

SOLIDのMMU活用アドレスバグ 検出機能を使ってみよう！

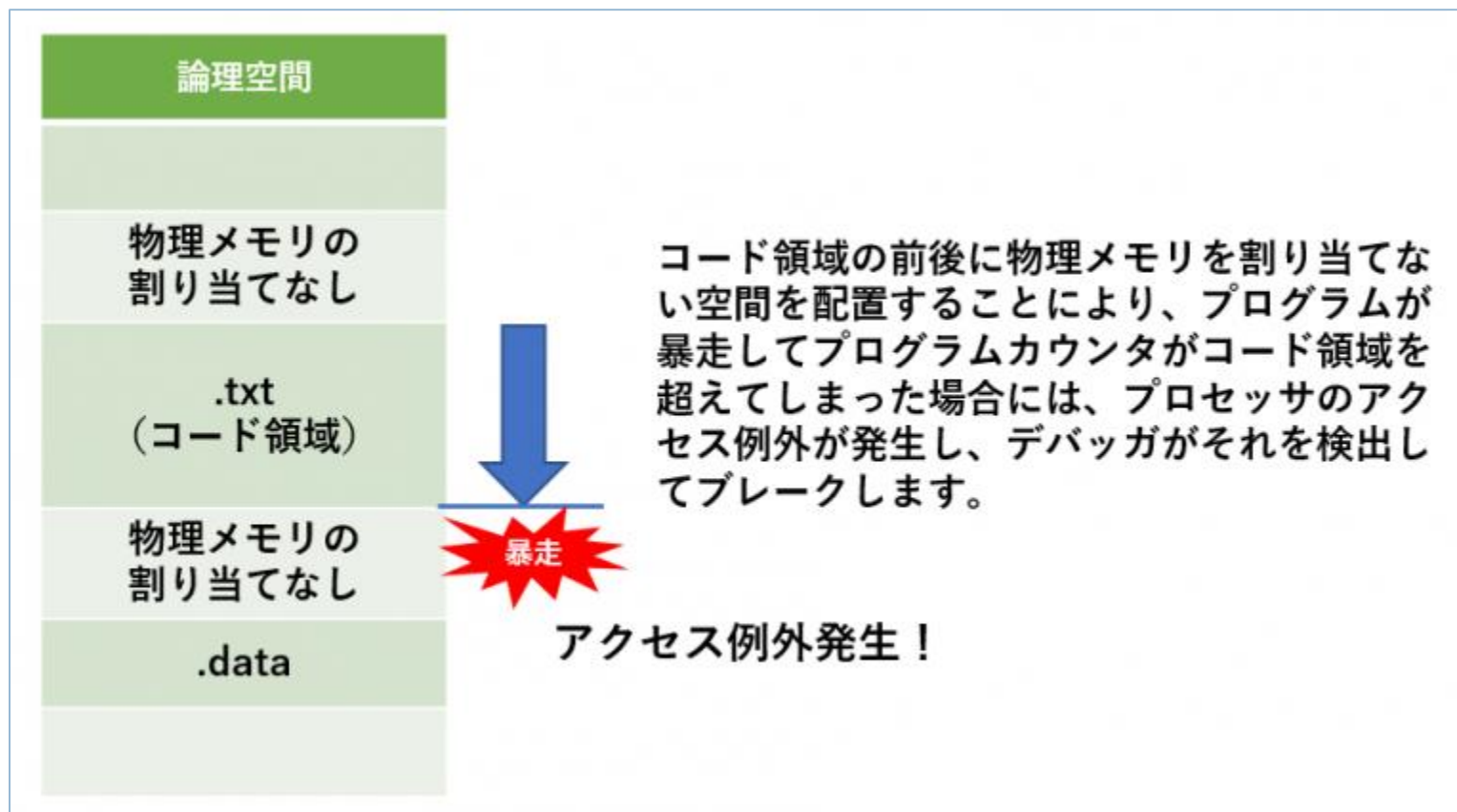
2019.07.08

京都マイクロコンピュータ

MMU活用アドレスバグ 検出機能

何ができるか

不当アクセス・不当実行



不当アクセス・不当実行

- 以下のようなバグも実行時に例外として検出します
 - 実行コード以外の部分をコードとして実行する
 - 実行コード部分のメモリの内容を書き換える
 - 書き換え不可のデータ (C 言語で `const` 宣言されたデータ)があるアドレスの内容を書き換える

スタックオーバーフロー



MMU活用アドレスバグ 検出機能を使うための準備

基本的には何もしなくても有効になります

必要な準備はSOLIDが殆ど済ませています

- 不当アクセス・不当実行の検出については、デフォルトで有効になっています。※準備は不要※
- スタックオーバーフローについては、スタック領域のメモリ割り当てをSOLID-OSに任せることで可能になります。
※タスクの生成時に注意するだけ※

スタックオーバーフロー検出の準備

- 静的生成タスクなら

```
const TINIB _kernel_tinib_table[] = {  
    {(TA_ACT), (intptr_t)0, (TASK)app_init, 4, ROUND_STK_T(STACK_SIZE), NULL},  
}
```

↑
タスク生成パラメータの
6番目のメンバーを NULL または 0 にするだけ
↓

```
const TINIB _kernel_tinib_table[] = {  
    { (TA_ACT), 0, ((TASK)(child_task)), INT_PRIORITY(MID_PRIORITY), ROUND_STK_T(STACK_SIZE), NULL ,  
    (TA_NULL), (tex_routine), CFG_TASK_AFFINITY(1,CFG_PROC_MASK_ANY) },
```


スタックオーバーフロー検出の準備

- 動的生成タスクなら

```
T_CTSK ctsk_main;
```

ASP3

```
ctsk_main.tskatr = TA_ACT;  
ctsk_main.exinf = (VP_INT)0;  
ctsk_main.task = MainTask;  
ctsk_main.ipriority = MAINTASK_PRIORITY;  
ctsk_main.stksz = MAINTASK_STACK_SIZE;  
ctsk_main.stk = 0;
```

```
taskid = acre_tsk(&ctsk_main);
```

```
T_CTSK ctsk;
```

FMP

```
ctsk.tskatr = TA_NULL;  
ctsk.exinf = (intptr_t)(param[i]);  
ctsk.task = child_task;  
ctsk.itskpri = rtsk.tskpri - 1;  
ctsk.stksz = 0x1000;  
ctsk.stk = NULL;  
ctsk.iaffinity = 1;  
ctsk.affinity_mask = CFG_PROC_MASK_ANY;
```

```
taskid = acre_tsk(&ctsk);
```

タスク生成パラメータの
stk (スタック領域のポインタ)に
NULL または 0 を指定するだけ

MMU活用アドレスバグ 検出機能の使い方

実行する際の手順

使い方は通常通り実行するだけ

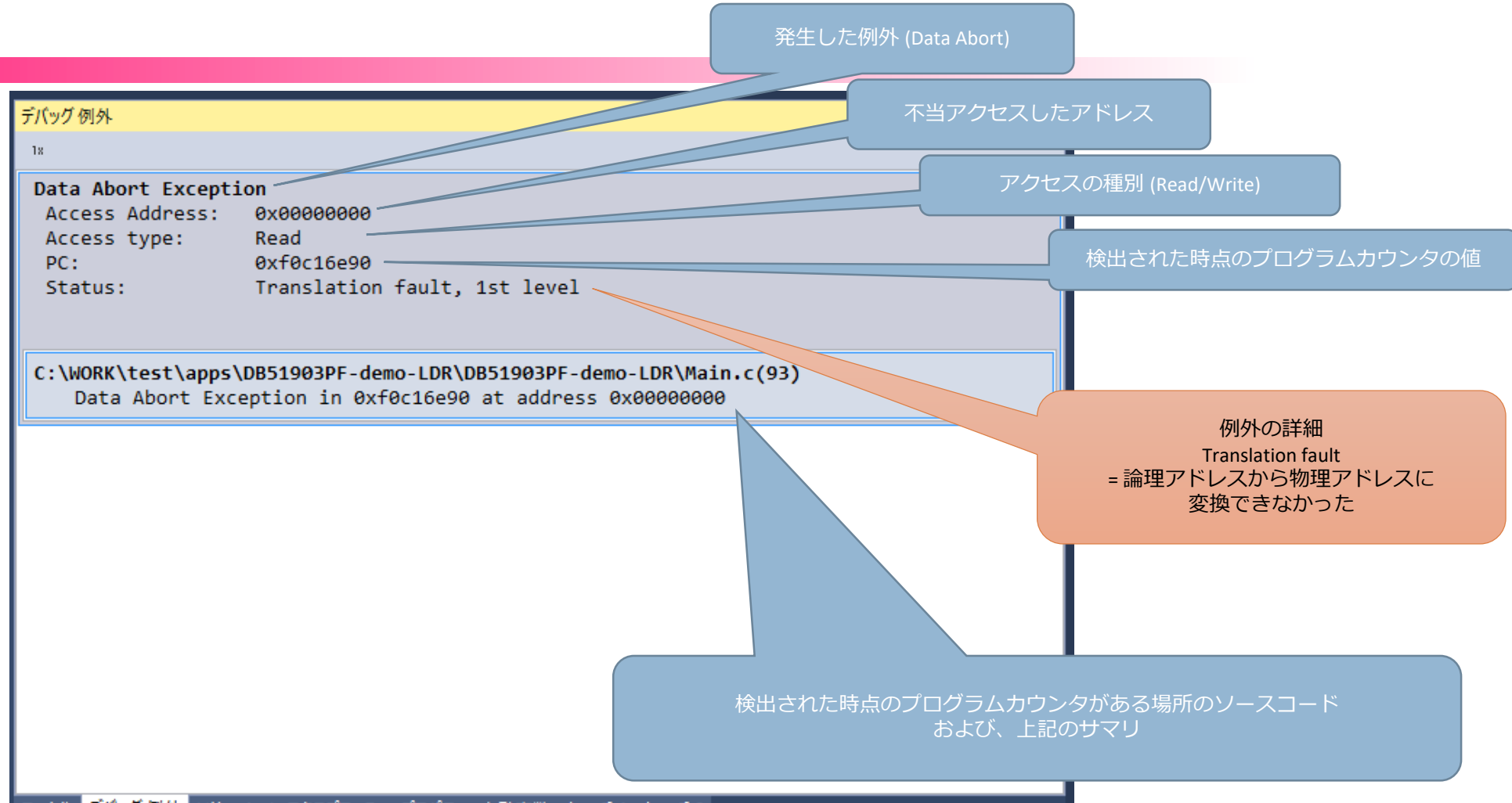
- 特に特別な実行手順は必要ありません。
- 前出の実行時バグが検出されるとその場で停まります。

不当アクセス(実メモリ未割当)の発生時

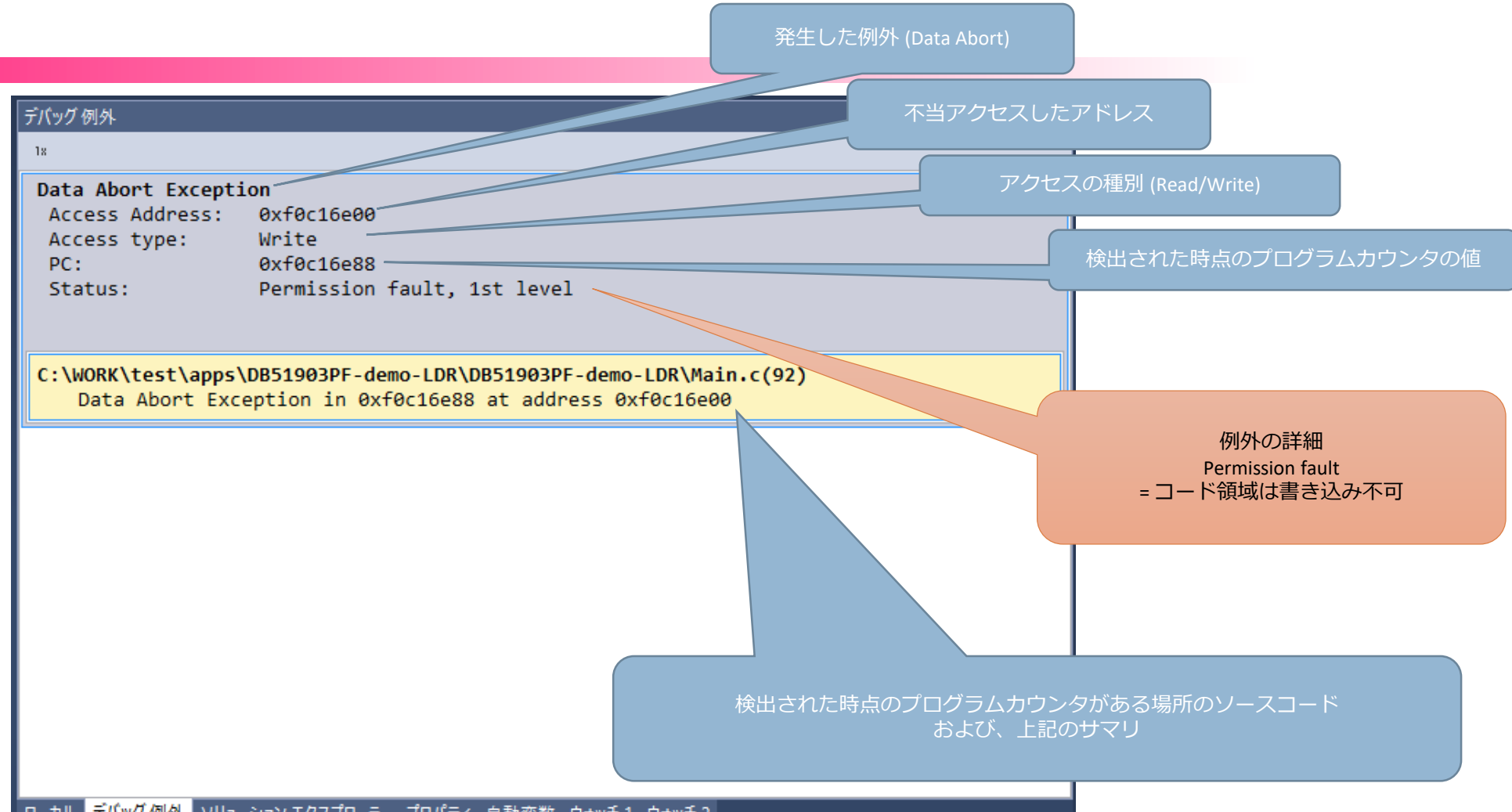
The screenshot displays a debugger interface with the following components:

- メモリ4 (Memory 4):** Shows a memory dump starting at address 0x00000000. The data is mostly unknown (??).
- kernel_cfg.c Main.c memory_map.smm:** The source code editor shows the `root_task()` function. Line 93, `var = *ptr;`, is highlighted in yellow. A blue callout bubble points to this line with the text: "バグが発生したソースコード行" (Source code line where the bug occurred).
- デバッグ例外 (Debug Exception):** A panel on the right shows the exception details:
 - Data Abort Exception**
 - Access Address: 0x00000000
 - Access type: Read
 - PC: 0xf0c16e90
 - Status: Translation fault, 1st levelA blue callout bubble points to this panel with the text: "検出されたバグの情報" (Information about the detected bug).
- Exception Message:** Below the exception details, the message reads: "C:\WORK\test\apps\DB51903PF-demo-LDR\DB51903PF-demo-LDR\Main.c(93) Data Abort Exception in 0xf0c16e90 at address 0x00000000". A grey callout bubble points to this message with the text: "まず、ここをクリックすると、左のように強調表示されます" (First, clicking here will highlight it like on the left).
- RTOSビューアー (RTOS Viewer):** At the bottom, it shows "RTOS: TOPPERS_ASP3".
- 呼び出し履歴 (Call Stack):** The bottom right shows the call stack with the entry: "solid_app_outroot_task(char * ptr) line 93".

不当アクセス(実メモリ未割当)の発生時



不当アクセス(コード領域書換)の発生時



不当アクセス(const宣言変数への書き込み) 発生時

The screenshot displays a debugger window titled "デバッグ 例外" (Debug Exception) showing a "Data Abort Exception". The exception details are as follows:

- Access Address: 0xf0cddba0
- Access type: Write
- PC: 0xf0c16e90
- Status: Permission fault, 1st level

The exception occurred in the source code at `C:\WORK\test\apps\DB51903PF-demo-LDR\DB51903PF-demo-LDR\Main.c(94)`, with the message: "Data Abort Exception in 0xf0c16e90 at address 0xf0cddba0".

Callouts provide further context:

- 発生した例外 (Data Abort)
- 不当アクセスしたアドレス
- アクセスの種別 (Read/Write)
- 検出された時点のプログラムカウンタの値
- 例外の詳細
Permission fault
= const の領域は書き込み不可
- 検出された時点のプログラムカウンタがある場所のソースコード
および、上記のサマリ

The debugger interface includes tabs for "ローカル", "デバッグ 例外", "ソリューション エクスプローラー", "プロパティ", "自動変数", "ウォッチ 1", and "ウォッチ 2".

不当実行(コードではないメモリの実行) 発生時

The screenshot shows a debugger window titled "デバッグ 例外" (Debug Exception) with a list of exceptions. The first exception is highlighted:

```
1x  
Prefetch Abort Exception  
Prefetch Address: 0xf0ceafc0  
Status: Permission fault, 1st level
```

Callouts provide further details:

- 発生した例外 (Prefetch Abort)**: Points to the exception type.
- 不当アクセスしたアドレス**: Points to the Prefetch Address: 0xf0ceafc0.
- 例外の詳細**
Permission fault
= 実行可能コードで無いメモリのデータを
実行しようとした: Points to the Status: Permission fault, 1st level.
- 実行した部分は、コードではないので、ソースコードや例外のサマリは表示されません**: Points to the empty area below the exception details.

The bottom of the window shows a tab bar with "ローカル", "デバッグ 例外", "ソリューションエクスプローラー", "プロパティ", "自動変数", "ウォッチ1", and "ウォッチ2".

スタックオーバーフロー発生時

検出されたバグの情報

バグが発生したソースコード行

まず、ここをクリックすると、左のように強調表示されます

RTOSビューアー

ID	Status	Priority	Entry	PC	StackUsage
6	Ready	7 (BasePriority:7)	kernel_itron4.c@cyclehnd_tsk	0xF0C76764	1032 / 4096
7	Waiting (SEM)	7 (BasePriority:7)	sample_app_msc.c@msc_conn_task	0xF0C123DC	1032 / 4096
8	Waiting (SEM)	7 (BasePriority:7)	Usbh_ReqProcTask	0xF0C7A690	1032 / 4096
9	Ready	7 (BasePriority:7)	Usbh_ReqCmpTask	0xF0C7B730	1032 / 4096
10	Running	3 (BasePriority:3)	child_task	0x100000	1032 / 1024

RTOSビューアーを立ち上げて、各タスクのスタック使用量やタスクのエントリアドレスを確認

スタックオーバーフロー発生時

発生した例外 (Data Abort Exception caused by stack overflow)

スタックオーバーフローした
タスクのタスクID

不当アクセスしたアドレス

アクセスの種別 (Read/Write)

スタック使用量

スタック上限アドレス

スタック下限アドレス

オーバーフロー発生時スタックポインタ

検出された時点のプログラムカウンタの値


検出された時点のプログラムカウンタがある場所のソースコード
および、上記のサマリ

```
デバッグ 例外
1x
Data Abort Exception caused by stack overflow
TID: 10
Access Address: 0xf0b12ffc
Access type: Write
Stack usage: 1032 bytes
Stack top: 0xf0b13000
Stack bottom: 0xf0b13400
SP: 0xf0b12ff8
PC: 0x000100b8

C:\WORK\test\apps\DB51903PF-demo-LDR\handson_app\app1.c(42)
Data Abort Exception caused by stack overflow in 0x000100b8
```

ローカル デバッグ 例外 ソリューション エクスプローラー プロパティ


まとめ



ほぼ無設定で使え オーバーヘッドはありません

- Cortex-A CPUでは
 - キャッシュを使わないと実行スピードが非実用的
 - キャッシュを有効にするには MMU を有効にしないといけない
- MMUを活用したアドレスバグ検出機能は、通常MMUがアドレス変換を行う際に使うテーブルの設定を、SOLIDが自動的に適切に行うことで実現しています。
- そのため、追加のメモリや実行時のオーバーヘッド無く使えます。

以上です



更新履歴

日付	更新内容
2019.07.08	• FMP向け記述の追加
2018.06.08	• 初期リリース