通过 Lauterbach 对 Mcu 的 Flash 中的代码打补丁

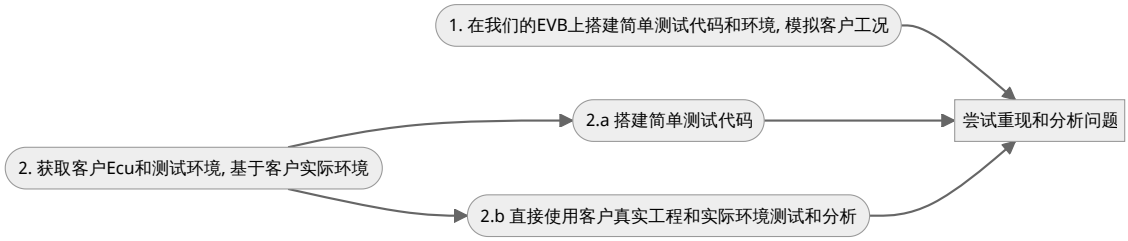
Key words: Lauterbach cmm Tips

https://autolab.wolai.com/vres5un3L8GUUG9Xcq4Y4T

一 · 场景

日常调试开发中有时候会遇到这种情况: 客户资源或者权限限制,无法提供完整源代码以及可编译的工程,本地软件配合能力有限；客户的问题仅在客户的工程上可以复现,甚至只在客户的 Ecu 以及网络节点环境下才能复现.

针对这样的场景, 我们通常的排查思路是:



1. 在我们的 EVB 上搭建简单测试代码和环境, 模拟客户工况, 尝试重现和分析问题.
2. 获取客户 Ecu 和测试环境, 基于客户实际环境

   a. 搭建简单测试代码, 尝试复现排查

   b. 直接使用客户真实工程和实际环境测试和分析.

      分析过程中，期望修改原有固件运行逻辑，尝试效果。由于没有源代码，我们需要直接对 Flash 进行 patch 补丁。

二 · 准备工作

对于 2.b 方式, 开展工作会比较挑战. 需要和客户沟通了解基本工作原理同时, 需要客户提供:

- 部分关键代码, 比如初始化, 休眠处理关键源代码. 即便整个工程无法提供, 也无法对工程做完整编译.我们仍然需要部分关键非业务敏感的代码供问题排查.
  - 包含源文件所在的目录结构
- map，包含 debug 信息的 elf
- 完整硬件测试平台

- 调试器接口(JTAG, 串口, CAN/LIN), 12V 电源, GND,相关线束，关键 toggle pin 或需要关注的测试点或 IO pin 脚引出
- 如果客户硬件有 sbc 或者硬件外部看门狗器件提供 watchdog 功能，建议他们先关闭或将其设置为调试模式避免干扰后续调试。

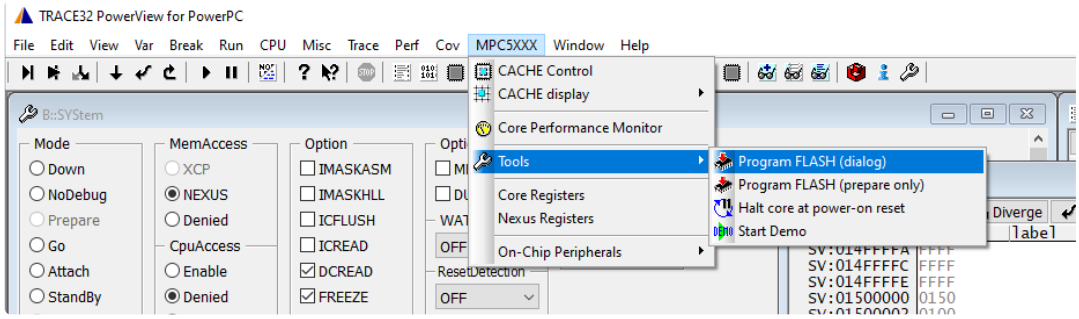
接下来分析中涉及到的工具准备好:

- Lauterbach T32
- S32DS IDE 环境以及简单样例工程.

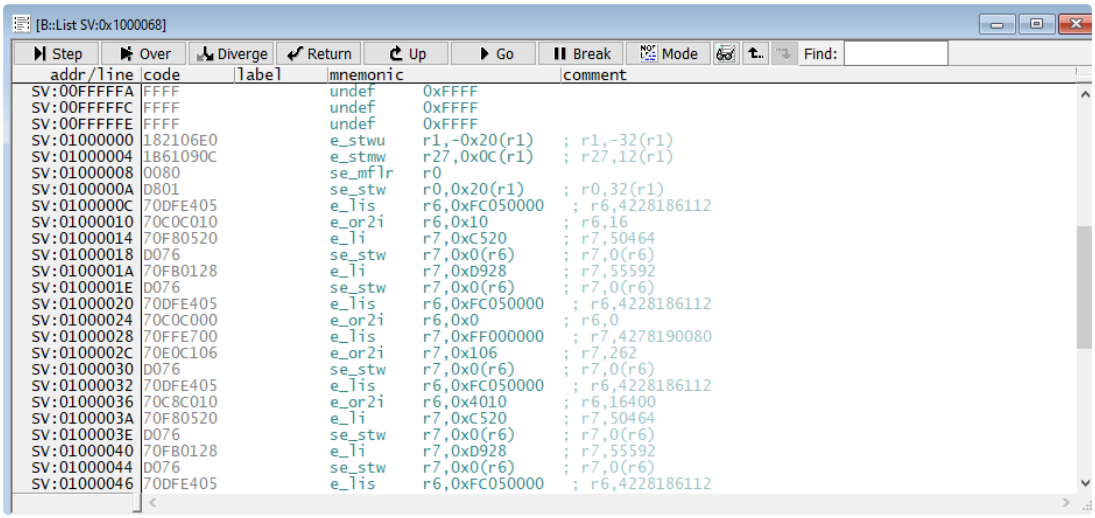
三 · 关键步骤

1. 将 elf 或者 Hex 通过 Lauterbach 对 Mcu 的 Flash 进行烧写.
   - 通过 run.cmm

     run.cmm 9.24 KB
   - 或者直接通过 Dialog 窗口进行 hex 或 elf 的 flash 编程



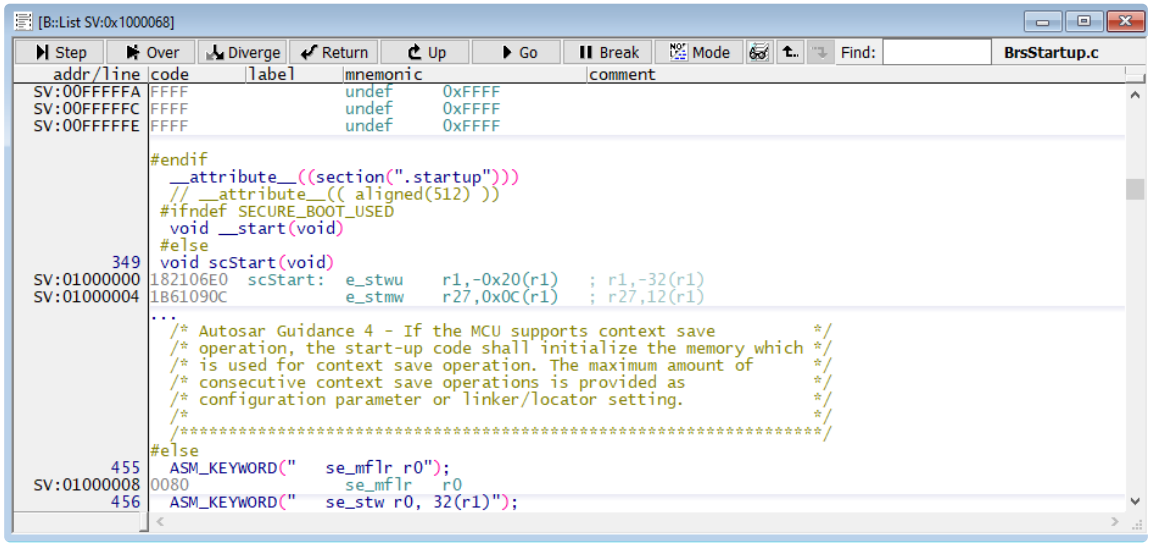
   如果缺 Dwarf 的调试信息, 下载完后通常看到的是汇编.



2. 要求客户提供的带调试信息 elf 和部分源代码, 以及涉及到的几个源文件在客户原始工程中的路径, 比如下例中的 F 盘即为客户的工程目录. 通过使用以下的脚本来将解析路径对接到自己的本地文件夹。这样我们在 Lauterbach T32 内就能看到我们关注功能的 C 源代码.

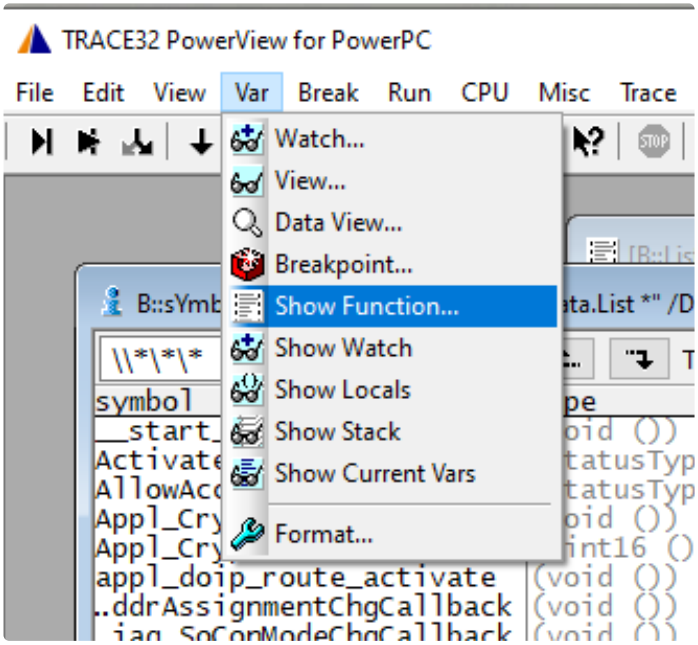
```
C
sys.attach
break
data.load.elf "C:\Project\Support\IHU_PBL.elf" /NOCODE
sYmbol.SourcePATH.Translate "F:\linysh\Applications\Appl\Source\" "C:\Project\Support\
```

通过 `v.l` 命令, function 搜索定位到_start 或者 main 函数，已经可以看到对应的 C 代码被正确解析出来.

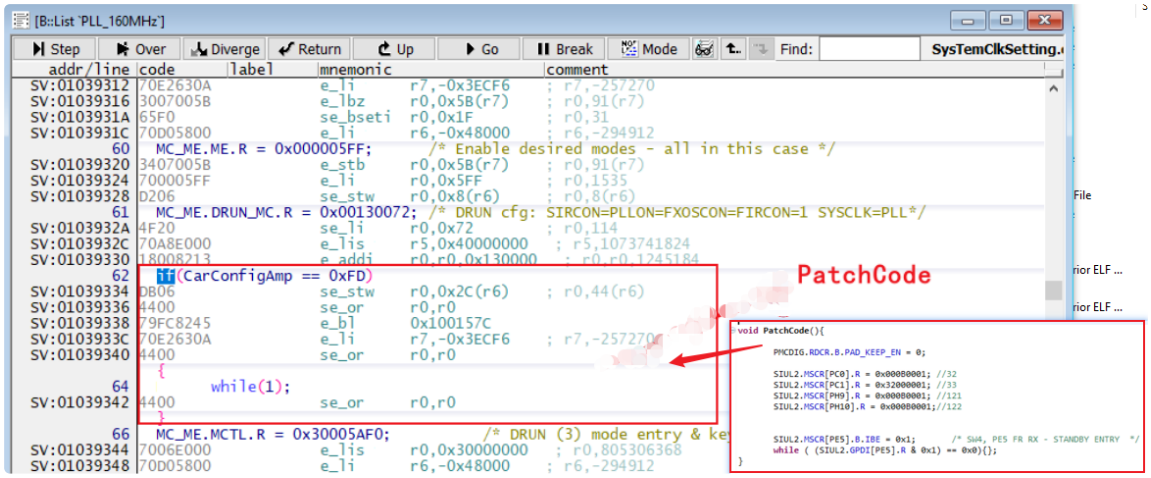


请注意带有 `NOCODE` 参数的 `data.load.elf "C:\Project\Support\IHU_PBL.elf" /NOCODE` 不会烧录 flash, 只是将 debug 信息, 比如函数名, 与具体代码映射关系, 变量等等导入 Lauterbach 环境中.

3. 调试过程中，打算对客户的代码部分做调整. 先找到对应的函数.



假设需要将下面的代码替换成一段新代码. 两种方案:



1. 直接将当前 Flash 内的代码更改替换成新代码

    - 此例中涉及的原始代码段很短, 能够被替换的代码数量非常有限，不一定放得下新补丁函数。

2. 新建函数来执行自定义补丁程序, 将当前的无效代码更换成对这个新函数的调用 `e_bl Fun1` ,把替换过程中此处的多余废弃代码用 se_nop 替换掉.

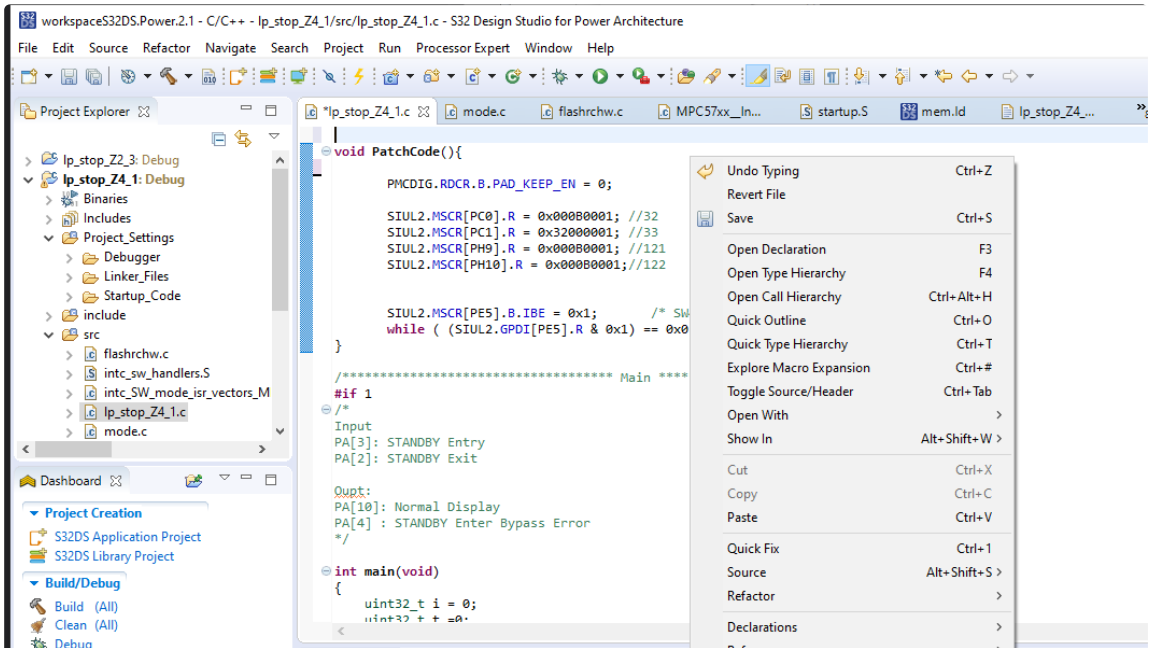
    - 此法相对容易实现

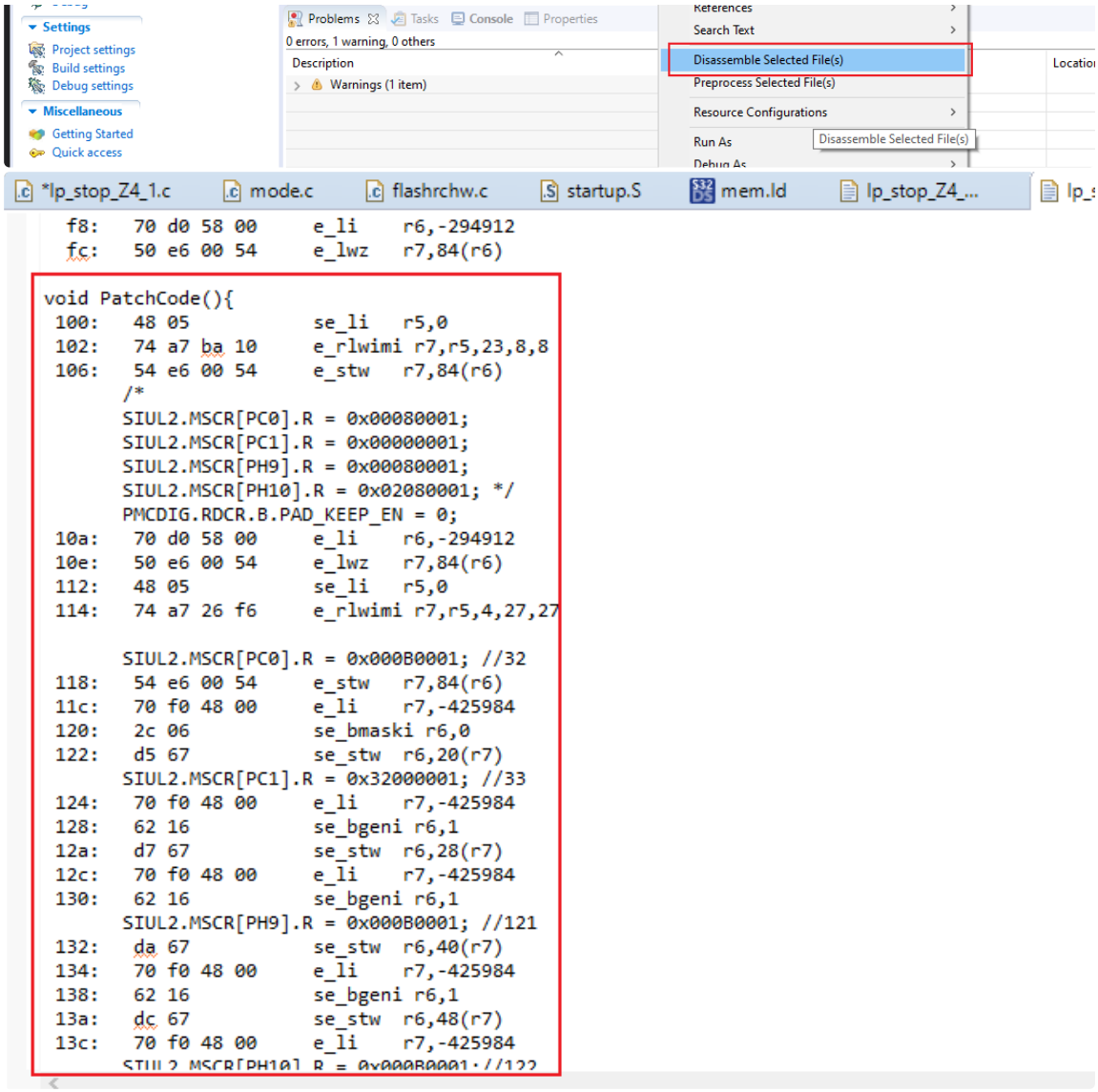
我们讨论第二种方法.

在之前准备的一个 S32DS 的简单工程中, 编写相关的 PatchCode 函数代码, 右键 Disassemble Selected File(s), 得到汇编代码.

对汇编不熟悉的话，可以借用这个工程来了解汇编语句。基本上 s 表示 store 存储，l 表示 load 取出，b 表示 branch 跳转或函数调用，c 或者 ne 之类基本是比较判断跳转。se 的前缀表示 short vle 类型节省代码存储空间。单条语句是两字节，四字节还需区分以免代码重叠冲突。

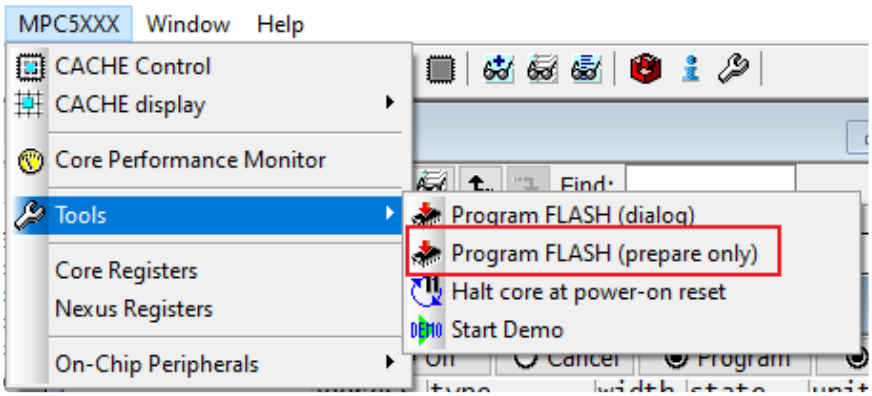
⏰ 检查该段汇编中是否有使用绝对地址，以免后续 load 到其他地址后部分代码失效。



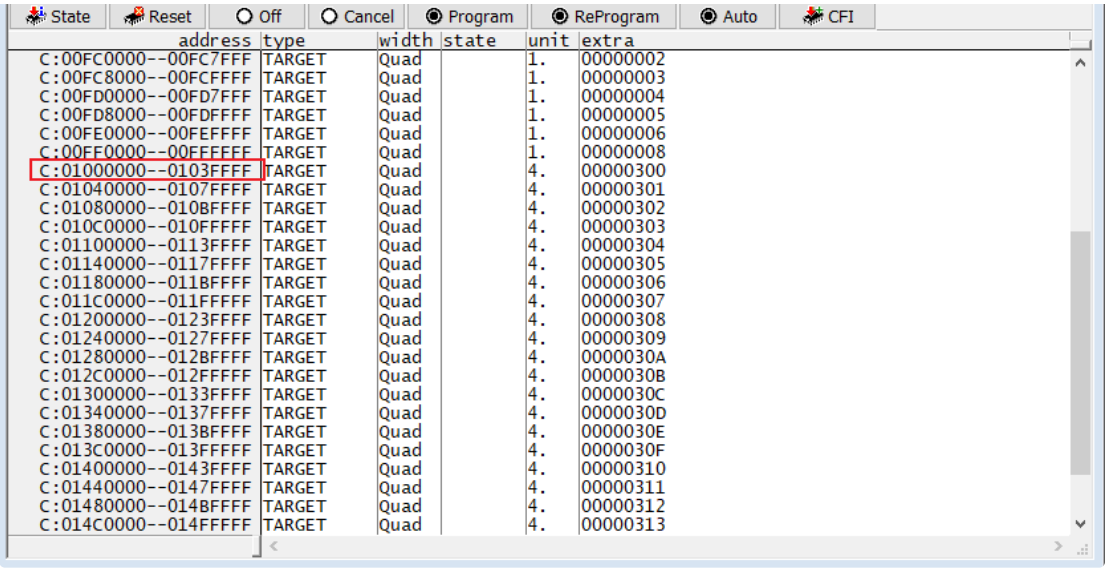


3. 接下来重头戏:

在 lauterbach T32 中 `Flash.list` 命令打开 Flash 的 list 清单或打开下图所示菜单



我们观察到,需要改动的代码区域 `01039334--01029342` 位于 blank `C:01000000--0103FFFF`



address	type	width	state	unit	extra
C:00FC0000--00FC7FFF	TARGET	Quad		1.	00000002
C:00FC8000--00FCFFFF	TARGET	Quad		1.	00000003
C:00FD0000--00FD7FFF	TARGET	Quad		1.	00000004
C:00FD8000--00FDFFFF	TARGET	Quad		1.	00000005
C:00FE0000--00FEFFFF	TARGET	Quad		1.	00000006
C:00FF0000--00FFFFFF	TARGET	Quad		1.	00000008
C:01000000--0103FFFF	TARGET	Quad		4.	00000300
C:01040000--0107FFFF	TARGET	Quad		4.	00000301
C:01080000--010BFFFF	TARGET	Quad		4.	00000302
C:010C0000--010FFFFF	TARGET	Quad		4.	00000303
C:01100000--0113FFFF	TARGET	Quad		4.	00000304
C:01140000--0117FFFF	TARGET	Quad		4.	00000305
C:01180000--011BFFFF	TARGET	Quad		4.	00000306
C:011C0000--011FFFFF	TARGET	Quad		4.	00000307
C:01200000--0123FFFF	TARGET	Quad		4.	00000308
C:01240000--0127FFFF	TARGET	Quad		4.	00000309
C:01280000--012BFFFF	TARGET	Quad		4.	0000030A
C:012C0000--012FFFFF	TARGET	Quad		4.	0000030B
C:01300000--0133FFFF	TARGET	Quad		4.	0000030C
C:01340000--0137FFFF	TARGET	Quad		4.	0000030D
C:01380000--013BFFFF	TARGET	Quad		4.	0000030E
C:013C0000--013FFFFF	TARGET	Quad		4.	0000030F
C:01400000--0143FFFF	TARGET	Quad		4.	00000310
C:01440000--0147FFFF	TARGET	Quad		4.	00000311
C:01480000--014BFFFF	TARGET	Quad		4.	00000312
C:014C0000--014FFFFF	TARGET	Quad		4.	00000313

构建如下 T32 脚本，保存为 updateCode.cmm 并运行。

updateCode.cmm 675 B

```
C
DO ~~/demo/powerpc/flash/mpc574xg PREPAREONLY
flash.Auto 0x01000000--0x0113ffff

;替换成对新补丁函数的调用:e_bl 0x0100157C, 并同时删除掉无用代码,
;注意不要破坏上下文中仍然会引用的临时寄存器比如r7.
data.Assemble 0x01039334 se_stw r0,0x2C(r6)
data.Assemble 0x01039336 se_nop
data.Assemble 0x01039338 e_bl 0x0100157C
data.Assemble 0x0103933C e_li r7, -0x3ECF6
data.Assemble 0x01039340 se_or r0,r0
data.Assemble 0x01039342 se_or r0,r0

;新函数. 为什么此处不是新函数的汇编代码而是bin文件? 稍后详细介绍.
data.load.binary patchCode0100057C_0100061F.bin 0x0100157C--0x0100161F

flash.Auto off
```

通过以上的步骤, 就可以对客户 Mcu 中的程序进行强制修改，进一步根据执行结果来调试分析.

TIPS:

1. 在修改之前, 建议对原先的内容做一个备份:

   可以通过 `data.save.binary test1000000_103FFFF.bin 0x1000000--0x103ffff` 保存所有的更改 blank 的内容.

   稍后可以通过 `data.load.binary test1000000_103FFFF.bin 0x01000000--0x0103ffff` 来恢复原有的 Flash 内容.

2. 可能大家已经注意到, 新函数部分我没有把汇编一条一条写进去, 而是使用了 data.load. 这是为什么呢? 大家可以回头来仔细看我们通过反汇编获取来的补丁函数代码. 前三条绿色的语句是已经手工修改的对应 Lauterbach T32 command. 后面的我还没有更新, 仍然保持原状. 看起来要改的部分挺多的, 是不是太麻烦了些? 所以我倾向另一个简单的办法, 就是我们不去手工修改, 而是把包含 PatchCode 代码的 S32DS 简单样例直接下载到 Mcu 中, 然后通过 `data.save.binary` dump 保存出 PatchCode 这个函数部分 flash 内容. 然后直接通过 `data.load.binary` 来指定新位置 load 到客户 Ecu 中。

仔细观察你会发现汇编代码部分有一定规律， 所以你也可以自行编写擅长的脚本来解析并批量替换，构造成对应的 T32 command 序列。

▼ PatchCode 函数反汇编代码改造进行中 （点击三角可展开）

C

```
~~100:  48 05          se_li   r5,0~~
~~102:  74 a7 ba 10    e_rlwimi r7,r5,23,8,8~~
~~106:  54 e6 00 54    e_stw   r7,84(r6)~~
 data.Assemble 0x0100157C se_li   r5,0
 data.Assemble 0x0100157E e_rlwimi r7,r5,23,8,8
 data.Assemble 0x01001582 e_stw   r7,84(r6)

 ;PMCDIG.RDCR.B.PAD_KEEP_EN = 0;
 10a:  70 d0 58 00    e_li    r6,-294912
 10e:  50 e6 00 54    e_lwz   r7,84(r6)
 112:  48 05          se_li   r5,0
 114:  74 a7 26 f6    e_rlwimi r7,r5,4,27,27

     SIUL2.MSCR[PC0].R = 0x000B0001; //32
 118:  54 e6 00 54    e_stw   r7,84(r6)
 11c:  70 f0 48 00    e_li    r7,-425984
 120:  2c 06          se_bmaski r6,0
 122:  d5 67          se_stw  r6,20(r7)
     SIUL2.MSCR[PC1].R = 0x32000001; //33
 124:  70 f0 48 00    e_li    r7,-425984
 128:  62 16          se_bgeni r6,1
 12a:  d7 67          se_stw  r6,28(r7)
 12c:  70 f0 48 00    e_li    r7,-425984
 130:  62 16          se_bgeni r6,1
     SIUL2.MSCR[PH9].R = 0x000B0001; //121
 132:  da 67          se_stw  r6,40(r7)
 134:  70 f0 48 00    e_li    r7,-425984
 138:  62 16          se_bgeni r6,1
 13a:  dc 67          se_stw  r6,48(r7)
 13c:  70 f0 48 00    e_li    r7,-425984
     SIUL2.MSCR[PH10].R = 0x000B0001;//122
 140:  2c 06          se_bmaski r6,0
 142:  dd 67          se_stw  r6,52(r7)
 144:  70 f0 48 00    e_li    r7,-425984
 148:  2c 06          se_bmaski r6,0
 14a:  d5 67          se_stw  r6,20(r7)
 14c:  70 f0 58 00    e_li    r7,-294912


     SIUL2.MSCR[PE5].B.IBE = 0x1;       /* SW4, PE5 FR RX - STANDBY ENTRY  */
```

```
150:  62 76          se_bgeni r6,7
152:  54 c7 01 e4    e_stw    r6,484(r7)
156:  70 f0 58 00    e_li     r7,-294912
15a:  70 da e0 00    e_lis    r6,53248
15e:  70 cb c2 f0    e_or2i   r6,23280
    while ( (SIUL2.GPDI[PE5].R & 0x1) == 0x0){};
162:  d1 67          se_stw   r6,4(r7)
164:  70 f0 58 00    e_li     r7,-294912
168:  70 da e0 00    e_lis    r6,53248
16c:  70 d4 c5 0f    e_or2i   r6,42255
170:  d1 67          se_stw   r6,4(r7)
172:  18 00 d0 00    e_nop
176:  18 00 d0 00    e_nop
17a:  48 07          se_li    r7,0
17c:  d2 7f          se_stw   r7,8(r31)
17e:  78 00 00 0e    e_b      18c <main+0x18c>
182:  18 00 d0 00    e_nop
}
186:  c2 7f          se_lwz   r7,8(r31)
188:  20 07          se_addi  r7,1
18a:  d2 7f          se_stw   r7,8(r31)
18c:  c2 6f          se_lwz   r6,8(r31)
18e:  70 e0 e0 1e    e_lis    r7,30
```

3. 如果在 Flash 操作过程中 Flash.Auto 和 Flash.Auto off 没有协调配对好, 后续再进行 Flash 编程会出错. 请在 F.L 的界面点击 off 按钮🔘或者 Cancel 来实施相关的代码变更/取消。然后再重新烧写程序.

> 📌 请注意：`Flash.auto` 后,对 flash 更改操作实际上是更新到虚拟内存中，只有在 `Flash.Auto off` 后，才真正刷写到 flash

4. 系统运行时打开 T32 `SYStem.OPTION DUALPORT ON`，可比较方便地实时查看结果. 建议日常打开这个开关。

```
SYStem.RESet
SYStem.BdmClock 5MHz
SYStem.CPU MPC5748G
SYStem.CONFIG.CORE 1. 1.
SYStem.CONFIG.SLAVE OFF
SYStem.Option.WATCHDOG OFF
SYStem.OPTION DUALPORT ON
SYStem.DETECT CPU

;System.Option LPMDEBUG ACTIVE
;restart the CPU with debug mode enable

system.Option SLOWRESET ON
SYSTem.Up
```