SMART ARM-based MCUs

AT13878: Atmel SMART SAM V7x TCM Memory

APPLICATION NOTE

Introduction

This application note provides information about configuring and using Tightly Coupled Memory (TCM) with Atmel® SMART SAM V7x microcontrollers using IAR™ development tool chain.

The application note includes:

- An overview of the Atmel SMART SAM V7x
- Differences between Tightly Coupled Memory and Cache memory
- Guidelines for configuring the TCM in Software

Features

Atmel SMART SAM V7x flash microcontrollers have the following features.

- High Performance Core
  - ARM® Cortex®-M7 processor running up to 300MHz
  - 16KB Instruction-Cache and 16KB Data-Cache
  - Memory Protection Unit (MPU) with 16 zones
  - Single- and double-precision HW Floating Point Unit
- Advanced Memory Architecture
  - Up to 2048KB embedded Flash
  - Up to 384KB embedded four-port SRAM
  - 16KB ROM with embedded Boot Loader routines
  - No-wait-state Tightly Coupled Memory (TCM) – programmable in one of four configurations to meet diverse application requirements

ITCM [KB]	DTCM [KB]
0	0
32	32
4	64
128	128

Table of Contents

1. SAM V7x Microcontroller

The automotive qualified SAM V70 and V71 series of microcontrollers offer a high performance core – an ARM Cortex-M7, with an Advanced Memory Architecture with up to 384KB of multi-port internal SRAM.

This coupled with a powerful set of peripherals, make these microcontrollers ideal for many interesting high-end applications, for example, Network gateways, infotainment connectivity and audio applications.

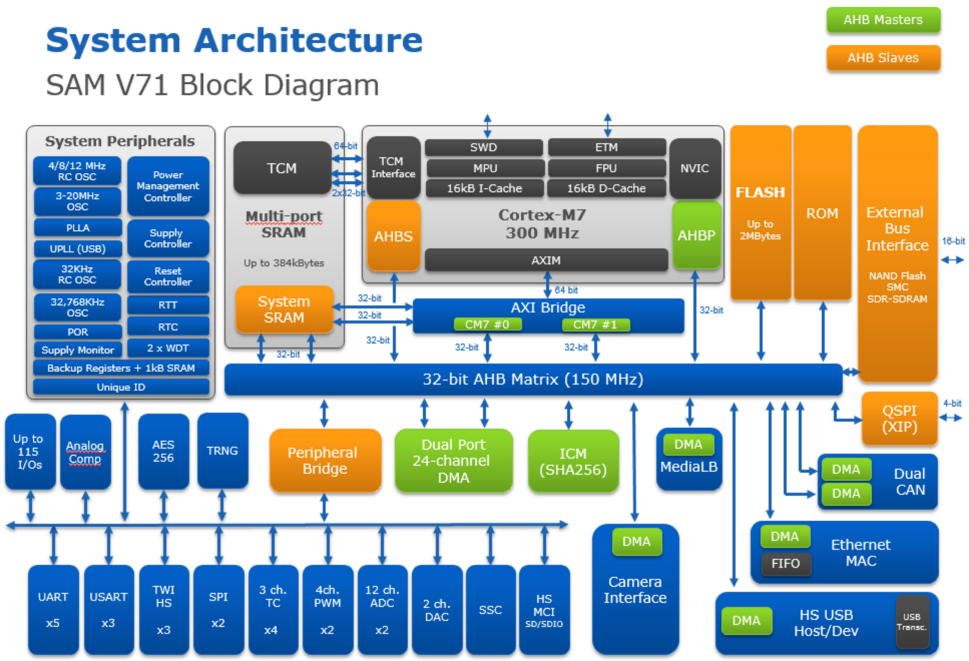
The peripheral set includes Ethernet MAC with hardware support to facilitate Ethernet AVB, high-speed USB with integrated PHY, I2S, QSPI, CAN-FD and MediaLB. Refer to their respective datasheet(s) for the peripherals and packages available for this device.

To ensure a high-speed, low latency, and deterministic access for time critical code and data, the ARM Cortex-M7 core is connected to TCM memory.

A multilayer Bus MATRIX interconnects the peripherals with the multi-port RAM.

The SAM V7x series provides the best combination of connectivity interfaces, highest performance ARM Cortex-M7 core, and a unique multi-port memory architecture.

Figure 1-1 SAM V7x Architecture Block Diagram



2. Cache Memory vs. TCM Memory

This chapter highlights the key differences between Cache and TCM.

Table 2-1 TCM vs. Cache

Tightly Coupled Memory	Cache Memory
Tightly Coupled Memory is a memory accessed by a *dedicated* connection from the core. In case of SAM V7x, there are *two* dedicated connections from Cortex-M7 to the internal SRAM - for Instruction TCM and another for Data TCM.	Cache memory is RAM memory integrated inside the Cortex-M7 core itself. In case of SAM V7x, 16KB of Instruction Cache and 16KB of data Cache is available.
Tightly coupled memory is part of the system memory map with a definite start address. The size of the TCM determines the end address.	Cache memory is not part of the system memory map. It does not have a physical memory address.
A programmer can decide the content to be stored in TCM at compilation time.	A *control logic* and not Programmer, determines what is stored in cache memory.
The TCM memory is directly accessible to software.	During program execution, cache is stored with instructions or data fetched from memory to the CPU.
TCM can be accessed both by CPU and by DMA.	Cache cannot be accessed by DMA.
Tightly coupled memory has deterministic access time. It always takes a single cycle to access contents from TCM.	Cache memory serves as an intermediate buffer between the processor and memory to reduce memory access time. The number of cycles needed to access a memory location differs for a *cache-hit* and a *cache-miss*.

Note: TCM memory contents are not cached. TCM memory is directly connected to the Cortex-M7 core by a bus. It can be accessed at similar speeds as accessing cache without the penalty of a cache-miss and cache coherence issues.

3. Atmel SAM V7x TCM Memory

SAM V7x microcontroller architecture supports two TCM memory instances; Instruction TCM (ITCM) and Data TCM (DTCM).

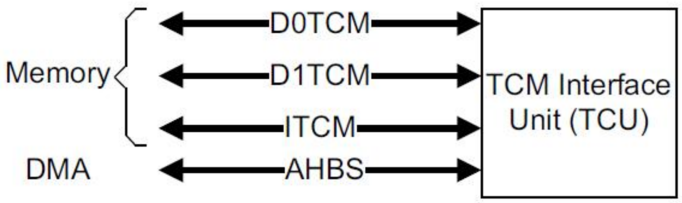
The base address of each TCM is fixed by the ARM System Address map:

- ITCM at 0x00000000
- DTCM at 0x20000000

The ITCM and DTCM occupy lower address ranges than Flash and internal SRAM respectively in the ARM microcontroller memory map.

1. **TCM Interface Unit (TCU)**
   The TCM Unit interfaces to the following:
   - **ITCM**: A single 64-bit Interface
   - **DTCM**: This is connected by 2x 32-bit interfaces
     - The Dual 32-bit interfaces enable two simultaneous access to DTCM. Two 32-bit data can be fetched concurrently in a single cycle.
   - **AHBS**: TCM memory can be accessed by DMA via the AHB Slave bus interface
     - This enables peripherals, for example GMAC, to directly access the TCM memory without any CPU intervention.

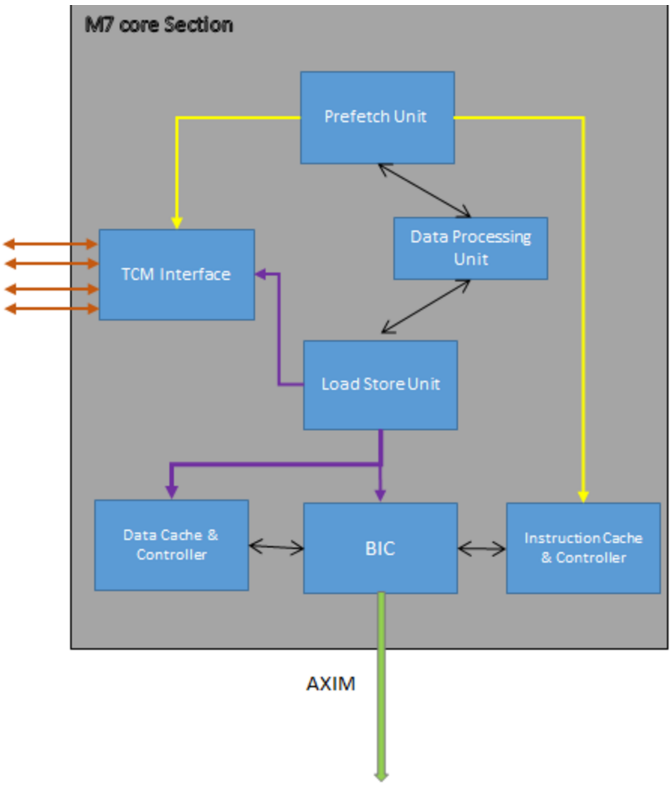
   Figure 3-1 TCM Interface



2. **TCM Access**
   Read or Write requests for each TCM interface are triggered from the following interfaces of the core:
   - Load Store Unit (LSU)
   - Pre-Fetch Unit (PFU)
   - System AHBS interface (AHBS)
   - Debug Unit (DGU)

   The TCM Interface Unit (TCU) contains *arbitration* logic to arbitrate between TCM access requests by various interfaces. The PFU can only read from the TCM whereas other interfaces can both read and write to the TCM memory.

Figure 3-2 Cortex-M7 Architecture for TCM Interface



3. **TCM Configuration**

TCM can be set to one of the *four* different configurations by software. Configuration can be changed by programming the GPNVM bits [8:7] with values as shown in the table TCM Configurations in KB. Multiple TCM configurations provide flexibility for application developers.

Table 3-1 TCM Configurations in KB

ITCM [KB]	DTCM [KB]	Available SRAM (in KB)		GPNVM Bits[8:7]
		384KB variant	256KB variant	
0	0	384	256	0
32	32	320	192	1
64	64	256	128	2
128	128	128	0	3

Note: There is no explicit/extra memory for TCM. It is the internal SRAM of the microcontroller that gets partitioned as TCM + SRAM. GPNVM bits control the amount of memory available as TCM.

Access to the TCM memory is at full *processor* clock speed (HCLK). In the case of SAM V7x, the HCLK can be up to a maximum of 300MHz without any wait states. All accesses from the core are configured for *single* cycle access. Hence, the *deterministic* behavior of TCM access.

The remaining part of the total internal SRAM, not used as TCM, is accessed at Master clock (MCK) frequency of up to 150MHz.

TCM memory can be explicitly enabled/disabled by a register configuration in the system control block of the Cortex-M7.

4. Software Implications of TCM

1. **Address 0x0**

   The Bit [1] of GPNVM selects the memory that is mirrored at address 0x0000 0000.

   - 0 – Internal ROM memory. ROM contents at 0x0080 0000 gets mirrored at address 0.
   - 1 – Internal Flash memory. Flash contents at 0x0040 0000 gets mirrored at address 0.

   In a fully erased microcontroller, the value of this GPNVM bit set to 0.

   This bit value enables the boot-loader code in ROM to execute and program the Flash.

   GPNVM Bit [1] should be set to 1 after programming a valid application in Flash.

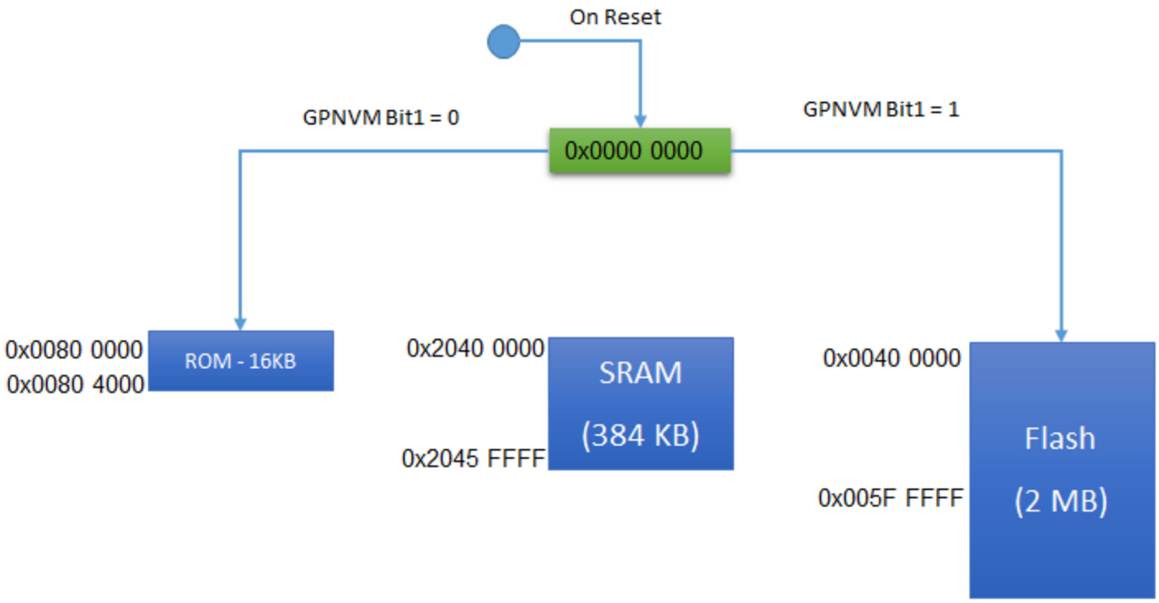
   This bit value will now map the internal Flash at address 0x0000 0000.

   Example: ATSAMV71Q21 microcontroller – 2MB Flash, 384KB internal SRAM.

   Consider a case wherein the GPNVM [8:7] = b'00.

   Here ITCM = 0KB; DTCM = 0KB. There is no reduction in available SRAM memory.

   Figure 4-1 Address "0" Mapping on Reset and GPNVM[8:7] = b'00



2. **ITCM is "also" at address 0x0**

   In SAM V7x, the ITCM RAM region is also mapped to 0x0000 0000 by ARMv7-M architecture.

   The address 0x0 mapped to ITCM memory can be accessed only if both the conditions below are satisfied.

   - GPNVM [8:7] bits setting indicates a non-zero TCM memory
   - Bit [0] of ITCM Control Register (ITCMCR) @ 0xE000EF90 is set

   The Bit [0] of ITCMCR register can be used to enable/disable the ITCM memory.

   - When this bit is SET, the Cortex-M7 core shall fetch all mapped instructions in ITCM range (between 0x0 – last address of ITCM) through the ITCM bus and not the AXIM interface
   - The flash memory is still available in its physical memory address starting at 0x0040 0000
   - On any reset, Bit [0] of ITCMCR gets cleared

– Vector tables and start-up code can be located in ROM or Flash irrespective of GPNVM bits configuration

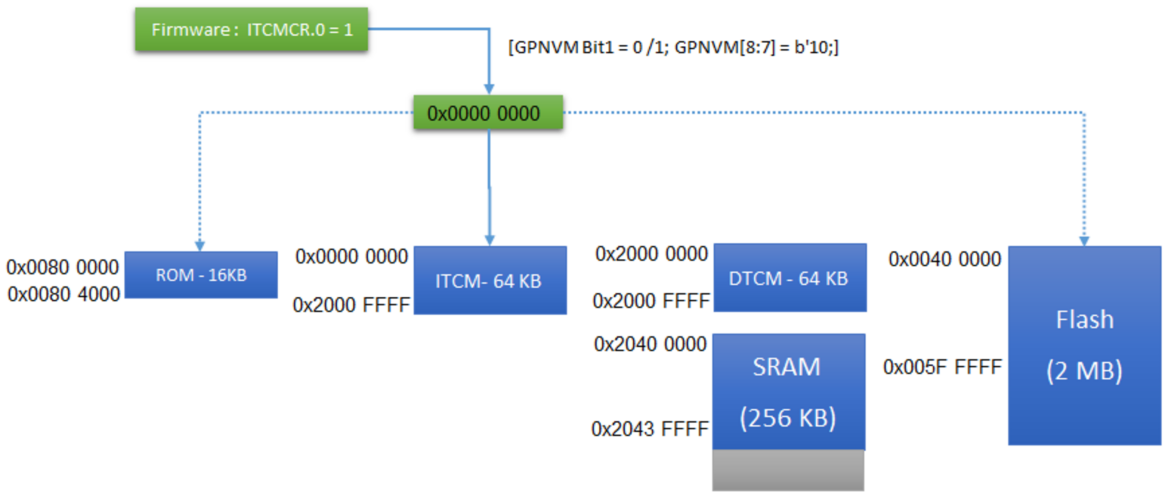
The Cortex-M7 core performs all instruction accesses after a reset through the AXI bus and does not access the TCM interface, refer to Figure 3.2. So an address 0x0 on a reset maps to the Flash or ROM memory, even if the GPNVM bits [8:7] are non-ZERO. However, as soon as the ITCMCR is configured to enable the ITCM, any further access of address 0 would be to ITCM through the TCM interface instead of the AXI bus.

Example SAMV71Q21 microcontroller – 2MB Flash, 384KB internal SRAM.

Consider a case wherein the GPNVM [8:7] = b'10.

Here, ITCM = 64KB; DTCM = 64KB. The internal SRAM memory reduces by 128KB. The available SRAM is 256KB instead of 384KB.

Figure 4-2 Address "0" Mapping on ITCM Configuration



3. **Mapping ITCM and Vector Table Relocation (VTOR) Register**

The default value for VTOR is 0x0000 0000. As specified in Section 4.1 , in cases where ITCM is not used, this address 0x0000 0000 mirrors the start of the Flash memory. Vector Tables are typically located at the beginning of the same, i.e. at address 0x00400000. Any interrupt triggered during the execution has its interrupt service routine correctly fetched from the VTOR offset.

However, after the ITCM is enabled, the VTOR does not point to the beginning of the Vector Table located in flash but to the beginning of ITCM. It is necessary to modify VTOR to map to the start address of the vector table.

There are two possible configurations for re-mapping VTOR register:

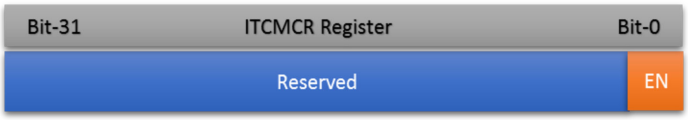
- Write the VTOR register with the address of the interrupt vector table in Flash
- Copy the Vector Table into ITCM after start-up and update VTOR with this address

The user can also copy/locate any critical interrupt routines to ITCM to increase the access speed of the routines and reduce latency in servicing the interrupt. For the consistently fastest interrupt response to time-critical events, both the vector table and the time-critical event handlers should be located in the ITCM memory.

After changing the VTOR register at address 0xE000ED08 appropriately, Enable Bit [0] in ITCMCR register at address 0xE000EF90.

The memory map immediately gets altered. The new memory map would include interface into ITCM as described in Figure 4-2 Address "0" Mapping on ITCM Configuration on page 8.

Figure 4-3 ITCMCR Register



4. **A Word about DTCM**

   DTCM memory size is the same as the ITCM memory.

   A non-zero TCM size by GPNVM [8:7] configuration is sufficient to enable DTCM. Unlike ITCM, which must be explicitly enabled by software after a reset, the DTCM is enabled at reset if configured for a non-zero size.

   If needed, Bit [0] in DTCMCR can be used to disable/enable the DTCM memory. This memory remains unused in the system. It is necessary to select an appropriate TCM configuration based on application needs. Refer TCM Configuration for details.

5. **Software build**

   5.1. The TCM memory should not be dynamically changed. The memory map used in the linker script must *match* the GPNVM configuration of the TCM's.

   5.2. The reasons for this are:

   5.2.1. The code/data inside TCM are not *mirrored* but actually copied to the memory in the start-up code.

   5.2.1.1. Copy routines should know the boundaries of the TCM memory.

   5.2.2. Configuration of the TCM memory results in a corresponding reduction of available internal SRAM.

   5.2.2.1. TCM memory are partitions within the SRAM. Available SRAM is dependent on the chosen TCM memory configuration.

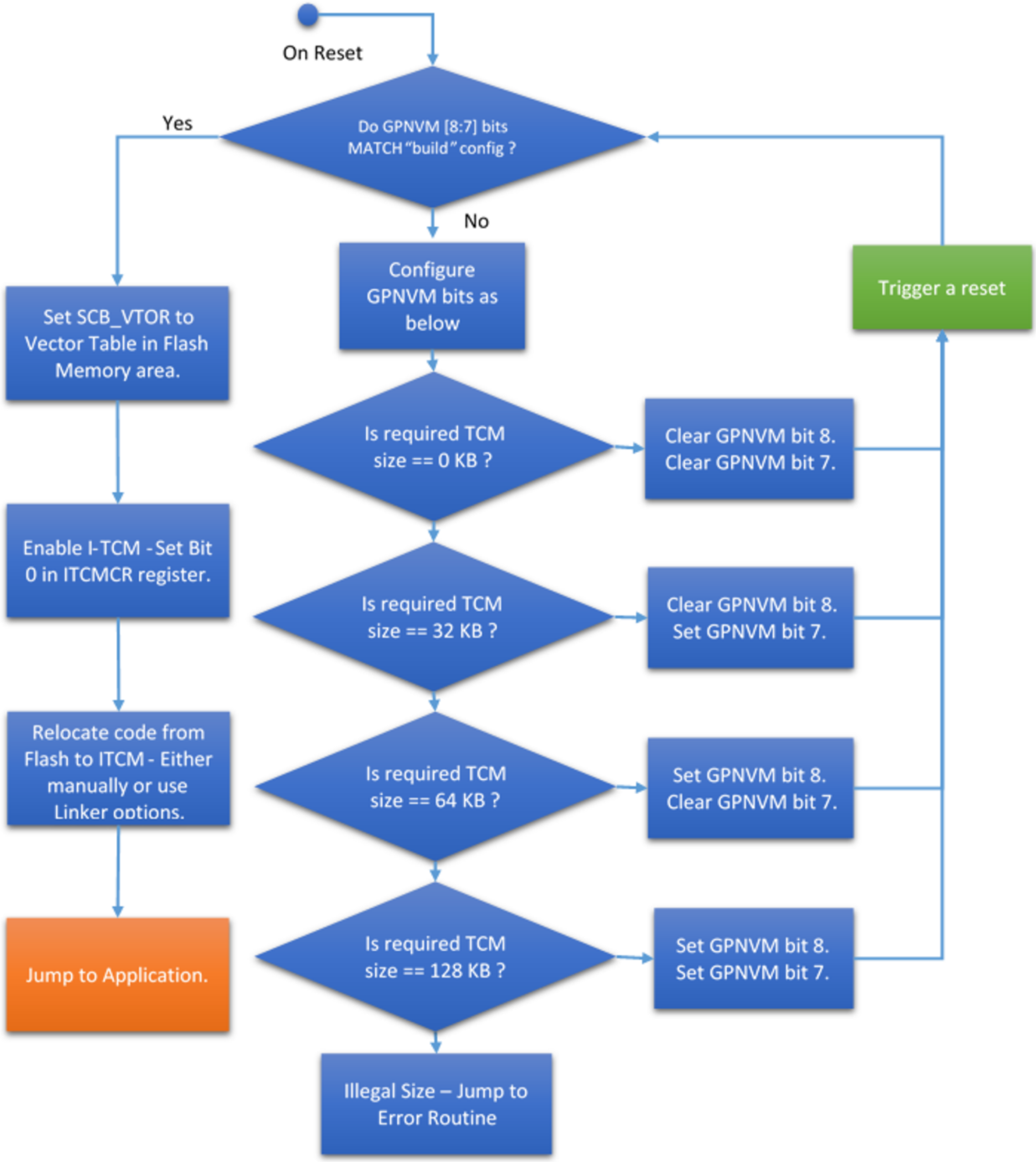
   5.2.2.2. The Flash download routines need to be *aware* of the available SRAM when the TCM is configured.

   5.3. A reset must be triggered whenever the GPNVM configuration bits are changed to execute the changes.

5. Programming Sequence for TCM Configuration

The GPNVM bits configuration sequence is illustrated for the TCM memory configuration.

Figure 5-1 GPNVM Configuration in Software



6. Example Benchmarking Software

The IAR project provided with this application note contains the linker file(s) – `flash.icf` / `sram.icf` – located in the `config` folder. It is available in the path - `build\ewarm\config`

The `flash.icf` file is the linker file for executing the code from Flash. The `sram.icf` contains all the code placed in the internal SRAM.

By default, the linker files are not set up to use the TCM memory.

The following sections contains the changes required and their implications.

1. **Linker configurations in IAR.**

1.1. **Memory Region Definitions:**

```
define symbol __ICFEDIT_region_ITCM_start__ = 0x0;
define symbol __ICFEDIT_region_DTCM_start__ = 0x20000000;
```

The preceding lines defines a symbol to indicate the start address of the ITCM and DTCM in the memory map respectively. These start addresses are fixed and do not vary for any application.

```
define symbol __ICFEDIT_size_itcm__         = 0x8000;
define symbol __ICFEDIT_size_dtcm__         = 0x8000;
```

The preceding code snippet defines the size of the ITCM and DTCM respectively. The size can only be one of the following; 0x0, 0x8000, 0x10000, and 0x20000 corresponding to 0KB, 32KB, 64KB, and 128KB. In the preceding code snippet, the 32KB is defined.

The size may vary depending on the application needs. As specified in Software Build, the GPNVM configuration must match the settings here. The application should also ensure that the GPNVM bits are correctly programmed.

```
define region ITCM_region    = mem:[from
__ICFEDIT_region_ITCM_start__ size __ICFEDIT_size_itcm__];
define region DTCM_region    = mem:[from
__ICFEDIT_region_DTCM_start__ size __ICFEDIT_size_dtcm__];
```

The preceding lines define memory regions. Each memory region have a well defined start address and end address.

1.2. **Code relocation into ITCM using Linker:**

The TCM memory is mapped to internal SRAM in Atmel SAM V7x. Internal SRAM is a volatile memory. We need to explicitly copy code into the TCM memory at start-up.

```
initialize by copy  { section code_TCM };
```

This line directs the linker that contents in the section `code_TCM` needs to be *copied* by the start-up code from internal Flash just like an *initialized* RAM variable.

```
place in ITCM_region  {section code_TCM };
place in DTCM_region {section data_TCM };
```

The preceding code block inform the linker that the *destination* for the copy is the ITCM and DTCM memory regions.

So, the start-up code that initializes the RAM variables also copies the code fragments in section `code_TCM` into the ITCM memory region.

Note: It is important that ITCM is *explicitly* enabled in the ITCMCR register **before** this copy routine is executed. `LowLevelInit()` API is used for writing ITCMCR as defined in Software flow in IAR.

1.3. **Source File directives to locate code in a section:**
The code segments within a file can be located in a specific section by using a pair of `#pragma` directives to the linker. All functions must be placed between these two directives in this section.

Example:

- ```
  #pragma default_function_attributes = @ "code_TCM"
  ```
  – *< All functions to be placed in ITCM >*
- ```
  #pragma default_function_attributes =
  ```

  The data segments within a file can be located in a given section by using a `#pragma` directive of the linker. This pragma applies to the *immediate* variable only. The pragma needs to be repeated for every variable that is placed inside the section.
- ```
  #pragma location = "data_TCM"
  ```
  – *< Variable to be located in DTCM >*

2. **Software flow in IAR**
The code flow at startup in IAR is from `__iar_program_start()`.

`__iar_program_start()`
– `__low_level_init()`
Sets up VTOR register with the correct address of the interrupt vector table.
  - `LowLevelInit()`
  Programs correct GPNVM bits [8:7] and triggers reset if needed. Otherwise, enables ITCM.

– `CopyRoutines API`
There is no need for an explicit copy routine. The copy routine in the standard start-up code is sufficient.

The function `__low_level_init()` is called *before* the data segments are initialized. The ITCM is enabled in it before the copy routine is executed.

So, we can safely use IAR linker *initialize by copy* directive to automatically load a code segment to ITCM memory.

3. **Example software results.**
An example software is attached with this application note.

In the example code,
- In `TcmGpnvmConfig()` API:
  – A parameter, `TCM_MemorySize`, specifies the TCM size to be configured
  – Programs the GPNVM bits according to the required TCM size
  – All this code is protected under a pre-processor macro - `ENABLE_TCM`
  – Ensure to define this macro to get the TCM configurations to work
  ```

- In `main.c` file,
  - Two buffers – `TCM_SrcBuff` and `TCM_DestBuff` – are defined and placed in DTCM memory
  - Two buffers – `SrcBuff` and `DestBuff` – are defined and placed in SRAM memory
  - A function – `TCM_memcpy()` – is in ITCM memory
  - A function – `SRAM_memcpy()` – is in SRAM memory
  - A function – `Flash_memcpy()` – is in Flash memory
- The linker configurations are done in `flash.icf` file

Execution metrics collected is in the Table 6-1 TCM Memory Configuration - 32KB, Optimization - None on page 13 and Table 6-2 TCM Memory Configuration - 32KB, Optimization - Speed on page 14. The tables show the advantages of using ITCM and DTCM over traditional Flash/SRAM combination.

It is observed that there is only minimal advantage when Cache is enabled.

This because the example program is extremely small and available entirely within the Cache. However, real world applications are much more complex and cannot be accommodated inside Cache. Such applications will find TCM to be more helpful in placing *critical* and *deterministic* code fragments.

Table 6-1 TCM Memory Configuration - 32KB, Optimization - None

Configuration	Code	Buffer	Cycles
Instruction Cache = OFF; Data Cache = OFF;	Flash	SRAM	592854
	Flash	DTCM	529644
	SRAM	SRAM	569889
	SRAM	DTCM	541637
	ITCM	SRAM	280475
	ITCM	**DTCM**	**217994**
Instruction Cache = ON; Data Cache = OFF;	Flash	SRAM	367245
	Flash	DTCM	311773
	SRAM	SRAM	337919
	SRAM	DTCM	277111
	ITCM	SRAM	280875
	ITCM	**DTCM**	**219744**
Instruction Cache = ON; Data Cache = ON;	Flash	SRAM	105601
	Flash	DTCM	102509
	SRAM	SRAM	102528
	SRAM	DTCM	102565
	ITCM	SRAM	102559
	ITCM	**DTCM**	**102551**

Table 6-2 TCM Memory Configuration - 32KB, Optimization - Speed

Configuration	Code	Buffer	Cycles
Instruction Cache = OFF; Data Cache = OFF;	Flash	SRAM	223503
	Flash	DTCM	222405
	SRAM	SRAM	222127
	SRAM	DTCM	183331
	ITCM	SRAM	210761
	ITCM	**DTCM**	**149325**
Instruction Cache = ON; Data Cache = OFF;	Flash	SRAM	210721
	Flash	DTCM	149084
	SRAM	SRAM	210631
	SRAM	DTCM	149129
	ITCM	SRAM	210533
	ITCM	**DTCM**	**149091**
Instruction Cache = ON; Data Cache = ON;	Flash	SRAM	35091
	Flash	DTCM	28820
	SRAM	SRAM	29392
	SRAM	DTCM	28837
	ITCM	SRAM	29355
	ITCM	**DTCM**	**28858**

7. Summary

Tightly Coupled Memory (TCM) bring a lot of advantages to application developers.

It provides the same performance as if the code is located in Cache.

Some suggestions to better utilize TCM in applications:

- Application stack can be located in DTCM
- Application heap can be located in DTCM
- Critical variables can be located in DTCM
- Frequently updated variables in DTCM
- Critical functions/routines can be located in ITCM
- A copy of Interrupt vector table can be located in ITCM
- Interrupt service routines can be located in ITCM

8. Revision History

Doc. Rev.	Date	Comments
42510A	10/2015	Initial Document Release

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**