

TOPPERS 活用アイデア・アプリケーション開発 コンテスト

部門 : 活用アイデア部門

作品のタイトル : athrill(アスリル)

作成者 : 森崇((株)永和システムマネジメント)

共同作業 :

対象者 : 車載向け TOPPERS ソフトウェア開発者(V850)

使用する開発成果物 : TOPPERS/ASP3 カーネル

目的・狙い

■目的

V850 実機レス開発環境下で、TOPPERS ソフトウェアを利用できるようにする。

■狙い

フリーの実機レス開発環境として QEMU/ Skyeye 等(主に ARM 系)があるが、V850 の環境は存在しない。V850 は車載系で広く利用されており、TOPPERS ソフトウェアを車載向けでも手軽に利用しやすくすることは有意義と考える。

アイデア/アプリケーションの概要

OSS・V850 実機レス開発環境として、V850 CPU エミュレータ(athrill)を開発する。本エミュレータ機能は以下の通りであり、本環境を使用して ASP3 を移植する。

- ・ CPU 命令エミュレーション
- ・ デバイスエミュレーション(タイマ, シリアル, CAN 等)
- ・ デバッグ機能(ソースレベルデバッグ, 割り込み発生, プロファイラ等)

1. 課題

実機レス開発環境は、高度化・複雑化するソフトウェア開発を行う上で欠かせないものである(図 1).

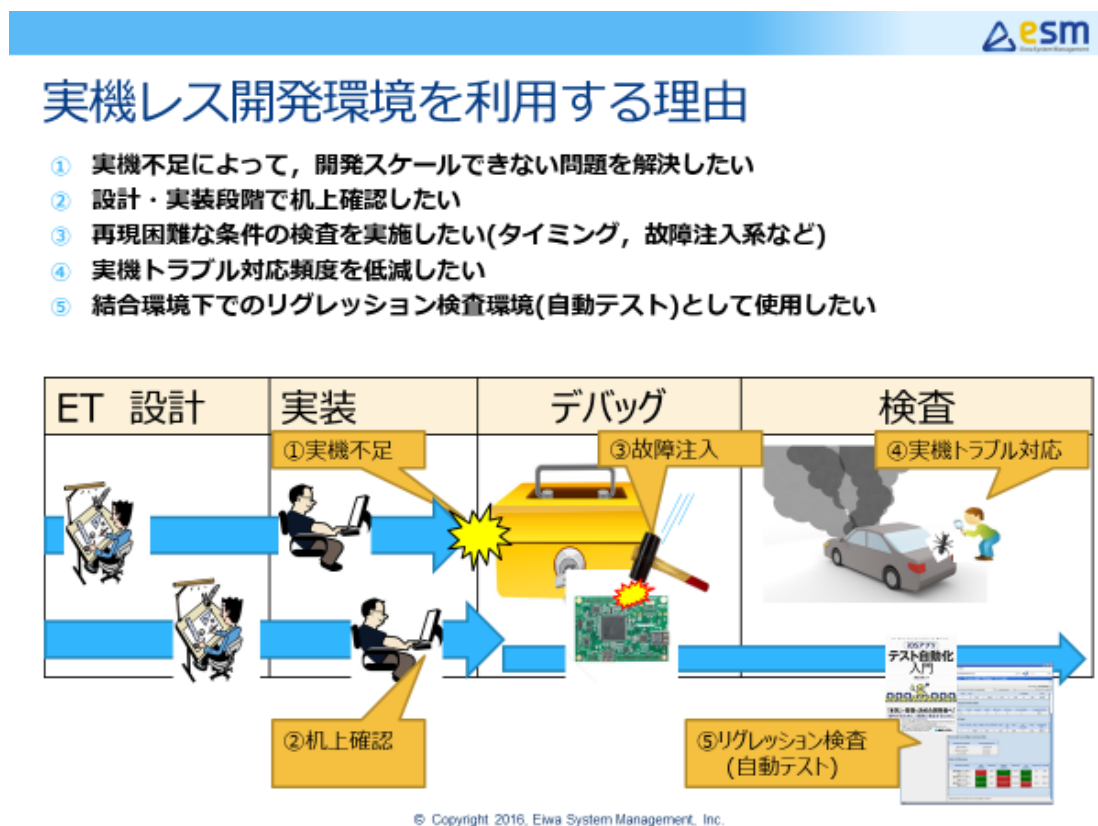


図 1 実機レス開発環境を利用する理由

一方、現市場では、実機レス開発環境を利用する場合、手軽に使える製品/OSS を見つけることは困難である(図 2).

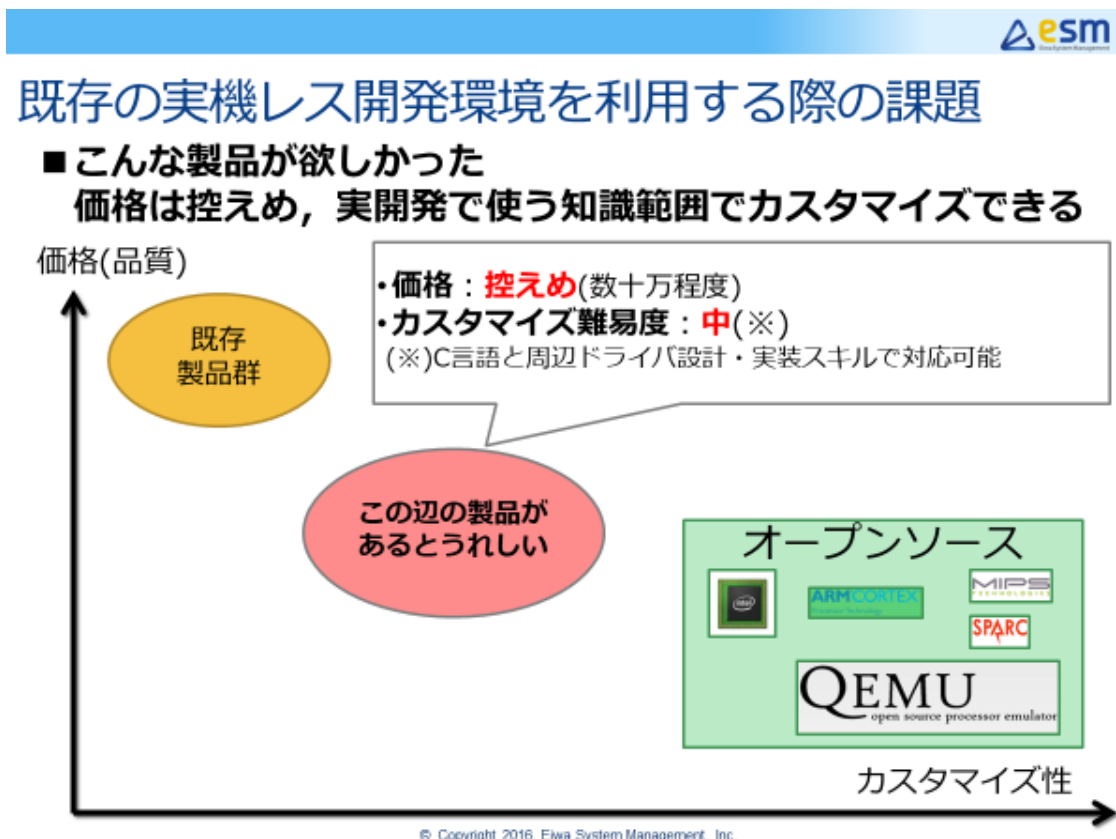


図 2 実機レス開発環境を利用する際の課題

特に、OSS 方面では、ARM 系の実機レス開発環境として QEMU や Skyeye は存在しているが、車載系(V850/RH850 等)の実機レス開発環境は存在しないに等しい状況である。

また、OSS の実機レス開発環境は使用用途に応じて独自改造等は可能であるが、対象ソースは大規模(数百万行)・複雑であり、容易に利用できるというものではない。

2. 提案内容

V850 は車載系で広く利用されており、TOPPERS ソフトウェアを車載向けでも手軽に利用しやすくすることは有意義と考える。そのため、OSS・V850 実機レス開発環境として athrill(アスリル)を開発した。

ここで、実機レス開発環境を有効活用していくためには、単に CPU 命令をエミュレーションするだけでは不十分であり、デバイスエミュレーション/タイミング系の検査支援機能等が揃っていないと実用的ではないと考えている。また、実機レス開発環境の独自カスタマイズを容易にするために、プログラム構成は単純設計にし、ソース規模も爆発しない方がよい。以上の開発要件をまとめたものを図 3 に示す。

要件	内容	詳細
仮想化範囲	CPU	V850
	割込みコントローラ	多重割込み
	デバイス	CAN, タイマ, シリアル
	マルチノード対応	ECU間通信検査用
支援機能	対話型CUI機能	CUIでCPUエミュレータをコントロール
	デバッグ機能	ブレーク, ステップ実行, メモリ参照
	割込み発生機能	多重割込み検査用
	命令実行ログ機能	処理内容および不具合解析用
	プロファイル機能	性能解析用
プログラム構成	設計	単純設計 オブジェクト指向設計
	実装	ソース規模: 2万行程度

図 3 実機レス開発環境の機能要件

athrill の有効性を確認するため、asp3 を V850ES/FK3 向けに移植(※)およびデバッグを実施し、その成果も併せて公開する。

(※) 移植に要する時間は約 5 日程度しか確保できなかったため、asp3 のすべてを移植しきることはできなかったが、タスク/割り込み周りの移植およびデバッグは簡単に実施できた。

3. athrill 設計方針

今回, athrill がサポートするターゲットは V850ES/FK3 であるが, 将来的な拡張を考慮して, 1 ターゲットに限定した作りにはするべきではない. そのため, athrill の基本的な設計方針を以下のように定義した.

設計方針		内容
静的構造		基本的な CPU アーキテクチャをそのままソフトウェア構造に落とし込み, 直感的にわかりやすい機能配置を行う(図 4).
動的構造		デバッグを容易にするため, CPU エミュレータの実行シーケンスは, Main 処理が各機能実行部を順番にトリガする構造とする.
拡張性		将来的に V850 以外の CPU にも対応できるように, ターゲット依存層と非依存層を分離した構造とする (V850 は 1 インスタンスという想定).
エミュレーション範囲	CPU	使用頻度が高い CPU 命令を優先的に選定(図 5).
	デバイス	組み込みソフトウェア開発で利用頻度が高いデバイスを選定(図 6).

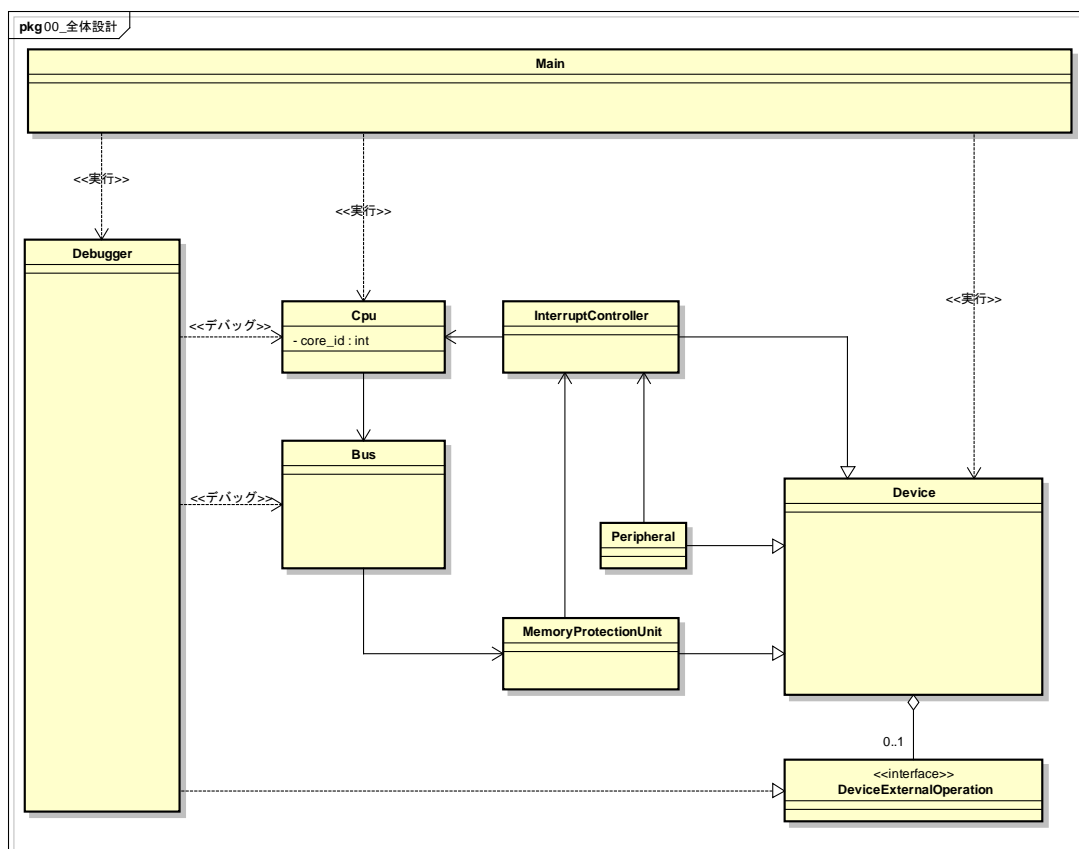


図 4 athrill 全体設計

CPU命令サポート範囲

総命令数80に対して71命令対応 (約88%)

命令	命令数	サポート有無	未サポート命令
ロード命令	10	○	-
ストア命令	6	○	-
乗算命令	4	○	-
算術演算命令	15	△	SASF
飽和演算命令	4	△	SATSUBR
論理演算命令	18	△	BSH, BSW, HSW
分岐命令	4	○	-
ビット操作命令	4	○	-
特殊命令	13	△	CALLT, CTRET
デバッグ機能用命令	2	×	DBRET, DBTRAP

© Copyright 2016, Elwa System Management, Inc.

図 5 athrill CPU 命令サポート範囲

デバイスサポート機能範囲

使用頻度が高い機能を優先的にサポート

デバイス	対応機能	割込み有無
割込みコントローラ	<ul style="list-style-type: none"> ・多重割込み ・割り込みマスク設定 ・割り込み優先度設定 	○
タイマ	<ul style="list-style-type: none"> ・周期タイマ ・インプットキャプチャ 	○
ADC	<ul style="list-style-type: none"> ・連続スキャンモード 	○
CAN	<ul style="list-style-type: none"> ・CAN送信 ・CAN受信 ・拡張フォーマット 	○
シリアル	<ul style="list-style-type: none"> ・シリアル送信 ・シリアル受信 	△ (受信のみ)

© Copyright 2016, Elwa System Management, Inc.

図 6 athrill デバイスサポート範囲

4. athrill 開発環境

athrill(※1)を作成および使用するための環境は以下の通り。

必要とする環境	用途	動作確認バージョン
Windows PC	実行環境	Windows 7, Windows 10
MinGW	コンパイル環境	MINGW32_NT-6.2
コンパイラ(gcc)	athrill コンパイル	5.3.0
ライブラリ	pthread(デバッガが使用)	(MinGW 標準組み込み)
	wsock32(デバッガが使用)	(MinGW 標準組み込み)
クロスコンパイラ(V850)	athrill 動作確認用クロスコンパイル(※2)	v850-elf-gcc.exe (GCC_Build_2.01) 4.9-GNUV850_v14.01
サクラエディタ	ソースデバッグ用	2.2.01

(※1) athrill 公開リポジトリ : <https://github.com/tmori/athrill>

(※2) クロスコンパイラ入手元 : <https://gcc-renesas.com/ja/v850/v850-download-toolchains/>

athrill のインストール手順は以下の通り。

step	手順概要	手順詳細
1	MinGW 起動	<ul style="list-style-type: none"> MinGW の端末を立ち上げる
2	環境変数登録	<ul style="list-style-type: none"> Windows の環境変数 Path に以下を追加する <ul style="list-style-type: none"> <athrill ルートフォルダパス>/src/bin サクラエディタバイナリ配置フォルダパス
3	athrill のビルド	<ul style="list-style-type: none"> athrill のビルドフォルダに移動 <pre>\$ cd <athrill ルートフォルダパス>/src/target/v850esfk3</pre> ビルド実施 <pre>\$ make clean; make</pre>
4	athrill のインストール確認	<ul style="list-style-type: none"> 任意のフォルダに移動し athrill を空打ちする <pre>\$ athrill</pre> 図 7 のメッセージが出力されることを確認する

```

MINGW32~
tmori@Chagall ~
$ athrill
Usage:athrill [OPTION]... <load_file>
-i                : execute on the interaction mode. if -i is not set, execute on the background mode.
-r                : execute on the remote mode, this option is valid on the interaction mode.
-t<timeout>      : set program end time using <timeout> clocks. this option is valid on the background mode.
-p<fifo config file> : set communication path with an another emulator.
-d<device config file> : set device parameter.
tmori@Chagall ~
$
  
```

図 7 athrill インストール確認

5. athrill を使用して asp3 をデバッグ！

athrill を使用して V850ES/FK3 向けに移植した asp3 のデフォルトサンプルアプリケーション(※)をデバッグする様子をお見せする。デバッグ内容は以下の通り。

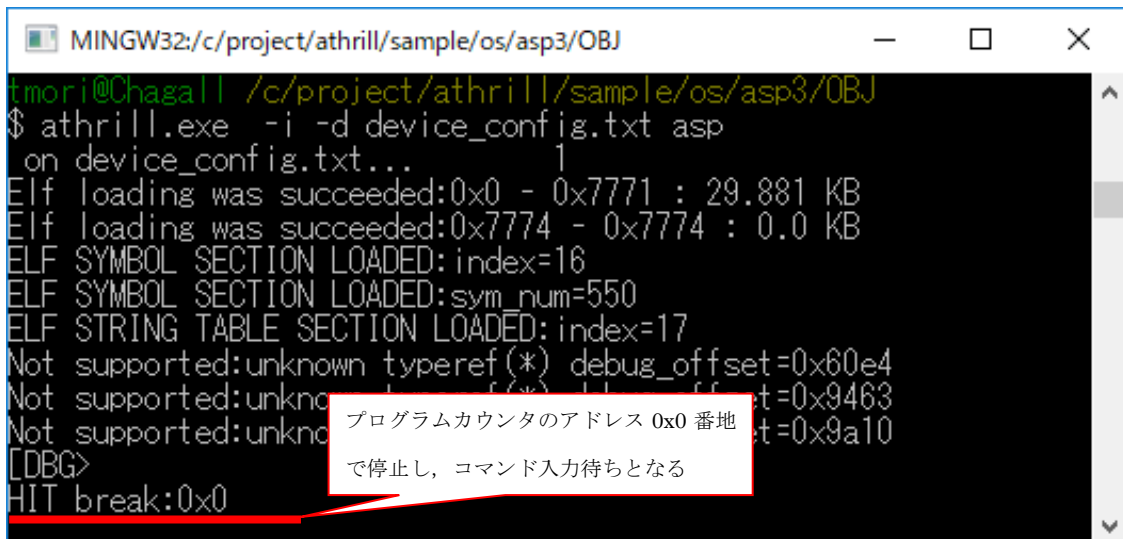
step	デバッグ内容	参照
1	athrill を起動する	5.1
2	asp3 を実行する	5.2
3	TASK1 の特定ポイント(sample1.c 150 行目)にブレークポイント設定	5.3
4	割り込み番号 36 の割り込みを発生させる	5.4
5	命令ログを参照し、TASK1 のブレークポイントから割り込みが発生し、元の TASK1 の処理に戻っていることを確認する	5.5

(※) ビルド済みの asp3 のバイナリ(ファイル名:asp)を以下に配置している(asp3 のサンプルビルド手順でビルド可能)。

V850ES/FK3 向け asp3 : <https://github.com/tmori/athrill/tree/master/sample/os/asp3/OBJ>

5.1. athrill 起動

athrill に asp バイナリファイル(ELF 形式)を渡して起動する。athrill が起動すると、ELF ファイルのローディングメッセージ出力後、プログラムカウンタのアドレス 0x0 番地で停止し、コマンド入力待ちとなる。



```
MINGW32/c/project/athrill/sample/os/asp3/OBJ
tmori@Chagall /c/project/athrill/sample/os/asp3/OBJ
$ athrill.exe -i -d device_config.txt asp
on device_config.txt... 1
Elf loading was succeeded:0x0 - 0x7771 : 29.881 KB
Elf loading was succeeded:0x7774 - 0x7774 : 0.0 KB
ELF SYMBOL SECTION LOADED:index=16
ELF SYMBOL SECTION LOADED:sym_num=550
ELF STRING TABLE SECTION LOADED:index=17
Not supported:unknown typeref(*) debug_offset=0x60e4
Not supported:unknown typeref(*) debug_offset=0x9463
Not supported:unknown typeref(*) debug_offset=0x9a10
[DBG>
HIT break:0x0
```

※オプション説明

※ -i : インタクションモード(デバッグするためのオプション)

※ -d : デバイスコンフィグファイル(athrill のカスタマイズ情報を指定する)

5.2. asp3 実行

athrill のコマンド上で, `continue` コマンド(c)を実行すると, `asp3` の起動メッセージが出力され, `TASK1` のメッセージが連続出力される.

```
MINGW32/c/project/athrill/sample/os/asp3/OBJ
[LNEXT> pc=0x94c kernel_cfg_asm.S 23
c
[CPU>
TOPPERS/ASP3 Kernel Release 3.2.0 for V850-ESFK3 (Aug 12 2017, 17:21:34)
Copyright (C) 2000-2003 by Embedded and Real-Time Systems Laboratory
Toyohashi Univ. of Technology, JAPAN
Copyright (C) 2004-2017 by Embedded and Real-Time Systems Laboratory
Graduate School of Information Science, Nagoya Univ., JAPAN

System logging task is started.
Sample program starts (exinf = 0).
E_OBJ (-1) reported by 'serial_opn_por'.
task1 is running (001).
task1 is running (002).
task1 is running (003).
q
[LNEXT> pc=0x94c sample1.c 152
[DBG>
```

continue コマンド(c)を実行

asp3 の起動メッセージが出力され, TASK1 のメッセージが連続出力される

q コマンドを実行すれば, コマンド入力モードに切り替わる

athrill の実行を中断するには, 任意のタイミングで `q` コマンドを実行すれば, コマンド入力モードに切り替わる.

5.3. ブレークポイント設定

ブレークポイントは, 「関数名」, 「ファイル名と行番のセット」, 「アドレス番地」のいずれかで設定できる. 今回は, `TASK1` の特定ポイント(`sample1.c` 150 行目)に設定し, `continue` すると, 設定したポイントでブレーク発生する.

```
MINGW32/c/project/athrill/sample/os/asp3/OBJ
[DBG>b sample1.c 150
break 0x916
[DBG>c
[CPU>task1 is running (004).
HIT break:0x916 task(+0xa6)
[LNEXT> pc=0x916 sample1.c 150
```

ブレークポイント設定

ブレーク発生

5.4. 割り込み発生

割り込み発生コマンド(i コマンド)を使用すると、任意の割り込みを発生させることができる。今回は、36番の割り込みを発生させ、ステップ実行(n コマンド)する。

```
MINGW32/c:/project/athrill/sample/os/asp3/OBJ
[DBG]>v
VIEW_MODE=ON
[DBG]>i 36
[DBG]>n
[DONE]> pc=0x916 task(+a6) 0x916: MOV imm32(26792),r11(38):26792
[NEXT]> pc=0x2c0 kernel_cfg_asm.S 415
[DBG]>n
[DONE]> pc=0x2c0 null(null) 0x2c0: JR disp22(11588):0x3004
[NEXT]> pc=0x3004 kernel_cfg_asm.S 2302
```

0x916番地の命令実行後、プログラムカウンタが0x2c0番地に移動している。0x2c0番地は36番の割り込み発生時のアドレス番地であり、割り込み発生したことになる。割り込み発生してから、プログラムカウンタがTASK1に戻るまでの命令の流れを記録するため、vコマンドでロギングする。

5.5. 命令ログ確認

命令ログはathrill起動フォルダ直下にlog.txtとしてテキスト出力される。内容を確認すると、vコマンド実行からのログ情報が大量に生成されている(1行1命令)。

```
選択 MINGW32/c:/project/athrill/sample/os/asp3/OBJ
[DONE]> pc=0x916 task(+a6) 0x916: MOV imm32(26792),r11(38):26792
[DONE]> pc=0x2c0 null(null) 0x2c0: JR disp22(11588):0x3004
[DONE]> pc=0x3004 null(null) 0x3004: ADDI imm5(-80),r3(100654988) r3(100654988):100654908
[DONE]> pc=0x3008 null(null) 0x3008: ST.W r10(0xfffffad), disp16(52) r3(0x5ffdf3c):0xfffffad
[DONE]> pc=0x300c null(null) 0x300c: ST.W r11(0x68a8), disp16(48) r3(0x5ffdf3c):0x68a8
[DONE]> pc=0x3010 null(null) 0x3010: ST.W r12(0x20), disp16(44) r3(0x5ffdf3c):0x20
[DONE]> pc=0x3014 null(null) 0x3014: MOVHI imm16(0),r0(0) r10(-83):0
[DONE]> pc=0x3018 null(null) 0x3018: MOVEA imm16(12416),r10(0) r10(0):12416
[DONE]> pc=0x301c null(null) 0x301c: MOV imm5(4),r11(26792)
[DONE]> pc=0x301e null(null) 0x301e: MOV imm32(36),r12(32):36
[DONE]> pc=0x3024 null(null) 0x3024: JR disp22(514):0x3226
```

1行目のログ情報は、TASK1の命令実行ポイントであり、その直後に36番地の割り込み発生し、割り込み処理が実行されていることがわかる。

最終行を確認すると、割り込み復帰命令(RETI)実行後、無事、TASK1 の命令アドレス(0x91c)に戻っていることが確認できた。

```
MINGW32... gcc/athrill/sample/os/asp3/OBJ
[DONE] pc=0x32e4 null(null) 0x32e4: SLD.W disp8(36), r30(0x5ffdf3c), r14(0x0):0x43
[DONE] pc=0x32e6 null(null) 0x32e6: SLD.W disp8(32), r30(0x5ffdf3c), r15(0x25):0x25
[DONE] pc=0x32e8 null(null) 0x32e8: SLD.W disp8(28), r30(0x5ffdf3c), r16(0x64):0x64
[DONE] pc=0x32ea null(null) 0x32ea: SLD.W disp8(24), r30(0x5ffdf3c), r17(0x6c):0x6c
[DONE] pc=0x32ec null(null) 0x32ec: SLD.W disp8(20), r30(0x5ffdf3c), r18(0x73):0x73
[DONE] pc=0x32ee null(null) 0x32ee: SLD.W disp8(16), r30(0x5ffdf3c), r19(0x5ffdf9c):0x5ffdf9c
[DONE] pc=0x32f0 null(null) 0x32f0: SLD.W disp8(8), r30(0x5ffdf3c), r31(0x329c):0x93c
[DONE] pc=0x32f2 null(null) 0x32f2: ADDI imm5(80), r30(100654908) r3(100654908):100654988
[DONE] pc=0x32f6 null(null) 0x32f6: SLD.W disp8(12), r30(0x5ffdf3c), r30(0x5ffdf3c):0x7444
[DONE] pc=0x32f8 null(null) 0x32f8: RETI:0x91c
[DONE] pc=0x91c task(+ac) 0x91c: ST.W r11(0x68a8), disp16(0) r3(0x5ffdf8c):0x68a8
```

今回お見せしたデバッグ内容は **athrill** のほんの一部のデバッグ機能である。この他に、CPU レジスタ参照機能やデータウォッチ機能等、ポーティング作業をする上で有効なデバッグ機能が用意されているので、ぜひ試してみてください(※)。

(※) 詳細は **athrill** マニュアルを参照のこと。

URL : <https://github.com/tmori/athrill/athrill-manual.xlsx>

6. 今後の展望

今回開発した CPU エミュレータに **athrill**(アスリル)という名前を付けたのは、この開発期間中にモノづくりをする楽しさ・苦しさを満喫していたからだ。開発当初は、CPU 命令を C 言語でちまちまと実装していくというのはハードルが高く躊躇していたが、少しずつ形になっていくと、あれもこれもとドンドン出来上がっていき、日々わくわく感(**athrill**)を味わうことができた。

現状の **athrill** は、まだ1ターゲットしか対応していないが、RH850 や ARM 系のサポートもしたいと考えているし、設計上はマルチコア/マルチパーティション対応も視野に入れている。

また、**athrill** は外部ツールと連携することはそれほど難しくはない。GDB スタブ対応すると、GDB が持つデバッグ機能を利用できるようになる。MATLAB/Simulink 連携すれば SILS 環境構築が可能となる。CAN ベースで CANoe 連携することで、自動テスト環境構築も可能であり、この方面の機能拡張はまさに今進めている最中である。

athrill が TOPPERS ソフトウェア普及の一助になれば幸いである。

