Embedding it better...

µTasker Document

i.MX RT 1052 – the µTasker way

Table of Contents

1. Introduction

See i.MX RT 1021 document – only changes are here

The **MIMXRT1050-EVK** is used as test vehicle throughout the document but any board using the part can be easily configured based on the contained details.

2. Clocking

1.1. ARM Core Clock

The ARM core is clocked by an internal clock signal called **AHB_CLK_ROOT**, which can be up to 600MHz.

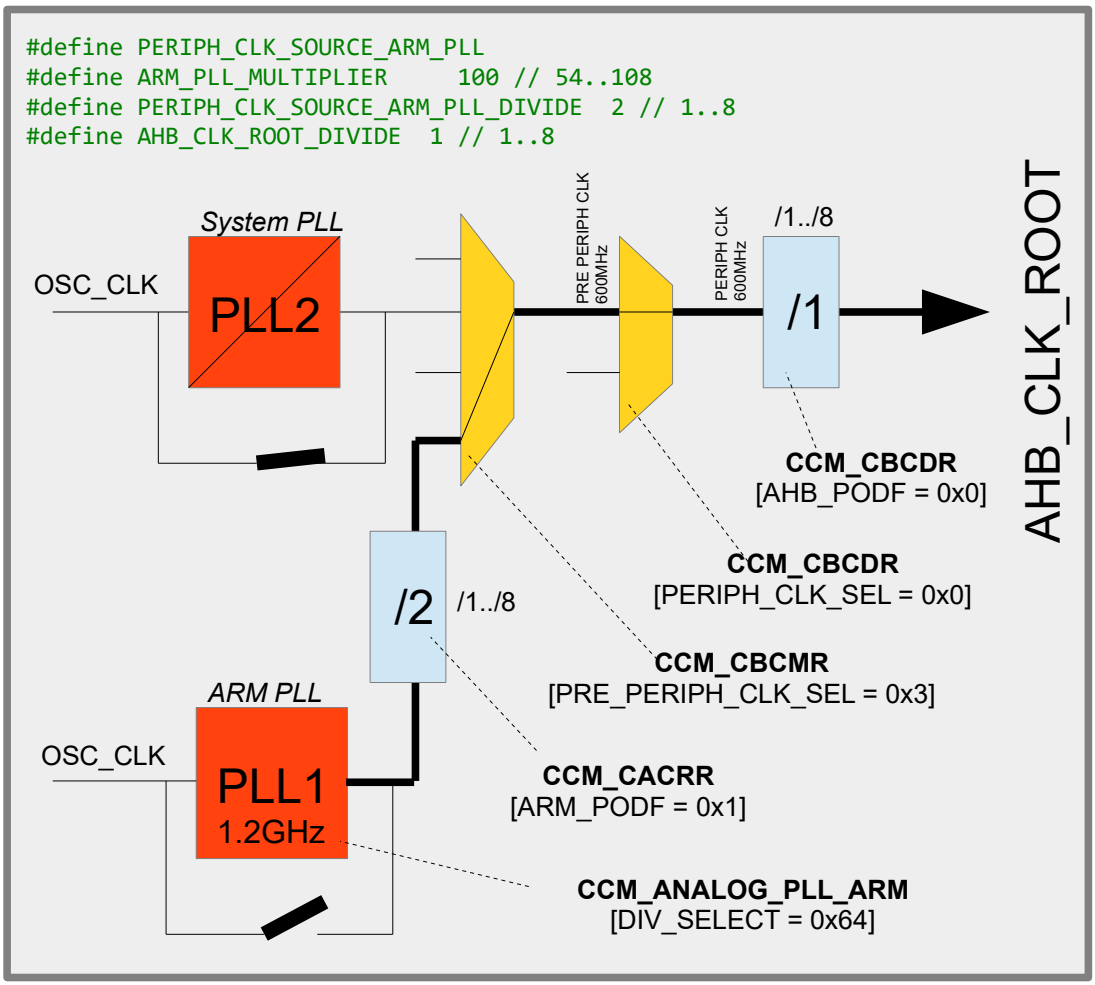
Fraction	System PLL (528MHz PLL2) ((528MHz * 18) / fraction)	USB1 PLL (480MHz PLL3) ((480MHz * 18) / fraction)
12	792MHz	720MHz - default PFD0
13	731.0769231MHz	664.6153846MHz - default PFD1
14	678.8571429MHz	617.1428571MHz
15	633.6MHz	576MHz
16	594MHz - default PFD1 and PDF3	540MHz
17	559.0588235MHz	508.2352941MHz - default PFD2
18	528MHz	480MHz
19	500.2105263MHz	454.7368421MHz - default PFD3
20	475.2MHz	432MHz
21	452.5714285MHz	411.4285714MHz
22	432MHz	392.7272727MHz
23	413.2173913MHz	375.6521739MHz
24	396MHz - default PFD2	360MHz
25	380.16MHz	345.6MHz
26	365.5384615MHz	332.3076923MHz
27	352MHz - default PFD0	320MHz
28	339.4285714MHz	308.5714286MHz
29	327.7241379MHz	297.9310345MHz
30	316.8MHz	288MHz
31	306.58064524MHz	278.7096774MHz
32	297MHz	270MHz
33	288MHz	261.8181818MHz
34	279.5294118MHz	254.1176471MHz
35	271.5428571MHz	246.8571429MHz

The i.MX RT 1052 clock selection possibilities differs to that of the i.MX RT 1021 in the fact that it doesn't support PLL6 (500MHz Ethernet PLL) to be selected as source and instead has the ability to select PLL1, or ARM_PLL (not available in the i.MX RT 1021) instead. PLL1 is a programmable integer frequency multiplier capable of output frequencies from 650MHz up to 1.3GHz, with 864MHz default, which can be divided by 1..8 (divide by 2 – for 432MHz - is set out of reset).

The complete set of frequencies possible are shown in the following table:

Multiplier	ARM PLL PLL1) ((24MHz / multiplier) / 2)	Multiplier	ARM PLL PLL1) ((24MHz / multiplier) / 2)
54	648MHz	82	984MHz
55	660MHz	83	996MHz
56	672MHz	84	1008MHz
57	684MHz	85	1020MHz
58	696MHz	86	1032MHz
59	708MHz	87	1044MHz
60	720MHz	88	1056MHz
61	732MHz	89	1068MHz
62	744MHz	90	1080MHz
63	756MHz	91	1092MHz
64	768MHz	92	1104MHz
65	780MHz	93	1116MHz
66	792MHz	94	1128MHz
67	804MHz	95	1140MHz
68	816MHz	96	1152MHz
69	828MHz	97	1164MHz
70	840MHz	98	1176MHz
71	852MHz	99	1188MHz
72	864MHz	100	1200MHz
73	876MHz	101	1212MHz
74	888MHz	102	1224MHz
75	900MHz	103	1236MHz
76	912MHz	104	1248MHz
77	924MHz	105	1260MHz
78	936MHz	106	1272MHz
79	948MHz	107	1284MHz
80	960MHz	108	1296MHz
81	972MHz		

```
#define PERIPH_CLK_SOURCE_ARM_PLL
```



Furthermore the PLL2 fractional divider outputs that can be used are different.

The choice of the core clock represents the major work of setting up the clocks. The following details are then specific to peripherals used in the system. *Peripherals of no interest don't need to be specifically configured since they will use defaults and be gated off by the the control code.*

1.2. IPG Clock – used for the access of most peripherals

No change

1.3. PERCLK – used by PIT and GPT

No change

1.4. UART_CLK_ROOT – used by all LPUARTs

No change

1.5. USDHC1_CLK_ROOT/USDHC2_CLK_ROOT

To add..

1.6. SEMC_CLK_ROOT – external memory controller clock

No change

1.7. FLEXSPI_CLK_ROOT

No change

1.8. LPSPI_CLK_ROOT

To add..

1.9. TRACE_CLK_ROOT

To add..

1.10. SAI1_CLK_ROOT/SAI2_CLK_ROOT/SAI2_CLK_ROOT

To add..

1.11. LPI2C_CLK_ROOT – used by all LPI2C controllers

No change

1.12. CAN_CLK_ROOT

No change

1.13. SPDIF0_CLK_ROOT

To add..

1.14. FLEXIO1_CLK_ROOT

To add..

1.15. USB1 Clock

USB1's PHY requires a 480MHz clock which is derived exclusively from PLL3's fixed 480MHz output – for this reason PLL3 is known also as USB1 PLL. This means that, logically, PLL3 must be operating at 480MHz for USB1 to be used.

When USB is enabled with

```
#define USB_INTERFACE
```

and

```
#define USE_USB2_ONLY
```

is NOT defined, it is automatically configured as part of the clock initialisation so that it is subsequently available for use by USB1. *If any other clocks are configured to use PLL3 or any of its PFD outputs it will also be available even without USB1 usage.*

1.16. USB2 Clock

USB2's PHY requires a 480MHz clock which is derived exclusively from PLL7's fixed 480MHz output – for this reason PLL7 is known also as USB2 PLL. This means that, logically, PLL7 must be operating at 480MHz for USB2 to be used.

When USB is enabled with

```
#define USB_INTERFACE
```

and either

```
#define USE_USB2_ONLY
```

or

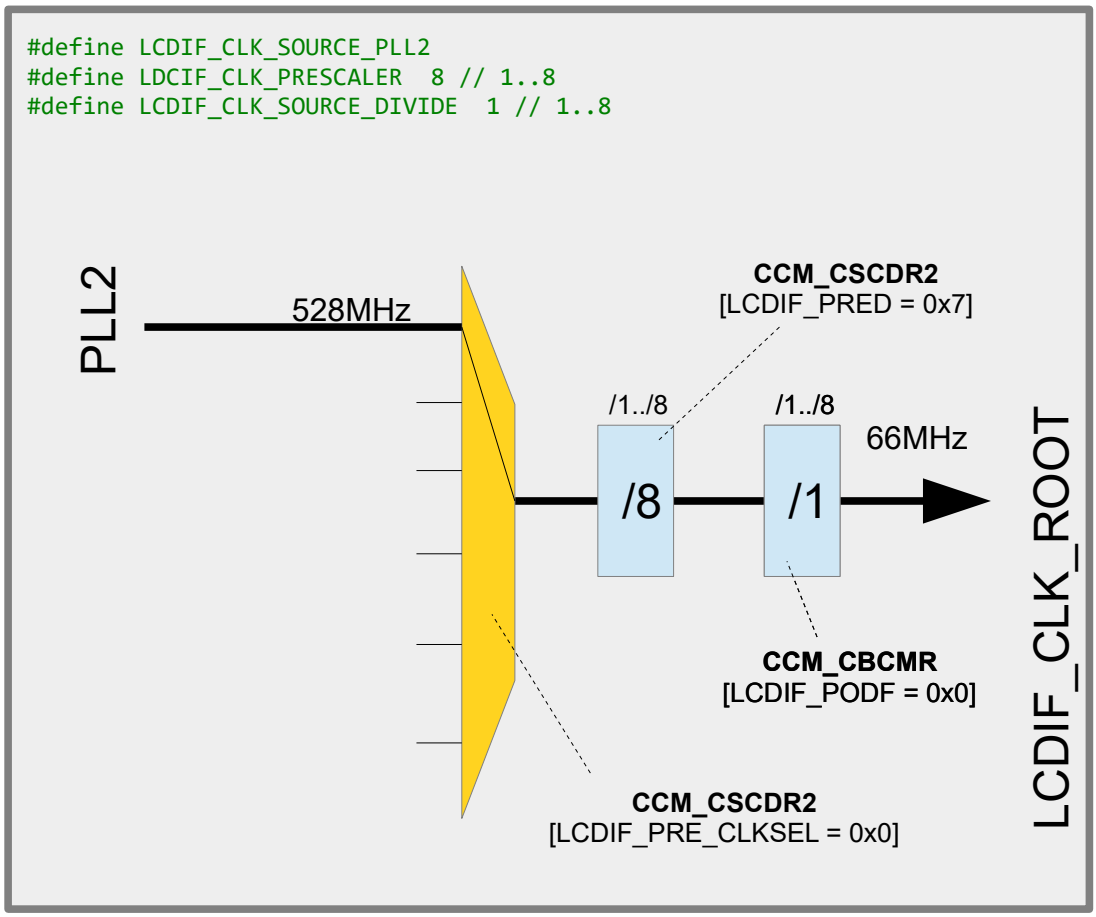
```
#define USE_BOTH_USB
```

are enabled it is automatically configured as part of the clock initialisation so that it is subsequently available for use by USB2. If USB1 alone is used and the USB2 defines not enabled PLL7 is kept in powered down state because it is not needed. *PLL7 can not be used for any other functions in the i.MX RT 1052.*

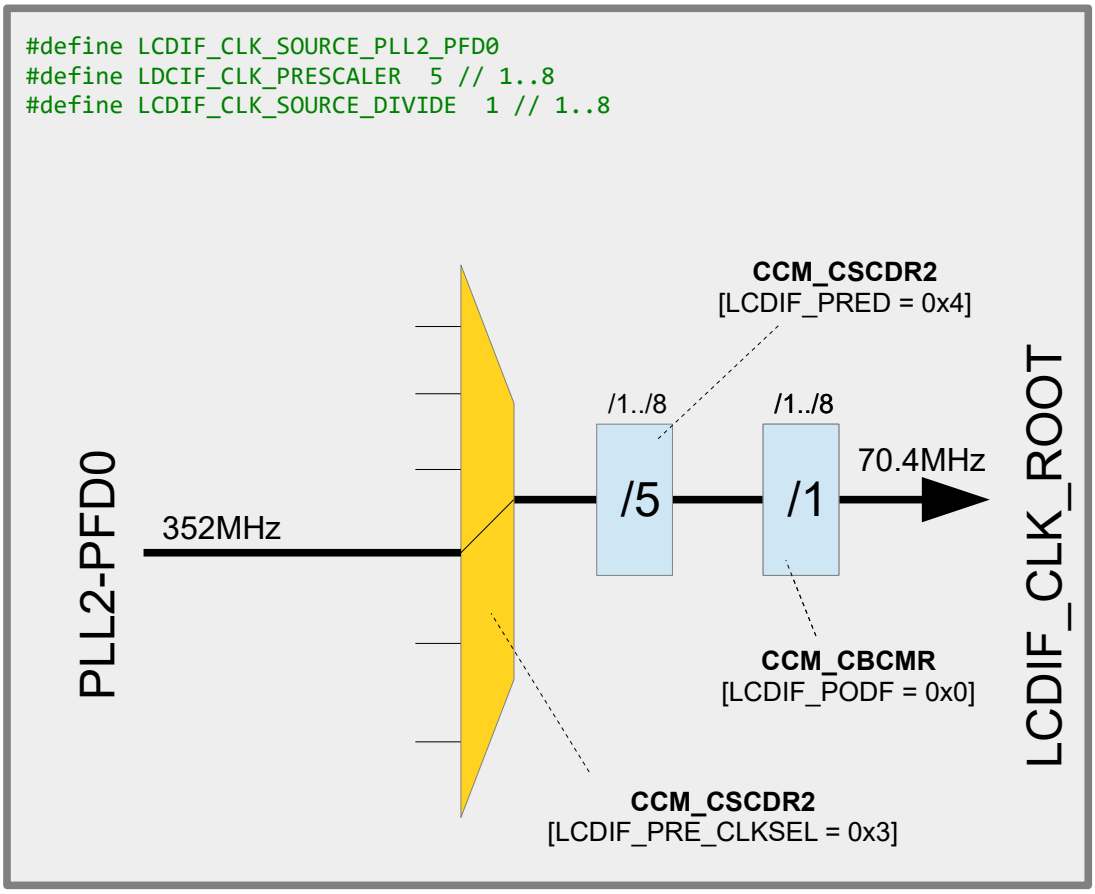
1.17. LCDIF_CLK_ROOT

The maximum LCDIF_CLK_ROOT frequency allowed is 75MHz and it can be sourced form 6 different clock sources

```
#define LCDIF_CLK_SOURCE_PLL2
```

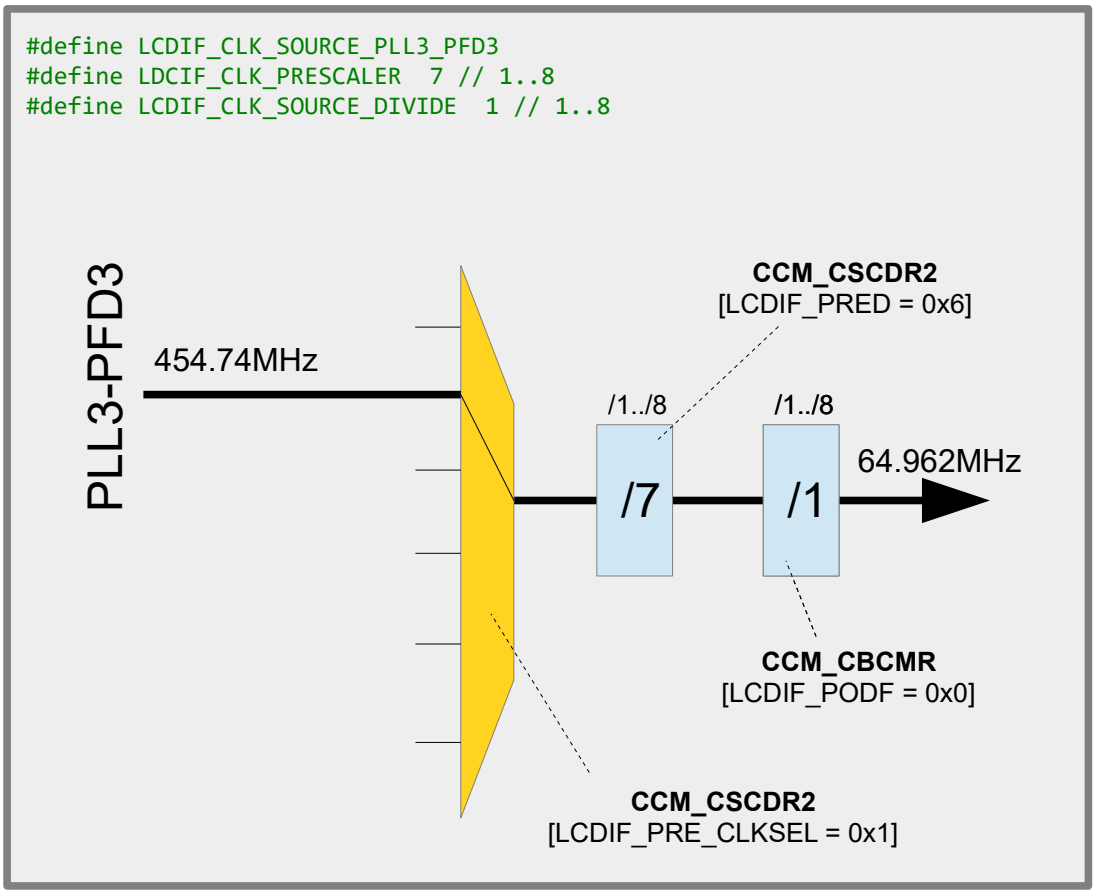


```
#define LCDIF_CLK_SOURCE_PLL2_PFD0
```

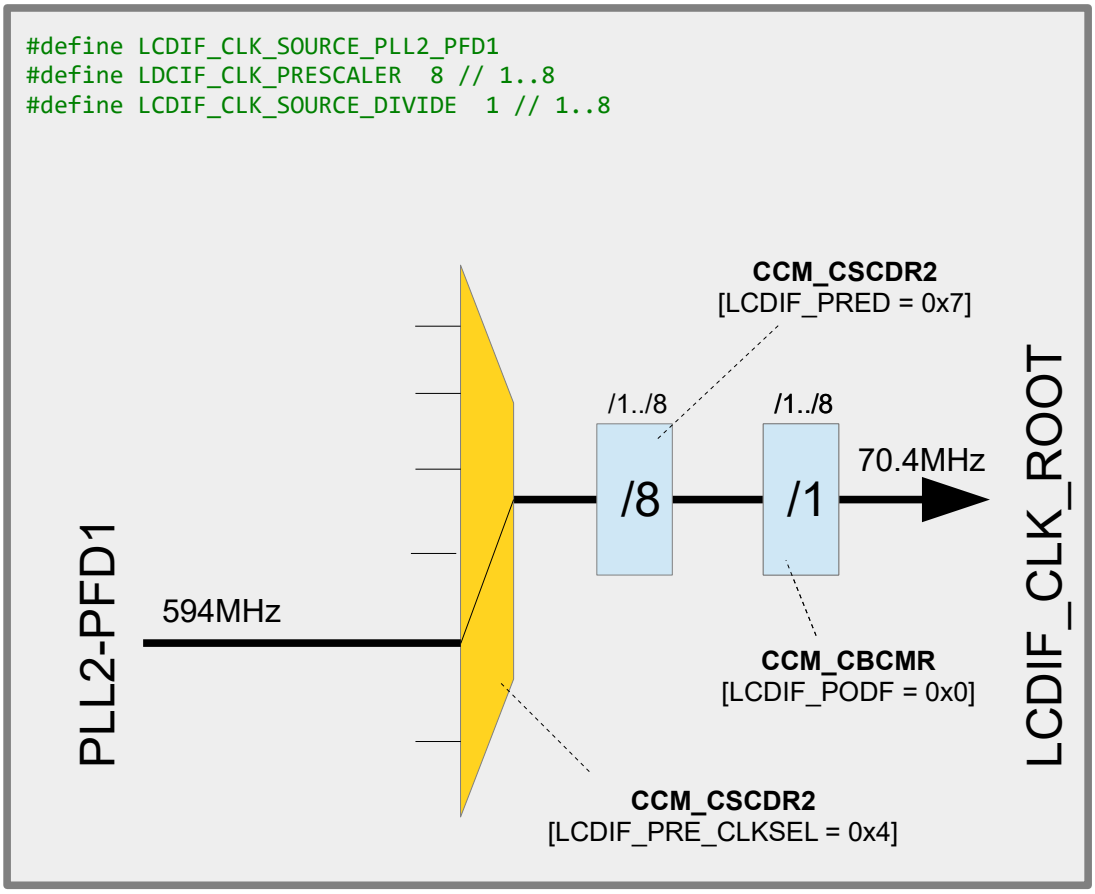



```
#define LCDIF_CLK_SOURCE_PLL3_PFD3
```

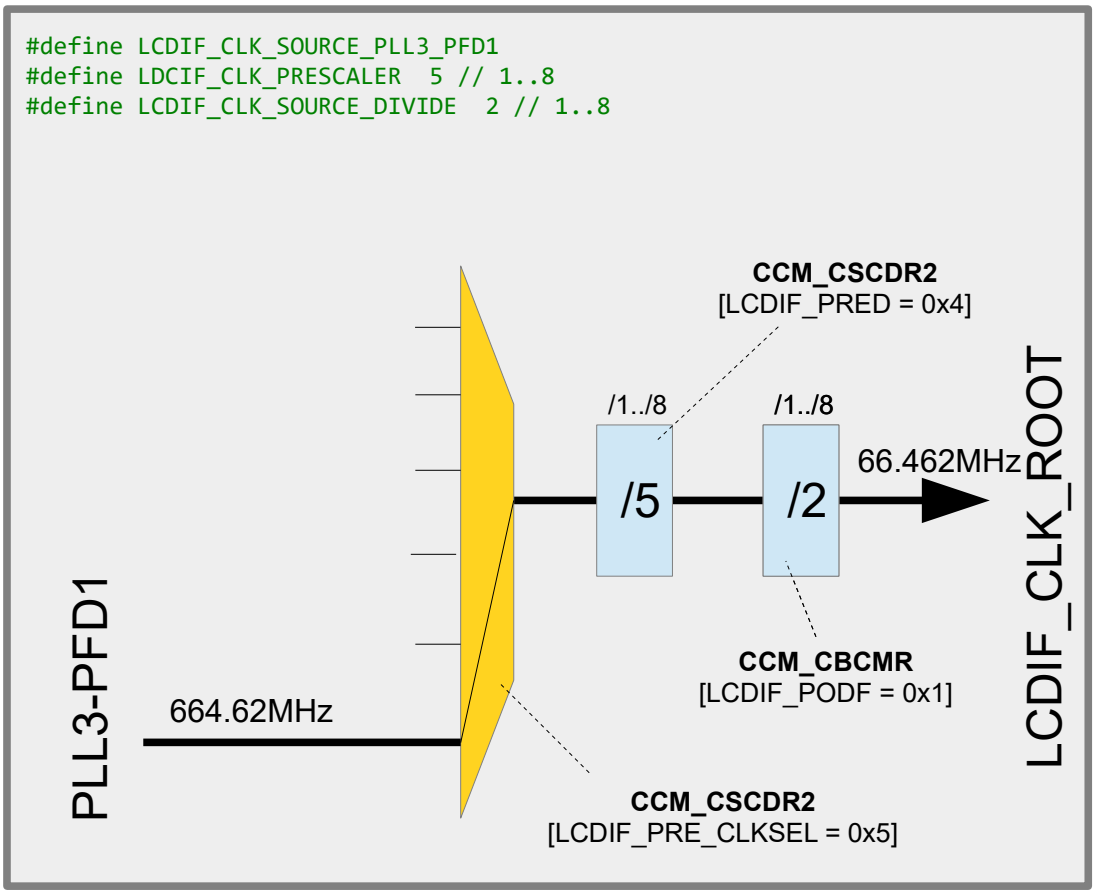
```
#define LCDIF_CLK_SOURCE_PLL3_PFD3
#define LDCIF_CLK_PRESCALER  7 // 1..8
#define LCDIF_CLK_SOURCE_DIVIDE  1 // 1..8
```



```
#define LCDIF_CLK_SOURCE_PLL2_PFD1
```



```
#define LCDIF_CLK_SOURCE_PLL3_PFD1
```



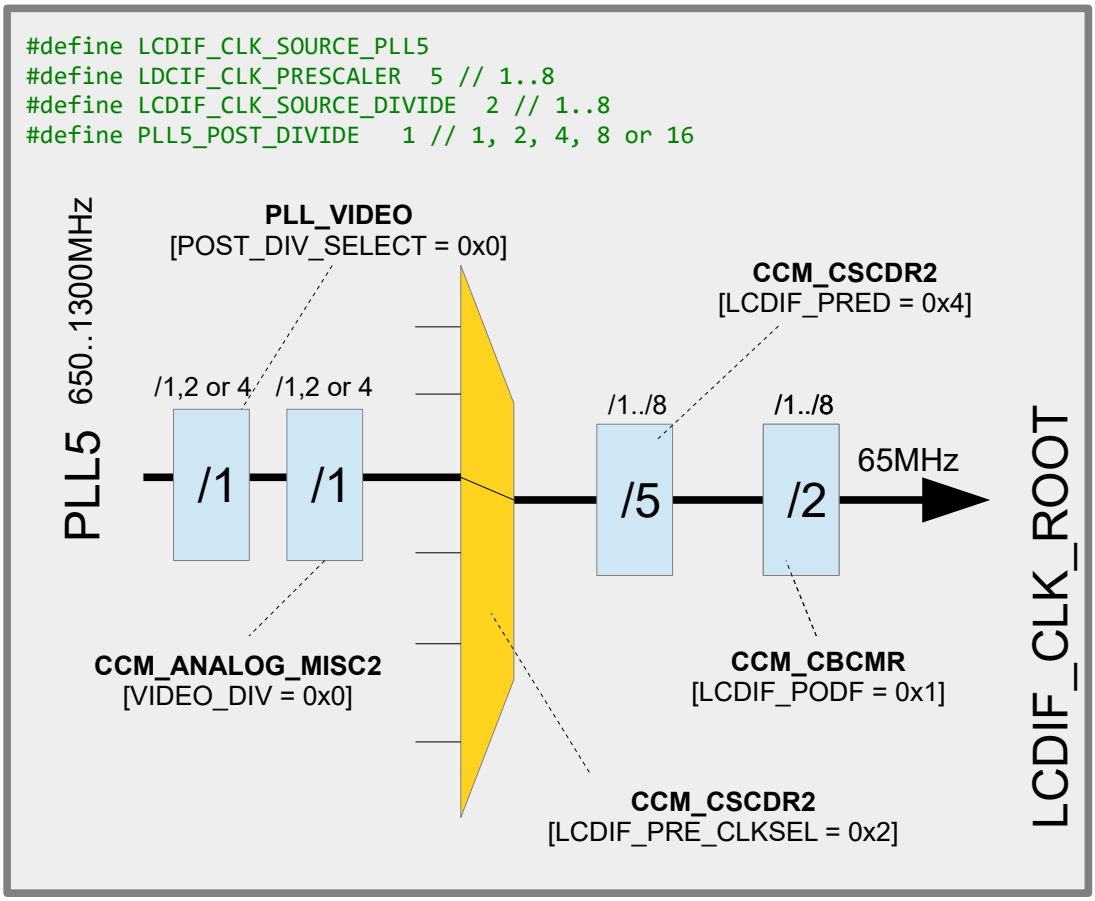
PLL5 is referred to as the Video PLL. It is a fractional multiplier PLL for operation in the range 650MHz to 1300MHz and is the default source for the LCDIF_CLK_ROOT.
Its exact frequency is defined by

```
PLL frequency = 24MHz * (VIDEO_PLL_LOOP_DIVIDER +
                         (VIDEO_PLL_LOOP_NUMERATOR/VIDEO_PLL_LOOP_DENOMINATOR))
```

where the loop divider can range from 27 to 54, the numerator and denominator from 0..0x3fffffff, (where the numerator should never be greater than the denominator) which allows a frequency in the range to be defined to better than 1Hz.

In additional it can be post divided by 1, 2, 4 8 or 16 if required.

```
#define LCDIF_CLK_SOURCE_PLL5
```



The following shows a reference configuration for this clock source set to 30MHz pixel clock:

```
// Define LCDIF clock source and speed
//
#define LCDIF_CLK_SOURCE_PLL5      // LCDIF_CLK_ROOT sourced from pll5 (video PLL
                                      frequency = 24MHz * (VIDEO_PLL_LOOP_DIVIDER +
                                (VIDEO_PLL_LOOP_NUMERATOR/VIDEO_PLL_LOOP_DENOMINATOR))

#define VIDEO_PLL_LOOP_DIVIDER    30 // possible range 27..54 [648MHz..1.296MHz]
#define VIDEO_PLL_LOOP_NUMERATOR  0  // if not set to 0 make sure that the numerator
                                  is smaller than the denominator (range 0..0x3fffffff)
#define VIDEO_PLL_LOOP_DENOMINATOR  0 // (range 0..0x3fffffff)
#define PLL5_POST_DIVIDE          2   // 1, 2, 4, 8 or 16

#define LDCIF_CLK_PRESCALER       4   // 1..8
#define LCDIF_CLK_SOURCE_DIVIDE   3   // 1..8

#define LCD_READ_PRIORITY         3   // 1..15 (15 is highest priority) - increase if
flicker or a shft is encountered (default 1 should be adequate for smaller displays
and when the buffer is in internal RAM but larger displays with SDRAM frame buffer may
need an increase)
```

The uTaskerV1.4 project define `SUPPORT_GLCD` enables a reference application in the display matching the board's display type and dimensions.
See also the uGLCDLIB guide at https://www.utasker.com/docs/uTasker/uTaskerLCD.PDF

3. Real Time Clock

No change

4. Internal Clock Monitoring

The i.MX RT 1052 has two peripheral outputs called CCM_CLKO1 (on GPIO_SD_B0_04, or GPIO3-16) and CCM_CLKO2 (on GPIO_SD_B0_05, or GPIO3-17) which can be attached to various internal clocks. This can be useful to verify that these clocks really have the frequencies that are expected, as well as generating signals for external usage. Fast internal signals can also be divided down by a pre-scaler with a value between 1 and 8.

These are the clocks that can be selected:

CCM_CLKO1

```
USB1_PLL_DIV2
SYS_PLL_DIV2
VIDEO_PLL_DIV2
SEMC_CLK_ROOT
LCDIF_PIX_CLK_ROO
AHB_CLK_ROOT
IPG_CLK_ROOT
PERCLK_ROOT
CKIL_SYNC_CLK_ROOT
PLL4_MAIN_CLK
```

CCM_CLKO2

```
USDHC1_CLK_ROOT
LPI2C_CLK_ROOT
OSC_CLK_ROOT
LPSPI_CLK_ROOT
USDHC2_CLK_ROOT
SAI1_CLK_ROOT
SAI2_CLK_ROOT
SAI3_CLK_ROOT
TRACE_CLK_ROOT
CAN_CLK_ROOT
FLEXSPI_CLK_ROOT
UART_CLK_ROOT
SPDIF0_CLK_ROOT
```

Two macros are made available to configure the pin and output the desired signals:

```
fnSetClock1Output(CLK, div)
fnSetClock2Output(CLK, div)
```

whereby examples of utilisation are:

```
    fnSetClock1Output(SYS_PLL_DIV2, 4);
                              // output PLL2/2 with pre-scaler 4 on CCM_CLKO1
    fnSetClock2Output(UART_CLK_ROOT, 1);
                              // output UART_CLK_ROOT with no pre-scaler on CCM_CLKO2
```

5. LPUART

The i.MX RT 1052 LPUART driver is shared with the Kinetis LPUART driver and supports interrupt and DMA driven modes. Minor differences due to the i.MX RT 1052 hardware are controlled by the platform definition `_iMX`. Sharing the driver is possible due to the high compatibility between the LPUART used in the Kinetis parts and i.MX RT 1052 and improves maintenance since only one source needs to be managed and the i.MX RT 1052 inherits the features from the mature Kinetis driver.

See the UART user's manual for general details of usage:
http://www.utasker.com/docs/uTasker/uTaskerUART.PDF

It should be kept in mind that i.MX RT 1052 peripherals tend to be numbered 1..n and not 0..n-1, as is the case with Kinetis peripherals. To avoid confusion it is recommended to use the defines

```
iMX_LPUART_1
iMX_LPUART_2
iMX_LPUART_n
```

instead of channel numbers, whereby `iMX_LPUART_1` is in fact 0.

The LPUARTs can be multiplexed onto various physical pins. When a specific LPUART is used it defaults to a certain set of pins if nothing else is specified – the following gives a list of the LPUART multiplex pins and the definition that can be used to control the use of alternatives where possible and if required:

Channel	Default	Alternative 1	Alternative 2
iMX_LPUART_1 (0) LPUART1_RXD LPUART1_TXD	 GPIO_AD_B0_13 GPIO_AD_B0_12		
iMX_LPUART_2 (1) LPUART2_RXD LPUART2_TXD	 GPIO_AD_B1_03 GPIO_AD_B1_02	**LPUART2_ON_SD** GPIO_SD_B1_10 GPIO_SD_B1_11	
iMX_LPUART_3 (2) LPUART3_RXD LPUART3_TXD	 GPIO_AD_B1_07 GPIO_AD_B1_06	**LPUART3_ON_B0** GPIO_B0_09 GPIO_B0_08	**LPUART3_ON_EMC** GPIO_EMC_14 GPIO_EMC_13
iMX_LPUART_4 (3) LPUART4_RXD LPUART4_TXD	 GPIO_B1_01 GPIO_B1_00	**LPUART4_ON_SD** GPIO_SD_B1_00 GPIO_SD_B1_01	**LPUART4_ON_EMC** GPIO_EMC_20 GPIO_EMC_19
iMX_LPUART_5 (4) LPUART5_RXD LPUART5_TXD	 GPIO_B1_13 GPIO_B1_12	**LPUART5_ON_EMC** GPIO_EMC_24 GPIO_EMC_23	
iMX_LPUART_6 (5) LPUART6_RXD LPUART6_TXD	 GPIO_EMC_26 GPIO_EMC_25	**LPUART6_ON_AD** GPIO_AD_B0_03 GPIO_AD_B0_02	
iMX_LPUART_7 (6) LPUART7_RXD LPUART7_TXD	 GPIO_EMC_32 GPIO_EMC_31	**LPUART7_ON_SD** GPIO_SD_B1_09 GPIO_SD_B1_08	
iMX_LPUART_8 (7) LPUART8_RXD	 GPIO_EMC_39	**LPUART8_ON_AD** GPIO_AD_B1_11	**LPUART8_ON_SD** GPIO_SD_B0_05

LPUART8_TXD	GPIO_EMC_38	GPIO_AD_B1_10	GPIO_SD_B0_04

For example, if LPUART2 is used its default pin-out Tx/RX is on
GPIO_AD_B1_02/GPIO_AD_B1_03 [GPIO1-IO18 and GPIO1-IO19]
but can be set instead to

GPIO_SD_B1_11/GPIO_SD_B1_10 [GPIO3-IO11 and GPIO3-IO10]
by enabling the define LPUART2_ON_SD. This define is set in `app_hw_iMX.h`

It is to be noted that the µTasker project is often chosen due to its immediate support for free-running UART Rx DMA on all serial interfaces, which is something that is generally not found in other solutions. The i.MX RT 1052 thus could immediately inherit this operation.

6. LPI2C

.

7. FLEXCAN

8. PIT

9. DMA

10. GPIO

The i.MX RT 1052 GPIO / peripheral concept is quite different to the Kinetis concept. See the following video for an overview and also details concerning how the project was solved to ensure compatibility between Kinetis and i.MX RT:
https://www.youtube.com/watch?v=SmFTi8hlba0&list=PLWKlVb_MqDQFZAulrUywU30v869JBYi9Q&index=29

GPIOs can also be used as interrupts (see `IRQ_TEST` in `Port_Interrupts.h` as reference to use).

Each GPIO can be configured to generate an interrupt on low levels, high levels, falling edges or rising edges (or both falling and rising edges). The µTasker GPIO interrupt driver allows the user to assign an individual interrupt callback to each GPIO but it is useful to understand that the i.MX RT 1052 actually has the following interrupt vectors:

- `PORT1-0` – individual vector for these pins
- `PORT1-1`
- `PORT1-2`
- `PORT1-3`
- `PORT1-4`
- `PORT1-5`
- `PORT1-6`
- `PORT1-7`

- `PORT1-15..PORT1-8` – these 8 pins share a single vector

- `PORT1-31..PORT1-16` – these 16 pins share a single vector

- `PORT2-15..PORT2-0` – these 16 pins share a single vector
- `PORT2-31..PORT2-16` – these 16 pins share a single vector

- `PORT3-15..PORT3-0` – these 16 pins share a single vector
- `PORT3-31..PORT3-16` – these 16 pins share a single vector

- `PORT5-15..PORT5-0` – these 16 pins share a single vector
- `PORT5-31..PORT5-16` – these 16 pins share a single vector

`PORT1-0..PORT1-7` are the most efficient interrupts since the handler doesn't need to identify which port bits caused the interrupt before dispatching the user interrupt callback. Ports with an interrupt vector shared by more than one pin are slightly less efficient due to the need to identify which source or sources caused the interrupt and then dispatch one or more call-backs. When multiple GPIO input interrupts are pending at the same time the callbacks are dispatched in the order of the lower pin number up to the highest pin number.

It is possible to trigger DMA transfers on some GPIO inputs when they are connected to the XBAR. See the following video for more details: https://www.youtube.com/watch?v=zNWIG-O7ZW0&feature=youtu.be

11. RAM and Cache

The i.MX RT 1052 contains 512k of internal RAM which is constructed of 16 banks of 32k each. These banks can each be assigned to one of 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 1052 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

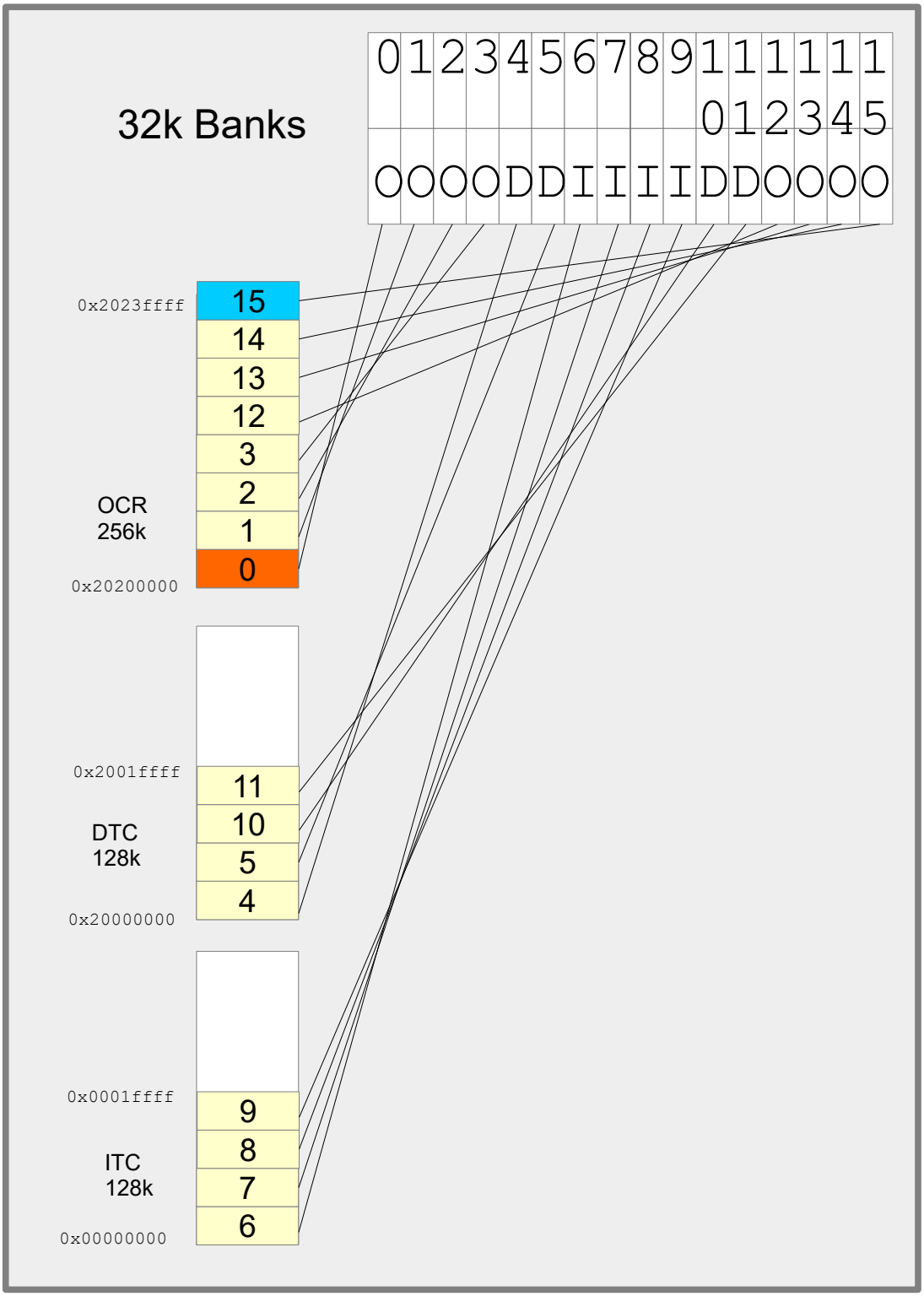
```
#define ENABLE_INSTRUCTION_CACHE
```

and

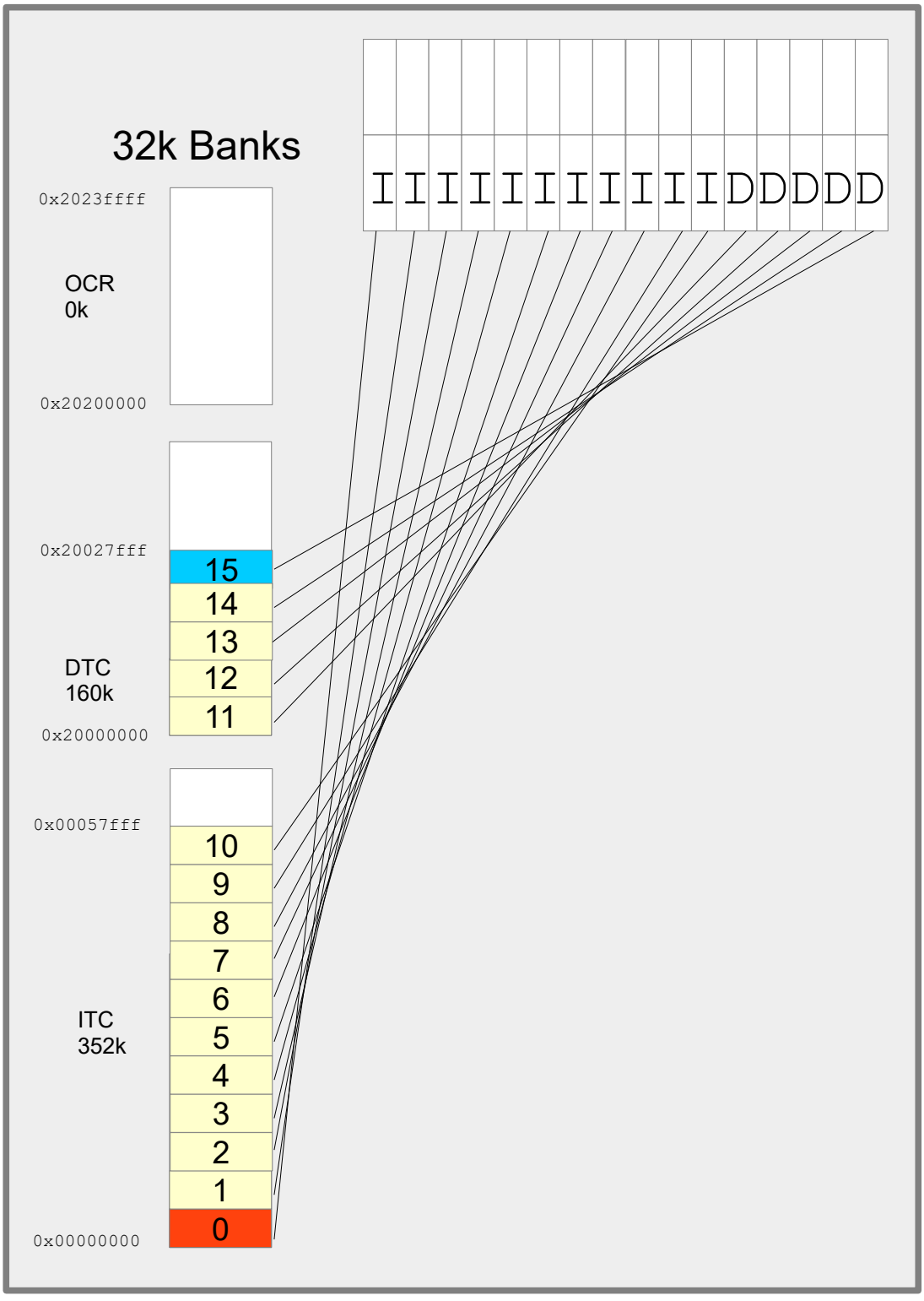
```
#define ENABLE_DATA_CACHE
```

The FlexRAM controller configures the RAM banks at reset based on eFuse settings. The standard setting (when nothing else has been programmed) is for 256k OCRAM, 128k DTCM and 128k ITCM; the ROM loader may use the first 64k of the OCRAM when it operates.

This default setting is assumed in the µTasker project to avoid special configuration requirements and therefore out of reset the banks are configured to give this memory map and layout:



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 1052, 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).

12. Boot Mode

13. Ethernet

14. LCD

15. Conclusion

Modifications:
V0.06 18.11.2019: Initial version in development

Appendix A – Hardware Dependencies

a) Space for first Appendix