

# Application note

---

## N32H7xx Methods For Running In TCM Application note

---

### Introduction

The purpose of this document is to help users quickly learn how to run code on TCM for the N32H7xx series microcontrollers (MCUs) across different development environments.

## Contents

<b>Contents.....</b>	<b>1</b>
<b>1 Overview.....</b>	<b>2</b>
<b>2 Development Environment .....</b>	<b>2</b>
2.1 Software.....	2
2.2 Hardware .....	2
2.3 Code Location .....	2
2.4 Resource Overview.....	2
<b>3 KEIL Configuration Notes.....</b>	<b>3</b>
3.1 Scatter Load Configuration Notes .....	3
<b>4 IAR Configuration Notes .....</b>	<b>5</b>
4.1 Scatter Load Configuration Notes .....	5
<b>5 VSCode+GCC Configuration Notes.....</b>	<b>6</b>
5.1 Boot File Section .....	6
5.2 .ld File Section.....	7
<b>6 Interrupt Vector Table Copy .....</b>	<b>10</b>
<b>7 Version history .....</b>	<b>11</b>
<b>8 Notice .....</b>	<b>12</b>

## 1 Overview

This document uses the N32H76x series MCU as an example to illustrate methods for enhancing code execution efficiency in a Windows environment. It demonstrates how to deploy code within the internal TCMSRAM using different compilers.

## 2 Development Environment

### 2.1 Software

- 1) KEIL MDK-ARM V5.34
- 2) VSCode + GCC
- 3) IAR EWARM 8.50.1

### 2.2 Hardware

- 1) Core board N32H787XIB7-HMI V1.1

### 2.3 Code Location

The code section for this application note is released with the SDK. The project directory structure is as follows (using Nations.N32H76x\_78x\_Library.1.2.0 as an example):

Nations.N32H7xx_Library.1.2.0 > projects > n32h7xx_EVAL > applications >		
名称	修改日期	类型
 run_In_TCM	2026/1/7 15:35	文件夹

### 2.4 Resource Overview

#### ➤ TCMSRAM

TCMSRAM is constructed from SRAM memory and comprises the following types: TCM (ITCM, D0TCM, D1TCM) used as the Cortex-M7's tightly coupled memory, and AXISRAM (AXISRAM2/3) serving as on-chip general-purpose SRAM.

Users may configure the size of ITCM and DTCM in 128KB increments; the remaining memory space is automatically allocated to the two AXI SRAM2/3 regions.

## 3 KEIL Configuration Notes

### 3.1 Scatter Load Configuration Notes

- **Keil Environment:** The following example illustrates a 512K ITCM + 512K DTCM configuration

```
LR_IROM1 0x15000000 0x00100000 {    ; load region size_region
  ER_IROM1 0x15000000 0x00100000 {    ; load address = execution address
    *.o (RESET, +First)
    *(InRoot$$Sections)
    startup_n32h7*.o (+R0)
    system_n32h7xx.o (+R0)
    n32h7xx_it.o (+R0)
  }
}

;ITCM
RW_ITCM 0x00000400 0x0007FC00 {    ; 512K ITCM   offset 0x400 for VECTOR TABLE
  .ANY (+R0)
  .ANY (+X0)
}

;AXI SRAM
RW_AXI_SRAM 0x24000000 0x00020000 {    ; 128K AXI-SRAM  stack
  .ANY (+RW +ZI)
  *(STACK)
}

;DTCM
RW_DTCM 0x20000000 0x00080000 {    ; 512K DTCM
  .ANY (+RW +ZI)
}

}
```

LR\_IROM1 defines the loading region as 0x15000000 with a size of 1M; this region indicates that all code is loaded from this area:

1. ER\_IROM1 defines an execution region at address 0x15000000 with a size of 1M; within this region, the reset vector table, standard library initialisation code, and the startup\_n32h7\*.o, system\_n32h7xx.o, and n32h7xx\_it.o code execute;
2. RW\_ITCM defines an additional execution region named ITCM, located at address 0x00000400 with a size of 512K;

indicating that all code except startup\_n32h7\*.o, system\_n32h7xx.o, and n32h7xx\_it.o shall run within this region, whose size is user-configurable;

3. RW\_AXI\_SRAM defines the read/write data region as AXI SRAM, located at address 0x24000000 with a size of 128K; (user-configurable), indicating the stack region is defined within this area;
4. RW\_DTCM defines an additional read/write data region as DTCM, located at address 0x20000000 with a size of 512K; (user-configurable). This indicates that all global variables within the project are defined within this region.

*Note:*

1. *ITCM and DTCM can be freely configured within the code;*
2. *Do not define the stack area within the DTCM region until the TCM configuration is complete;*
3. *This scatter load example uses 512K ITCM + 512K DTCM. Clients may adjust this according to their actual requirements. For TCM configuration details, please refer to the CN\_UG\_N32H7xx\_TCM\_User\_Guide.pdf.*

## 4 IAR Configuration Notes

### 4.1 Scatter Load Configuration Notes

```
34  define block ITCM_CODE      { readwrite code };
35  define block ITCM_DATA      { readwrite data }
36  |
37  |   except {
38  |   |   section .bss,
39  |   |   section .data
40  |   };
41  do not initialize { section .noinit };
42  initialize by copy { readwrite };
43  initialize by copy with packing = none { readonly }
44  |   except {
45  |   |   section .intvec,
46  |   |   ro object system_n32h7*.o,
47  |   |   ro object startup_n32h7*.o,
48  |   |   ro object n32h7xx_it.o,
49  |   |   ro object *.a
50  |   };
51
52  if (isdefinedsymbol(__USE_DLIB_PERTHREAD))
53  {
54  |   // Required in a multi-threaded application
55  |   initialize by copy with packing = none { section __DLIB_PERTHREAD };
56  |   }
57
58  place at address mem:__ICFEDIT_intvec_start__ { readonly section .intvec };
59
60  place in EITCM_region { block ITCM_CODE, block ITCM_DATA};
61  place in IROM_region { readonly };
62  place in IRAM_region { readwrite, block CSTACK, block PROC_STACK, block HEAP };
63
```

IROM\_region denotes the definition of a load region at 0x15000000 with a size of 1M, indicating that all code is loaded from this region; IRAM\_region denotes the read/write data region; EITCM\_region denotes the read/write execution region.

1. Except for the reset vector table, standard library initialisation code, and the code from startup\_n32h7\*.o, system\_n32h7\*.o, n32h7xx\_it.o, which are not copied to TCM for execution, all other user code is copied to RAM for execution by default.
2. Place the copied code from step 1 (with attributes readwrite code and readwrite data) in ITCM, at address 0x00000400, with a size of 1M;

## 5 VSCode+GCC Configuration Notes

1. The GCC environment demonstrates solely how to store code in FLASH and transfer it to SRAM for execution upon power-up.
2. This document primarily explains how to transfer code stored in FLASH to SRAM for execution under the VSCode+GCC environment. For configuration and detailed guidance on the VSCode+GCC environment, please refer to the Application Notes: AN\_N32H7xx\_GCC Development Environment Application Note.

### 5.1 Boot File Section

Additional code must be incorporated into the boot file(startup\_n32h76x\_ITCM\_gcc.s) to copy data/code from FLASH to designated SRAM locations.

1. Define the segment names for each segment file in the .ld file within the boot file.

```
/* start address for the initialization values of the .data section.
defined in linker script */
.word _sidata
/* start address for the .data section. defined in linker script */
.word _sdata
/* end address for the .data section. defined in linker script */
.word _edata
/* start address for the .bss section. defined in linker script */
.word _sbss
/* end address for the .bss section. defined in linker script */
.word _ebss
/*some param defined in linker script*/
.word _siram_code
.word _s_ram_code
.word _e_ram_code
.word _fp_flash_rodata
.word _fp_s_rodata
.word _fp_e_rodata
.word __exidx_start
.word __exidx_end
.word _fp_exidx
.word __preinit_array_start
.word __preinit_array_end
.word _sipreinit_array
.word __init_array_start
.word __init_array_end
.word _siinit_array
.word __fini_array_start
.word __fini_array_end
.word _sifini_array
```

2. Define a function macro to copy data segments to SRAM.

```

/* 宏定义：统一复制过程 */
.macro COPY_SECTION section_start, section_end, load_address
    ldr r0, =\section_start /* RAM目标地址 */
    ldr r2, =\load_address /* Flash源地址 */
    ldr r1, =\section_end /* RAM结束地址 */
    subs r3, r1, r0 /* 计算长度 */
    ble 1f /* 如果长度为0则跳过 */
/* 复制循环（每次4字节） */
0: ldr r4, [r2], #4
    str r4, [r0], #4
    subs r3, r3, #4
    bgt 0b
1:
.endm

```

- Place the data copy operation after SystemInit, as ITCM/DTCM initialisation occurs within the SystemInit function. Access to this region is prohibited prior to initialisation.

```

124 /* Call the clock system initialization function.*/
125 bl SystemInit
126
127 /* Copy all the segments that need to be initialized and select whether to block or not according to the configuration of the ld file */
128 COPY_SECTION _s_ram_code, _e_ram_code, _siram_code /* code snippet */
129 COPY_SECTION _fp_s_rodata, _fp_e_rodata, _fp_flash_rodata /* Read only data */
130 COPY_SECTION _sdata, _edata, _sidata /* data segment */
131 /* COPY_SECTION __exidx_start, __exidx_end, _fp_exidx ARM Exception Index */
132 /* COPY_SECTION __preinit_array_start, __preinit_array_end, _spreinit_array
133 COPY_SECTION __init_array_start, __init_array_end, _sinit_array
134 COPY_SECTION __fini_array_start, __fini_array_end, _sfini_array
135 COPY_SECTION _sirq_vector_ram, _eirq_vector_ram, _sivector Interrupt Vector Table */
136 /* Call static constructors */
137 bl __libc_init_array
138 /* Call the application's entry point.*/
139 bl main

```

## 5.2 .ld File Section

The .ld file(n32h76x\_ITCM\_flash.ld) defines code/data locations. Key points include:

- Defining block regions

\_estack defines the stack top position; \_Min\_Heap\_Size/\_Min\_Stack\_Size define stack size;

This routine partitions SRAM into four segments: AXI\_SRAM (128KB), ITCM\_SRAM (512KB), DTCM\_SRAM (512KB), and AHB\_SRAM (352KB). As the execution code is placed in ITCM\_SRAM for execution in this example, the first 1KB of ITCM\_SRAM is reserved for storing the interrupt vector table. The same principle applies when executing from other regions.

FLASH is utilised for storing various data and code.



```

51  /* Highest address of the user mode stack */
52  _estack = 0x30058000;    /* end of RAM */
53  /* Generate a link error if heap and stack don't fit into RAM */
54  _Min_Heap_Size = 0x2000;    /* required amount of heap */
55  _Min_Stack_Size = 0x1000;    /* required amount of stack */
56
57  /* Specify the memory areas */
58  MEMORY
59  {
60      AXI_SRAM (xrw): ORIGIN = 0x24000000, LENGTH = 0x00020000    /* 128KB */
61      ITCM_SRAM (xrw): ORIGIN = 0x00000400, LENGTH = 0x0007FC00    /* 512KB - 1KB */
62      DTCM_SRAM (xrw): ORIGIN = 0x20000000, LENGTH = 0x00080000    /* 512KB */
63      AHB_SRAM (xrw): ORIGIN = 0x30000000, LENGTH = 0x58000    /* 352KB */
64      FLASH (rx) : ORIGIN = 0x15000000, LENGTH = 0x1E0000    /* 2*1024K - 128K */
65  }

```

## 2. Defining Storage and Execution Locations for Code

The interrupt vector table is stored and executed in FLASH by default. After all code is transferred to designated SRAM locations, a jump to the main function immediately executes the transfer of the vector table to SRAM. See Chapter 6 for details.

```

69  /* The startup code goes first into FLASH */
70  .isr_vector :
71  {
72      . = ALIGN(4);
73      KEEP(*(.isr_vector)) /* Startup code */
74      . = ALIGN(4);
75  } >FLASH

```

System initialisation functions and boot files are placed in FLASH for execution, handling clock configuration and code/data relocation.

All other executable code is placed in the ITCM\_SRAM region, while data resides in AHB\_SRAM.

Refer to the diagram below: '>FLASH' denotes storage and execution in FLASH; '>ITCM\_SRAM AT > FLASH' indicates storage in FLASH with execution in the ITCM\_SRAM region; other cases follow the same principle.

The '.text' section signifies all executable code resides in this region, stored in FLASH and executed in ITCM\_SRAM; However, the '.text\_flash' section takes precedence over '.text'. Consequently, code from the 'startup\_n32h76x\_ITCM\_gcc' and 'system\_n32h7xx' files is stored in FLASH and executed from FLASH, whilst the remaining code executes from ITCM\_SRAM.

```
78  .init :
79  {
80      . = ALIGN(4);
81      KEEP(*(.init))
82      . = ALIGN(4);
83  } >FLASH
84
85  .text_flash :
86  {
87      . = ALIGN(4);
88      build/startup_n32h76x_ITCM_gcc.o(.text*)
89      build/startup_n32h76x_ITCM_gcc.o(.rodata*)
90      build/system_n32h7xx.o(.text*)
91      build/system_n32h7xx.o(.rodata*)
92      . = ALIGN(4);
93  } >FLASH
94
95  /* The program code and other data goes into ram */
96  .text :
97  {
98      . = ALIGN(4);
99      _s_ram_code = .; /* section start address*/
100     *(.text)          /* .text sections (code) */
101     *(.text*)         /* .text* sections (code) */
102     *(.glue_7)        /* glue arm to thumb code */
103     *(.glue_7t)       /* glue thumb to arm code */
104     *(.eh_frame)
105
106     KEEP (*(.init))
107     KEEP (*(.fini))
108
109     . = ALIGN(4);
110     _e_ram_code = .; /* define a global symbols at end of code */
111 } >ITCM_SRAM AT > FLASH
112
113 _siram_code = LOADADDR(.text);
```

## 6 Interrupt Vector Table Copy

As the vector table resides in FLASH, for code acceleration, the initialisation process requires copying the FLASH vector table to the acceleration region. The function is as follows:

### ➤ TCMSRAM

```
#define      ITCM_BASE      (0x00000000UL)
void CopyVectTableToITCM(void)
{
    volatile uint32_t* pSrcVect = (uint32_t*)(FLASH_BASE);
    volatile uint32_t* pDestVect = (uint32_t*)(ITCM_BASE);
    uint32_t numVECT = 0x400 / 4;
    for(uint32_t i = 0; i < numVECT; i++)
    {
        pDestVect[i] = pSrcVect[i];
    }
    #ifndef CORE_CM4
        pDestVect[15] = pSrcVect[250];
    #endif
    SCB->VTOR = ITCM_BASE;
    __ISB();
    __DSB();
}
```

## 7 Version history

Date	Version	Modify
2025.12.29	V1.2.0	Initial version

## 8 Notice

This document is the exclusive property of NSING TECHNOLOGIES PTE. LTD. (Hereinafter referred to as NSING). This document, and the product of NSING described herein (Hereinafter referred to as the Product) are owned by NSING under the laws and treaties of Republic of Singapore and other applicable jurisdictions worldwide. The intellectual properties of the product belong to NSING Technologies Inc. and NSING Technologies Inc. does not grant any third party any license under its patents, copyrights, trademarks, or other intellectual property rights. Names and brands of third party may be mentioned or referred thereto (if any) for identification purposes only. NSING reserves the right to make changes, corrections, enhancements, modifications, and improvements to this document at any time without notice. Please contact NSING and obtain the latest version of this document before placing orders. Although NSING has attempted to provide accurate and reliable information, NSING assumes no responsibility for the accuracy and reliability of this document. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. In no event shall NSING be liable for any direct, indirect, incidental, special, exemplary, or consequential damages arising in any way out of the use of this document or the Product. NSING Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, Insecure Usage'. Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, all types of safety devices, and other applications intended to supporter sustain life. All Insecure Usage shall be made at user's risk. User shall indemnify NSING and hold NSING harmless from and against all claims, costs, damages, and other liabilities, arising from or related to any customer's Insecure Usage Any express or implied warranty with regard to this document or the Product, including, but not limited to. The warranties of merchantability, fitness for a particular purpose and non-infringement are disclaimed to the fullest extent permitted by law. Unless otherwise explicitly permitted by NSING, anyone may not use, duplicate, modify, transcribe or otherwise distribute this document for any purposes, in whole or in part.