Memory management for algorithms in Codec Engine based applications

DSP
TEXAS INSTRUMENTS
TECHNOLOGY

Application / Component Advantages

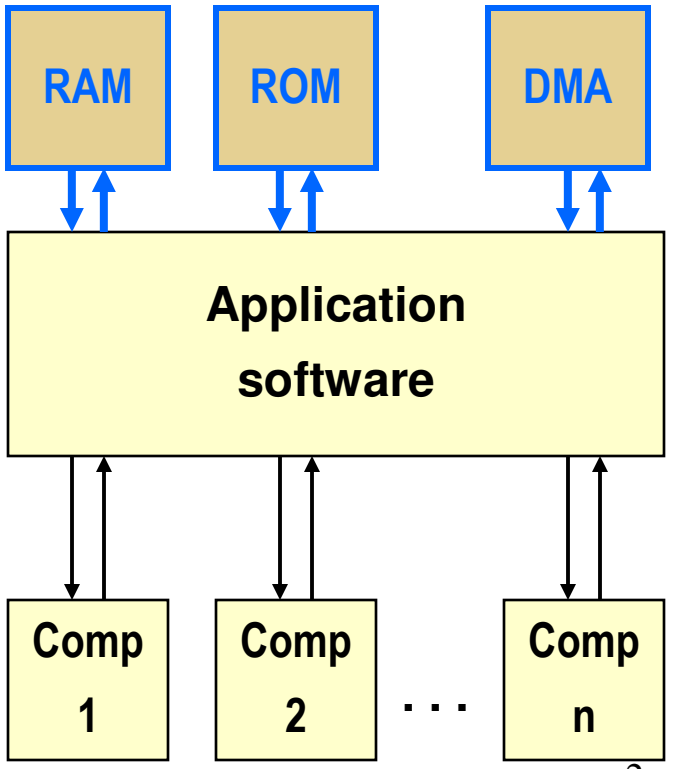
Dividing software between components and system integration provides optimal reuse partitioning, allowing:

- *System Integrator/framework:* full control of system resources
- *Algorithm Author:* to write components that can be used in any kind of system

What are "system resources"?

- **CPU Cycles**
- **RAM (internal, external) : Data Space**
- **ROM : Code Space**
- **DMA hardware**
  - Physical channels
  - PaRAMs
  - TCCs

Let's focus on memory resources. How does the framework mange them?



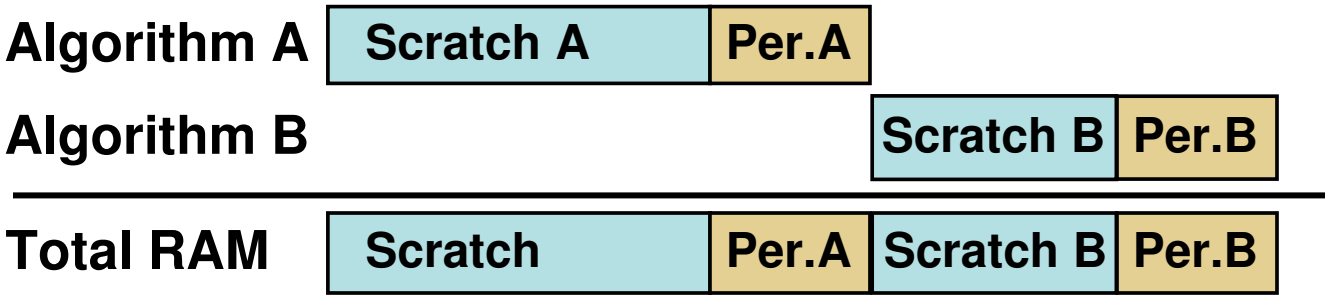
Resource Management : RAM Allocation

- **Algos *never* 'take' memory directly**
  - Algos tell system its needs (*algNumAlloc(), algAlloc()*)
  - Framework determines what memory to give/lend to algo (*MEM_alloc()*)
  - Framework tells algo what memories it may use (*algInit()*)
- **Framework can give algo memory permanently (static systems) via declaration or just during life of algo (a "dynamic system") via malloc-like action**
- **Algos may request internal or external RAM, but must function with either**
  - Allows framework more control of system resources
  - Framework should note algo cycle performance can/will be affected
- **Algo authors can request memory as 'scratch' or 'persistent'**
  - Persistent : ownership of resource must persist during life of algo
  - Scratch : ownership or resource required only when algo is running

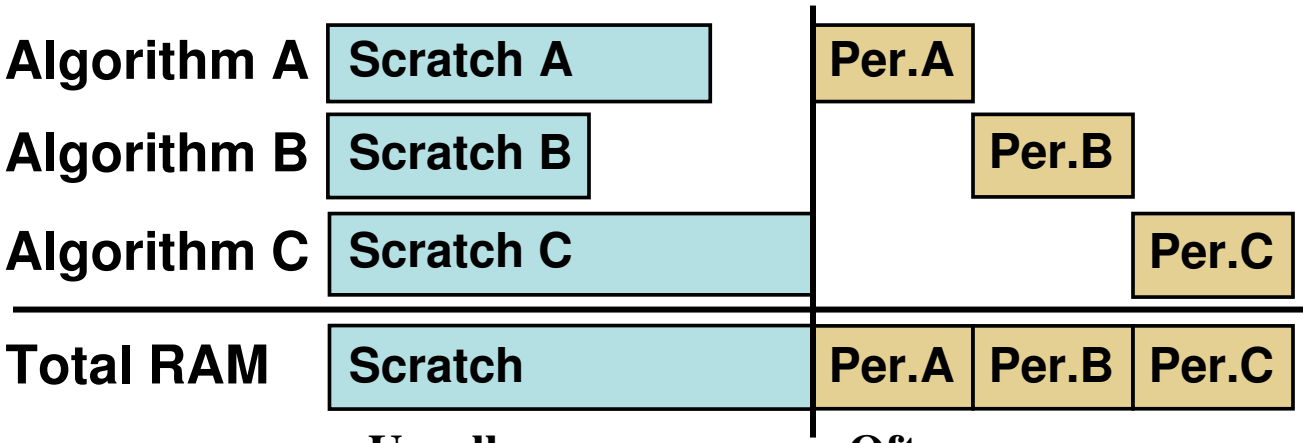
Looking closer at Scratch/Persistent ...

Scratch vs Persistent Memory

- **Scratch** : used by algorithm **during execution only**
- **Persistent** : used to **store** state information **during instance lifespan**



Okay for speed-optimized systems, but not where minimum memory usage is desired ...



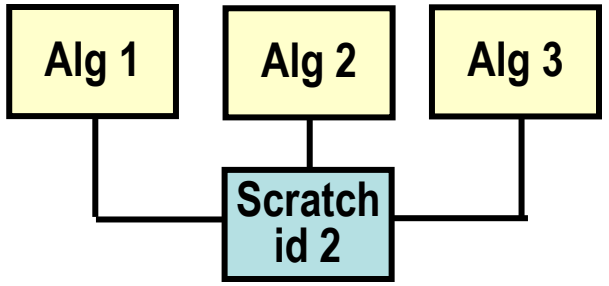
Usually a:
Limited Resource
e.g.: Internal RAM

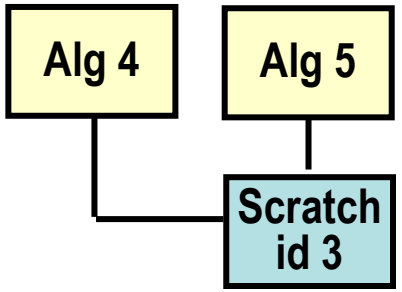
Often an:
Extensive Resource
e.g.: External RAM

DSP
TEXAS INSTRUMENTS
TECHNOLOGY

Resource Management : Scratch Memory

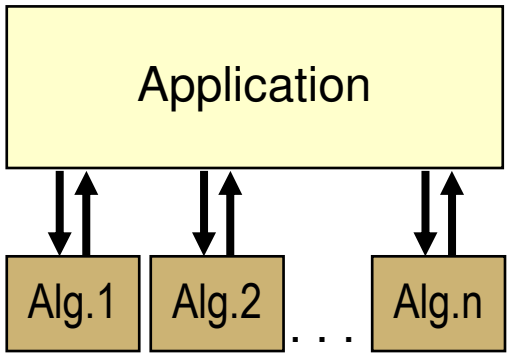
- **Framework can assign a permanent resource to a Scratch request**
  - Easy - requires no management of sharing of temporary/scratch resources
  - Requires more memory in total to satisfy numerous concurrent algos
- **Framwork must assure that each scratch is only lent to one algo at a time**
  (*algActivate(), algDeactivate()*)
- **No preemption amongst algos sharing a common scratch is permitted**
  - Best: share scratch only between equal priority threads – preemption is implicitly impossible
  - *Tip:* limit number of thread priorities used to save on number of scratch pools required
  - Other scratch sharing methods possible, but this is method used by CE
- **Scratch management can yield great benefits**
  - More usage of highly prized internal RAM
  - Smaller total RAM budget
  - Reduced cost, size, and power when less RAM is specified





So, the real benefit of scratch is...

2 vs 3 Layer Topology

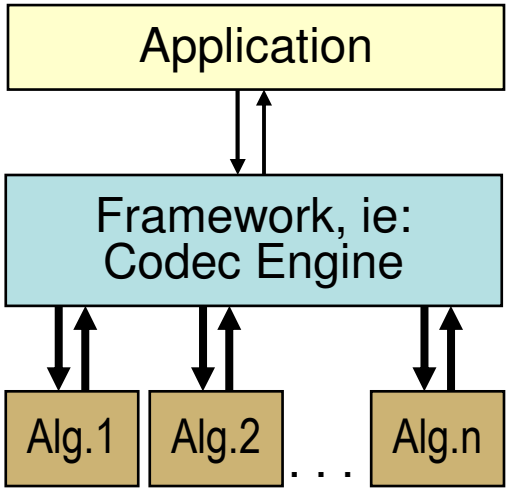


Algo can be directly managed by Application, ie: a '2 layer' topology

- Allows most direct interface between system and component
- Requires more coding effort since algo I/F is at a low level
- All management and details must be managed by System Integrator/App author
- *But, if all algos work identically, couldn't this management be proceduralized to an intermediary level/layer?*

The framework reduces app code's algo management to higher level constructs, eg: Create, Execute, Delete

- Frameworks can be independently written to manage algos in whatever manner the system integrator desires (static v dynamic, scratch mgmt policies, etc)
- DaVinci Codec Engine includes two key framework components:
  - "DSKT2" to manage memory
  - "DMAN3" to manage DMA resources (channels, PaRAMs, TCCs)



DSKT2 Module

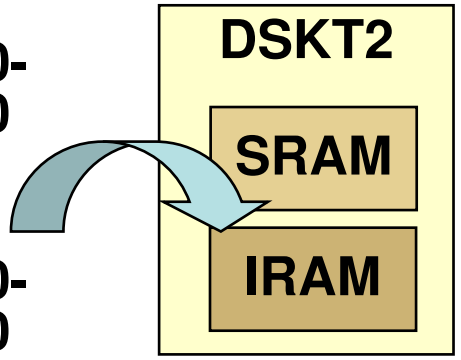
Initialization Phase (config-time)

SRAM:

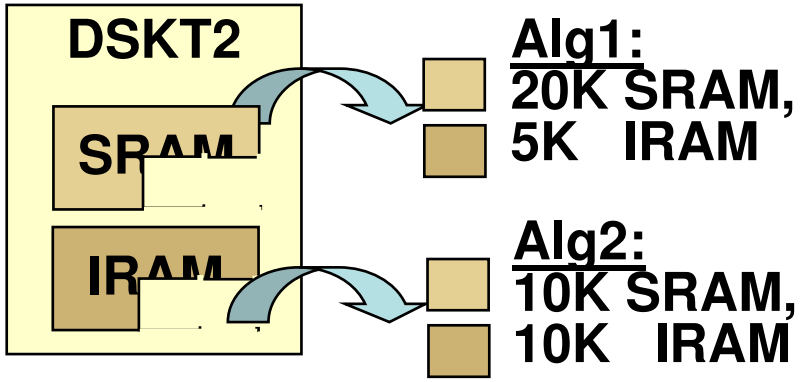
0x8000_0000-
0x8010_0000

IRAM:

0x0000_0000-
0x0004_0000



Usage Phase (run-time)



- **Acts as a warehouse for Memory resources**
- **System integrator initializes the DSKT2 module with available memory resources**
- **Algorithms "check out" memory from the DSKT2 module at runtime when they are instantiated.**

DSP
TEXAS INSTRUMENTS
TECHNOLOGY

DSKT2 Framework

- **Designed to manage all memory needs of all xDAIS algos in system**
- **Presents a simple "Create, Execute, Delete" interface to application level**
- **Dynamic framework – persistent memories are only allocated when an algorithm is created, and are returned to heap when algo is deleted**
- **Scratch memory requests cause allocation of a memory 'pool' that can be shared by other algos with the same "Scratch Group ID"**
  - Allows reuse/sharing of scratch pool RAM resource
  - Scratch pools only created when a user in that group has been created
  - *Tip* : assign same SGID to algos of matching priority
  - *Tip* : Predefine default size of SG pools or create largest users first

DSKT2 method	Sub operations performed…
DSKT2_createAlg	algNumAlloc algAlloc MEM_alloc algInit
DSKT2_activateAlg	algActivate
DSKT2_controlAlg	*algControl*
DSKT2_deactivateAlg	algDeactivate
DSKT2_freeAlg	algNumAlloc algFree MEM_free

C

X

D

- *These API are are managed by the Codec Engine in DaVinci systems*
- *They are presented here "FYI" for Codec Engine users*
- *Non Codec Engine systems can use these API directly to manage xDAIS algos*

DSKT2 internal scratch memory allocation

- DSKT2 allows the framework to organize algorithms into "scratch-groups", and when possible arranges for the algorithm instances within the same scratch group to share a common scratch memory pool.
  - If a scratch memory request cannot be satisfied from the shared scratch buffer, DSKT2 allocates as much of the scratch memory as it can from any other shared scratch buffer available to the same scratch group (ie. DARAM vs. SARAM).
  - If that fails, DSKT2 allocates it as 'persistent' using one of the internal heaps.
  - When ALLOW_EXTERNAL_SCRATCH=true, DSKT2 tries to allocate the memory in an external heap if internal (shared or persistent) memory is not available.
  - Otherwise DSKT2 will indicate failure.
- Only one algorithm in a given scratch group can be active at a given time.

Scratch buffer sizing

- When the first XDAIS algorithm instance at a given scratch-group requests scratch memory, DSKT2 dynamically allocates a 'shared' scratch buffer for that group.
- Size of the common scratch buffer = max {

  (the size of scratch memory requested by the first algorithm),

  (the DSKT2 configuration provided default shared scratch buffer size for the given group-id)

  }

Tip: Use DARAM_SCRATCH_SIZES[] and SARAM_SCRATCH_SIZES[], to improve chances of "shared" scratch memory allocation for each scratch group.

DSKT2 configuration in a CE application

- For DSKT2 to fully honor algorithm memory requests it must know the DSP/BIOS memory heap segments that are available to allocate from.
- This can be configured in the DSP server's .cfg configuration file
- Scratch memory assignment on a per scratch group basis can also be specified at this stage
- Always specify a group id for each algorithm.
  - Otherwise the latter will be given an undetermined group id, which can cause algorithms to occasionally fail at creation time.

DSKT2 configuration in CE applications

Configuration parameter	*Description*
ALLOW_EXTERNAL_SCRATCH	if a scratch request in internal memory cannot be granted AND there is insufficient memory in persistent internal memory to allocate for the request, then DSKT2 attempts allocation using external Memory
DARAM0, DARAM1, DARAM2, SARAM0, SARAM1, SARAM2, ESDATA, IPROG, EPROG	Used to map IALG memory space types to specific heaps defined in DSP/BIOS .tcf configuration file. Each buffer request from an algorithm is allocated from the corresponding heap based on its memory space type
DSKT2_HEAP	The heap that DSKT2 will use by default to allocate internal objects
debug	Enables the debug profile of the DSKT2 library. This results in a larger and slower version of the library being linked in. It provides extra parameter checking and causes debug trace statements to be generated in the DSP/BIOS SYS trace buffer
DARAM_SCRATCH_SIZES[]	Array to assign size of scratch 'pool' for each scratch group corresponding to a DARAM request
SARAM_SCRATCH_SIZES[]	Array to assign size of scratch 'pool' for each scratch group corresponding to a SARAM request
cacheWritebackInvalidateFxn	function to implement cache writeback and Invalidation, called before granting memory to algorithms (default is BCACHE_wbInv)

Example DSKT2 configuration

MyServer.cfg

```
/*
 *  ======== DSKT2 Configuration ========
 *  XDAIS Algorithm Memory and DMA allocation
 */
   var DSKT2 = xdc.useModule('ti.sdo.fc.dskt2.DSKT2');
   DSKT2.DARAM0     = "L1DSRAM";
   DSKT2.DARAM1     = "L1DSRAM";
   DSKT2.DARAM2     = "L1DSRAM";
   DSKT2.SARAM0     = "L1DSRAM";
   DSKT2.SARAM1     = "L1DSRAM";
   DSKT2.SARAM2     = "L1DSRAM";
   DSKT2.ESDATA     = "DDRALGHEAP";
   DSKT2.IPROG      = "L1DSRAM";
   DSKT2.EPROG      = "DDRALGHEAP";
   DSKT2.DSKT2_HEAP = "DDR";

   // H264BP ENC has largest scratch requirements - group 0 is sized for it
   // All video codecs share same group (0)

   DSKT2.DARAM_SCRATCH_SIZES = [ 64900, 0,0,0,0,0,0, /* ... */ 0 ];

   DSKT2.SARAM_SCRATCH_SIZES = [ 64900, 0,0,0,0,0,0, /* ... */ 0 ];

   DSKT2.debug = true;
```

Tips on configuration

- **Always specify the DARAM_SCRATCH_SIZES[] and SARAM_SCRATCH_SIZES[] arrays.**
  - **To find out how much space to give to each scratch group, you can use the algutil utility in servertools to print out algorithm scratch requests for DARAM and SARAM.**
- **When constrained in amount of internal memory heap space:**
  - **set DSKT2.DSKT2_HEAP to an external heap**
  - **Double-check the scratch sizes arrays to ensure you are not allocating more scratch space then necessary for each scratch group**
- **Turn DSKT2.debug on during debugging to obtain better visibility into the module's activities.**
- **See xDAIS DSKT2 User's Guide for more details on DSKT2**
  - **more than you ever need to know since CE hides most of DSKT2 usage internally**