

Rom Patch for Andes Platform



Outline

- ❖ Introduction
- ❖ Main Project
- ❖ Patch Code
- ❖ Load and Debug
- ❖ Conclusion

Introduction

❖ Why patch

- When IC is produced with non-erasable ROM, if errors are in the ROM code, it can only be fixed with patch. The patch code is put at the external storage, like flash, to replace the error prone code in the ROM

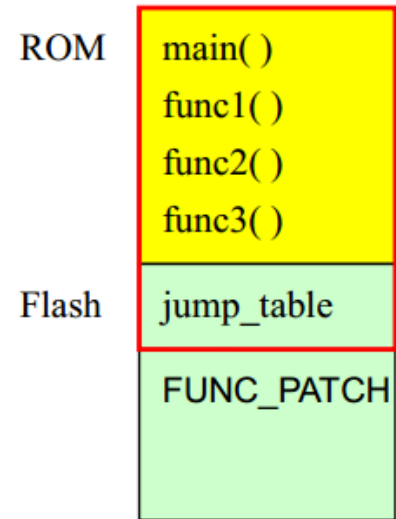
❖ Rom patch

- Put the addresses of the functions at rewriteable storage. When a new address of a function is put at a specific location to replace the old address, it means the new function will be invoked.

Introduction

❖ Storage

- Yellow block: Un-rewritable ROM
 - ◆ `main()`, `func1`, `func2`, `func3`
- Green block: Rewriteable flash
 - ◆ `jump_table`, `FUNC_PATCH`



❖ Codes

- `main` invokes `func1`, `func2`, `func3`
 - ◆ Addresses of `func1`, `func2`, `func3` are at `jump_table`
 - ◆ Indirect invoke through `jump_table`, not direct invoke
- Patch code are put at `FUNC_PATCH` block
 - ◆ Example: Put patch code of `func2` at `FUNC_PATCH` block

Example Main Project

❖ Main code : main.c

- Use `__attribute__` to specify jump_table to a section

```
main.c
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 int func1(int);
5 int func2(int);
6 int func3(int);
7 int num1=1;
8 int num2=2;
9 int num3=3;
10
11 typedef struct strfunptr {
12     int (*func_a)(int);
13     int (*func_b)(int);
14     int (*func_c)(int);
15 }sfptr;
16
17 sfptr jump_table __attribute__((section (".FUNC_TABLE"))) = {func1, func2, func3};
18
19 int main(void) {
20
21     printf("func1(30)=%d\n", jump_table.func_a(30));
22     printf("func2(30)=%d\n", jump_table.func_b(30));
23     printf("func3(30)=%d\n", jump_table.func_c(30));
24
25     return EXIT_SUCCESS;
26 }
27
28 int func1(int x){
29     return x*num1;
30 }
31 int func2(int x){
32     return x*num2;
33 }
34 int func3(int x){
35     return x*num3;
36 }
37
```

Example Main Project

- ❖ Sag file: specify the .FUNC_TABLE to address 0x40001

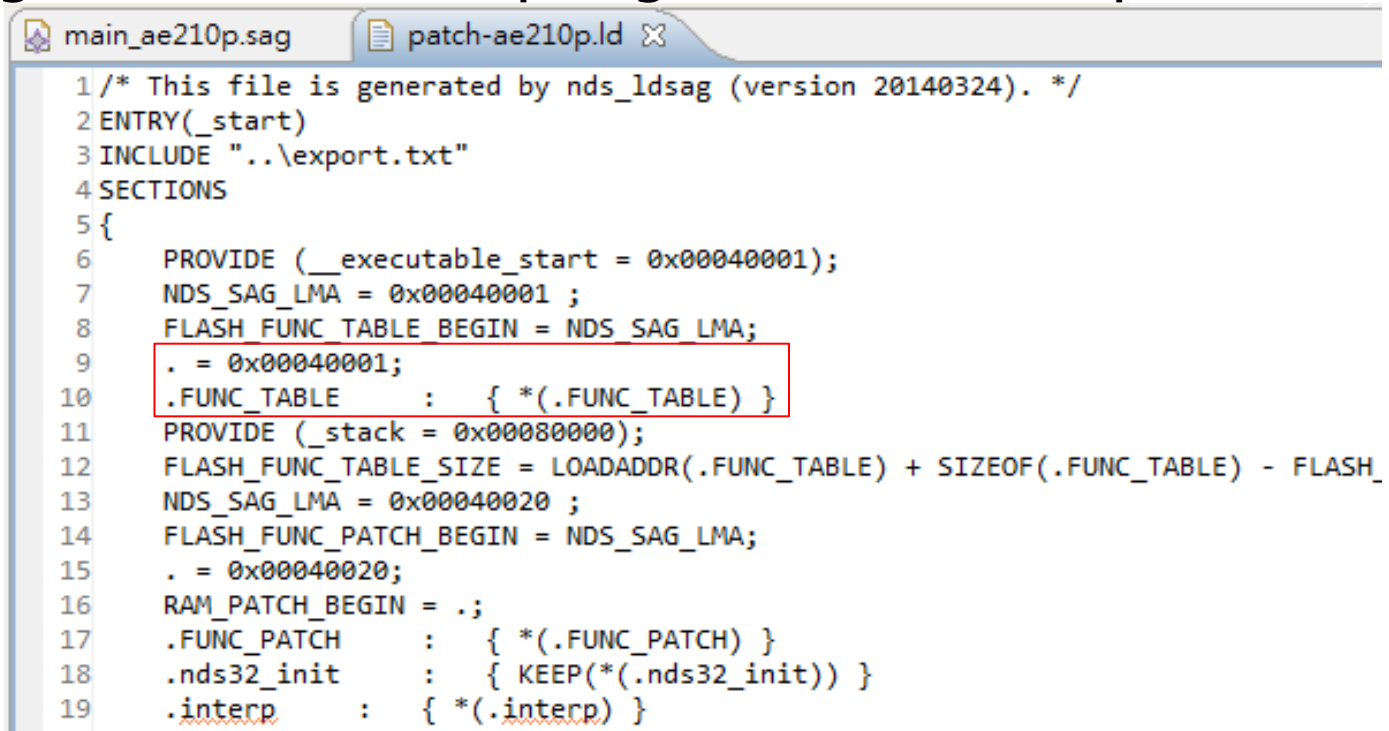
```
main_ae210p.sag
1 USER_SECTIONS .FUNC_TABLE
2
3 ROM 0x00000000 0x00040000 ; address base 0x00000000, max_size=256k
4 {
5     EXEC 0x00000000
6     {
7         * (+RO,+RW,+ZI)
8         STACK = 0x00040000
9     }
10 }
11
12 FLASH 0x00040001 0x00000020 ; FUNC_TABLE address 0x00040001
13 {
14     TABLE_SECTION 0x00040001
15     {
16         * (.FUNC_TABLE)
17     }
18 }
```

Example Main Project

❖ Use nds_ldsag tool to generate ld script

■ In cygwin command line mode :

```
$/cygdrive/c/Andestech/AndeSightxxxMCU/utils/nds_ldsag.exe main_ae210p.sag -o main_ae210p.ld
```



```
main_ae210p.sag patch-ae210p.ld X
1 /* This file is generated by nds_ldsag (version 20140324). */
2 ENTRY(_start)
3 INCLUDE "..\export.txt"
4 SECTIONS
5 {
6     PROVIDE (__executable_start = 0x00040001);
7     NDS_SAG_LMA = 0x00040001 ;
8     FLASH_FUNC_TABLE_BEGIN = NDS_SAG_LMA;
9     . = 0x00040001;
10    .FUNC_TABLE : { *(.FUNC_TABLE) }
11    PROVIDE (_stack = 0x00080000);
12    FLASH_FUNC_TABLE_SIZE = LOADADDR(.FUNC_TABLE) + SIZEOF(.FUNC_TABLE) - FLASH_
13    NDS_SAG_LMA = 0x00040020 ;
14    FLASH_FUNC_PATCH_BEGIN = NDS_SAG_LMA;
15    . = 0x00040020;
16    RAM_PATCH_BEGIN = .;
17    .FUNC_PATCH : { *(.FUNC_PATCH) }
18    .nds32_init : { KEEP(*(nds32_init)) }
19    .interp : { *(.interp) }
```

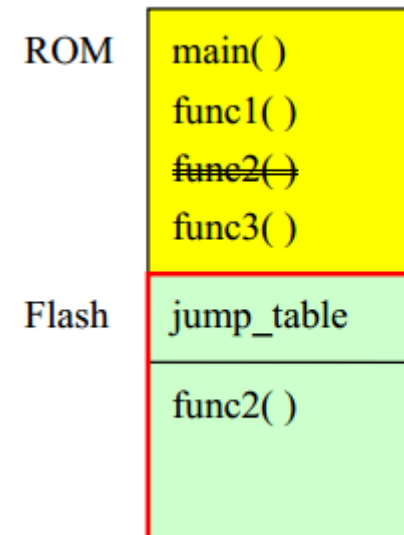
Example Main Project

❖ Result of main program :

```
func1(30)=30  
func2(30)=60  
func3(30)=90
```


Main Program and Patch Code

- ❖ While finding func2 needs to be patched for some reason, the patched diagram is as below.
 - Red block: Needs to be re-compiled
 - ◆ It is flash
 - The address of func2 in jump_table is replaced by new address. The addresses of func1 and func3 are the same.



Main Project

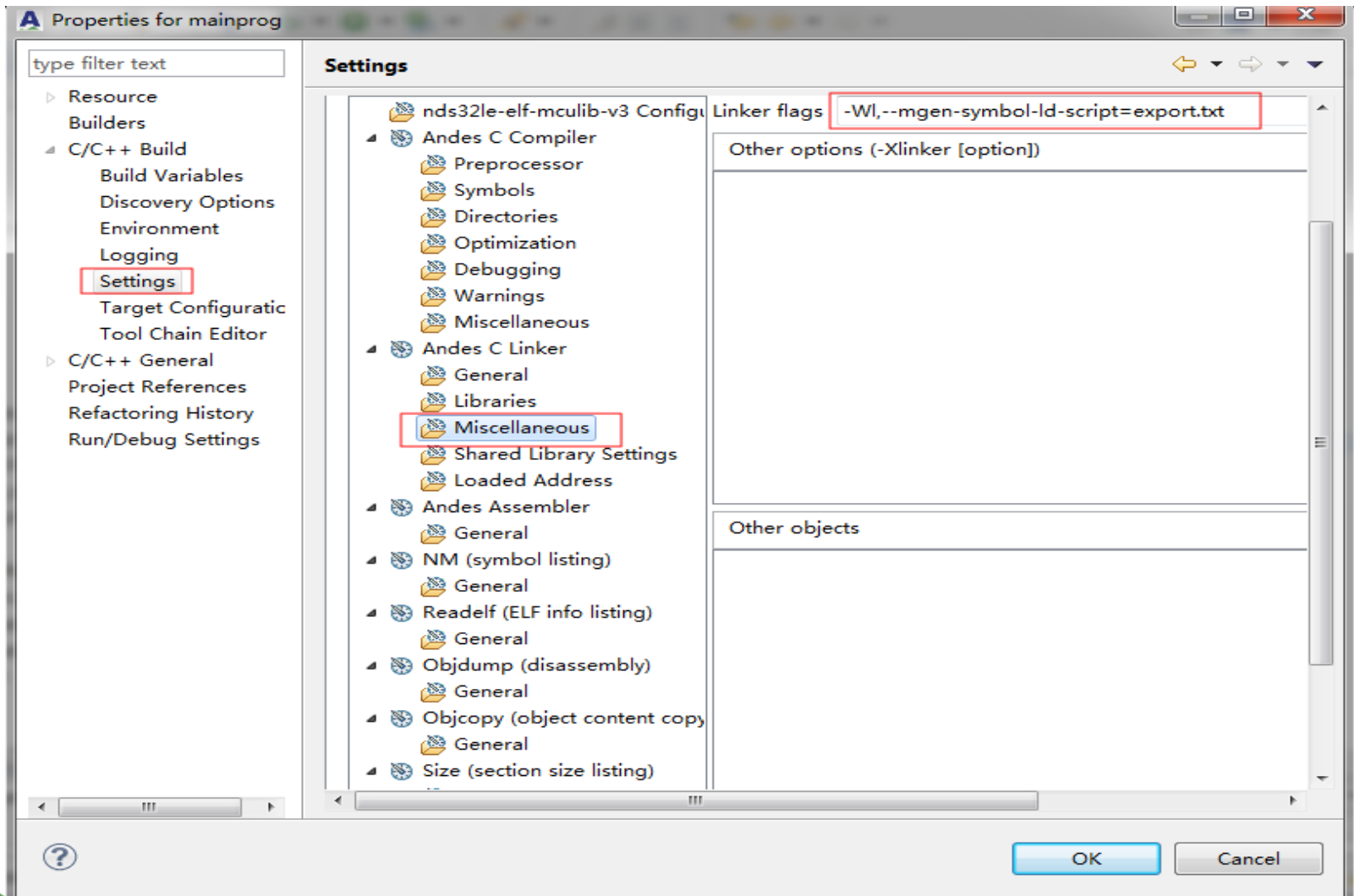
❖ Implementation

■ Export symbol table

- ◆ Including variables or functions that are needed to patch
- ◆ Add "-Wl,--mgen-symbol-ld-script=export.txt" in linker.
"export.txt" is the exported file name
- ◆ Delete func2 in the export.txt
- ◆ Copy export.txt to patch project

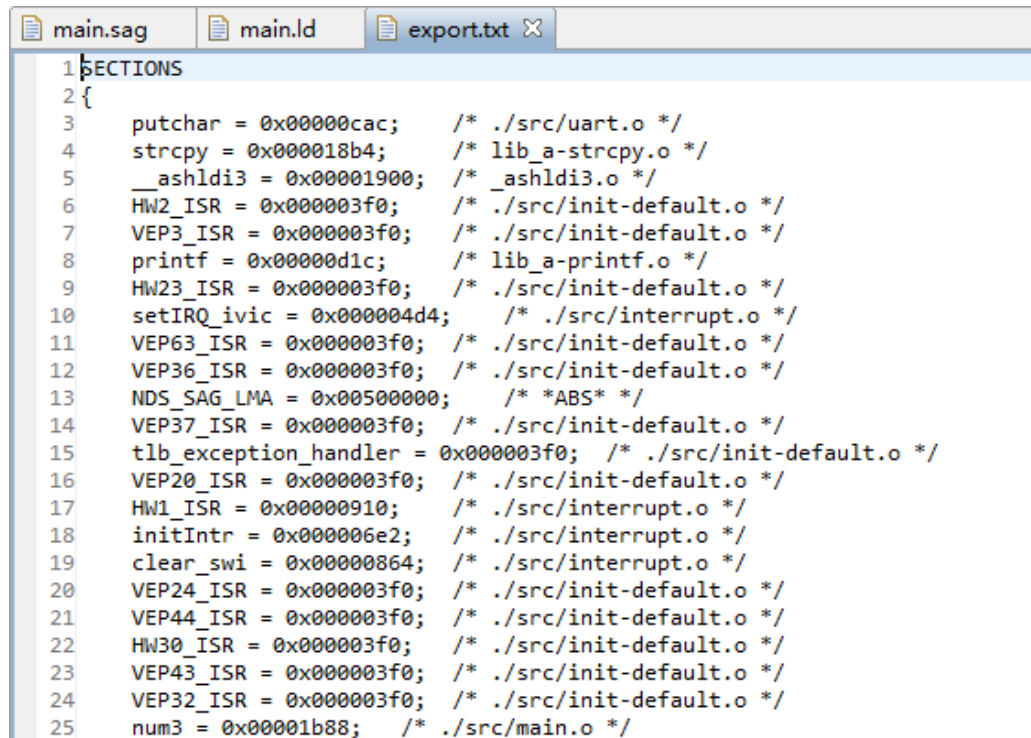
Main Project

❖ Generate symbol table via AndeSight



Main Project

- ❖ Symbol table is the addresses all variables and functions in the main project.



```
1 SECTIONS
2 {
3     putchar = 0x00000cac; /* ./src/uart.o */
4     strcpy = 0x000018b4; /* lib_a-strcpy.o */
5     __ashldi3 = 0x00001900; /* __ashldi3.o */
6     HW2_ISR = 0x000003f0; /* ./src/init-default.o */
7     VEP3_ISR = 0x000003f0; /* ./src/init-default.o */
8     printf = 0x00000d1c; /* lib_a-printf.o */
9     HW23_ISR = 0x000003f0; /* ./src/init-default.o */
10    setIRQ_ivic = 0x000004d4; /* ./src/interrupt.o */
11    VEP63_ISR = 0x000003f0; /* ./src/init-default.o */
12    VEP36_ISR = 0x000003f0; /* ./src/init-default.o */
13    NDS_SAG_LMA = 0x00500000; /* *ABS* */
14    VEP37_ISR = 0x000003f0; /* ./src/init-default.o */
15    tlb_exception_handler = 0x000003f0; /* ./src/init-default.o */
16    VEP20_ISR = 0x000003f0; /* ./src/init-default.o */
17    HW1_ISR = 0x00000910; /* ./src/interrupt.o */
18    initIntr = 0x000006e2; /* ./src/interrupt.o */
19    clear_swi = 0x00000864; /* ./src/interrupt.o */
20    VEP24_ISR = 0x000003f0; /* ./src/init-default.o */
21    VEP44_ISR = 0x000003f0; /* ./src/init-default.o */
22    HW30_ISR = 0x000003f0; /* ./src/init-default.o */
23    VEP43_ISR = 0x000003f0; /* ./src/init-default.o */
24    VEP32_ISR = 0x000003f0; /* ./src/init-default.o */
25    num3 = 0x00001b88; /* ./src/main.o */
```

- ❖ Delete the func2 in symbol table
 - Patch code will use new func2 function

Patch Code

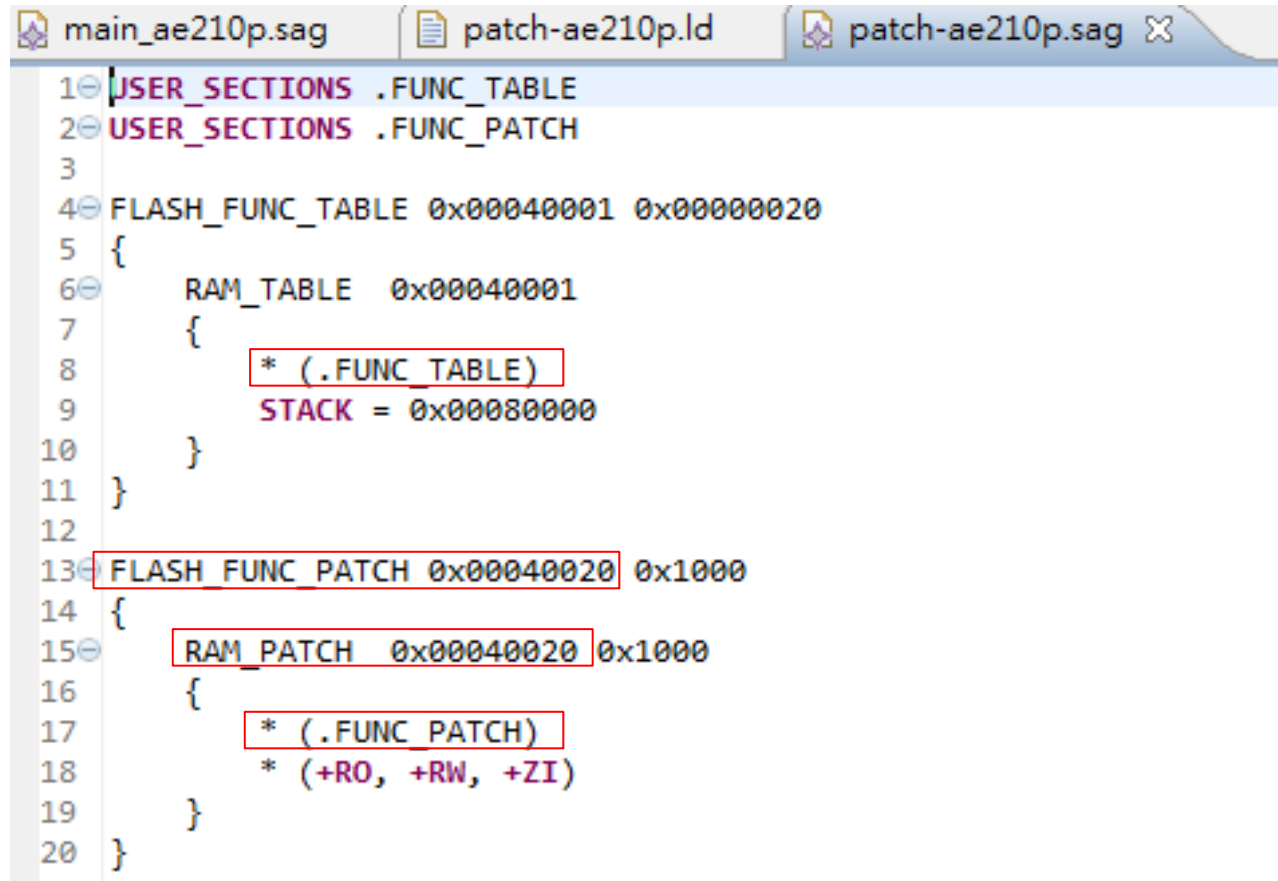
❖ Patch code

- Redefine func2 function and add to new jump_table
 - ◆ func1/func3 use the old symbol in the jump_table
- Use num2 in main project
- No main function, nor start file are needed in compile time

```
patchprog.c
2+ * patchprog.c
7 #include <stdio.h>
8 #include <stdlib.h>
9
10 extern int func1(int);
11 extern int func3(int);
12 int func2(int) __attribute__((section (".FUNC_PATCH")));
13 extern int num2;
14
15 typedef struct strfunptr {
16     int (*func_a)(int);
17     int (*func_b)(int);
18     int (*func_c)(int);
19 }sfptr;
20
21 sfptr jump_table __attribute__((section (".FUNC_TABLE"))) = {func1, func2, func3};
22
23 int func2(int x){
24     return x*num2*100;
25 }
26
27
```

Patch Code

❖ Sag file in patch



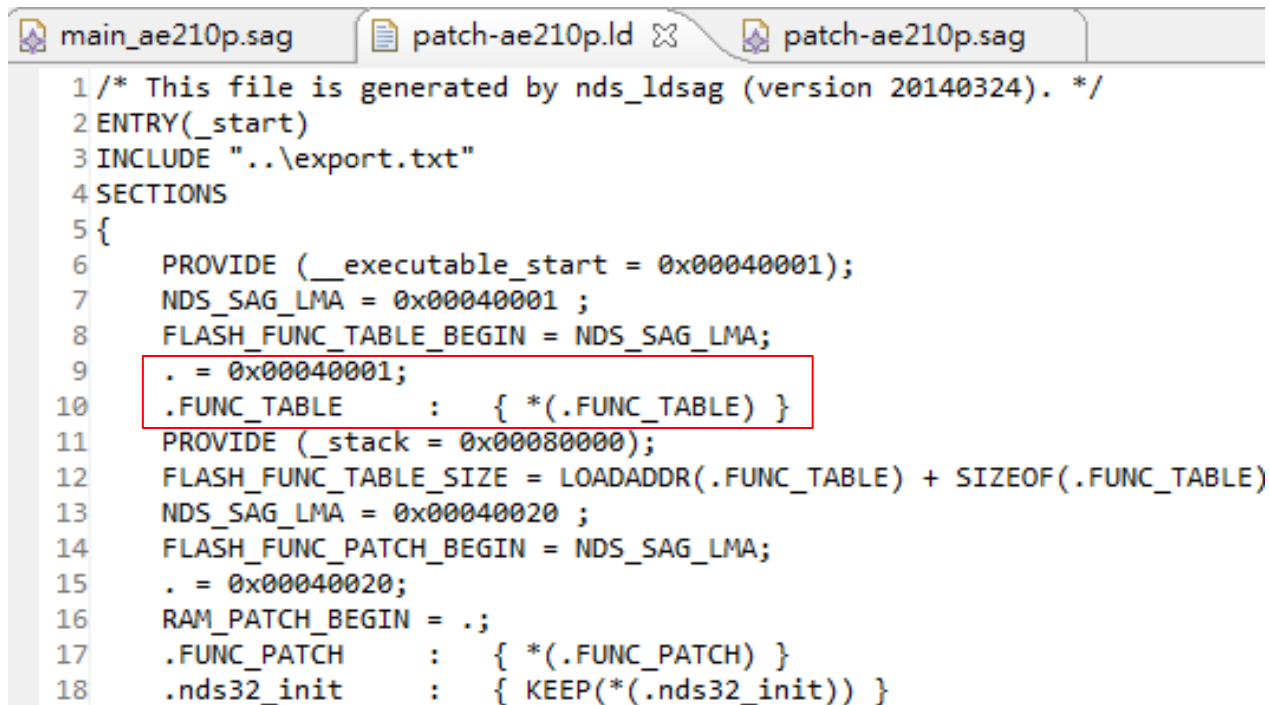
```
main_ae210p.sag | patch-ae210p.ld | patch-ae210p.sag X
1 USER_SECTIONS .FUNC_TABLE
2 USER_SECTIONS .FUNC_PATCH
3
4 FLASH_FUNC_TABLE 0x00040001 0x00000020
5 {
6     RAM_TABLE 0x00040001
7     {
8         * (.FUNC_TABLE)
9         STACK = 0x00080000
10    }
11 }
12
13 FLASH_FUNC_PATCH 0x00040020 0x1000
14 {
15     RAM_PATCH 0x00040020 0x1000
16     {
17         * (.FUNC_PATCH)
18         * (+RO, +RW, +ZI)
19     }
20 }
```

Patch Code

❖ Use nds_ldsag tool to generate ld script

■ In cygwin command line mode :

```
$/cygdrive/c/Andestech/AndeSightxxxMCU/utils/nds_ldsag.exe patch_ae210p.sag -o patch_ae210p.ld
```



```
main_ae210p.sag patch_ae210p.ld patch_ae210p.sag
1 /* This file is generated by nds_ldsag (version 20140324). */
2 ENTRY(_start)
3 INCLUDE "..\export.txt"
4 SECTIONS
5 {
6     PROVIDE (__executable_start = 0x00040001);
7     NDS_SAG_LMA = 0x00040001 ;
8     FLASH_FUNC_TABLE_BEGIN = NDS_SAG_LMA;
9     . = 0x00040001;
10    .FUNC_TABLE      :    { *(.FUNC_TABLE) }
11    PROVIDE (_stack = 0x00080000);
12    FLASH_FUNC_TABLE_SIZE = LOADADDR(.FUNC_TABLE) + SIZEOF(.FUNC_TABLE)
13    NDS_SAG_LMA = 0x00040020 ;
14    FLASH_FUNC_PATCH_BEGIN = NDS_SAG_LMA;
15    . = 0x00040020;
16    RAM_PATCH_BEGIN = .;
17    .FUNC_PATCH      :    { *(.FUNC_PATCH) }
18    .nds32_init      :    { KEEP(*(.nds32_init)) }
```

Load and Debug

❖ Load main project and patch code

- The following example uses RAM to simulate ROM and flash. Load main project and patch code to RAM.
- Start ICEman: ICEman -p 1234
- In cygwin: nds32le-elf-gdb
- The following are gsb commands:
 - core00(gdb) target remote :1234 (connect to ICEman port 1234)
 - core00(gdb) file mainprog.adx (open main project)
 - core00(gdb) load (load main project to RAM)

Load and Debug

❖ Load main project and patch code

- core00(gdb) file ../../patch/Debug/patch.adx (open patch code using relative path.)
- core00(gdb) load (load patch code to RAM)
- core00(gdb) file mainprog.adx (reopen main project to debug)
- core00(gdb) add-symbol-file ../../patch/Debug/patch.adx 0x500000 -s FUNC_TABLE 0x500000 -s FUNC_PATCH 0x500020
 - ◆ Add patch code symbol to main project

Load and Debug

❖ gdb debug

- core00(gdb) set \$pc=0x0
- core00(gdb) b main
- Breakpoint 1 at 0xbd8: file ../src/main.c, line 21.
- core00(gdb) c
- Continuing.
- Breakpoint 1, main () at ../src/main.c:21
- 21 printf("func1(30)=%d\n",jump_table.func_a(30));
- core00(gdb) s
- func1 (x=30) at ../src/main.c:29 <- main project
- 29 return x*num1;
- core00(gdb) n
- 30 }
- core00(gdb) n
- main () at ../src/main.c:22
- 22 printf("func2(30)=%d\n",jump_table.func_b(30));

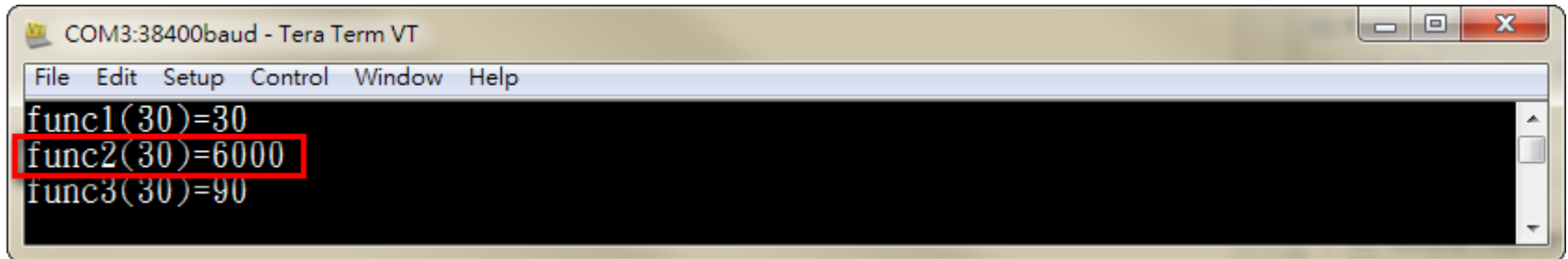
Load and Debug

❖ gdb debug

- core00(gdb) s
- func2 (x=30) at ../patchprog.c:24 <- patch code
- 24 return x*num2*100;
- core00(gdb) n
- 25 }
- core00(gdb) n
- main () at ../src/main.c:23
- 23 printf("func3(30)=%d\n",jump_table.func_c(30));
- core00(gdb) s
- func3 (x=30) at ../src/main.c:35 <- main project
- 35 return x*num3;
- core00(gdb) n
- 36 }
- core00(gdb) n
- main () at ../src/main.c:25
- 25 return EXIT_SUCCESS;
- core00(gdb)

Load and Debug

❖ Result after patch



A screenshot of a Tera Term VT window titled "COM3:38400baud - Tera Term VT". The window has a menu bar with "File", "Edit", "Setup", "Control", "Window", and "Help". The main text area displays three lines of output: "func1(30)=30", "func2(30)=6000", and "func3(30)=90". The line "func2(30)=6000" is highlighted with a red rectangular box.

```
func1(30)=30  
func2(30)=6000  
func3(30)=90
```

Conclusion

❖ jump_table

- It is indirectly invoked. The invoked functions in the main process should be replaced with an indirect call. A function table is needed.
- Size effect : Increase codesize, lower performance
 - ◆ A reasonable design is needed. Too many patch functions are avoided.

❖ Patch code and main project

- To reduce code size, if a patch code needs some functions or variables that already exist in the main project, avoid redefining them in the patch code and use "extern" key word to declare.



- ❖ Andes **e-service** is a web-based support ticket system. It is a convenient way to get quick response of your question.
- ❖ How to apply for a new account?
 - Please send your information to es.admin@andestech.com
 - ◆ Including your **name, e-mail, company name and telephone number**

How to Use E-service?

- E-service website: <http://es.andestech.com/ilogin.php>

The screenshot shows the Andes Embedded World e-service website. The top navigation bar includes links for About Andes, Products, Showroom, Partnership, Technical Support, Download, Licensees, News/Events, and Contact Us. The 'e-service' link is highlighted in the 'Technical Support' dropdown menu. A red box with the text 'Click e-service to log in' points to this link. Below the navigation bar, there is a section for 'Andes-Embedded World™ GPS Solution' and a 'Product' section. The bottom part of the screenshot shows the 'SUPPORT TICKET SYSTEM' interface. It includes a search bar with the text 'Query:' and a 'Search' button. Below the search bar, there are buttons for 'New Ticket', 'My Tickets', 'Group Tickets', 'Change Password', and 'Log Out'. A table of tickets is displayed, showing columns for Ticket #, Created on, Status, Subject, IPS staff, and Customer Full Name. The table contains four rows of tickets, all assigned to Anthony Liao.

Andes-Embedded World™
GPS Solution

Click e-service to log in

SUPPORT TICKET SYSTEM

Andes TECHNOLOGY

New Ticket **My Tickets** **Group Tickets** **Change Password** **Log Out**

Query:

Showing 1 - 6 of 6 All Tickets

Ticket #	Created on	Status	Subject	IPS staff	Customer Full Name
1005	2011/12/26	Assigned	test ticket to check the content	Anthony Liao	pryn liao
997	2011/12/22	Assigned	測試 TICKET	Anthony Liao	pryn liao
996	2011/12/22	Assigned	abc test	Anthony Liao	pryn liao
995	2011/12/22	Assigned	test ticket	Anthony Liao	pryn liao

You can track your ticket in e-service system