

Figure: Transfer path for SMC access via Cache

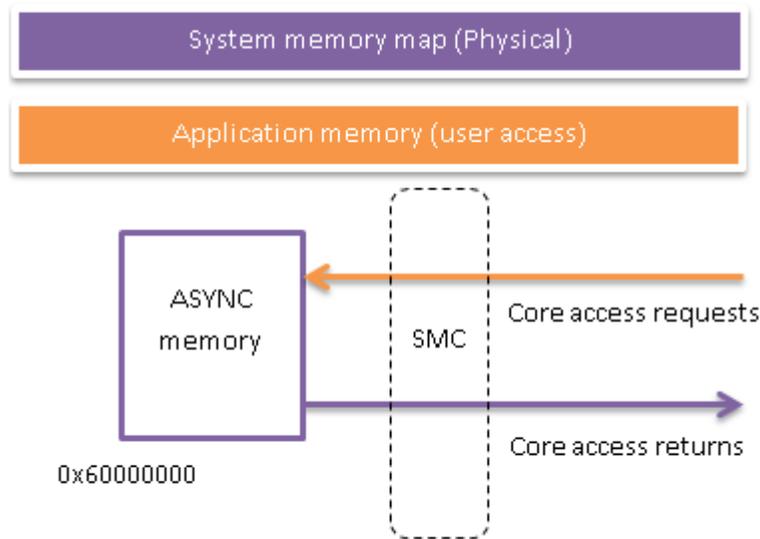
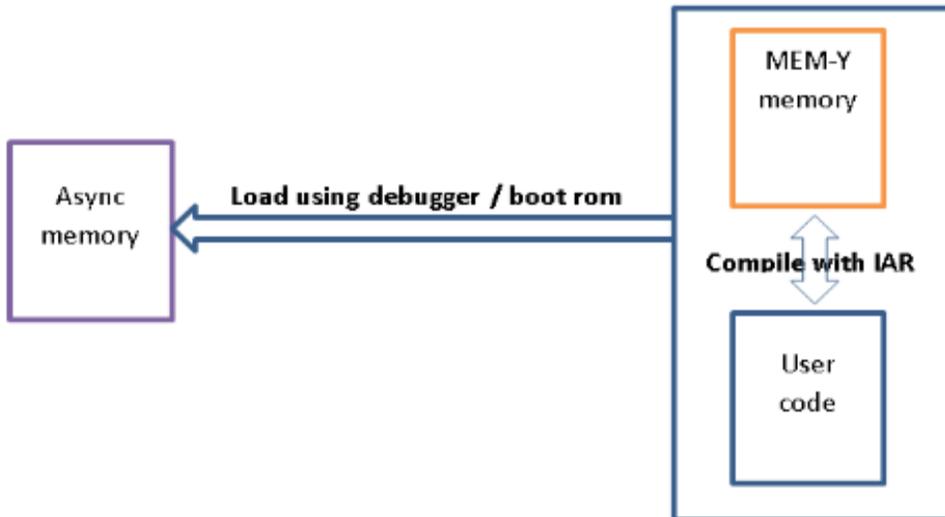


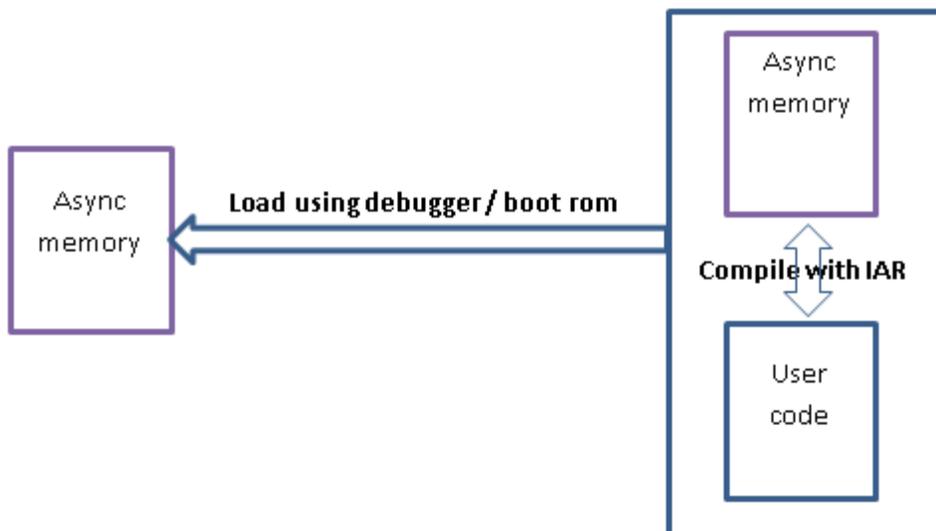
Figure: Transfer path for SMC access without using Cache

Developing application

Code building and loading for case of Cache



Code building and loading for case without Cache



Developing SMC connected SRAM based projects without using Cache

SMC connected SRAM can be accessed directly, similar to internal SRAM. Here is an example Script that can be added in IAR ICF

```
// setup linker file
define symbol ASYNC_BASE      = 0x60000000;
define symbol ASYNC_SIZE     = 0x00080000;
define region ASYNC_code_region = mem:[from ASYNC_BASE size ASYNC_SIZE ];
place in ASYNC_code_region    { section .async_code};
```

```
// test code mapped in SMC SRAM
#pragma location = ". async_code "
void CallSomeCode() {...}
```

Developing SMC connected SRAM based projects using Cache

As mentioned in the first page, developing SRAM based projects that support Cache has requirement of: different compile memory (0x1900_0000) and load memory (0x6000_0000). IAR Tools support a way to assist code development in this scenario. The idea is to inform the Linker that the code for MEMORY locations are made as 'initialize manually', but do not perform the actual initialization (memory translation in hardware takes care of that). Then, have the initialization source copy inside the Async memory itself that is retained for the internal access.

```
// setup linker file
define symbol ASYNC_BASE      = 0x60000000;
define symbol ASYNC_SIZE     = 0x00080000;
define symbol MEMORY_BASE    = 0x19000000;
define symbol MEMORY_SIZE    = 0x01000000;
define region MEMORY_code_region = mem:[from MEMORY_BASE size ASYNC_SIZE ];
define region MEMORY_INIT     = mem:[from ASYNC_BASE size ASYNC_SIZE];
place in MEMORY_code_region   { section .memy_code };
place in MEMORY_INIT         { section .memy_code_init };
initialize manually          { section .memy_code };
```

```
// test code mapped in SMC SRAM
#pragma location = ".memy_code"
void CallSomeCode() {...}
```

```
// check the Map file after link
"P4":
0x3c
P4 s2      0x19000000 0x3c <Init block>
.memy_code  initd 0x19000000 0x3c Blinky.o [1]
- 0x1900003c 0x3c
"P5":
0x3c
Initializer bytes ro data 0x60000000 0x3c <for P4 s2>
- 0x6000003c 0x3c
```

Figure: Test example screenshot from IAR

The screenshot displays the IAR Embedded Workbench interface with two main windows: the source code editor and the memory/disassembly view.

Source Code Editor: The code defines a macro `PLACE_FLASH` and a function `CallSomeFlashCode`. The function uses a `while(1)` loop with nested `for` loops to toggle an LED. The `for` loop `for(Cnt=0;Cnt<BlinkCnt;Cnt++);` is highlighted in green, and a green arrow points to the start of the `while` loop.

```
defining MACRO PLACE_FLASH places this or
/*
#ifdef PLACE_FLASH
#pragma location = ".flash_code"
#endif
#ifdef PLACE_ASYNC SRAM
#pragma location = ".async_code"
#endif
#ifdef PLACE_MEMORY CODE
#pragma location = ".memy_code"
#endif
void CallSomeFlashCode(int BlinkCnt)
{
    int Cnt,i;

    /* loop forever and keep blinking */
    while(1)
    {
        LED2_ON = LED2_PIN;
        for(Cnt=0;Cnt<BlinkCnt;Cnt++);

        LED2_OFF = LED2_PIN;
        for(Cnt=0;Cnt<BlinkCnt;Cnt++);
    }
}
/*
```

Memory View: Shows memory addresses from `0x60000000` to `0x60000040` with their corresponding hexadecimal values and ASCII representations.

Address	Hex	ASCII
5fffffff0	---	---
5fffffff8	---	---
60000000	08 4a 40 23 13 60 00 22	.J@#...
60000008	11 00 81 42 01 da 49 1c	...B...I.
60000010	fb e7 05 4a 40 23 13 60	...J@#...
60000018	00 22 11 00 81 42 ef da	...B...
60000020	49 1c fb e7 10 52 00 40	I...R.@
60000028	14 52 00 40 64 e0 e3 f3	.R@d...
60000030	05 f6 63 14 c2 19 21 bf	...c...!
60000038	09 1a 9c 63 32 08 8a 88	...c2...
60000040	45 8a 79 cf 2c 6f 8e 80	E.v...o...

Disassembly View: Shows the assembly code for `CallSomeFlashCode`, including instructions like `LDR.N`, `MOVS`, and `STR`.

```
Disassembly
Go to CallSomeFlashCode Memory
Disassembly
0x18ffffffd: ---- ---
0x18ffffffe: ---- ---
0x18ffffff: ---- ---
LED2_ON = LED2_PIN;
CallSomeFlashCode:
??CallSomeFlashCode_0:
0x19000000: 0x4a08 LDR.N R2, ??
0x19000002: 0x2340 MOVS R3, #6
0x19000004: 0x6013 STR R3, [R
for(Cnt=0;Cnt<BlinkCnt;Cnt++);
0x19000006: 0x2200 MOVS R2, #0
0x19000008: 0x0011 MOVS R1, R2
for(Cnt=0;Cnt<BlinkCnt;Cnt++);
??CallSomeFlashCode 2:
```