre al flusso di programma, la ETM ha anche la possibilità di fornire informazioni relative all'accesso ai dati. Ciò può causare problemi al tracciamento del flusso di programma dovuto alla grande quantità di informazioni necessarie (per es. un accesso a un dato di tipo long richiede 32-bit di indirizzo e altrettanti di dati).

Troppi accessi ai dati in un breve intervallo di tempo possono causare un "sovraccarico" della porta ETM, che portano alla presenza di gap nel tracciamento del programma.

Dato che la ETM è sempre collegata direttamente alla CPU, ciò che fornisce sono sempre indirizzi logici, cosa appropriata poiché il codice è eseguito proprio in questo spazio di indirizzi, così come variabili, puntatori, stack, e così via. In questo caso non viene data importanza all'indirizzo fisico in cui il processo sta girando, poiché la traccia e la memoria vengono presentate esattamente come viste dal processore.

Cosa accade quando la MMU è riprogrammata durante il run-time causando un cambiamento nella visione della memoria da parte del processore? In questo scenario, alcuni indirizzi logici perdono validità, altri la acquistano e altri ancora si riferiscono a codice diverso rispetto a quello cui si riferivano all'inizio dell'esecuzione. Siccome la ETM mostra solamente informazioni relative a indirizzi logici destinazione

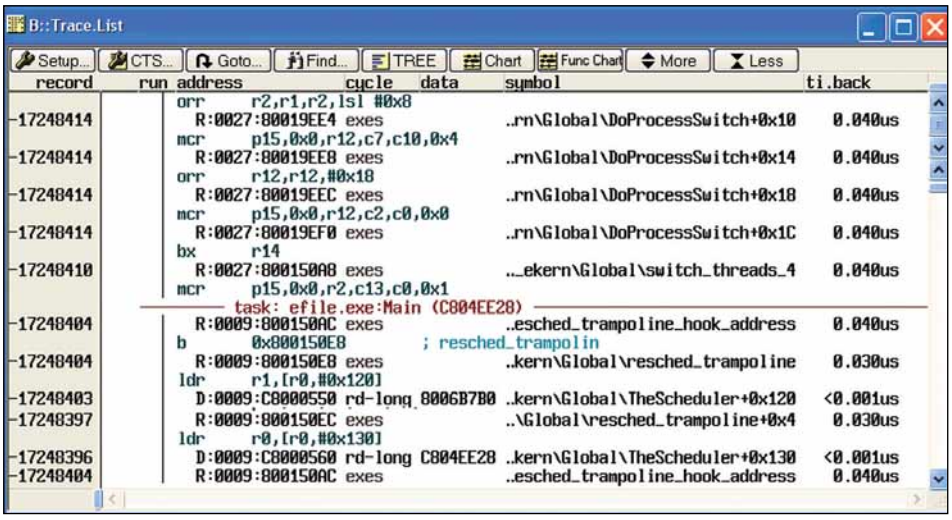


Fig. 3 - Il process ID compare in aggiunta all'indirizzo di destinazione del branch

di istruzioni di tipo branch, e non riferite al codice effettivamente eseguito, diventa ora difficoltoso ricostruire il flusso del programma.

I sistemi operativi che gestiscono dinamicamente la MMU sono tipici esempi di questo scenario; essi includono Windows CE, Linux, QNX, LynxOS e Symbian.

In questi OS ogni processo riceve uno spazio di indirizzi logici a se stante, che ovviamente risiederanno in aree di memoria fisicamente diverse. Dal punto di vista logico però, ogni processo utilizza lo stesso spazio di indirizzi, che solitamente parte dall'indirizzo zero. In questo caso il processore lavorerà sempre nello spazio di indirizzi del processo correntemente attivo, e a un cambio di processo la MMU del processore sarà riprogrammata dal sistema operativo affinché il processore gestisca un processo completamente differente (codice e dati) nello stesso spazio logico di indirizzi.

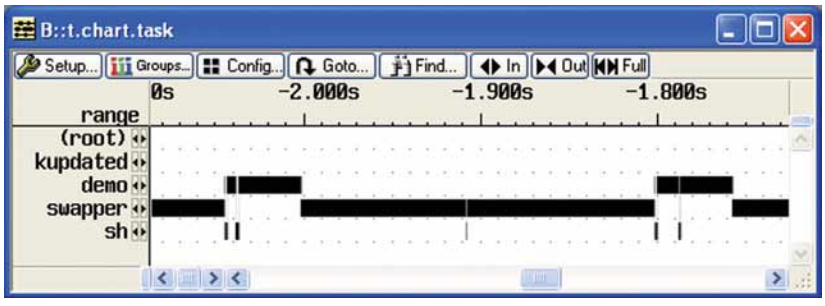
Che accade quindi quando il trace del programma contiene uno (o più) cambi di processo?

Si ipotizzi che il sistema stia dapprima eseguendo il processo "A". In questo caso la ETM registrerà l'esecuzione del programma utilizzando lo spazio logico di indirizzi del processo "A". Si assuma ora che il processo "B" sia visibile alla CPU e venga perciò eseguito. D'ora in poi la ETM registrerà il codice eseguito nello spazio logico di indirizzi del processo "B".

Fermando ora il processore, nel tentativo di ricostruire il flusso di programma si presentano due problemi:

1: dato che i due processi usano lo stesso spazio logico, per entrambi apparirà nel buffer di trace come se eseguissero dallo stesso spazio di indirizzi, sebbene "A" e "B" siano due processi differenti, ognuno con il prorpio codice univoco. Ora è compito del debugger distinguere quali parti del trace fanno riferimento a ciascun processo.

2: come già descritto, la ETM registra solo informazioni riguardo le destinazioni di istruzioni di tipo branch e non il codice effettivamente eseguito; da ciò si evince che il debugger deve essere in grado di accedere al codice



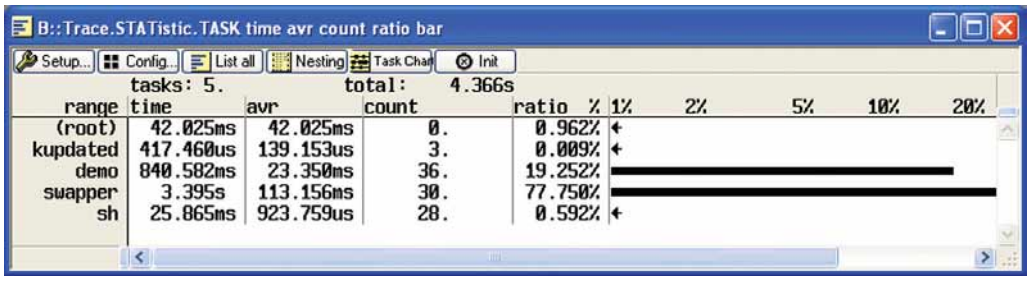
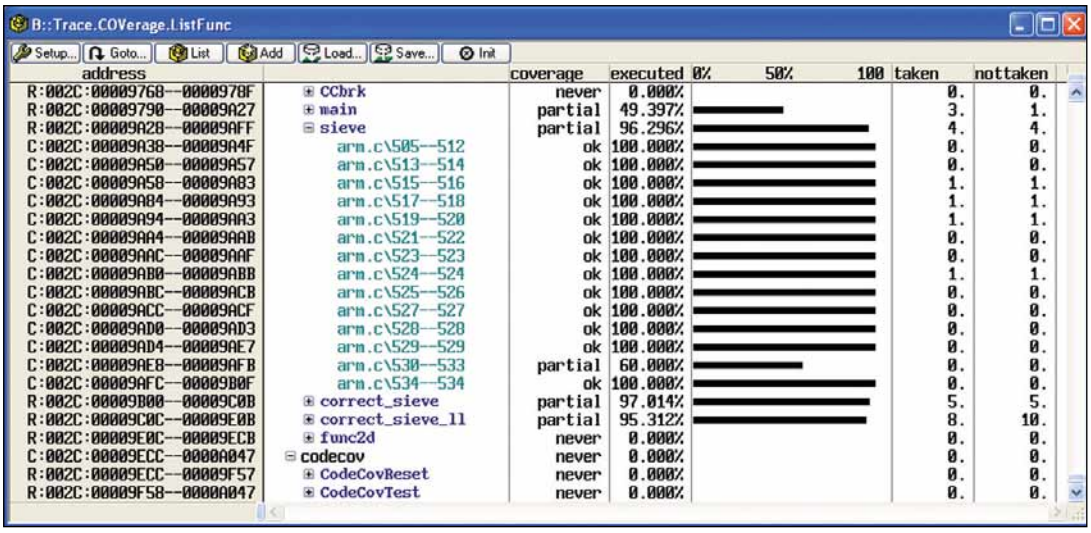


Fig. 4 - Analisi grafica e statistica dei cambi di processo registrati nel trace



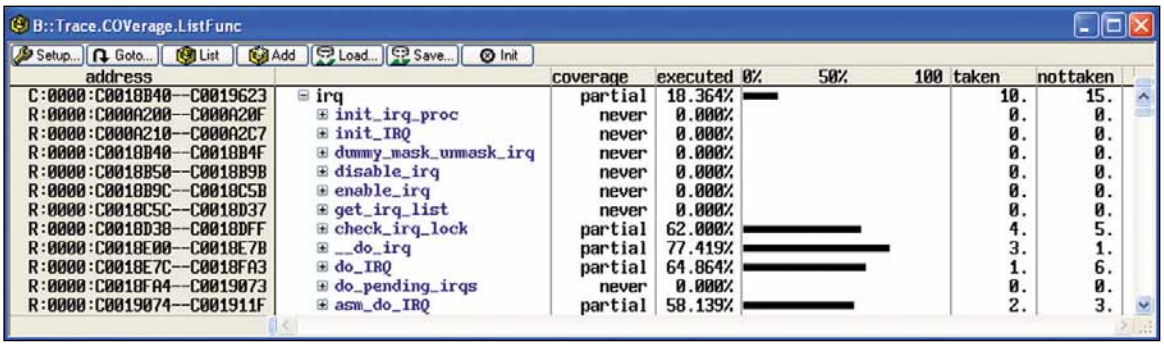


Fig. 5 - Analisi della copertura del codice utilizzato dai processi e dai segmenti del kernel basata sul tracciamento del flusso di programma

eseguito nella memoria attraverso target, affinché sia possibile ricostruire il flusso di programma.
Quando è stato fermato però il processore era nel contesto del processo "B" e non può accedere al codice del processo "A", rendendone impossibile la ricostruzione.
È perciò fondamentale permettere al debugger di accedere al codice del processo "A" pur avendo come contesto attuale nel processo "B".

La soluzione

Innanzitutto è necessario risolvere il primo problema. Per correlare il ramo di trace registrato al codice effettivamente eseguito il debugger deve sapere esattamente quando è avvenuto il cambio di processo. Per far ciò è necessario includere un meccanismo nel tracciamento per catturare i cambi di processo durante l'esecuzione. Tipicamente, ogni sistema operativo dispone di una variabile che memorizza quale processo è attivo in un determinato momento e che viene modificata ogniqualvolta il processo cambia (con un valore relativo al nuovo processo). Registrando gli accessi in scrittura a questa variabile di sistema il debugger può correlare correttamente a quale processo fa capo ciascuna sezione del trace. Nell'intervallo tra due scritture alla suddetta variabile è noto il processo attivo in quel momento e il codice ad esso associato (Fig. 2).
Per avere cognizione dei cambi di contesto, la ETM dovrà quindi essere programmata in modo da registrare gli accessi in scrittura alla succitata variabile oltre al normale flusso di programma. Deve inoltre essere possibile escludere gli altri accessi ai dati non relativi alla gestione dei processi. Registrare troppe informazioni relative ai dati può però portare a eccedere la capacità di trasmissione della porta ETM.
Questo metodo permette perciò di assegnare ciascuna entità di trace a un determinato processo, ma comporta un alcuni svantaggi. Innanzitutto, per poter assegnare un segmento di trace a un processo, è necessario che la variabile di processo sia già stata scritta; da ciò deriva l'impossibilità di associare il primo segmento di trace, dal principio fino alla prima scrittura di tale variabile, poiché non c'è modo di sapere quale fosse il processo in esecuzione precedente il primo cambio di processo. In secondo luogo, deve essere disponibile un filtro di trace, correttamente programmato, che possa essere utilizzato

esclusivamente al solo scopo di registrare gli accessi alla variabile di processo.
Con l'introduzione della versione v1.2 della ETM, sono stati introdotti nuovi meccanismi che permettono di intercettare con una diversa modalità i cambi di processo. La ETM v1.2 include un registro chiamato Context ID, che contiene l'ID del processo in esecuzione.
È compito del sistema operativo aggiornare il registro Context ID a ogni cambio di processo. Il contenuto di questo registro viene registrato assieme alle consuete informazioni di trace (Fig. 3). Grazie a questa informazione aggiuntiva ora il debugger è in grado di determinare l'ID di processo di ciascuna entità di trace, senza bisogno di registrare gli accessi alla corrispondente variabile di processo. Essendo quindi il contesto del codice sempre noto, anche l'inizio del trace può essere associato al codice effettivamente eseguito.
In ogni caso appare ovviamente necessario che il debugger sia informato su come queste informazioni vadano analizzate. Il debugger Lauterbach è in grado di farlo tramite un file in cui è descritta la OS Awareness, un file specifico per ogni sistema operativo che ne descrive le caratteristiche uniche.
Ora che esiste la possibilità di determinare con precisione la relazione tra indirizzi logici e i processi corrispondenti ai vari segmenti trace, si ha tutto ciò che serve per ricostruire la storia completa dell'esecuzione del codice da parte del processore. I pacchetti catturati dalla ETM contengono però solo gli address di destinazione di eventi di tipo discontinuo nel codice eseguito (branch). Per conoscere il codice effettivamente eseguito tra due pacchetti consecutivi è necessario quindi accedere alla memoria sul target.
Accedere a codice del processo corrente non è un problema, poiché la MMU sarà già configurata per permetterne la lettura. Per leggere il codice di altri processi il debugger dovrà bypassare la MMU, poiché quest'ultima non permette accessi ad aree di codice non inerente al processo attivo.
Le informazioni di traduzione degli indirizzi logici in indirizzi fisici sono "nascoste" nel sistema operativo in apposite strutture dati, e a ogni scambio di processi vengono utilizzate per riprogrammare la MMU. Attraverso una ricerca nel sistema operativo, il debugger può creare una tabella di corrispondenza tra indirizzi logici e fisici per tutti i processi. In questo modo non sarà necessario per il debugger cambiare il contesto di esecuzione sul microprocessore per accedere alle aree di memoria di pertinenza di processi non al momento attivi, ma basterà semplicemente disattivare la MMU e accedere direttamente alla memoria fisica relativa al dato processo.
Ora che il debugger ha accesso a ogni parte del codice, compresi i processi attivi e inattivi, la ricostruzione del programma eseguito può essere fatta per intero, anche attraversando parecchi cambi di processo. Con quote informazioni, i debug tool Lauterbach offrono la possibilità di analizzare il tempo di esecuzione dei processi (Fig. 4), l'utilizzo della cache o la code coverage (Fig. 5). In aggiunta, i tool Lauterbach di ultima generazione dispongono di 2 Gigabyte di memoria trace, permettendo di condurre analisi su intervalli di tempo considerevolmente lunghi.

Lauterbach Italia
readerservice.it n. 28