SOLIDのMMU活用のアドレスバグ検出機能を使ってみよう！

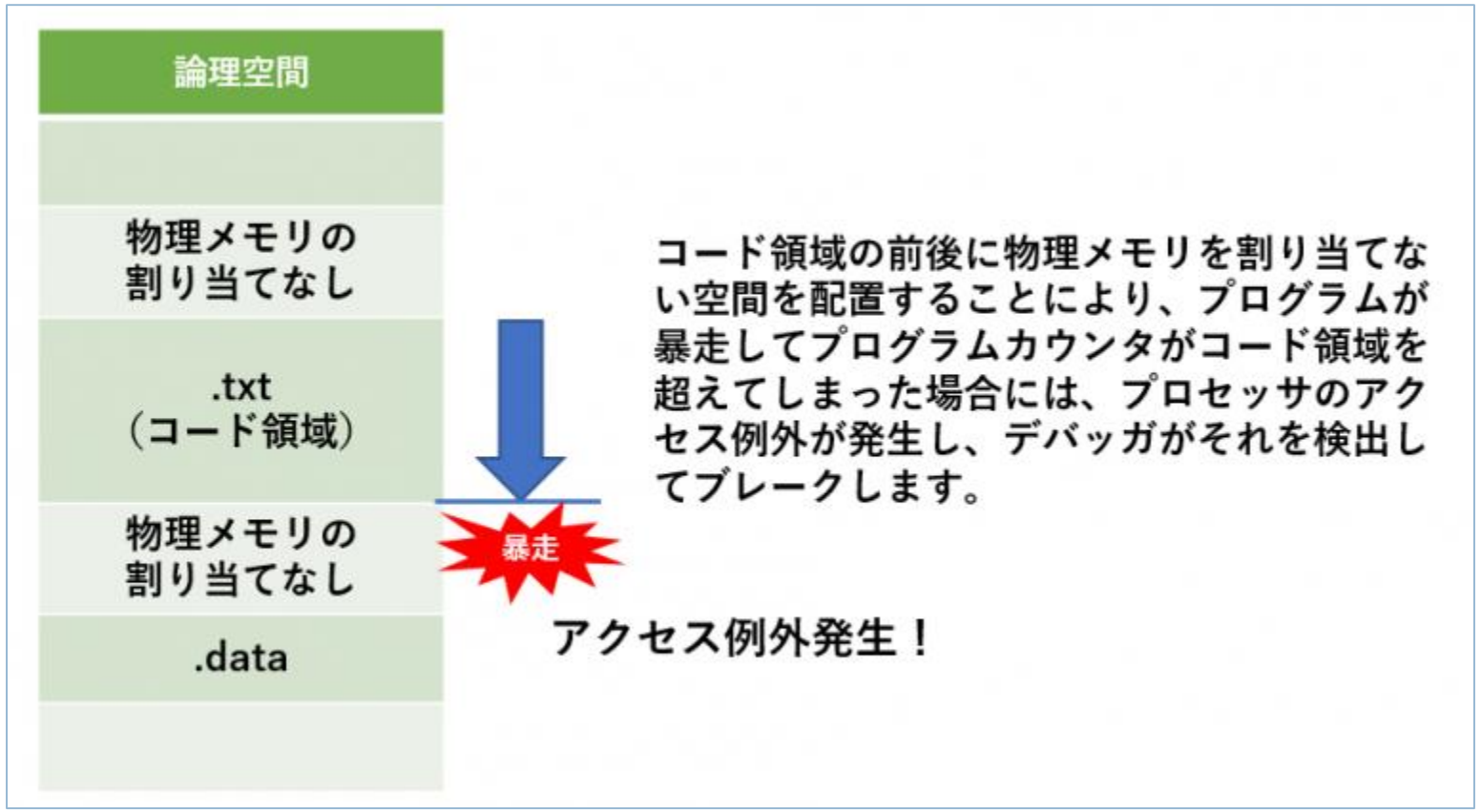
2019.07.08

京都マイクロコンピュータ

MMU活用の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


論理空間
物理メモリの
割り当てなし
.txt
（コード領域）
物理メモリの
割り当てなし
.data
暴走
コード領域の前後に物理メモリを割り当てない空間を配置することにより、プログラムが暴走してプログラムカウンタがコード領域を超えてしまった場合には、プロセッサのアクセス例外が発生し、デバッガがそれを検出してブレークします。
アクセス例外発生！

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活用の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

ASP3

```
T_CTSK ctsk_main;

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);
```

FMP

```
T_CTSK ctsk;

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

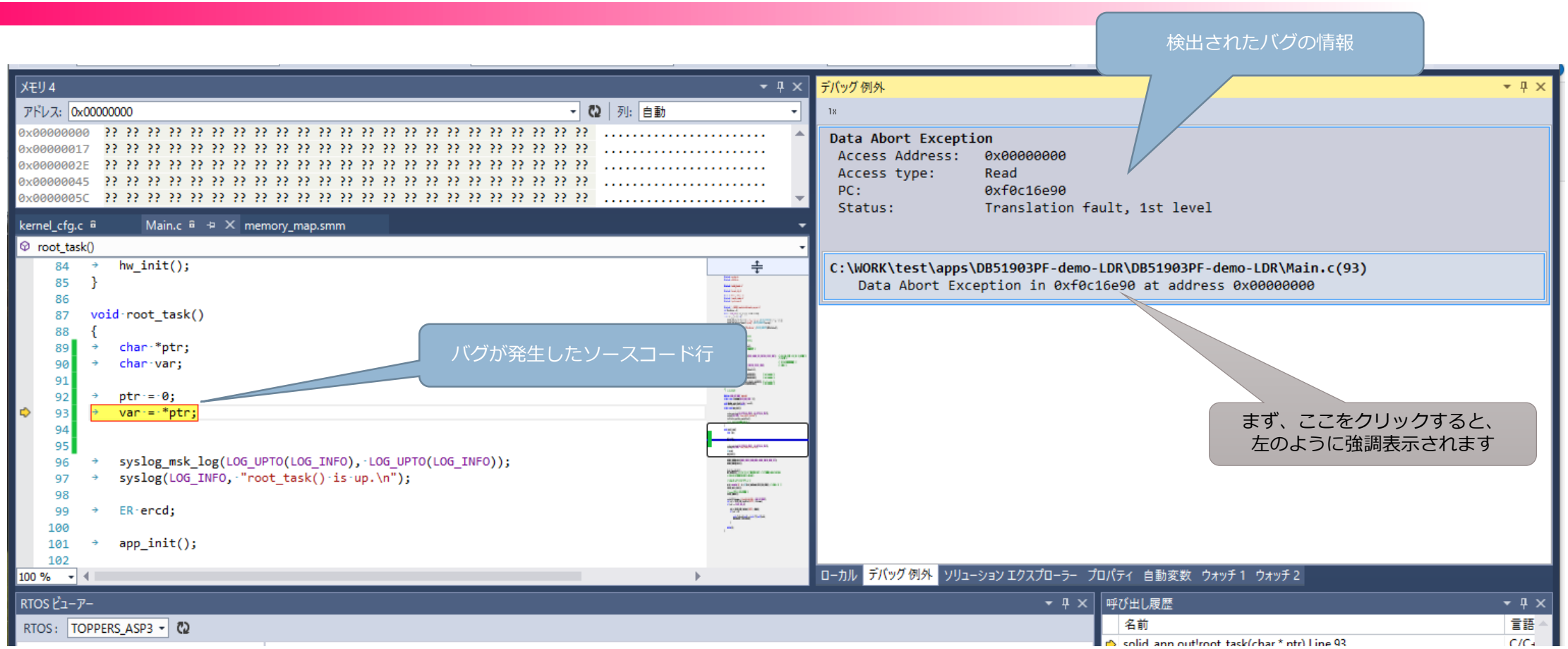
MMU活用のアドレスバグ検出機能の使い方

実行する際の手順

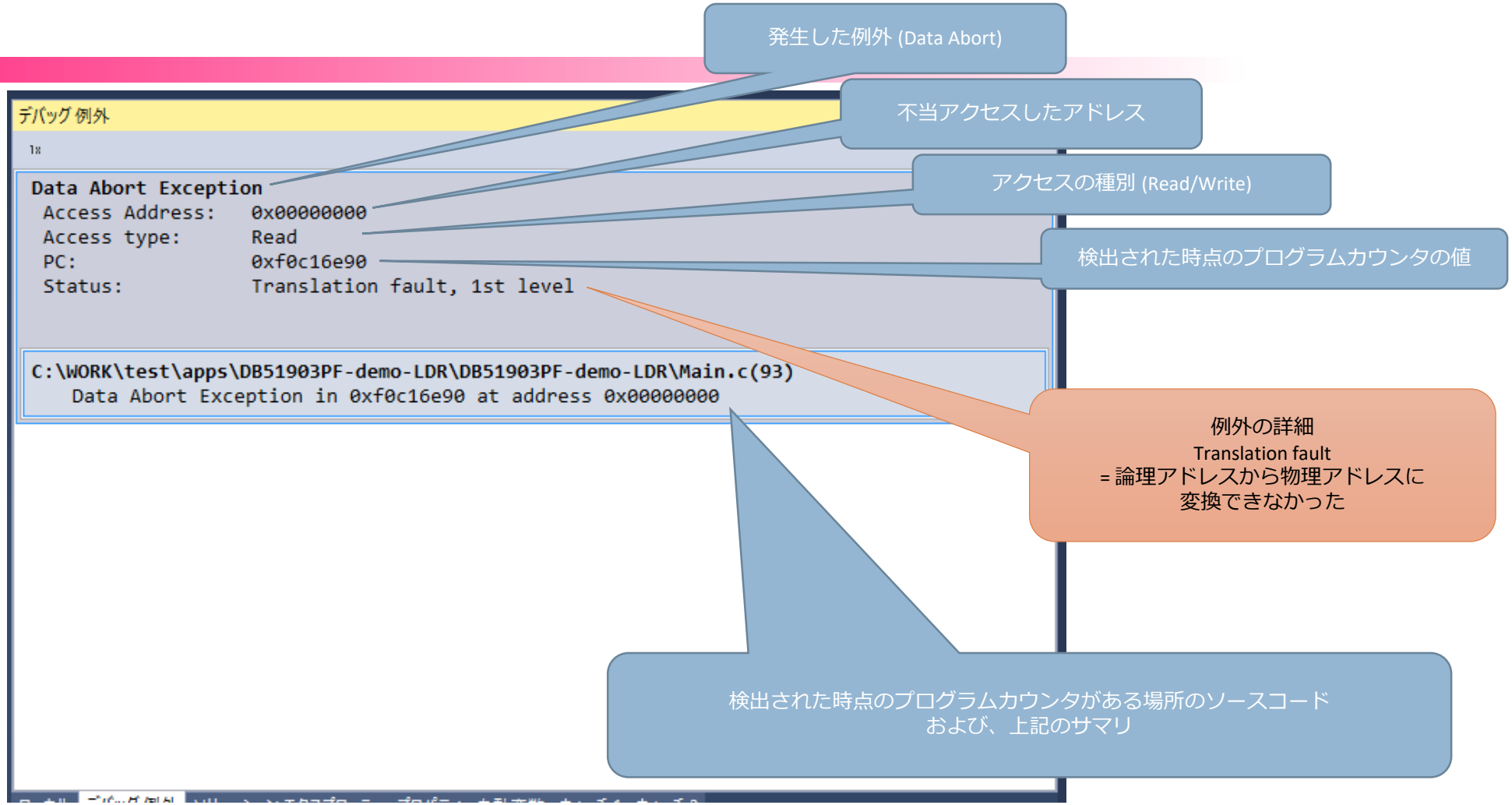
使い方は通常通り実行するだけ

- 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。

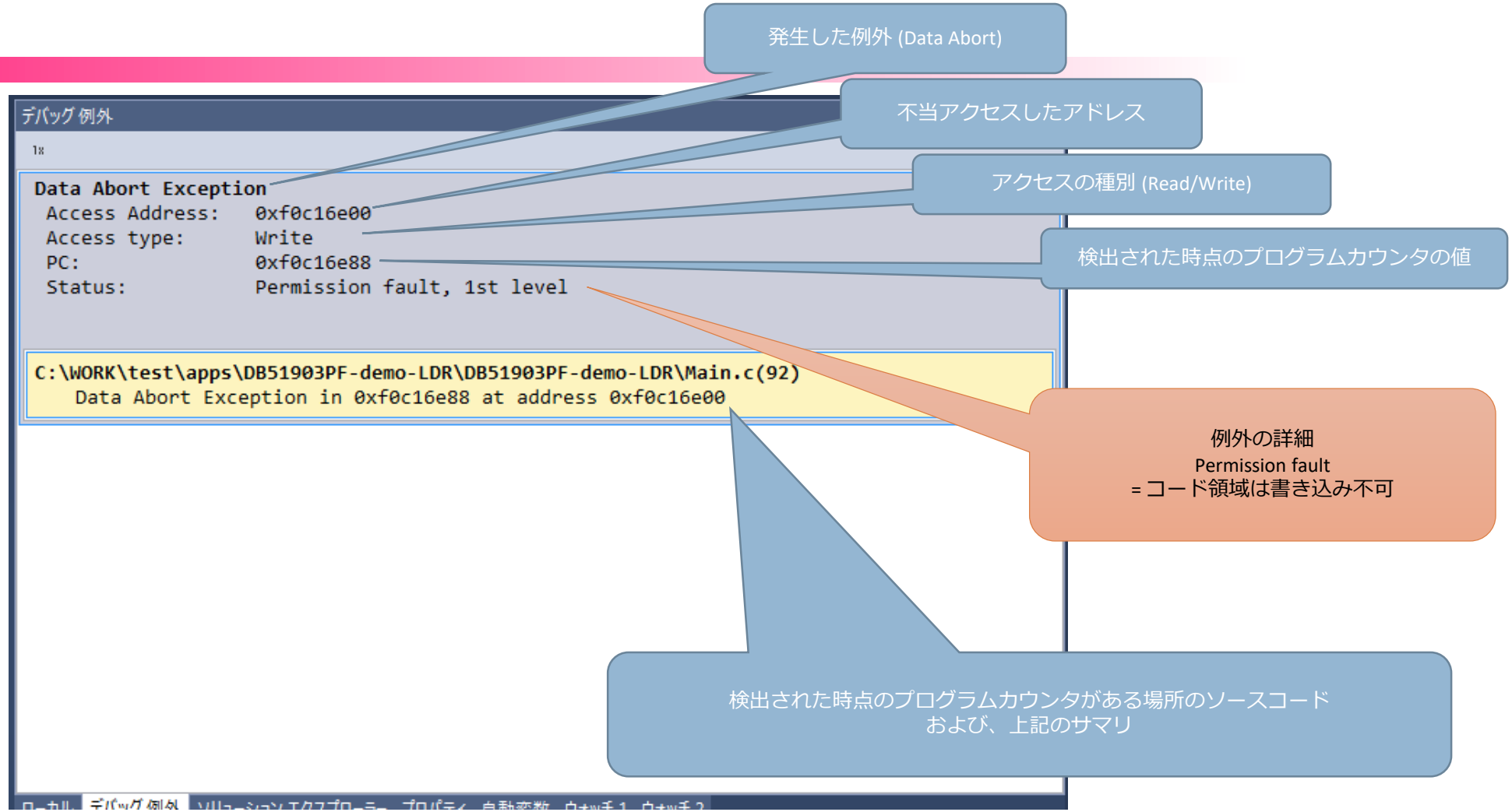
不当アクセス(実メモリ未割当)の発生時



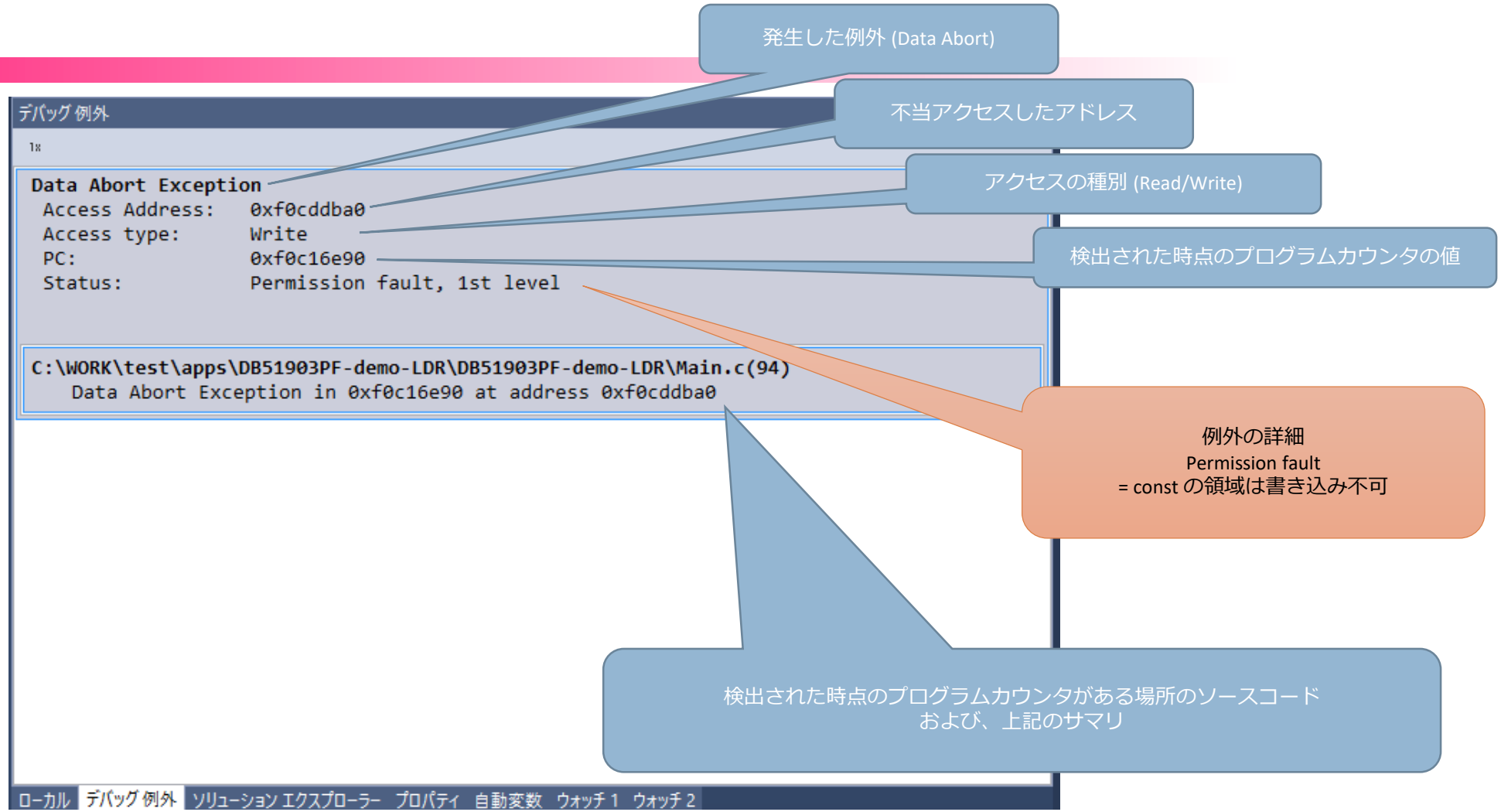
不当アクセス(実メモリ未割当)の発生時



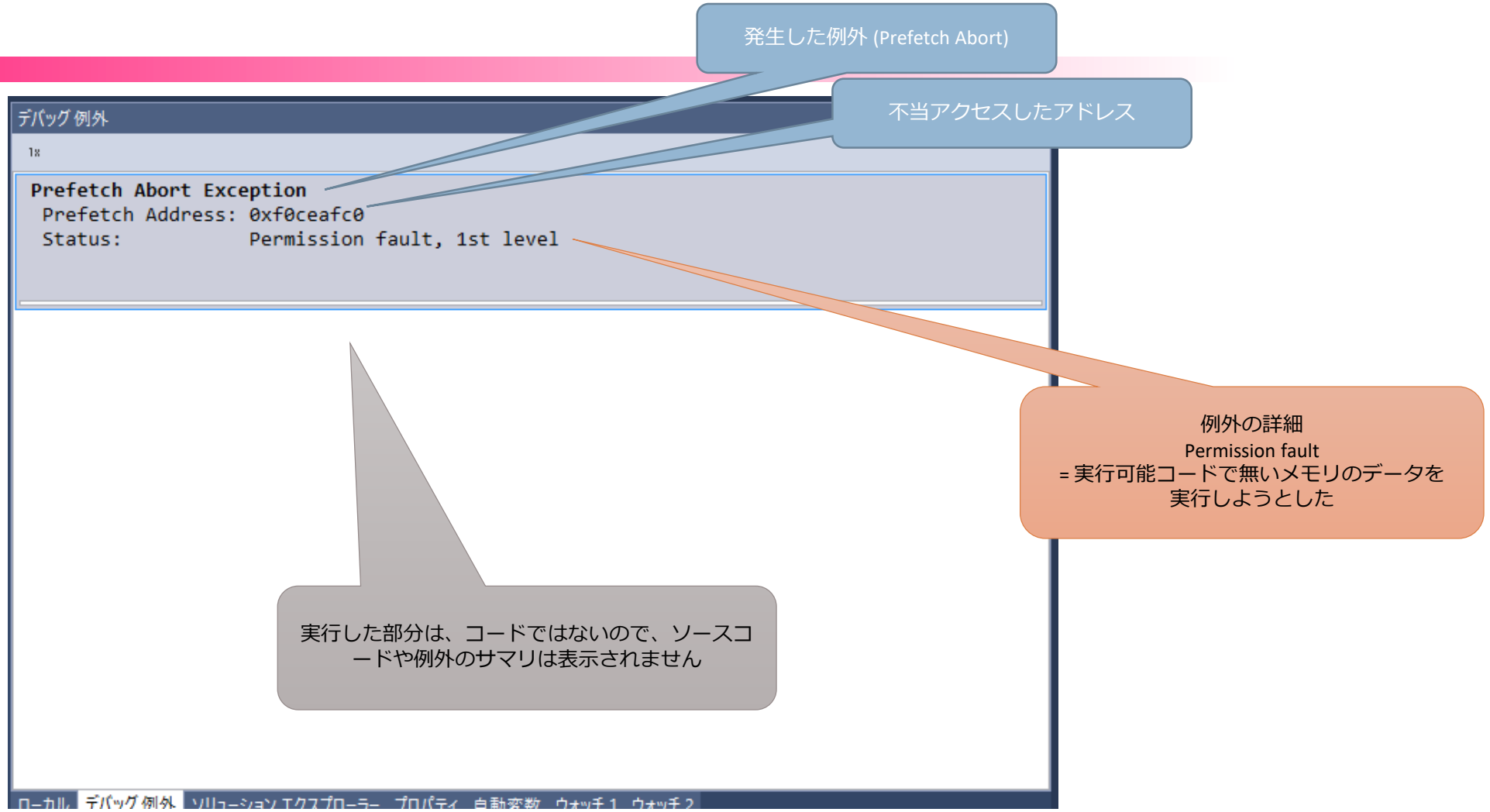
不当アクセス(コード領域書換)の発生時



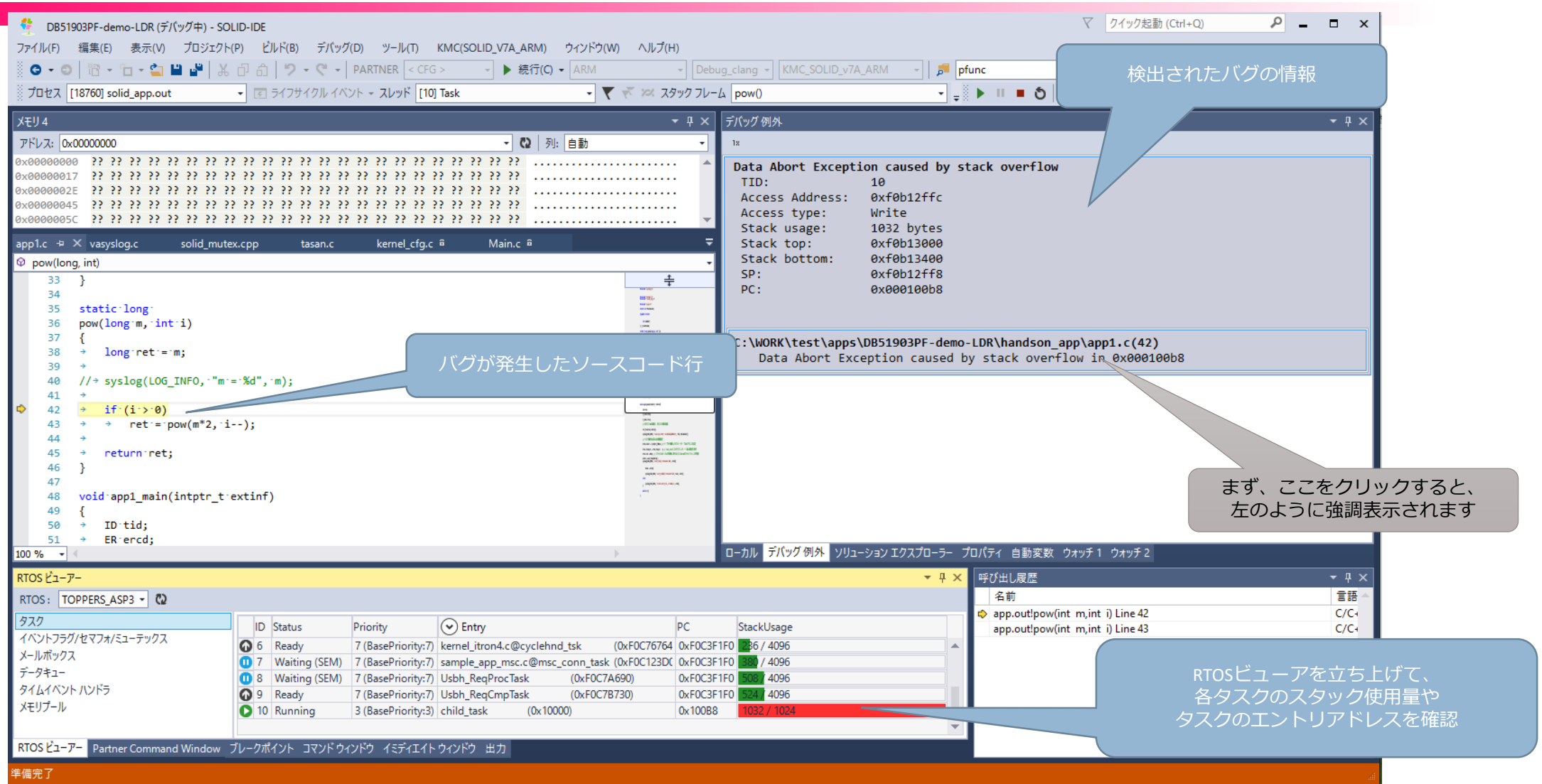
不当アクセス(const宣言変数への書き込み)発生時



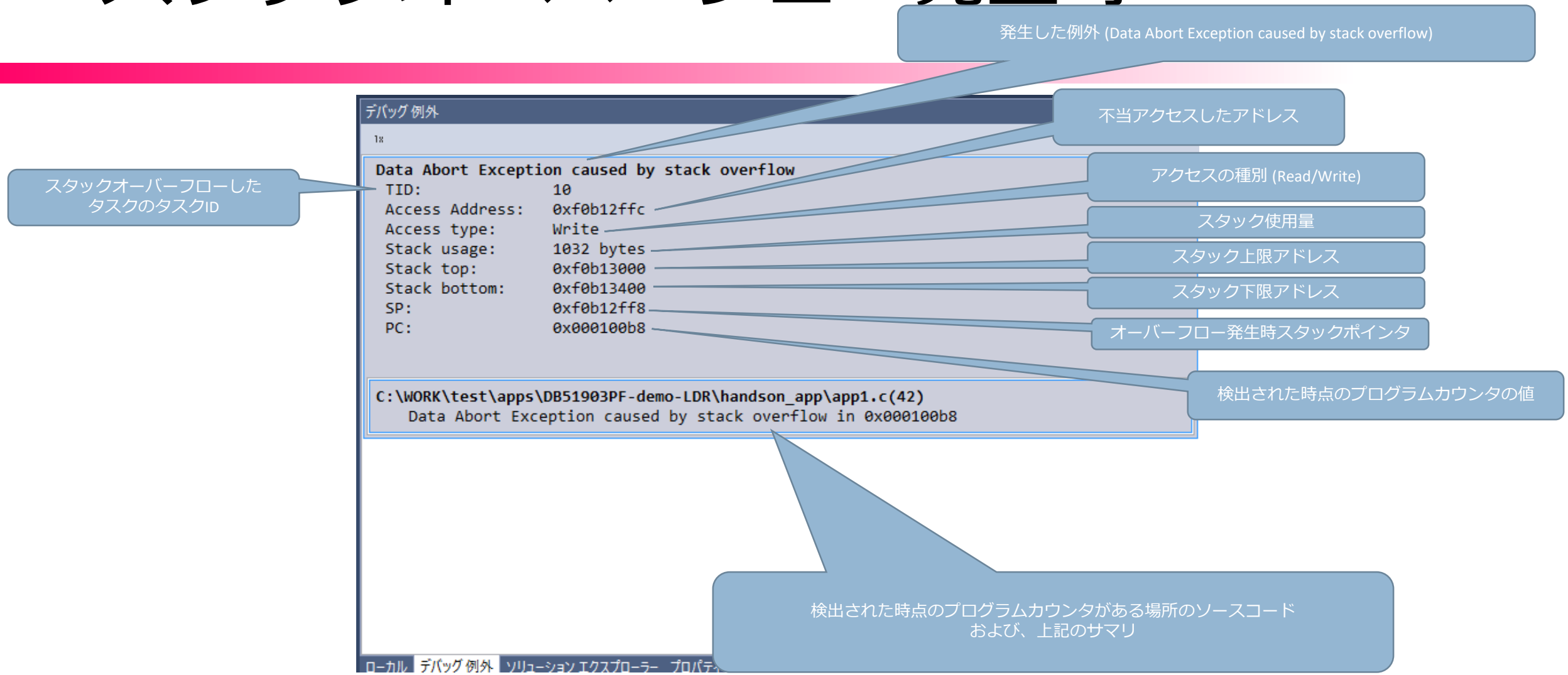
不当実行(コードではないメモリの実行)発生時



スタックオーバーフロー発生時



スタックオーバーフロー発生時



まとめ

ほぼ無設定で使え
オーバーヘッドは有りません

- 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