軟體 Overlay 程式編寫與除錯

Software Overlay：How to coding and debugging

發表人：賴歆雅，技術經理，
晶心科技股份有限公司
hylai@andestech.com

軟體 Overlay：程式編寫與除錯

賴歆雅，技術經理，晶心科技股份有限公司

近幾年來，SOC 為了支援更大的硬體資源，及更精確的演算法，很多應用中的韌體程式碼越來越大，但是售價卻要越來越便宜。各家廠商無不絞盡腦汁尋找降低成本的方法。

SRAM 在 SOC 上，是一個快速但單位面積較大的元件，而單位面積較大代表成本較昂貴。有一個降低成本的方法，是將程式碼放在較慢但單位面積較小的 flash 或 ROM 上，當系統需要執行裡面的某些程式碼時，才載入到記憶體裡執行。

如果用商店來比喻的話，有一個小店租在都市裡的黃金店面裡，小店的展示櫃很小，當客人想要看架上沒有的商品時，店員才從後面較大的倉庫裡，把商品拿出來放到展示櫃上。這裡的展示櫃就像 SRAM，昂貴但是有效率，倉庫就像 flash，便宜容積大但是存取較麻煩。

本文介紹的是軟體 overlay 的技術。除此之外，晶心科技也發展了硬體 overlay 的技術，使得 overlay 執行更快，實作更為簡單。期望本文章能對使用者有所助益，也希望讀者不吝指教提供您寶貴的意見。

1. 軟體 Overlay 技術介紹及實作

我們拿一個情境實例做為說明，比方說程式碼的大小為 210KB，RAM 只有 64KB，我們把 RAM 規劃成一格一格的大小，比方說每 4KB 切成一塊。每 4KB 的大小可以在不同時間，更換成不同的程式碼，可以重覆利用 RAM 的空間。程式碼儲存在 ROM 或 flash 裡，只有在執行之前會將函式從 ROM 或 flash 裡動態載入 SRAM 裡。當這個函式執行完成，下一個函式要執行前，再載入下一個函式。

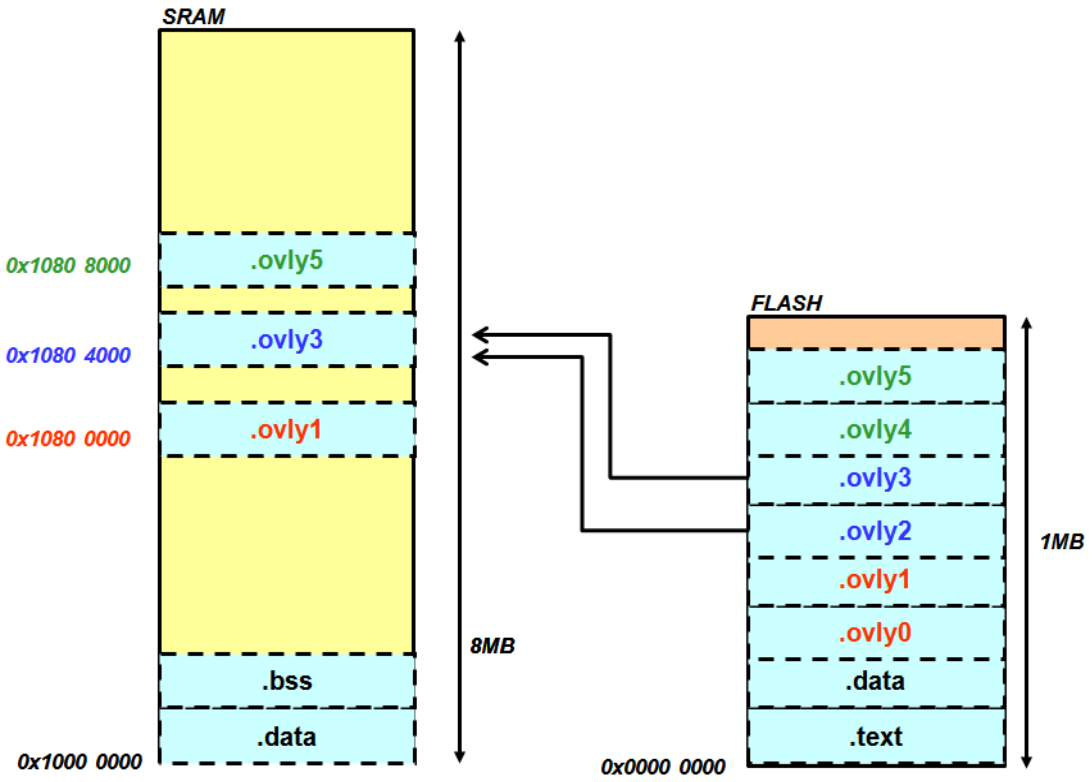
值得注意的是，每一格 SRAM 裡可載入的程式碼是互斥的，比方說有些不會同時使用的功能可以放在同一格裡，比方說 mp3 播放器，錄音和播放不會同時使用，就可以規劃重覆利用同一格 SRAM。

1.1 系統架構

請參考圖表 1，右邊長方形是 flash 的內容。0x0 起 1MB 的空間，flash 裡存放了程式碼和.data，及各個即將要被 overlay 的 sections。

圖表1的左邊長方形是SRAM規劃，位址從0x10000000開始，我們切出三格提供overlay的SRAM，分別是0x10800000, 0x10804000及0x10808000。Overlay要規劃成幾格，或者每一格要切成多大塊，都是由使用者規劃。這裡的SRAM與flash的地址是以通用型Andes FPGA開發板作例子。讀者設計SOC時，可以根據實際需求訂定合理的位址。

程式執行時，0x10800000可以載入.ovly0或是.ovly1。0x10804000可以載入.ovly3或是.ovly2。0x10808000可以載入.ovly4或是.ovly5。



圖表1. 系統架構圖

1.2 overlay 的 sag 檔編寫

圖表 2 是範例 sag 檔。Sag 檔是 Andes linker script generator 所需要的輸入檔，執行 linker script generator 後，輸出會產生 GNU linker 需要的 linker script。詳細語法說明可以參考 Andes BSP v3.2.0 User manual 第 12 章。

我們簡單介紹圖表 2 的語法。第 1 行關鍵字 USER_SECTIONS 表示後面接的這幾個 sections 都是由使用者自訂的 sections。在後面的章節，筆者會介紹如何把函式指定為這些自訂 sections。

```
01
    USER_SECTIONS .overlay0, .overlay1, .overlay2, .overlay3, .overla
y4, .overlay5
02  ROM 0x0
03  {
04    RAM 0x0
05      {
06        STACK = 0x10800000
07        * (+RO)
08      }
09    OVLY0 0x10800000 OVERLAY 0x0
10      {
11        .overlay0 {* (.overlay0)}
12        .overlay1 {* (.overlay1)}
13      }
14    OVLY1 0x10804000 OVERLAY 0x0
15      {
16        .overlay2 {* (.overlay2)}
```

```
17        .overlay3 {* (.overlay3)}
18      }
19    OVLY2 0x10808000 OVERLAY 0x0
20      {
21        .overlay4 {* (.overlay4)}
22        .overlay5 {* (.overlay5)}
23      }
24    RAM1 0x10000000
25      {
26        LOADADDR __data_lmastart
27        ADDR __data_start
28        * (+RW, +ZI)
29      }
30  }
```

圖表2. sw-ovly.sag檔

圖表2中，第4行表示從0x0開始的區域是唯讀區，包含程式碼(.text section)及唯讀資料段(.rodata section)。第9行，OVLY0從0x10800000開始，裡面有2個sections可overlay，一個是.overlay0，另一個是.overlay1。以此類推，在OVLY1和OVLY2都各有2個sections可以overlay。第24行的RAM1裡放的是.data及.bss sections，執行時期會從0x10000000開始。

1.3 sag 檔轉成 linker script

如圖表3，在cygwin下執行nds_ldsag軟體，將sw-ovly.sag轉成sw-nds32.ld檔。參數-o sw-nds32.ld為指定輸出檔的檔名。nds_ldsag軟體可以在AndeSight 2.0.1 MCU或是BSP v3.2.0裡取得。

```
$./nds_ldsag.exe sw-ovly.sag -o sw-nds32.ld
```

圖表3. Sag檔轉檔

1.4 程式裡指定函式或變數放在自訂的 sections

GNU ld (linker)可連結目的檔為可執行檔，排列上的最小單位是section，基本的sections為.text，.data及.bss這3個sections。為了達成分區overlay的功能，必須指定函式或是變數在自訂的sections上。在前一節裡我們介紹了我們切出3個區域可以做overlay，分別是OVLY0(從0x10800000起)，OVLY1(從0x10804000起)及OVLY2(從0x10808000起)三個區域。指定函式overlay0放在自訂section .overlay0裡，要使用`__attribute__((section(".overlay0")))`語法，完整寫法請參考圖表4a。圖表4b.是另外一種寫法。

```
void overlay0 (void) __attribute__((section(".overlay0")));
```

圖表4a. 指定函式放在自訂section

```
__attribute__((section(".overlay0"))) void overlay0(void)
{
…
```

```
}
```

圖表4b. 指定函式放在自訂section的另一種寫法

指定全域變數gdata1放在自訂section .overlay4裡，要使用 `__attribute__((section(".overlay4")))`語法，完整寫法請參考圖表5。

```
int gdata1 __attribute__((section(".overlay4"))) = 0x1234;
```

圖表5. 指定變數放在自訂section

1.5 各 sections 的 LMA 與 VMA

圖表6，是各個section的LMA和VMA。在這個表上，可以看.andes32_init到.sdata_w的LMA從0x0~0x29dc，這些section的LMA是連續的。.overlay0與.overlay1做overlay，所以有共同的VMA 0x10800000。同樣的，.overlay2和.overlay3，具有共同的VMA 0x10804000。.Overlay4和.overlay5，也有同樣的VMA 0x18008000。

```
Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .nds32_init   00000430  00000000  00000000  00001000  2**3
  1 .text         00001f40  00000430  00000430  00001430  2**2
  2 .rodata       0000035c  00002370  00002370  00003370  2**2
  3 .overlay0     000000a8  10800000  000026cc  00004000  2**2
  4 .overlay1     000000a8  10800000  00002774  00005000  2**2
  5 .overlay2     000000a8  10804000  0000281c  00006000  2**2
  6 .overlay3     000000a8  10804000  000028c4  00007000  2**2
  7 .overlay4     00000004  10808000  0000296c  00008000  2**2
  8 .overlay5     00000004  10808000  00002970  00009000  2**2
  9 .nds32.ovly.tbl 00000068  10000000  00002974  0000a000  2**0
10 .sdata_w      00000010  10000068  000029dc  0000a068  2**2
11 .sbss_w       00000028  10000078  10000078  0000a078  2**2
12 .bss          00000224  100000a0  100000a0  0000a078  2**3
```

圖表6. 各sections的LMA和VMA

1.6 overlay 程式的載入

前面已經介紹overlay section的sag檔寫法。那麼如何載入使用者想要用的overlay程式呢？

請看圖表7，這是overlay的執行程式碼。第5行`OverlayLoad(0)`表示載入section .overlay0。第6行`OverlayLoad(4)`表示載入section .overlay4。第7行在.overlay0被載入後，執行overlay0()，可以正常工作。

```
01  Void overlay ()
02  {
03
04    puts("/* Enter overlay Root portion */\n");
05    OverlayLoad (0);
06    OverlayLoad (4);
07    overlay0 ();
08
09    OverlayLoad (1);
10    overlay1 ();
11
12    OverlayLoad (2);
13    OverlayLoad (5);
14    overlay2 ();
15
16    OverlayLoad (3);
17    overlay3 ();
18    return;
19  }
20
```

圖表7. Overlay sections的使用方法

再來我們介紹一下Overlay manager的程式運作，Overlay manager即為圖表7中的函式**OverlayLoad**。圖表8列出Overlay manager程式碼片段，主要做了兩件事。一，修改mapped table `_ovly_table`，標示overlay section是mapped或是unmapped。`_ovly_table`的用途是讓gdb知道目前載入的是哪一個section，使得gdb在debug時，能自動切換為正確的除錯資訊。

二，在程式執行時期將函式載入，函式`ovly_copy`是一個memcpy函式，將函式從LMA複製到VMA上。當`OverlayLoad(0)`執行完後，`overlay0`函式主體便存在於VMA上，可正確的執行。

```
for (i = 0; i < _novlys; i++)
{
  if (i == ovlyno)
    _ovly_table[i][MAPPED] = 1;     /* this one now mapped */
  else if (_ovly_table[i][VMA] == _ovly_table[ovlyno][VMA])
    _ovly_table[i][MAPPED] = 0;     /* this one now un-mapped */
}
ovly_copy (_ovly_table[ovlyno][VMA],
       _ovly_table[ovlyno][LMA],
       _ovly_table[ovlyno][SIZE]);
```

```
  _ovly_debug_event ();
```

圖表8. Overlay manager程式碼片段

圖表9為_ovly_table的內容，要標示每一個overlay section的vma, size, lma, 和是否mapped。必須要注意的一點，_ovly_table要位在一個lma等於vma的區域裡。

```
struct
{
unsigned long vma;
unsigned long size;
unsigned long lma;
unsigned long mapped;
}
```

圖表9. _ovly_table的內容

2. 除錯Overlay的程式

開啟自動overlay除錯功能的gdb命令是*overlay auto*。當overlay auto開啟後，對於使用者來說，與一般程式的除錯方法相同。

圖表8的最後一行`_ovly_debug_event()`的用途是讓gdb能把中斷點加在正確的地址上，這一行要寫在`OverlayLoad`的後面。必須要有這一行，gdb的自動overlay除錯才能正常。

當使用者加一個中斷點在被overlay的區域，gdb會在函式被載入之後(即為執行完`OverlayLoad`)，遇到_ovly_debug_event時，自動的把中斷點加到overlay的地址上。

3. 參考資料

Overlay Commands
https://sourceware.org/gdb/onlinedocs/gdb/Overlay-Commands.html

Automatic Overlay Debugging
https://sourceware.org/gdb/onlinedocs/gdb/Automatic-Overlay-Debugging.html#Automatic-Overlay-Debugging

Debugging Programs That Use Overlays
http://davis.lbl.gov/Manuals/GDB/gdb_11.html

Andes BSP v3.2.0 User Manual
Chapter 12 “Linker Script Generation”

結語

善用 overlay 技術可以更有效率的使用快速但昂貴的 SRAM，在執行時，容納比 SRAM 實際大小更大的程式，設計出高效率小面積的 IC。