

# SOLIDのMMU活用アドレスバグ 検出機能を使ってみよう！

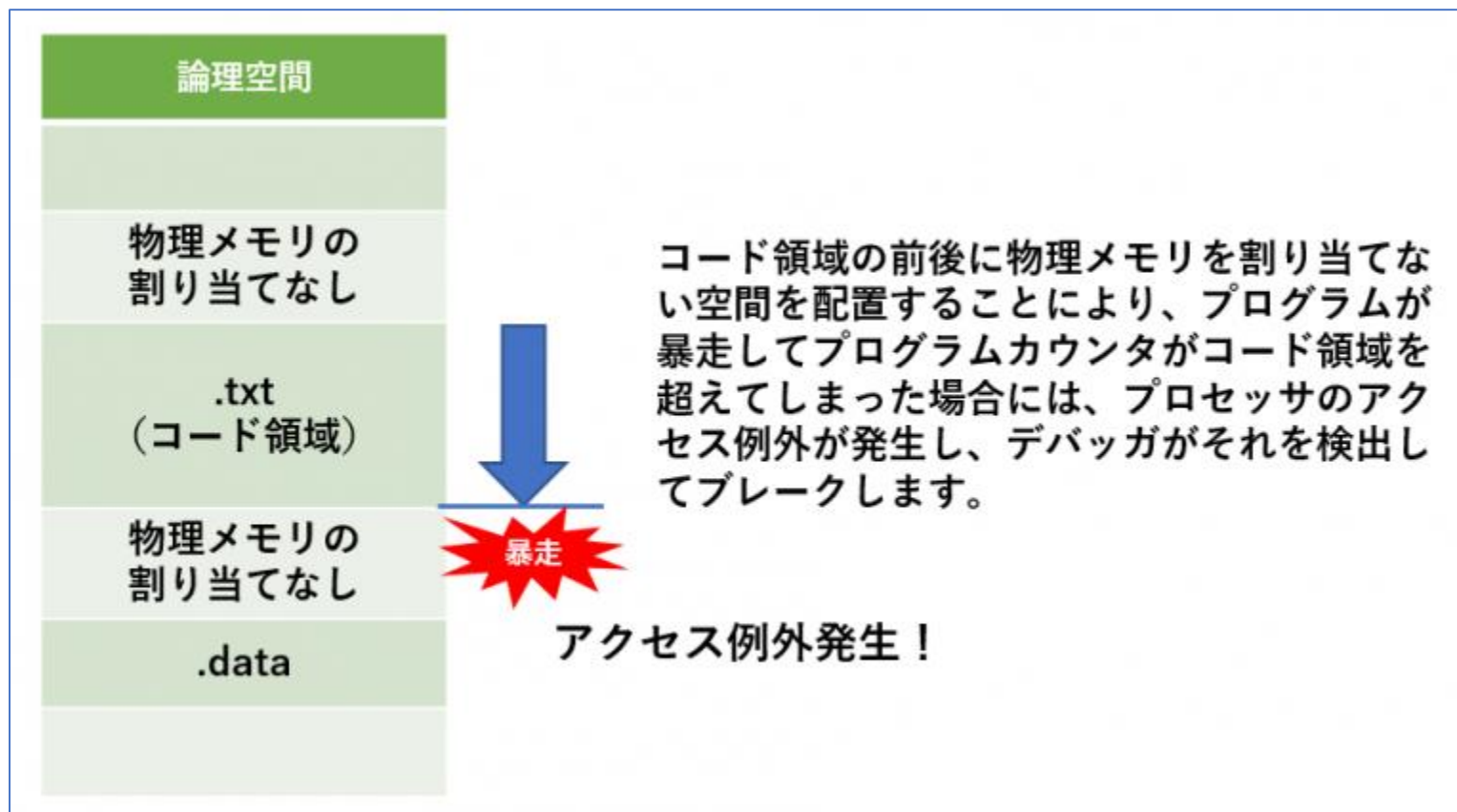
2018.06.07

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

# 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;  
  
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;  
  
tasked = acre_tsk(&ctsk_main);
```

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 several panels. The top-left panel shows memory dump data. The bottom-left panel shows the source code of `root_task()` in `kernel_cfg.c`. The top-right panel shows the exception details for a `Data Abort Exception`. The bottom-right panel shows the call stack.

**検出されたバグの情報** (Information about the detected bug):

**Debug Exception**

- Access Address: 0x00000000
- Access type: Read
- PC: 0xf0c16e90
- Status: Translation fault, 1st level

**Call Stack:**

- C:\WORK\test\apps\DB51903PF-demo-LDR\DB51903PF-demo-LDR\Main.c(93)  
Data Abort Exception in 0xf0c16e90 at address 0x00000000

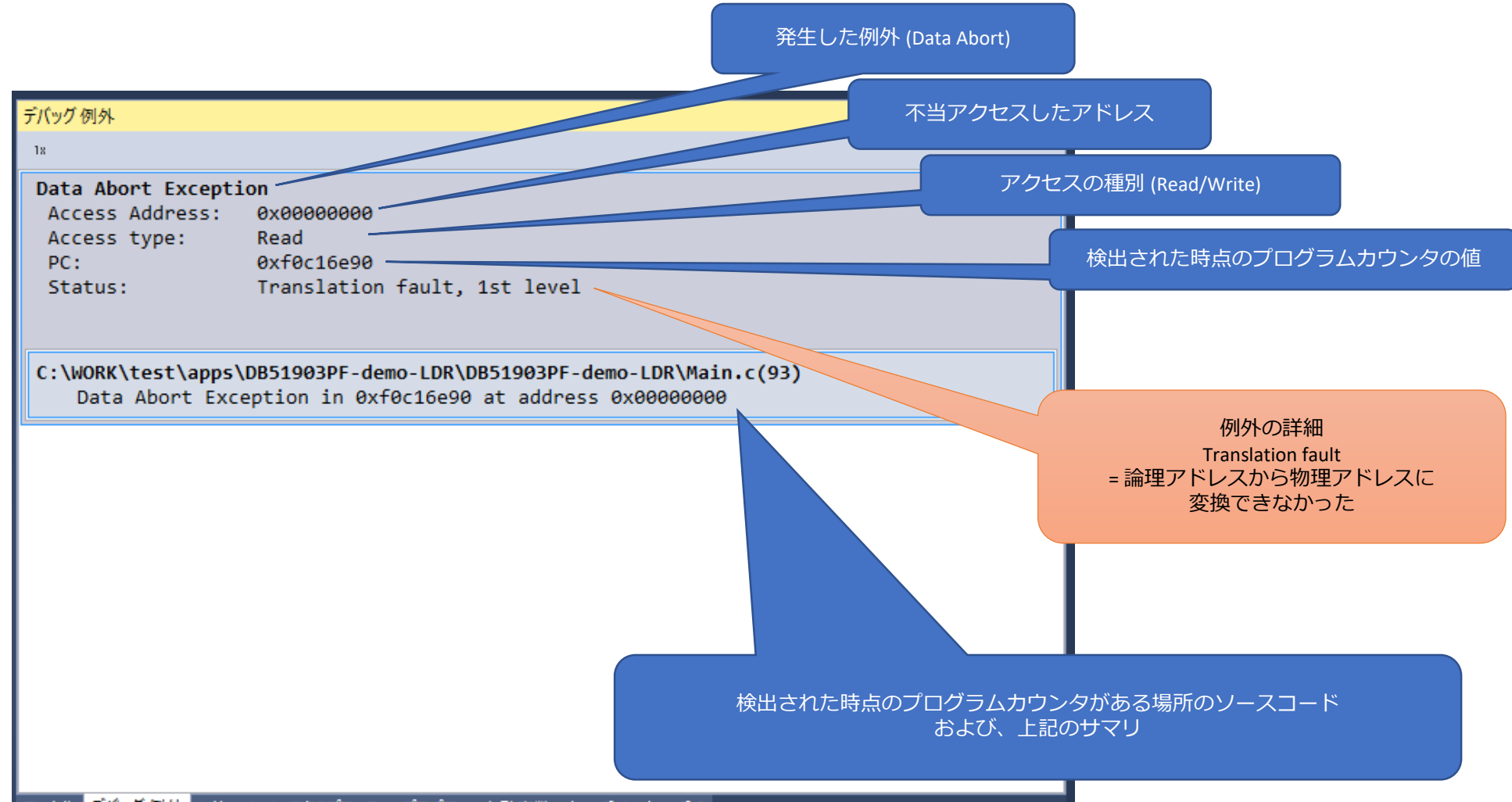
**バグが発生したソースコード行** (Source code line where the bug occurred):

```
84 → hw_init();  
85 }  
86  
87 void root_task()  
88 {  
89 → char *ptr;  
90 → char var;  
91  
92 → ptr = 0;  
93 → var = *ptr;  
94  
95  
96 → syslog_msg_log(LOG_UPTO(LOG_INFO), LOG_UPTO(LOG_INFO));  
97 → syslog(LOG_INFO, "root_task() is up.\n");  
98  
99 → ER-ercd;  
100  
101 → app_init();  
102
```

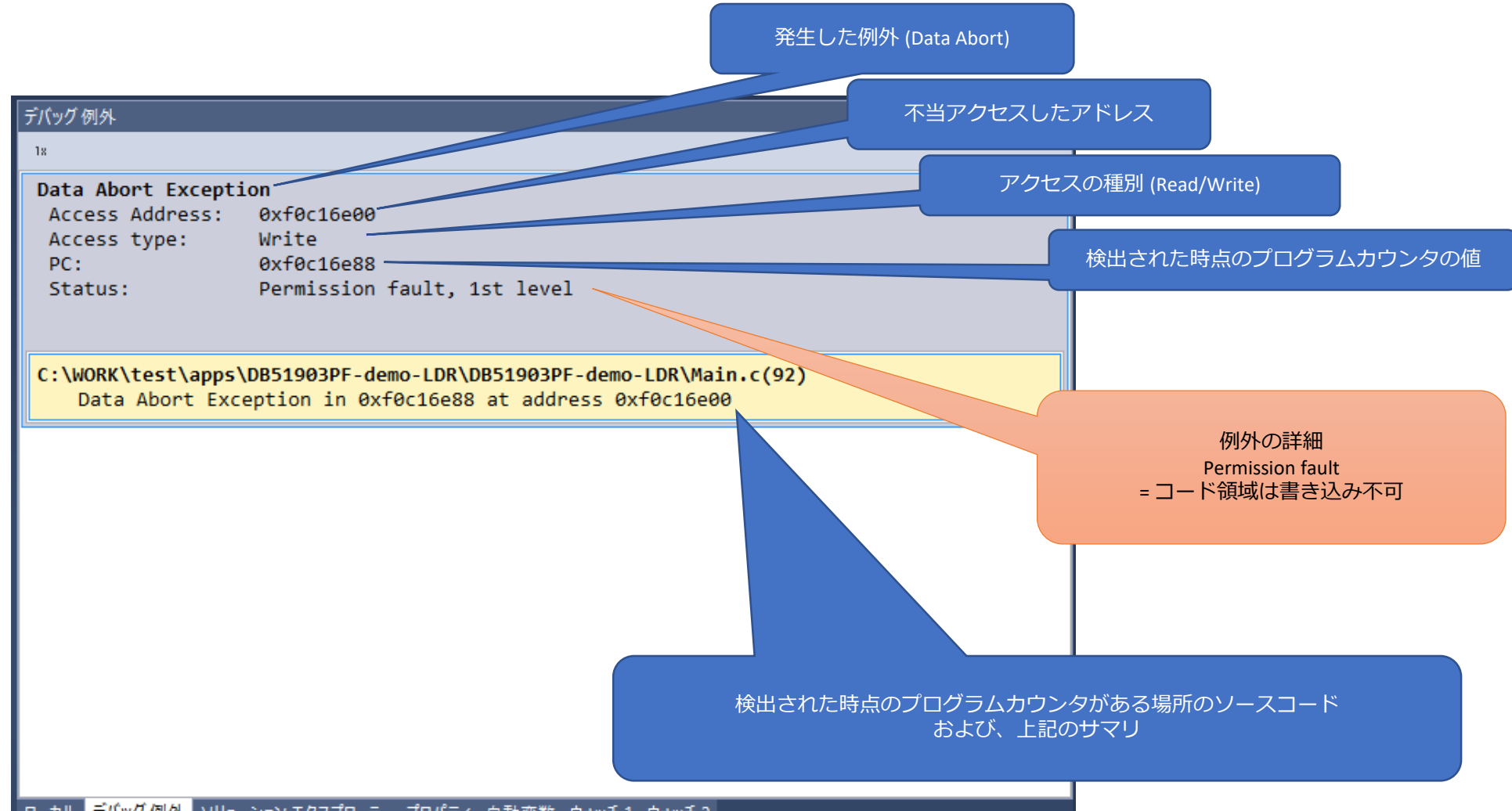
**まず、ここをクリックすると、左のように強調表示されます** (First, click here, and it will be highlighted like this on the left):

The call stack entry `C:\WORK\test\apps\DB51903PF-demo-LDR\DB51903PF-demo-LDR\Main.c(93)` is highlighted, and a green arrow points from it to line 93 of the source code, which is also highlighted.

# 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 shows the 'Debug Exception' window in Visual Studio. The exception is a 'Data Abort Exception' with the following details:

- 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)`. The callout box explains that the exception details are 'Permission fault', which means 'const' areas are not writable.

Callouts from the image:

- 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

# 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 "Prefetch Abort Exception" at address 18. The details of this exception are shown in a box below the list:

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

Callouts provide further explanation:

- 発生した例外 (Prefetch Abort)**: Points to the exception name in the list.
- 不当アクセスしたアドレス**: Points to the Prefetch Address in the details.
- 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 field.
- 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 space below the exception details.

The debugger interface at the bottom shows 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".

# 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	136 / 4096
7	Waiting (SEM)	7 (BasePriority:7)	sample_app_msc.c@msc_conn_task	0xF0C123DC	136 / 4096
8	Waiting (SEM)	7 (BasePriority:7)	Usbh_ReqProcTask	0xF0C7A690	136 / 4096
9	Ready	7 (BasePriority:7)	Usbh_ReqCmpTask	0xF0C7B730	136 / 4096
10	Running	3 (BasePriority:3)	child_task	0x10000	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ーヘッド無く使えます。

以上です

