

10. RAM and Cache

The i.MX RT 1060 contains 1M of internal RAM, whereby 512k is fixed (OCRAM2) and 512k is configurable (FlexRAM) which is constructed of 16 banks of 32k each. These banks can each be assigned to one three areas (FlexRAM controller):

- OCRAM General RAM operates at 1/4 the core clock speed (32 bit wide). This is cacheable, meaning that if L1 cache is enabled data content that is already in cache is used to avoid needing to perform the OCRAM access.
- ITCM Instruction Tightly Coupled Memory (64 bit wide) that is optimised for instruction execution at the maximum core speed. Non-cacheable (also since already optimally fast) and so no potential cache synchronisation problems.
- DTCM Data Tightly Coupled Memory (64 bit wide) that is optimised for data access at the maximum core speed. Non-cacheable (also since already optimally fast) and so no potential cache synchronisation problems.

For full details concerning the FlexRAM and optimal configuration to match an applications memory requirements NXP has prepared the application note AN12077 which can be found at <https://www.nxp.com/docs/en/application-note/AN12077.pdf>

The i.MX RT 1060 has L1 cache with 32kBytes instruction cache and 32kBytes data cache. NXP has prepared the application note AN12042 which can be found at <https://www.nxp.com/docs/en/application-note/AN12042.pdf>

Use of the cache can ensure high speed operation even when the source of code or data is in a slower memory by avoiding to have to unnecessarily fetch the data when it has been loaded once to the cache.

When the cache is enabled it caches from OCRAM and QSPI-Flash; it neither caches ITCM nor DTCM, which are already tightly coupled to the core.

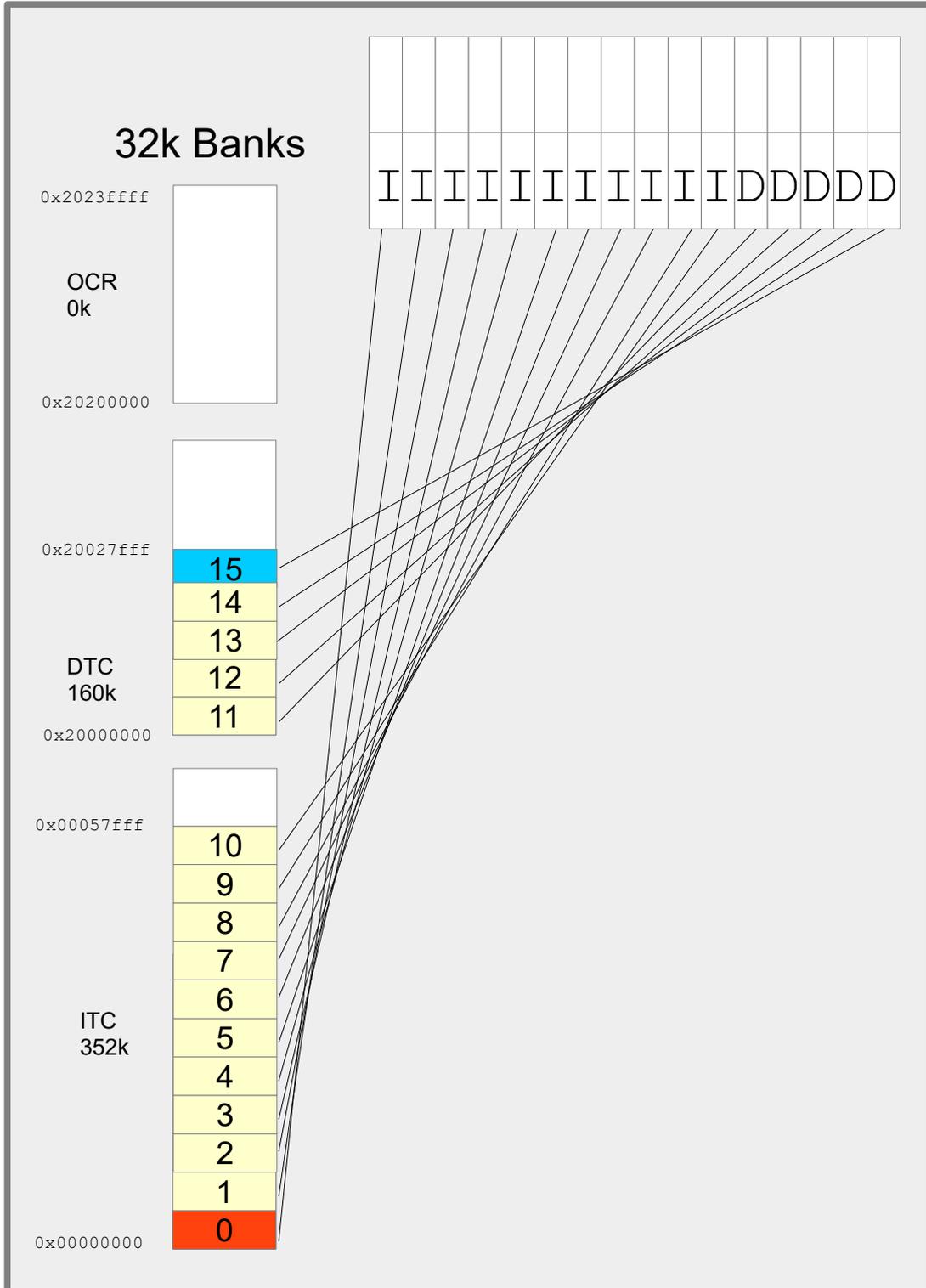
The application can decide whether it uses data or instruction cache with the defines

```
#define ENABLE_INSTRUCTION_CACHE
```

```
and
```

```
#define ENABLE_DATA_CACHE
```


Assuming it is decided that an application were best configured to have 11 banks for ITCM (so that the code could be completely located there – 352k) and 5 banks for data (so that up to 160k of data could be accesses at optimal speed) and no OCRAM the 16 banks could be configured as follows:



The µTasker FlexRAM driver always uses the bank order from instruction use to data use (if OCRAM were used it would be inserted between the two). The most important thing to understand is that when the bank use is modified the address range of the bank is also changed (it is moved in the memory map). This means that data that was in OCR before the change (in the default configuration) still exists in the bank memory but is now addressed in the ITC or DTC memory space instead.

This behaviour makes it complicated to change the memory configuration at run time because it means that any memory used before the change (eg. The stack or initialised variables) are usually at completely different locations after the change. Typically this will cause a program that simply changes the bank configuration without respecting the fact that its memory moves during the process to immediately fail. For this reason such changes are generally not performed during program operation; if such a configuration is changed it tends to be performed before any variable initialisation and also from code running in other sources and without stack dependency.

The µTasker concept assumes that code and variables fit in the internal RAM and so OCRAM is avoided. The division between ITC and OTC is performed at system initialisation automatically to allocate ITC banks to the code space and DTC banks to data space in such a way as to have as much DTC available as possible for heap and stack. If code of 340k were encountered it would thus assign 352k ITC and 160k DTC, as in the example. If less code were encountered additional banks would be assigned to DTC in order to maximise heap and stack availability. Code and data are automatically in the highest performance RAM areas and caching is not required to achieve optimal performance (without caching, no additional synchronisation of data is required).

There is an important reason for choosing the bank ordering: In the default configuration bank 7 is assigned to OCR but will not be used by the ROM loader (the ROM loader may use up to 64k only). After the bank swap is performed this bank is the last bank in DTM, whereby the stack pointer is located near the top, but leaving some additional space above it for 'preserved' variables. The advantage of this is that an application can always write values to the preserved area before a reset and these values will not be modified by the ROM loader. The µTasker boot loader or another application can then read these values, even if the application uses a different RAM bank configuration; as long as its stack pointer is put to near the end of the final bank it will automatically be referenced to the the preserved data area! The preserved area is used in the µTasker project for communicating between applications and the µTasker boot loaders, but can also be used by custom applications for holding data that is guaranteed to be preserved across warm resets.

Due to the nature of the memory operation of the i.MX RT 1060, its configuration requirement to achieve optimal performance and the desire to allow µTasker users to benefit from these with no additional effort the RAM bank management is an integral part of the µTasker boot strategy and the µTasker Boot Loader (see Boot Mode section) an integral part of every project (apart from when a stand-alone application is loaded in a debug environment for test purposes).