Andes SaG Application Examples

White Paper

……Press Contact……

Andes Technology Corporation

Marketing & Technical Service Division Joyce Chen

Tel: 886-3-6668300 ext. 254

E-mail: joycechen@andestech.com

Web: www.andestech.com

Andes Technology Corp.
2F, No.1, Li-Hsin First Road, Hsinchu Science Park,
Hsinchu City, Taiwan 300 R.O.C (SiSoft Research Center)
Tel: 886-3-6668300
Fax: 886-3-6668322
www.andestech.com

Andes SaG Application Examples

Memory mapping is essential to embedded system software design and often implemented in linker scripts. However, many programmers find linker scripts difficult to write and maintain due to its complicated syntax and long descriptions. For those who develop projects based on AndesCore™, they find Andes SaG is a boon to replace linker scripts with simple and intuitive descriptive language. More and more developers reveal their preference for Andes SaG over linker scripts when providing feedbacks to Andes. Having a technical article introducing and explaining SaG syntax before, in this article we center on four SaG application examples in an attempt to help developers better understand Andes SaG and provide them references during development.

1. Place a function or variable at a fixed address

The first example shows how to locate a function or variable at a fixed address – the address used here is a virtual address (VMA). There are a number of reasons for doing this. One is when the process address space of your SoC is not continuous and one is when you need to access high-efficient memory effectively. You have two steps: first, add your own defined section in a SaG file and specify its VMA to a fixed address; next, in your C code, specify functions or variables that are about to change to the section defined in the last step.

Figure 1 below demonstrates the first step. The key word “USER_SECTIONS” in line 1 denotes the following sections are all user-defined.

```
01  USER_SECTIONS .mysection0, .mysection1, .mysection2
02  ROM 0x0
03  {
04    RAM 0x0
05      {
06        STACK = 0x00020000
```

Andes Technology Corp.
2F, No.1, Li-Hsin First Road, Hsinchu Science Park,
Hsinchu City, Taiwan 300 R.O.C (SiSoft Research Center)
Tel: 886-3-6668300
Fax: 886-3-6668322
www.andestech.com


```
07        * (+RO)
08      }
09    MYRAM0 0x00014000
10      {
11         * (.mysection0)
12      }
13    MYRAM1 0x00018000
14      {
15        * (.mysection1)
16      }
17    MYRAM2 0x0001c000
18      {
19        * (.mysection2)
20      }
21    RAM1 0x00010000
22      {
23        LOADADDR __data_lmastart
24        ADDR __data_start
25        * (+RW, +ZI)
26      }
27  }
```

Figure 1. Samp1.sag

In Figure 1, line 4 to 8 show the memory region beginning from 0x0 is read-only and contains program codes (.text section) and read-only data (.rodata section). From line 9 to 12, the MYRAM0 part shows the VMA of .mysection0 begins from 0x00014000. Similarly, MYRAM1 and MYRAM2 parts present the beginning of VMA for .mysection1 and .mysection2 respectively. RAM1 in line 21 contains .data and .bss sections and its VMA begins with 0x00010000. That is, you should copy the data section from the load address (LMA) to the VMA in your C code and use __data_lmastart and __data_start to assign address values.

To specify functions to your own defined section, say, .mysection0, you have to use __attribute__((section(".mysection0"))) in your C code. Please refer to Figure 2A for the complete statement and Figure 2B for an alternative.

```
void functionAA (void) __attribute__((section(".mysection0")));
```

Figure 2A. Specify a function to a user-defined section in C code

Andes Technology Corp.
2F, No.1, Li-Hsin First Road, Hsinchu Science Park,
Hsinchu City, Taiwan 300 R.O.C (SiSoft Research Center)
Tel: 886-3-6668300
Fax: 886-3-6668322
www.andestech.com


```
__attribute__((section(".mysection0"))) void functionAA (void)
{
   …
}
```

Figure 2B. Another way to specify a function to a user-defined section

To specify a global variable, gdata1, to a user-defined section , .mysection1，define __attribute__((section(".mysection1"))) in your C code. Figure 3 below shows the complete statement.

```
   int gdata1 __attribute__((section(".mysection1"))) = 0x1234;
   long long gdata2 __attribute__((section(".mysection2"))) =
0xabcdabcd;
```

Figure 3. Specify a variable to a user-defined section

With functions and variables specified to fixed addresses like above, the ELF file generated after compilation reveals the VMA and LMA of each user-defined section, as shown in Figure 4.

```
   4 .mysection0 0000000c 00014000 000001a0 00002000 2**1
   CONTENTS, ALLOC, LOAD, READONLY, CODE
   5 .mysection1 00000004 00018000 000001ac 00003000 2**2
   CONTENTS, ALLOC, LOAD, DATA
   6 .mysection2 00000008 0001c000 000001b0 00004000 2**3
   CONTENTS, ALLOC, LOAD, DATA
```

Figure 4. ELF Header

Andes Technology Corp.
2F, No.1, Li-Hsin First Road, Hsinchu Science Park,
Hsinchu City, Taiwan 300 R.O.C (SiSoft Research Center)
Tel: 886-3-6668300
Fax: 886-3-6668322
www.andestech.com

2. Change IVB at run time

You may want to execute separate ISR for the same interrupt at boot time and at run time. There are several ways to achieve this. The one introduced here is to implement a vector table that contains the entry of desired ISR and then use a SaG file to specify the vector table to a fixed address. That way, if you like to switch ISR after the boot time, you only have to modify the register IVBASE (ir3).

Thus, the key is to build a vector table in assembly code and assign it to a user-defined section. See Figure 5 for how it is implemented：

```
    .section .nds32_aa, "ax"
    !============================================================
    !  NEW Vector Table
    !============================================================
    .align 2
    new_exception_vector:
    j _start                    !  (0) Trap Reset
    vector TLB_Fill             !  (1) Trap TLB fill
    vector PTE_Not_Present      !  (2) Trap PTE not present
    vector TLB_Misc             !  (3) Trap TLB misc
    vector TLB_VLPT_Miss        !  (4) Trap TLB VLPT miss
    vector Machine_Error        !  (5) Trap Machine error
    vector Debug_Related        !  (6) Trap Debug related
    vector General_Exception    !  (7) Trap General exception
    vector Syscall              !  (8) Syscall
       ......
```

Figure 5. Specify a vector table to a user defined section

The point of Figure 5 lies on the first line where ".section" is used to define a non-standard section ".nds32_aa". While Andes standard vector table is usually placed in the nds32_init

section, the newly-created vector table is placed in .nds32_aa. As for “ax”, “a” denotes “allocable” and “x” means “executable”. Next, use a SaG file to specify the VMA of .nds32_aa to a fixed address, as shown in Figure 6 below:

```
   USER_SECTIONS .vector, .nds32_aa
   SDRAM 0x00000000 0x00800000 ; address base 0x00000000,
max_size=8M
   {
     EXEC 0x00000000
     {
       VAR _ILM_BASE = 0x00600000 ; ILM base address
       VAR _DLM_BASE = 0x00700000 ; DLM base address
       VAR _ILM_SIZE = 0x00010000 ; 64Kb
       VAR _DLM_SIZE = 0x00010000 ; 64Kb
       * (.vector)
       * (+RO,+RW,+ZI)
       STACK = 0x00800000
     }
   }
   QUICKSRAM 0x000010000
   {
    QEXEC 0x000010000
    {
        * (.nds32_aa)
    }
   }
```

Figure 6. Specify the VMA of the user-defined section to a fixed address

Figure 6 shows that the first address of the new vector table is 0x10000 – this is the address you need to direct IVBASE to upon completion of the boot up process. That way, whenever an interrupt comes in, it will be led to the new vector table.

3. Specify sections of .o or .a files to a fixed address

The above two examples have something in common: both specify some part of the program to a user-defined section. They have something different too: in the first example, a C function or variable is assigned to a user-defined section; in the second example, an assembly function is assigned. Both cases involve modifications on source code. However, for some application scenarios that there is no source code but object files or static library like .o or .a, can you specify sections of these files to a fixed address as well? Please reference Figure 7A below. It exemplifies how to specify the VMA of an object file's (hello.o's) read-only segments, including program code (.text section) and read-only data (.rodata section), to a fixed address (0x10000 in this case).

```
HELLOSRAM 0x000010000
{
    HEXEC 0x000010000
    {
        hello.o (+RO)
    }
}
```

Figure 7A. Specify sections of an .o file to a fixed address

Though you can follow Figure 7A to list every .o file in your SaG file, it is not recommended for the SaG file will become difficult to read and maintain. A better solution is to exclude certain .o files with a directive like "EXCLUDE_FILE" in GNU linker script. Andes SaG also supports excluding files. Figure 7B demonstrates how to specify all sections of *uart.o files to a fixed address and in the meanwhile keep other files unchanged. As there is inconsistency between the LMA and VMA of *uart.o sections, the assignment in Figure 7B

copies these sections from the LMA to VMA to resolve the problem.

```
   USER_SECTIONS .vector
   SDRAM 0x00000000 0x00800000 ; address base 0x00000000,
max_size=8M
   {
   EXEC 0x00000000
      {
       VAR _ILM_BASE = 0x00600000 ; ILM base address
       VAR _DLM_BASE = 0x00700000 ; DLM base address
       VAR _ILM_SIZE = 0x00010000 ; 64Kb
       VAR _DLM_SIZE = 0x00010000 ; 64Kb
       * (.vector)
       * EXCLUDE_FILE(*uart.o) (+RO,+RW,+ZI)
       STACK = 0x00800000
      }
   EXEC0 0x00003000
      {
       LOADADDR NEXT __uart_lmastart
       ADDR __uart_start
          *uart.o KEEP(+RO,+RW,+ZI)
       LOADADDR __uart_lmaend
      }
   }
```

Figure 7B. Andes SaG supports “EXCLUDE_FILE”

Andes Technology Corp.
2F, No.1, Li-Hsin First Road, Hsinchu Science Park,
Hsinchu City, Taiwan 300 R.O.C (SiSoft Research Center)
Tel: 886-3-6668300
Fax: 886-3-6668322
www.andestech.com

4. Fix inconsistency between LMA and VMA

The first three examples show how to locate the VMA of some program section at a fixed address, which is also the basic function of linker scripts. When the LMA and VMA of a section are not equal, embedded software engineers need to copy the section from the LMA to VMA during program initialization. Both the LMA and VMA are assigned to specific values in the linker script for C code to reference.

Andes SaG also can assign specific values to variables LMA and VMA. Instead of close arrangement, there is alignment constraint on data storage. For example, the first address at which a 4-byte word stored needs to be 4-byte aligned; a program with specific optimization flags, such as -O3 using Andes compiler, also requires the first address of its functions to be 4-byte aligned. For gaps between sections resulting from such an alignment constraint, Andes SaG normally can handle them well. However, in some sophisticated cases, you'll need to give Andes SaG more instructions to ensure it works properly.

Take Figure 7B as an example, the keyword "NEXT" in the statement "LOADADDR NEXT __uart_lmastart" is used to notify SaG that the value of this variable must be gained from the beginning of the next section rather than the end of the last section. To make it clear, let's look at Figure 8, the ELF file header generated after project compilation with the SaG file in Figure 7B:

```
    5 .sbss_w         00000024  00001c24  00001c24  00002c24  2**2
                      ALLOC
    6 .bss            00000010  00001c48  00001c48  00002c24  2**3
                      ALLOC
    7 .text_*uart.o   0000005c  00003000  00001c60  00003000  2**1
                      CONTENTS, ALLOC, LOAD, READONLY, CODE
    8 .data_*uart.o   00000004  0000305c  00001cbc  0000305c  2**0
```

Figure 8. ELF header

Figure 8 shows the LMA of ".text_*uart.o" is 0x1c60. Namely, the LMA of the last section (.bss) must end at 0x1c48+0x10=0x1c58, which echoes the usage of "NEXT" in Figure 7B. The keyword explicitly indicates that "__uart_lmastart" represents the LMA start address of ".text_*uart.o" rather than the LMA end address of ".bss".

Now, turn to Figure 9 in which "LMA_FORCE_ALIGN" is used instead. In this case, there is a conflict between a section with size of 2 byte (i.e. not a 4-byte multiple) and a following section whose VMA start address has to be 4-byte aligned. To resolve the conflict, the keyword "LMA_FORCE_ALIGN" is used to ensure the LMA and VMA are aligned with the same value.

```
   USER_SECTIONS .vector
   SDRAM 0x00000000 0x00800000 ; address base 0x00000000,
max_size=8M
   {
     EXEC 0x00000000
     {
       VAR _ILM_BASE = 0x00600000 ; ILM base address
       VAR _DLM_BASE = 0x00700000 ; DLM base address
       VAR _ILM_SIZE = 0x00010000 ; 64Kb
       VAR _DLM_SIZE = 0x00010000 ; 64Kb
       *(.vector)
       ADDR __code_start_usr
       ./src/* ./test/*(+RO LMA_FORCE_ALIGN)
       ./test/*(+RW LMA_FORCE_ALIGN)
       ADDR __code_end_usr
       STACK = 0x00800000
     }
     EXEC1 0x30000
     {
```

Andes Technology Corp.
2F, No.1, Li-Hsin First Road, Hsinchu Science Park,
Hsinchu City, Taiwan 300 R.O.C (SiSoft Research Center)
Tel: 886-3-6668300
Fax: 886-3-6668322
www.andestech.com


```
        LOADADDR NEXT __data_lmastart_usr
        ADDR __data_start_usr
        ./src/*(+RW LMA_FORCE_ALIGN)
        ADDR __data_end_usr
        ADDR __bss_start_usr
        *(+ZI LMA_FORCE_ALIGN)
        ADDR __bss_end_usr
      }
    }
```

Figure 9. "LMA_FORCE_ALIGN" example

Figure 10 is the ELF header generated after project compilation with the SaG file in Figure 9. In this figure, we can see that the LMA of the .sbss_b section is changed from 0x1d42 to 0x1d44.

```
  7 .sdata_b_.test* 00000002  00001d36  00001d36  00002d36  2**0
                    CONTENTS, ALLOC, LOAD, DATA
  8 .sdata_h_.src*  00000002  00030000  00001d40  00003000  2**1
                    CONTENTS, ALLOC, LOAD, DATA
  9 .sbss_b         00000002  00030004  00001d44  00003004  2**0
                    ALLOC
 10 .sbss_h         00000002  00030006  00001d46  00003004  2**1
                    ALLOC
```

Figure 10. Result of using "LMA_FORCE_ALIGN"

Andes Technology Corp.
2F, No.1, Li-Hsin First Road, Hsinchu Science Park,
Hsinchu City, Taiwan 300 R.O.C (SiSoft Research Center)
Tel: 886-3-6668300
Fax: 886-3-6668322
www.andestech.com

5. Conclusion

Andes provides programmers with the easy-to-use SaG tool to replace the complex linker scripts, thereby significantly improving their software development efficiency on AndesCore platform. This paper demonstrates the competent and friendly Andes SaG tool by real examples that solve engineering problems efficiently. As the SaG tool gets more and more powerful, we would like developers to get deeper understanding of its functional design – this is actually what the last example aims for. We look forward to seeing more developers mastering Andes SaG tool and hearing more about how this tool helps to tackle software development problems agilely.

Reference document:

1: Andes Programming Guide for BSP v3.2.1 (Linker Script Generator chapter)

2: The GNU Linker Manual

Andes Technology Corp.
2F, No.1, Li-Hsin First Road, Hsinchu Science Park,
Hsinchu City, Taiwan 300 R.O.C (SiSoft Research Center)
Tel: 886-3-6668300
Fax: 886-3-6668322
www.andestech.com

……About Andes……

Andes Technology Corporation was founded in Hsinchu Science Park, Taiwan in 2005 to develop innovative high-performance/low-power 32-bit processor cores and its associated development environment to serve worldwide rapidly-growing embedded system applications. It delivers the best super low power CPU cores with integrated development environment and associated software and hardware solutions for SoC development.

In order to meet demanding requirements of today's electronic devices, Andes delivers configurable software/hardware IP and scalable platforms to respond to customers' needs for quality products and faster time-to-market. Andes' comprehensive CPU includes entry-level, mid-range, high-end, extensible and security families to address full range of embedded electronics products, especially for connected, smart and green applications.

For more information about Andes Technology, please visit http://www.andestech.com/