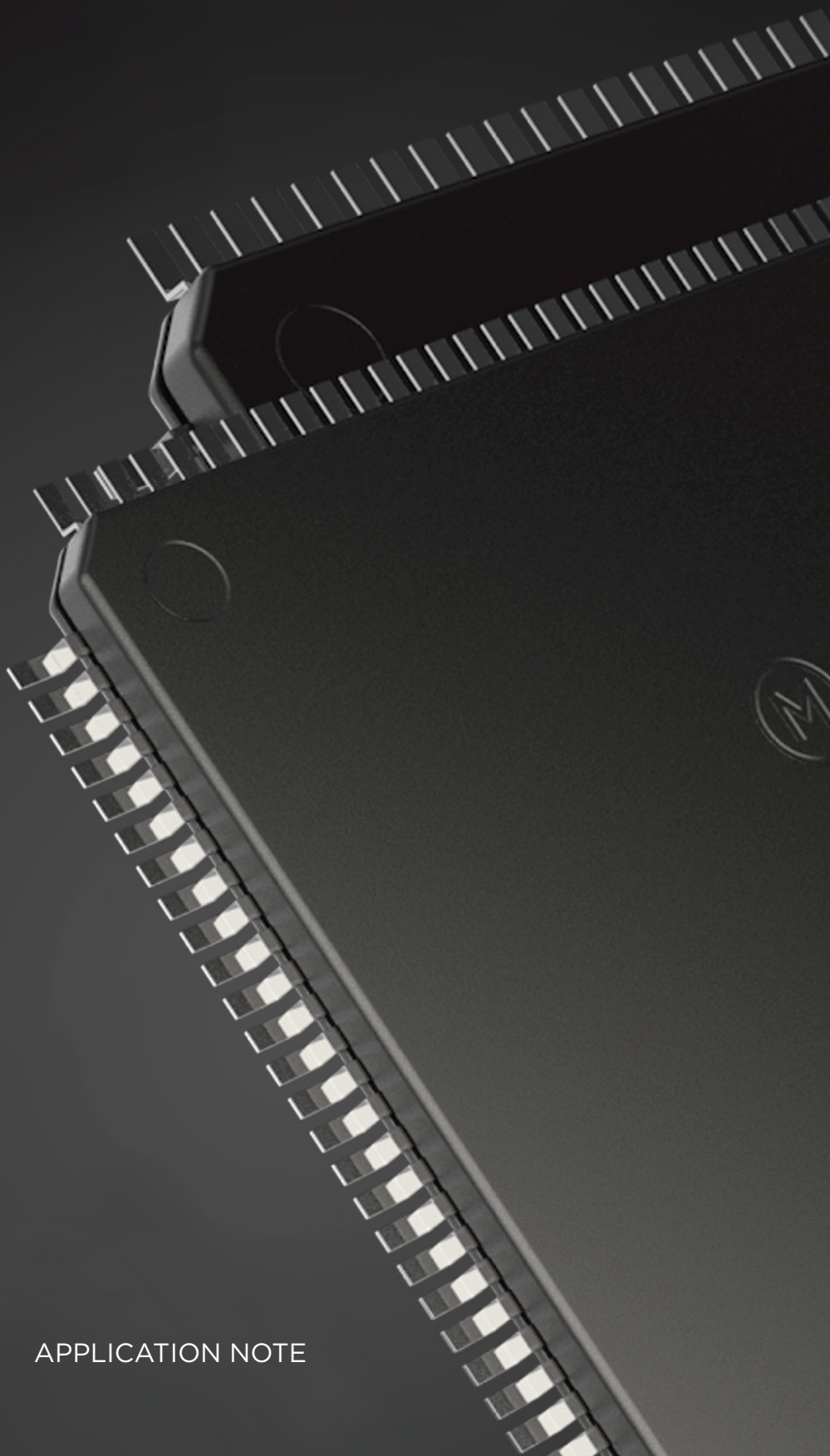
TASKING®

TRICORE COMPILER INTERNAL GENERATED SECTIONS



APPLICATION NOTE

TRICORE COMPILER INTERNAL GENERATED SECTIONS

INTRODUCTION

Besides normal code and data sections, the compiler also generates internal sections for various reasons. This application note provides more details about those sections and their content. The following list shows some of these compilers generated internal sections and describes how they can be utilized.

1. Initializers: *.<n>.ini.*
2. String literals: *.<n>.str.*
3. Compound literals: *.<n>.lit.*
4. Constants: *.<n>.cnt.*
5. Switch lookup tables: *.<n>.lkp.*
6. Switch jump tables: *.<n>.jmp.*
7. Sections like .bss.file_1._999001_test / .zbss.file_1._999001_test / .data.file_1._999001_test / .zdata.file_1._999001_test

1. INITIALIZERS: *.<n>.ini.* Sections

These types of constant sections are generated when a local array is initialized. E.g.

file_1.c

```
#include <stdio.h>
int main(void)
{
    //Generates Initializers: *.<n>.ini.* Sections
    char              a[]    = "text";
    int               b[]    = {1,2};
    long int   c[]    = {3,4};
    short int  d[]    = {5,6};
    double            e[]    = {0.1, 0.2};

    printf("%c\n", a[2]);

    for (int i = 0; i < sizeof(b); i++)
    {
        printf("%d\n", b[i]);
        printf("%ld\n", c[i]);
        printf("%u\n", d[i]);
        printf("%f\n", e[i]);
    }
}
```

file_1.lsl

```
section_layout mpe:vtc:linear
{
    group Initializers_Sections( ordered,contiguous, run_addr = mem:mpe:pflash0)
    {
      select "*.ini";
    }
}
```

Invocation: `cctc -Ctc27x -O0 -Wc-N0 -Wl-dfile_1.lsl file_1.c`
Referencing the generated .map file you can see the new generated sections.

```
+-----------------------------------------------------------------------------------------------------------------------------------------------------------+
| Chip        | Group                | Section                                | Size (MAU) | Space addr | Chip addr  | Alignment  |
|===========================================================================================================================================================|
| mpe:pflash0 | Initializers_Sections | .rodata.file_1..2.ini (3)              | 0x00000008 | 0x80000024 | 0x00000024 | 0x00000002 |
| mpe:pflash0 | Initializers_Sections | .rodata.file_1..3.ini (4)              | 0x00000008 | 0x8000002c | 0x0000002c | 0x00000002 |
| mpe:pflash0 | Initializers_Sections | .rodata.file_1..4.ini (5)              | 0x00000004 | 0x80000034 | 0x00000034 | 0x00000002 |
| mpe:pflash0 | Initializers_Sections | .rodata.file_1..5.ini (6)              | 0x00000008 | 0x80000038 | 0x00000038 | 0x00000002 |
| mpe:pflash0 | Initializers_Sections | .rodata.file_1..1.ini (2)              | 0x00000005 | 0x80000040 | 0x00000040 | 0x00000001 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------+
```

2. String literals: *.<n>.str.*

The compiler generates a constant string literal section(s) for a string like:

file_1.c

```
#include<stdio.h>
int main()
{
    /* Generates String Literals: *.<n>.str.* Sections */
    printf ("Hello World");
    printf ("Welcome to Tricore VX Tool Set");
    return 0;
}
```

file_1.lsl

```
section_layout mpe:vtc:linear
{
    group Strings_Literals( ordered,contiguous, run_addr = mem:mpe:pflash0 )
    {
    select "*.str";
    }
}
```

Invocation: `cctc -Ctc27x -O0 -Wc-N0 -dfile_1.lsl file_1.c`
Referencing the generated .map file you can see the following entries.

```
+-----------------------------------------------------------------------------------------------------------------------------+
| Chip        | Group           | Section                              | Size (MAU) | Space addr | Chip addr  | Alignment  |
|=============================================================================================================================|
| mpe:pflash0 | Strings_Literals | .rodata.file_1..1.str (2)           | 0x0000000c | 0x80000024 | 0x00000024 | 0x00000001 |
| mpe:pflash0 | Strings_Literals | .rodata.file_1..2.str (3)           | 0x0000001f | 0x80000030 | 0x00000030 | 0x00000001 |
+-----------------------------------------------------------------------------------------------------------------------------+
```

3. Compound literals: *.<n>.lit.*

Sections like .data.file_1..1.lit can be generated when compound literals are used. E.g.

file_1.c

```
#include<stdio.h>
/* Generates String Literals: *.<n>.str.* Sections */
char **foo = (char *[]) { "x", "y", "z" };

int main ()
{
    return 0;
}
```

file_1.lsl

```
section_layout mpe:vtc:linear
{
    group Compound_Liternals( ordered,contiguous, run_addr = mem:mpe:dspr0 )
    {
    select "*.lit";
    }
}
```

Invocation: `cctc -Ctc27x -O0 -Wc-N0 -Wl-OC -dfile_1.lsl file_1.c`
Referencing the generated .map file you can see the following entries.

```
+-----------------------------------------------------------------------------------------------------------------------------+
| Chip        | Group             | Section                            | Size (MAU) | Space addr | Chip addr  | Alignment  |
|=============================================================================================================================|
| mpe:dspr0   | Compound_Liternals | .data.file_1..1.lit (5)           | 0x0000000c | 0x70000000 | 0x0        | 0x00000004 |
| mpe:pflash0 |                   | [.data.file_1..1.lit] (199)        | 0x0000000c | 0x80000024 | 0x00000024 | 0x00000004 |
| mpe:pflash0 |                   | [.data.file_1.foo] (200)           | 0x00000004 | 0x80000030 | 0x00000030 | 0x00000004 |
| mpe:lmuram  |                   | .data.file_1.foo (6)               | 0x00000004 | 0x90000000 | 0x0        | 0x00000004 |
+-----------------------------------------------------------------------------------------------------------------------------+
```

4. Constants: *.<n>.cnt.*

The compiler generates a near addressable section like .zrodata.file_1..3.cnt which includes the initialization value of a variable initialized during run-time like E.g.

file_1.

```
#include<stdio.h>

near int var_1;
near int var_2;

far int var_3;
far int var_4;

int main()
{
    // Generating Constants: *.<n>.cnt.* Sections
    var_1= 0x10001;
    var_2= 0x10002;

    var_3= 0x10004;
    var_4= 0x10005;

    return 0;
}
```

file_1.lsl

```
section_layout mpe:vtc:abs18
{
    group Constants( ordered,contiguous, run_addr = mem:mpe:pflash0 )
    {
    select "*.cnt";
    }
}
```

Invocation: `cctc -Ctc27x -O0 -Wc-N8 -dfile_1.lsl file_1.c`
Reference the generated .map file to see the following entries.

```
+-----------------------------------------------------------------------------------------------------------------------------+
| Chip        | Group        | Section                                | Size (MAU) | Space addr | Chip addr  | Alignment  |
|=============================================================================================================================|
| mpe:pflash0 | Constants    | .zrodata.file_1..1.cnt (1)             | 0x00000004 | 0x80000024 | 0x00000024 | 0x00000002 |
| mpe:pflash0 | Constants    | .zrodata.file_1..2.cnt (2)             | 0x00000004 | 0x80000028 | 0x00000028 | 0x00000002 |
| mpe:pflash0 | Constants    | .zrodata.file_1..3.cnt (3)             | 0x00000004 | 0x8000002c | 0x0000002c | 0x00000002 |
| mpe:pflash0 | Constants    | .zrodata.file_1..4.cnt (4)             | 0x00000004 | 0x80000030 | 0x00000030 | 0x00000002 |
| mpe:lmuram  |              | .zbss.file_1.var_1 (6)                 | 0x00000004 | 0x90000000 | 0x0        | 0x00000002 |
| mpe:lmuram  |              | .zbss.file_1.var_2 (7)                 | 0x00000004 | 0x90000004 | 0x00000004 | 0x00000002 |
+-----------------------------------------------------------------------------------------------------------------------------+
```

Hint: You can prevent these sections by using the C compiler option --immediate-in-code or --default-near-size=0(-N0).

If the default near allocation is disabled, C compiler does not generate an alternative .rodata section including the initialization value instead of the near addressable .zrodata, because a far data access to load the value into a register is not efficient anymore.

5. Switch lookup tables: *.<n>.lkp

The compiler generates a section like .rodata.file_1..6.lkp when a lookup table is used for a switch case. E.g.

file_1.c

```
#include <stdio.h>
int main()
{
    int i=2;
    //Generating Switch lookup tables: *.<n>.lkp.* sections
#pragma switch lookup

    switch (i)
    {
    case 1:
        printf("Case1 ");
        break;
    case 2:
        printf("Case2 ");
        break;
    case 3:
        printf("Case3 ");
        break;
    case 4:
        printf("Case4 ");
        break;
    default:
        printf("Default ");
    }
#pragma switch restore

    return 0;
}
```

file_1.lsl

```
section_layout mpe:vtc:linear
{
    group Switch_Lookup_Table( ordered,contiguous, run_addr = mem:mpe:pflash0 )
    {
    select "*.lkp";
    }
}
```

Invocation: `cctc -Ctc27x -O0 -dfile_1.lsl file_1.c`
Refer to the generated .map file to see the following entries.

```
+-----------------------------------------------------------------------------------------------------------------------------------+
| Chip         | Group              | Section                                | Size (MAU) | Space addr | Chip addr  | Alignment  |
|===================================================================================================================================|
| mpe:pflash0 | Switch_Lookup_Table | .rodata.file_1..6.lkp (2)              | 0x00000020 | 0x80000024 | 0x00000024 | 0x00000004 |
+-----------------------------------------------------------------------------------------------------------------------------------+
```

Hint: You can change the switch case behavior using the C compiler option -switch or #pragma switch to have the compiler use a jump chain or a jump table instead of the lookup table.

6. Switch jump tables: *.<n>.jmp

The compiler generates a section like .rodata.file_1..4.jmp if a jump table is used for a switch case. E.g.

file_1.c

```
#include <stdio.h>
int main()
{
    int i=2;
    //Generating Switch Jump tables: *.<n>.jmp.* sections
#pragma switch jumptab

    switch (i)
    {
    case 1:
        printf("Case1 ");
        break;
    case 2:
        printf("Case2 ");
        break;
    case 3:
        printf("Case3 ");
        break;
    case 4:
        printf("Case4 ");
        break;
    default:
        printf("Default ");
  }
#pragma switch restore

  return 0;
}
```

file_1.lsl

```
section_layout mpe:vtc:linear
{
    group Switch_Jump_Table( ordered,contiguous, run_addr = mem:mpe:pflash0 )
    {
        select "*.jmp";
    }
}
```

Invocation: `cctc -Ctc27x -O0 -dfile_1.lsl file_1.c`
Refer to the generated .map file to see the following entries.

```
+-----------------------------------------------------------------------------------------------------------------------------+
| Chip         | Group             | Section                                | Size (MAU) | Space addr | Chip addr  | Alignment  |
|=============================================================================================================================|
| mpe:pflash0 | Switch_Jump_Table | .rodata.file_1..6.jmp (2)              | 0x00000010 | 0x80000024 | 0x00000024 | 0x00000004 |
+-----------------------------------------------------------------------------------------------------------------------------+
```

Hint: You can change the switch case behavior using the C compiler option -switch or #pragma switch to have the compiler use a lookup table or a jump table instead of the lookup table.

7. Local static variable section *._999001_test

Sections like .bss.file_1._999001_test / .zbss.file_1._999001_test / .data.file_1._999001_test / .zdata.file_1._999001_test do include function local static variables. E.g.

file_1.c

```
#include <stdio.h>

int main()
{
    static unsigned int a;
    static unsigned int b =20;
    __far static unsigned int c;
    __far static unsigned int d =20;
    a=b;
    c=d;
    printf("Value=%d %d",a,c);
    return 0;
}
```

file_1.lsl

```
section_layout mpe:vtc:linear
{
    group Special_Section_1( ordered,contiguous, run_addr = mem:mpe:lmuram )
    {
        select ".bss.file_1._999003_c";
        select ".data.file_1._999004_d";
    }
}

section_layout mpe:vtc:abs18
{
    group Special_Section( ordered,contiguous, run_addr = mem:mpe:lmuram )
    {
        select ".z*._999*";
    }
}
```

Invocation: `cctc -Ctc27x -O0 -Wc-N8 -dfile_1.lsl file_1.c`
Refer to the generated .map file to see the following entries.

```
+ Space mpe:vtc:linear
+-------------------------------------------------------------------------------------------------------------------------------------------+
| Chip         | Group              | Section                                  | Size (MAU) | Space addr | Chip addr  | Alignment  |
|===========================================================================================================================================|
| mpe:lmuram   | Special_Section_1  | .bss.file_1._999003_c (4)                | 0x00000004 | 0x90000008 | 0x00000008 | 0x00000002 |
| mpe:lmuram   | Special_Section_1  | .data.file_1._999004_d (5)               | 0x00000004 | 0x9000000c | 0x0000000c | 0x00000002 |
+-------------------------------------------------------------------------------------------------------------------------------------------+

+ Space mpe:vtc:abs18
+-------------------------------------------------------------------------------------------------------------------------------------------+
| Chip         | Group              | Section                                  | Size (MAU) | Space addr | Chip addr  | Alignment  |
|===========================================================================================================================================|
| mpe:lmuram   | Special_Section    | .zbss.file_1._999001_a (2)               | 0x00000004 | 0x90000000 | 0x0        | 0x00000002 |
| mpe:lmuram   | Special_Section    | .zdata.file_1._999002_b (3)              | 0x00000004 | 0x90000004 | 0x00000004 | 0x00000002 |
+-------------------------------------------------------------------------------------------------------------------------------------------+
```

Section Renaming is possible for such internal auto generated sections.

Additional Information: In order to avoid any confusion, it is suggested to place these compiler internal generated sections selection at the end of your linker script - after all other sections are selected.