

Embedding it better...



μTasker Document

i.MX RT 1064 – the μTasker way

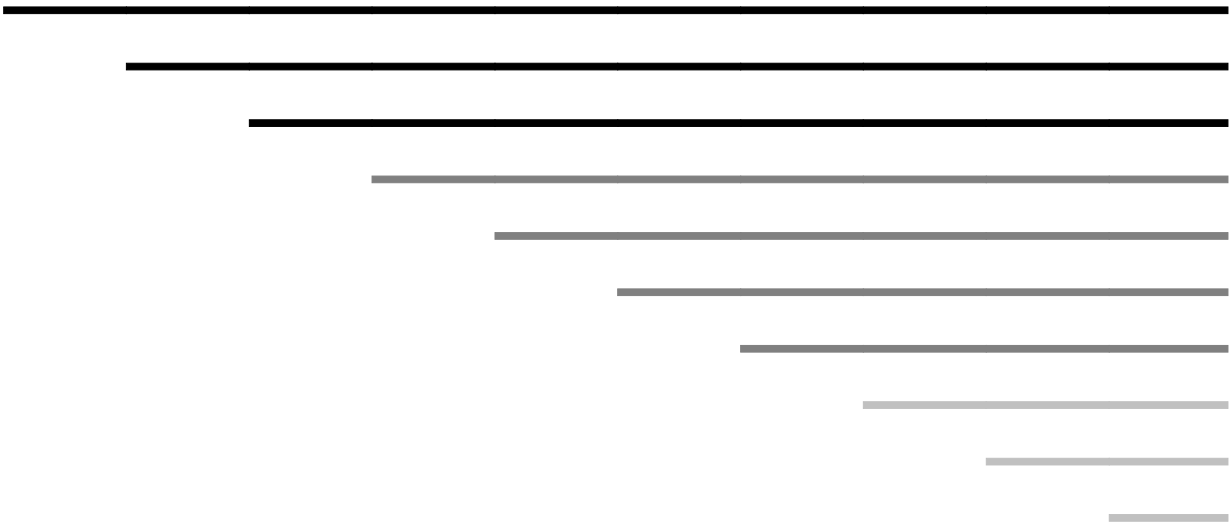


Table of Contents

| | |
|--|----|
| 1. Introduction..... | 3 |
| 2. Clocking..... | 4 |
| 1.1.ARM Core Clock..... | 5 |
| 1.2.IPG Clock – used by ADC and XBAR..... | 8 |
| 1.3.PERCLK – used by PIT and GPT..... | 9 |
| 1.4.UART_CLK_ROOT – used by all LPUARTs..... | 10 |
| 1.5.USDHC1_CLK_ROOT/USDHC2_CLK_ROOT..... | 12 |
| 1.6.SEMC_CLK_ROOT..... | 12 |
| 1.7.FLEXSPI_CLK_ROOT..... | 13 |
| 1.8.LPSPI_CLK_ROOT..... | 15 |
| 1.9.TRACE_CLK_ROOT..... | 15 |
| 1.10.SAI1_CLK_ROOT/SAI2_CLK_ROOT/SAI2_CLK_ROOT..... | 15 |
| 1.11.LPI2C_CLK_ROOT – used by all LPI2C controllers..... | 16 |
| 1.12.CAN_CLK_ROOT..... | 18 |
| 1.13.SPDIF0_CLK_ROOT..... | 20 |
| 1.14.FLEXIO1_CLK_ROOT..... | 20 |
| 3. Internal Clock Monitoring..... | 21 |
| 4. LPUART..... | 22 |
| 5. LPI2C..... | 23 |
| 6. FLEXCAN..... | 24 |
| 7. PIT..... | 25 |
| 8. DMA..... | 26 |
| 9. GPIO..... | 27 |
| 10. RAM and Cache..... | 28 |
| 11. Boot Mode..... | 32 |
| 12. Ethernet..... | 33 |
| 13. Conclusion..... | 34 |
| Appendix A – Hardware Dependencies..... | 35 |
| a)Space for first Appendix..... | 35 |

1. Introduction

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

The **MIMXRT1064-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.

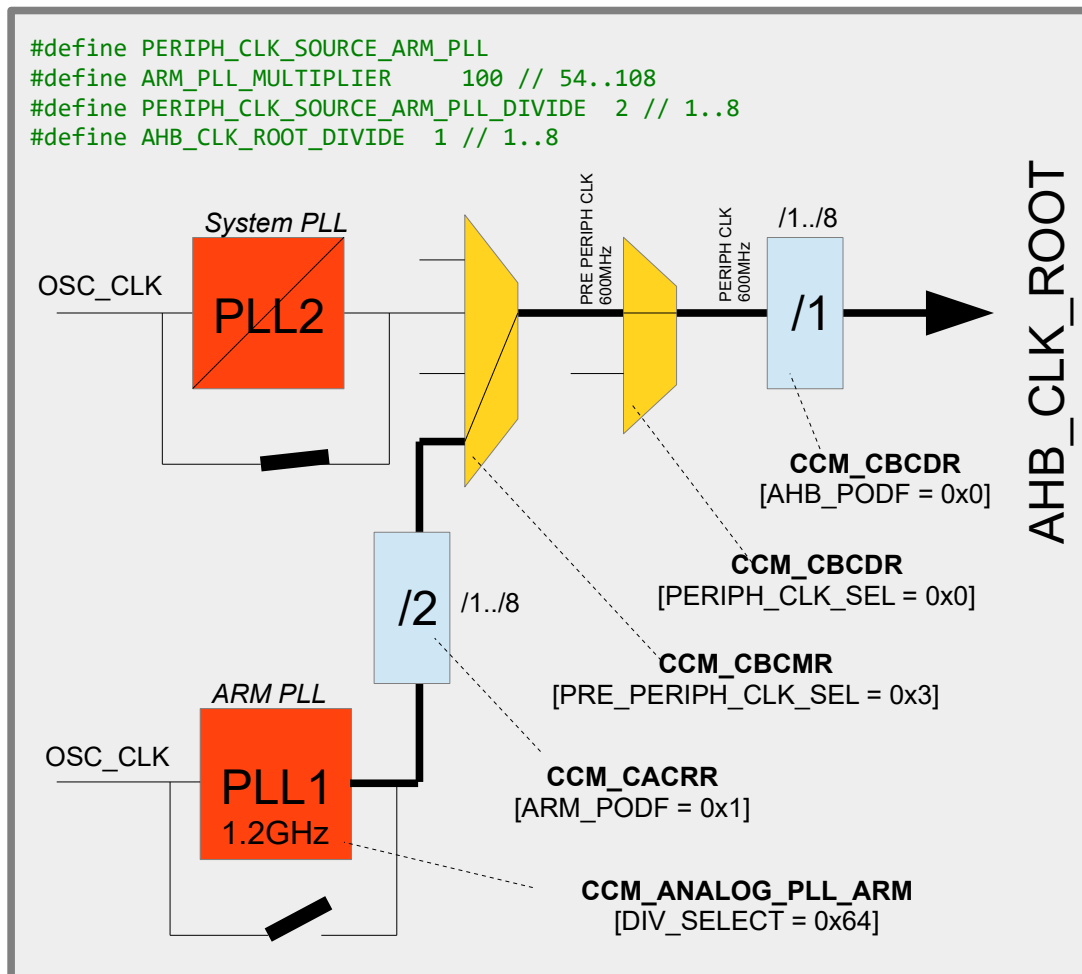
| Fraction | System PLL (528MHz PLL2) ((528MHz * 18) / fraction) | USB1 PLL (480MHz PLL3) ((480MHz * 18) / fraction) |
|----------|--|--|
| 12 | 792MHz | 720MHz - default PFD0 |
| 13 | 731.0769231MHz | 664.6153846MHz - default PFD1 |
| 14 | 678.8571429MHz | 617.1428571MHz |
| 15 | 633.6MHz | 576MHz |
| 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 1064 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 by ADC and XBAR

1.3. PERCLK – used by PIT and GPT

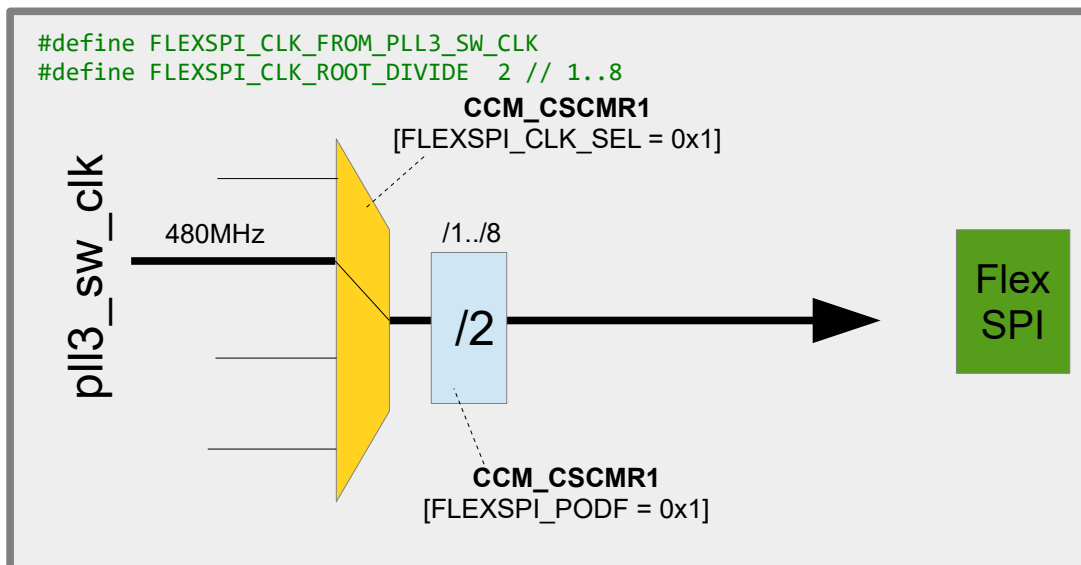
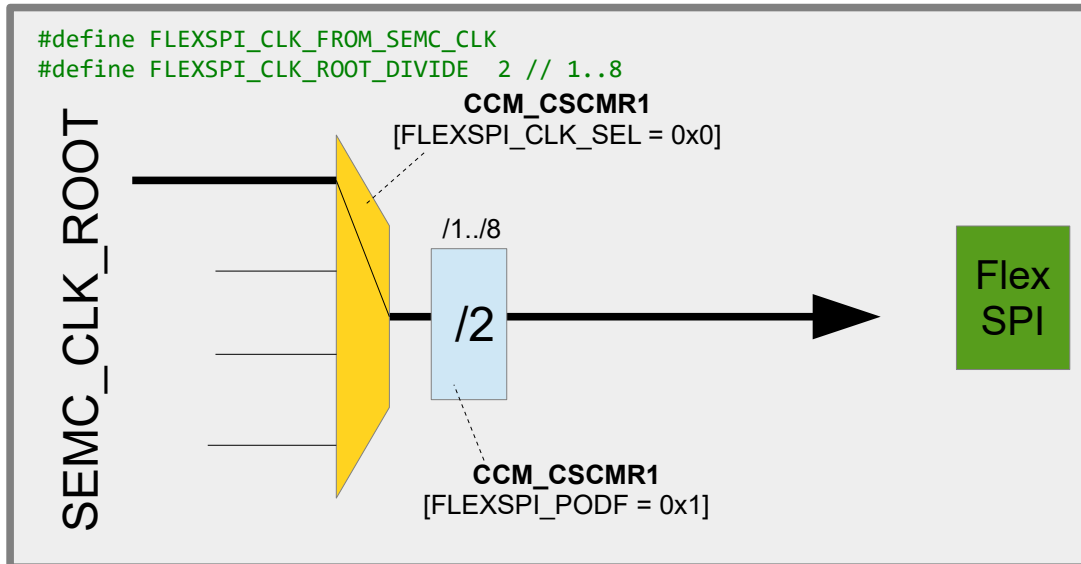
1.4. UART_CLK_ROOT – used by all LPUARTs

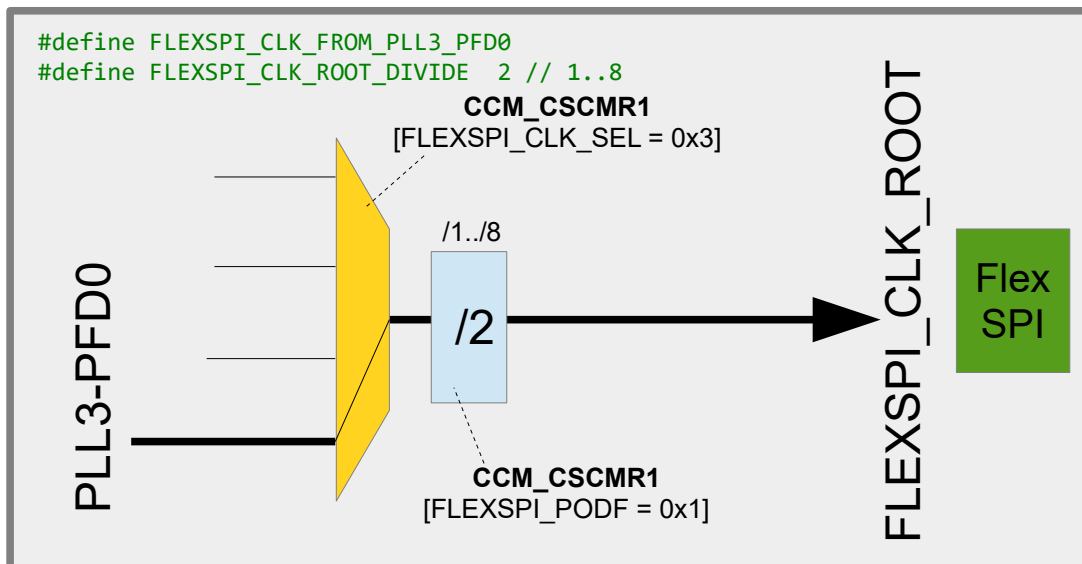
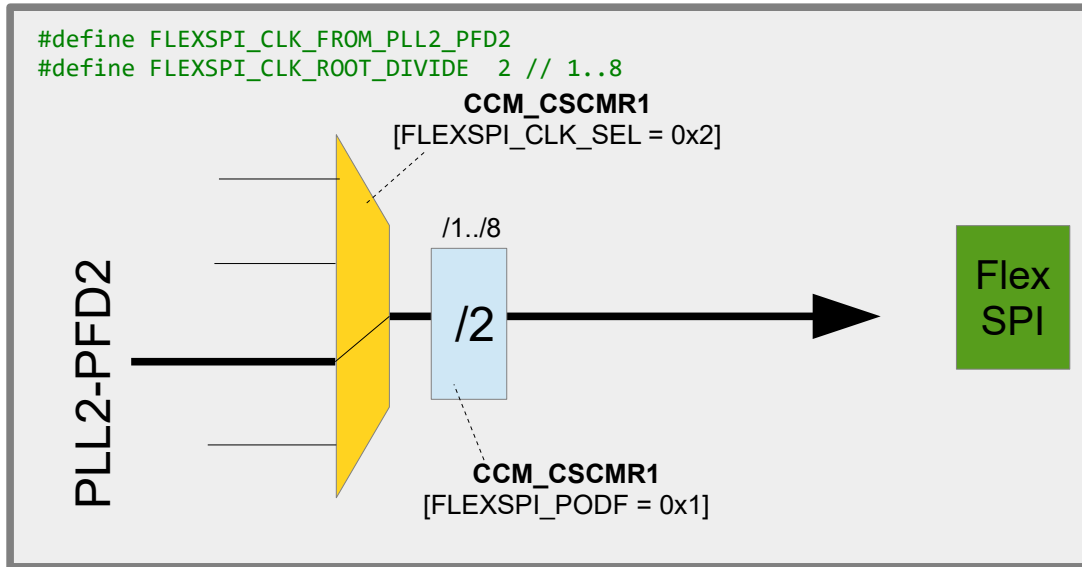
1.5. USDHC1_CLK_ROOT/USDHC2_CLK_ROOT

1.6. SEMC_CLK_ROOT

1.7. FLEXSPI_CLK_ROOT

The default source of the FLEXSPI_CLK_ROOT is from the SEMC_CLK_ROOT (see previous section) with a pre-scaler of 2. As discussed in the previous section, the SEMC_CLK_ROOT is per default the PERIPH_CLK (the clock root supplying the core clock pre-scaler) divided by 3. FLEXSPI_CLK_ROOT frequency must not exceed 322MHz.





FlexSPI2 Clk Root

Has the last 2 options but not the first two; instead it has the options for PLL2 (528MHz) and PLL3-PD1

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

1.12. CAN_CLK_ROOT

1.13. SPDIF0_CLK_ROOT

To add..

1.14. FLEXIO1_CLK_ROOT

To add..

3. Internal Clock Monitoring

The i.MX RT 1064 has two peripheral outputs called `CCM_CLKO1` (on `GPIO_SD_B1_02`, or `GPIO3-22`) and `CCM_CLKO2` (on `GPIO_SD_B1_03`, or `GPIO3-23`) 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

```
PLL3_SW_CLK_DIV2
PLL2_DIV2
ENET_PLL_DIV2
SEMC_CLK_ROOT
AHB_CLK_ROOT
IPG_CLK_ROOT
PERCLK_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(ENET_PLL_DIV2, 4);
// output ENET_PLL/2 with pre-scaler 4 on CCM_CLKO1
fnSetClock2Output(UART_CLK_ROOT, 1);
// output UART_CLK_ROOT with no pre-scaler on CCM_CLKO2
```

4. LPUART

The i.MX RT 1064 LPUART driver is shared with the Kinetis LPUART driver and supports interrupt and DMA driven modes. Minor differences due to the i.MX RT 1064 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 1064 and improves maintenance since only one source needs to be managed and the i.MX RT 1064 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 1064 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.

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 1021 thus could immediately inherit this operation.

5. LPI2C

The i.MX RT 1064 LPI2C driver is shared with the Kinetis LPUART driver and supports interrupt and DMA driven modes. Minor differences due to the i.MX RT 1064 hardware are controlled by the platform definition `_iMX`. Sharing the driver is possible due to the high compatibility between the LPI2C used in the Kinetis parts and i.MX RT 1064 and improves maintenance since only one source needs to be managed and the i.MX RT 1064 inherits the features from the mature Kinetis driver.

See the I²C user's manual for general details of usage:

http://www.utasker.com/docs/uTasker/uTasker_I2C.pdf

It should be kept in mind that i.MX RT 1064 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_LPI2C_1  
iMX_LPI2C_2  
iMX_LPI2C_n
```

instead of channel numbers, whereby `iMX_LPI2C_1` is in fact 0.

6. FLEXCAN

The i.MX RT 1064 CAN driver is shared with the Kinetis CAN driver. Minor differences due to the i.MX RT 1064 hardware are controlled by the platform definition `_iMX`. Sharing the driver is possible due to the high compatibility between the FLEXCAN used in the Kinetis parts and i.MX RT 1064 and improves maintenance since only one source needs to be managed and the i.MX RT 1021 inherits the features from the mature Kinetis driver.

The FLEXCAN in the i.MX RT 1064 supports 64 receive buffers as opposed to the 16 in the FLEXCAN in the Kinetis parts.

See the CAN user's manual for general details of usage:

<http://www.utasker.com/docs/uTasker/uTaskerCAN.PDF>

It should be kept in mind that i.MX RT 1064 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_FLEXCAN_1  
iMX_FLEXCAN_2
```

instead of channel numbers, whereby `iMX_FLEXCAN_1` is in fact 0.

7. PIT

The i.MX RT 1064 PIT driver is shared with the Kinetis PIT driver. Minor differences due to the i.MX RT 1064 hardware are controlled by the platform definition `_iMX`. Sharing the driver is possible due to the high compatibility between the PIT used in the Kinetis parts and i.MX RT 1064 and improves maintenance since only one source needs to be managed and the i.MX RT 1064 inherits the features from the mature Kinetis driver.

See the HW timer user's manual for general details of usage:

<http://www.utasker.com/docs/uTasker/uTaskerHWTimers.PDF>

(see `TEST_PIT` [`TEST_PIT_SINGLE_SHOT`, `TEST_PIT_PERIODIC` and `TEST_PIT_64_BIT` in `ADC_Timers.h` as reference to use).

8. DMA

The i.MX RT 1064 DMA driver is shared with the Kinetis eDMA driver. Minor differences due to the i.MX RT 1064 hardware are controlled by the platform definition `_iMX`. Sharing the driver is possible due to the high compatibility between the eDMA and DMA MUX used in the Kinetis parts and i.MX RT 1064 and improves maintenance since only one source needs to be managed and the i.MX RT 1064 inherits the features from the mature Kinetis driver.

9. GPIO

The i.MX RT 1064 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=PLWKIVb_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 1021 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.

10. RAM and Cache

The i.MX RT 1064 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 1064 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

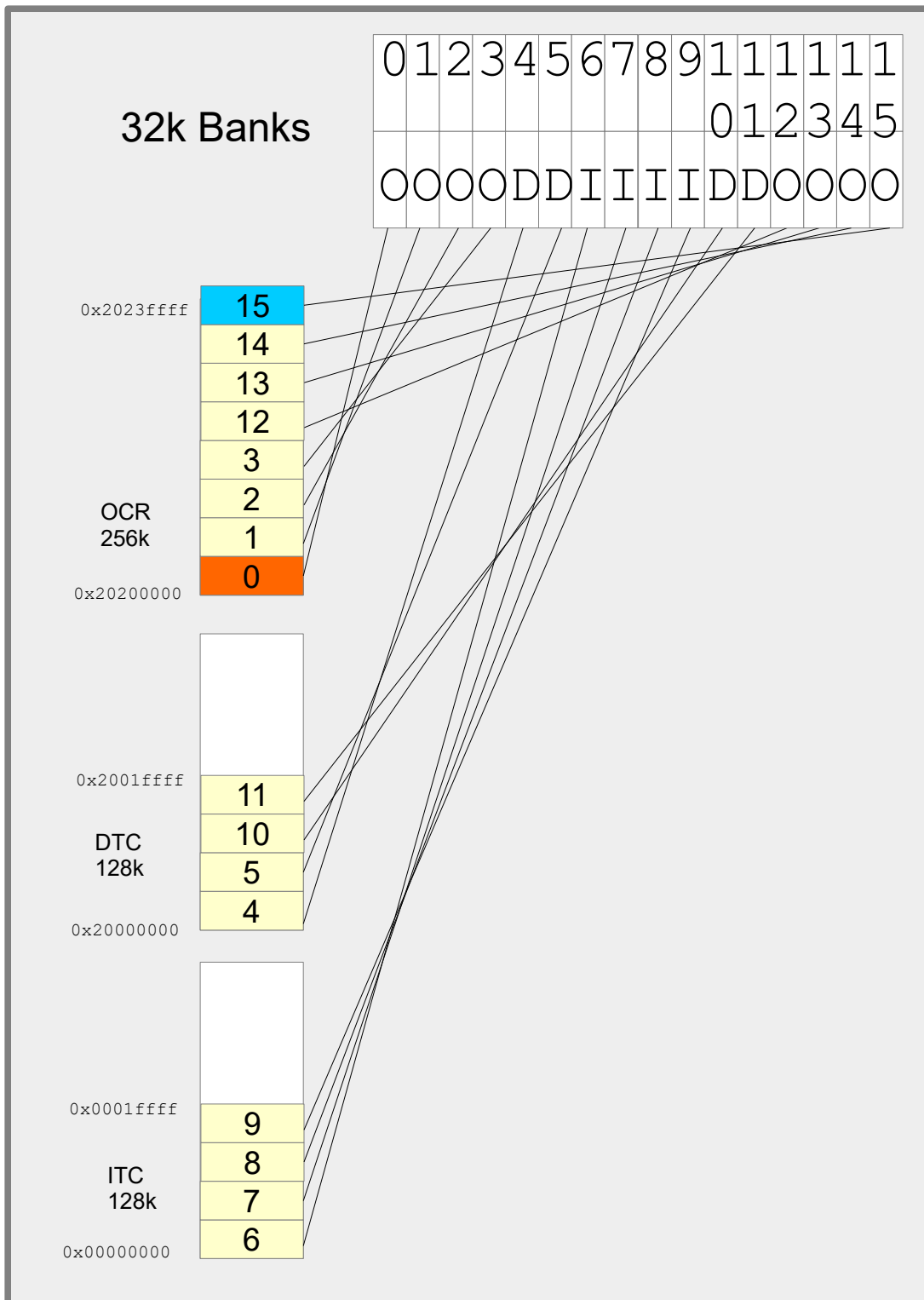
```
#define ENABLE_INSTRUCTION_CACHE
```

and

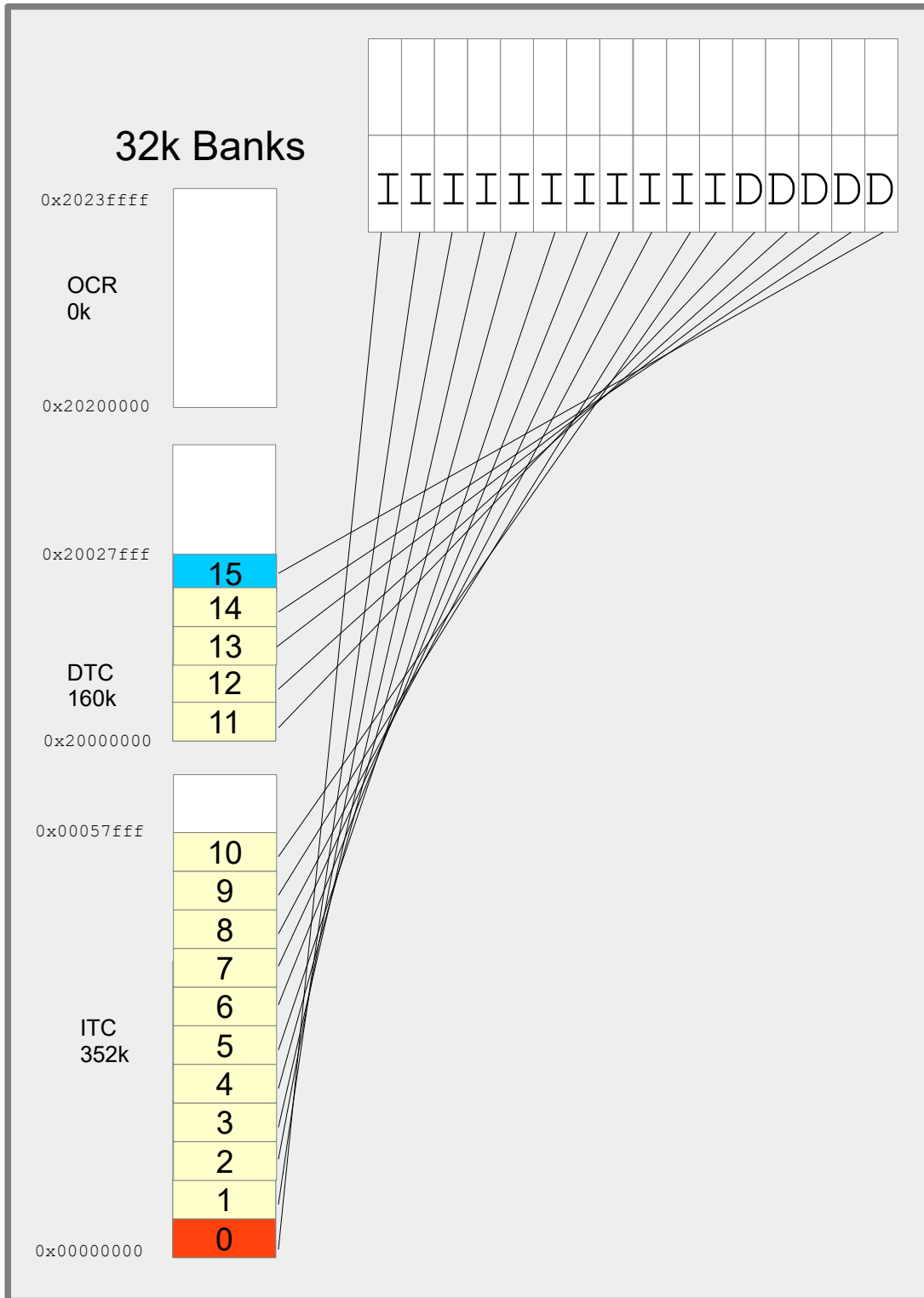
```
#define ENABLE_DATA_CACHE
```

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 1064, 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).

11. Boot Mode

12. Ethernet

13. Conclusion

Modifications:

V0.00 23.06.2020: Initial version in development

Appendix A – Hardware Dependencies

a) Space for first Appendix