**µTasker Document**

**i.MX RT 1170 – the µTasker way**

## Table of Contents

# 1. Introduction

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

The **MIMXRT1170-EVK** is used as test vehicle throughout the document but any board using the part can be easily configured based on the contained details.

The i.MX RT 117x has an optional 400MHz Cortex-M4 core in addition to its 1GHz Cortex-M7 core. When nothing else is mentioned in this document the details are relevant to the Cortex-M7 core.

## 2. Clocking

Unlike the i.MX RT 10xx clock tree which is quite specific to the device in question the i.MX RT 117x clock tree is of a more general matrix nature.

Whereby in the i.MX RT 10xx a certain peripheral clock root may have have two or three possible clock sources (with some of these also being other peripheral clock roots) the peripheral root clock selection in the i.MX RT 117x has the capability of selecting any of 8 reference clocks, divided down by a dedicated pre-scaler. Some of the reference clocks are available to all peripheral clock roots and in some cases there are various specific choices.

An all there are 23 reference clocks and so each of the peripheral clock roots can choose one of 8 of these overall possible sources, as show in this table

**CLK ROOT
ID - Name**

Column headers (left to right):
OSC_24M, OSC_32K, OSC_RC_16M, OSC_RC_48M_DIV2, OSC_RC_48M, OSC_RC_400M, PLL_VIDEO, PLL_AUDIO, SYS_PLL3_DIV2, SYS_PLL3_PFD3, SYS_PLL3_PFD2, SYS_PLL3_PFD1, SYS_PLL3_PFD0, SYS_PLL3_CLK, SYS_PLL2_PFD3, SYS_PLL2_PFD2, SYS_PLL2_PFD1, SYS_PLL2_PFD0, SYS_PLL2_CLK, SYS_PLL1_DIV5, SYS_PLL1_DIV2, SYS_PLL1_CLK, PLL_ARM

Row list:

0  – M7
1  – M4
2  – BUS
3  – LPSR
4  – SEMC
5  – CSSYS
6  – CSTRACE
7  – M4_SYSTICK
8  – M7_SYSTICK
9  – ADC1
10 – ADC2
11 – ACMP
12 – FLEXIO1
13 – FLEXIO2
14 – GPT1
15 – GPT2
16 – GPT3
17 – GPT4
18 – GPT5
19 – GPT6
20 – FLEXSPI1
21 – FLEXSPI2
22 – CAN1
23 – CAN2
24 – CAN3
25 – LPUART1
26 – LPUART2
27 – LPUART3
28 – LPUART4
29 – LPUART5
30 – LPUART6
31 – LPUART7
32 – LPUART8
33 – LPUART9
34 – LPUART10
35 – LPUART11
36 – LPUART12
37 – LPI2C1
38 – LPI2C2
39 – LPI2C3
40 – LPI2C4
41 – LPI2C5
42 – LPI2C6
43 – LPSPI1
44 – LPSPI2
45 – LPSPI3
46 – LPSPI4
47 – LPSPI5
48 – LPSPI6
49 – EMV1
50 – EMV2
51 – ENET1
52 – ENET2
53 – ENET_QOS
54 – ENET_25M
55 – ENET_TIMER1
56 – ENET_TIMER2
57 – ENET_TIMER3
58 – USDHC1
59 – USDHC2
60 – ASRC
61 – MQS
62 – MIC
63 – SPDIF
64 – SAI1
65 – SAI2
66 – SAI3
67 – SAI4
68 – GPU2D
69 – ELCDIF
70 – LCDIFV2
71 – MIPI_REF
72 – MIPI_ESC
73 – CSI2
74 – CSI2_ESC
75 – CSI2_UI
76 – CSI
77 – CLKO1
78 – CLKO2

## 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)<br>`((528MHz * 18) / fraction)` | USB1 PLL (480MHz PLL3)<br>`((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
```



```
#define PERIPH_CLK_SOURCE_ARM_PLL
#define ARM_PLL_MULTIPLIER    100 // 54..108
#define PERIPH_CLK_SOURCE_ARM_PLL_DIVIDE  2 // 1..8
#define AHB_CLK_ROOT_DIVIDE  1 // 1..8
```
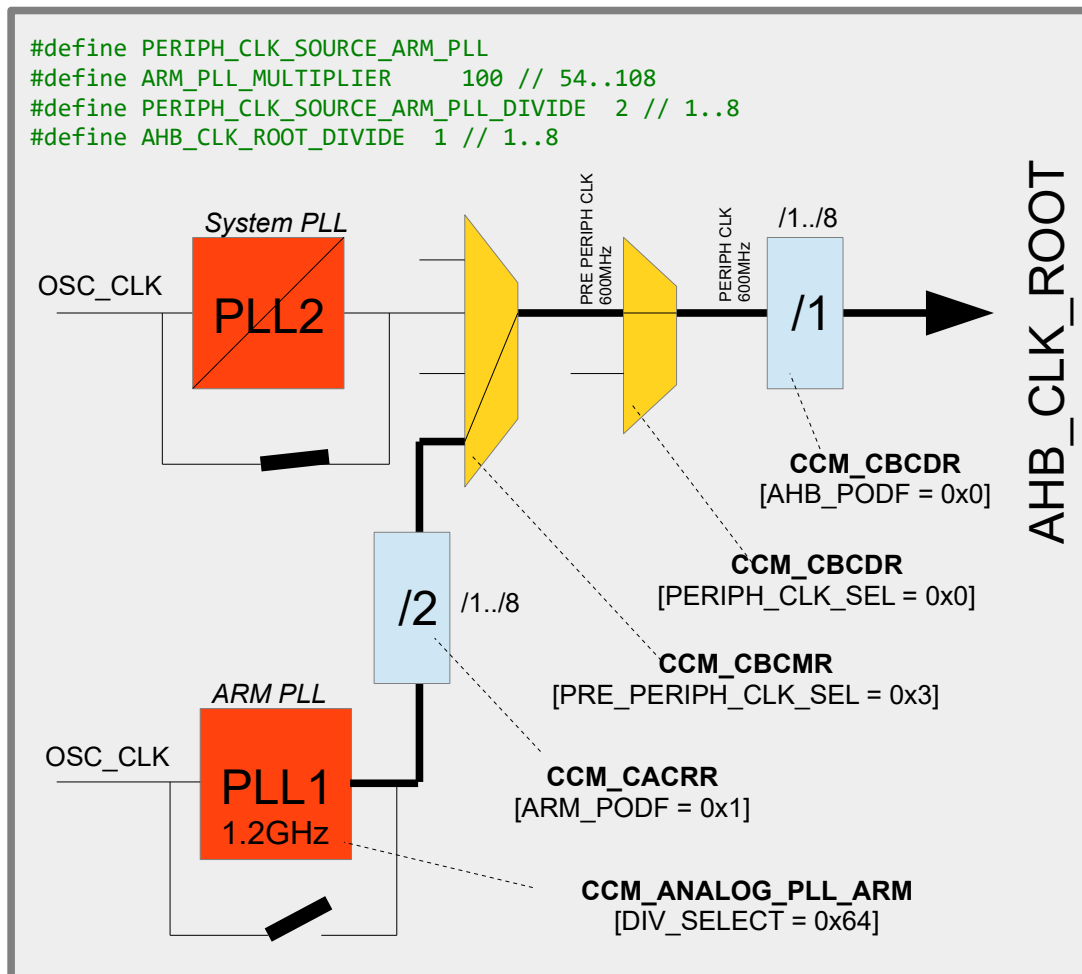
Furthermore the PLL2 fractional divider outputs that can be used are different.

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 GPOs, 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.

```
#define FLEXSPI_CLK_FROM_PLL2_PFD2
#define FLEXSPI_CLK_ROOT_DIVIDE  2 // 1..8
```

**CCM_CSCMR1**
[FLEXSPI_CLK_SEL = 0x2]

/1../8

/2

PLL2-PFD2

Flex SPI

**CCM_CSCMR1**
[FLEXSPI_PODF = 0x1]

```
#define FLEXSPI_CLK_FROM_PLL3_PFD0
#define FLEXSPI_CLK_ROOT_DIVIDE  2 // 1..8
```

**CCM_CSCMR1**
[FLEXSPI_CLK_SEL = 0x3]

/1../8

/2

PLL3-PFD0

FLEXSPI_CLK_ROOT

Flex SPI

**CCM_CSCMR1**
[FLEXSPI_PODF = 0x1]
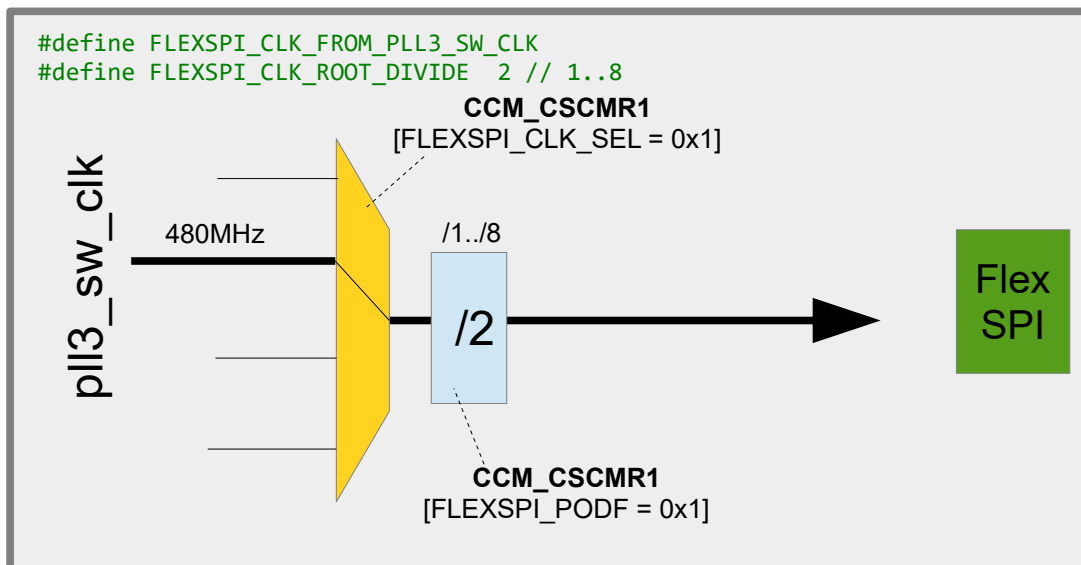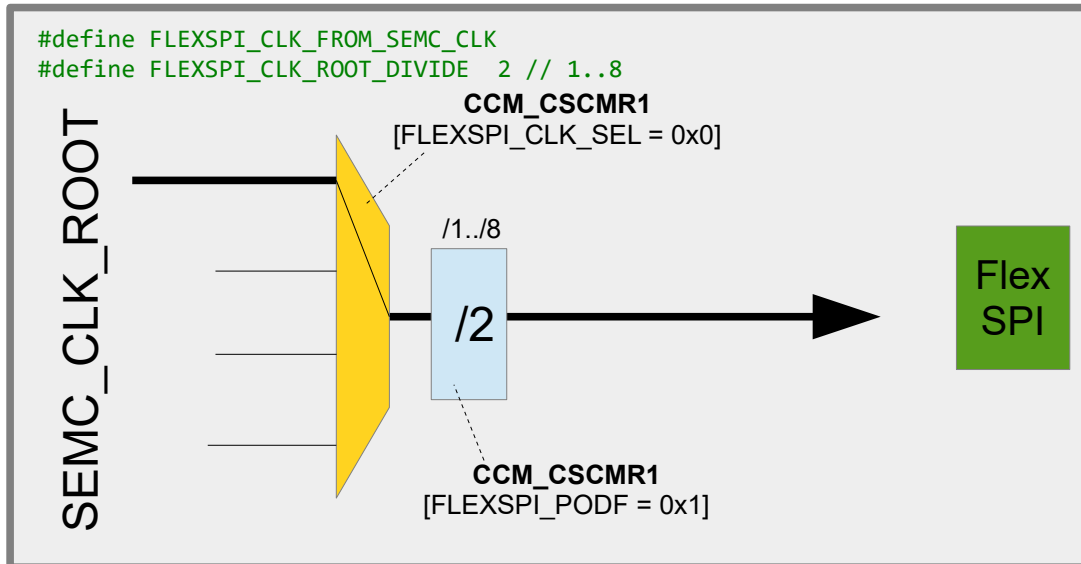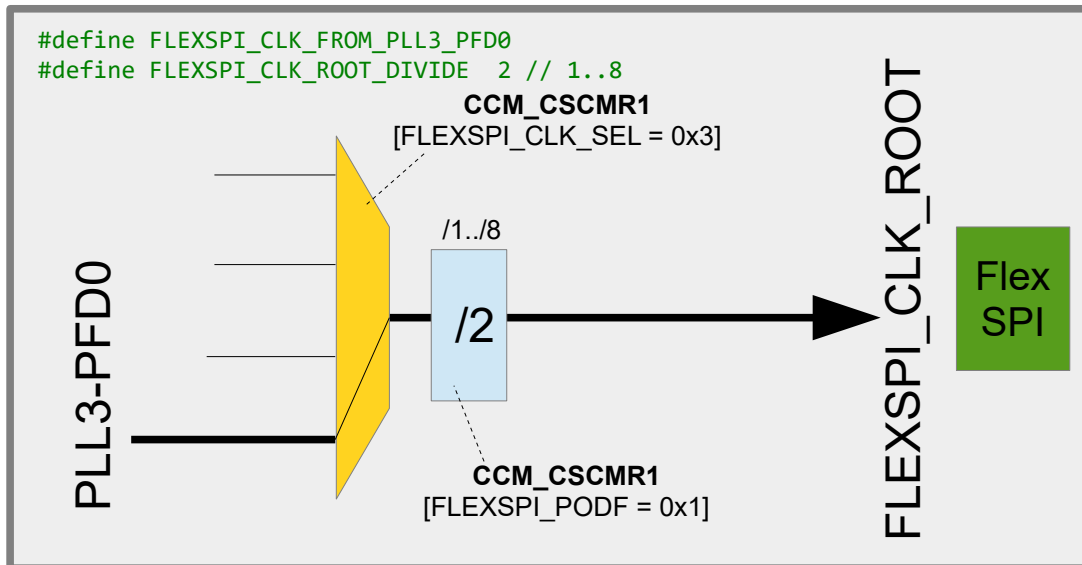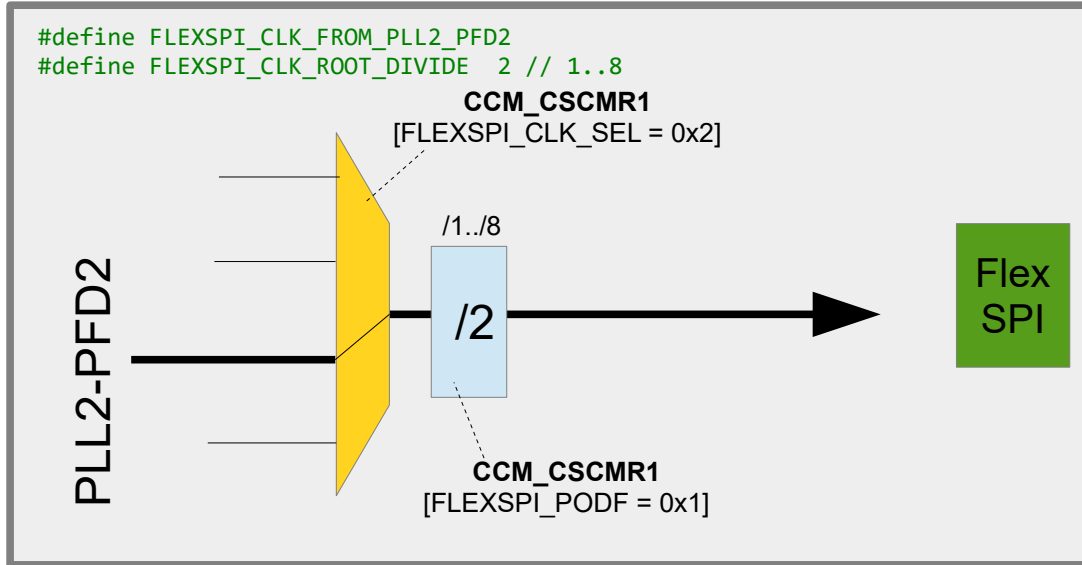
**FlexSPI2 Clk Root**
**Has the last 2 options but not the first two; instead it has the options for PLL2 (528MHz) and PLL3-PD1**

## 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 interal 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.

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)** <br> LPUART1_RXD <br> LPUART1_TXD | GPIO_AD_B0_07 <br> GPIO_AD_B0_06 | | |
| **iMX_LPUART_2 (1)** <br> LPUART2_RXD <br> LPUART2_TXD | GPIO_AD_B1_09 <br> GPIO_AD_B1_08 | **LPUART2_ON_EMC** <br> GPIO_EMC_23 <br> GPIO_EMC_22 | |
| **iMX_LPUART_3 (2)** <br> LPUART3_RXD <br> LPUART3_TXD | GPIO_AD_B0_15 <br> GPIO_AD_B0_14 | **LPUART3_ON_EMC** <br> GPIO_EMC_07 <br> GPIO_EMC_06 | |
| **iMX_LPUART_4 (3)** <br> LPUART4_RXD <br> LPUART4_TXD | GPIO_AD_B1_11 <br> GPIO_AD_B1_10 | **LPUART4_ON_EMC_LOW** <br> GPIO_EMC_03 <br> GPIO_EMC_02 | **LPUART4_ON_EMC_HIGH** <br> GPIO_EMC_33 <br> GPIO_EMC_32 |
| **iMX_LPUART_5 (4)** <br> LPUART5_RXD <br> LPUART5_TXD | GPIO_AD_B0_11 <br> GPIO_AD_B0_10 | **LPUART5_ON_EMC** <br> GPIO_EMC_39 <br> GPIO_EMC_38 | |
| **iMX_LPUART_6 (5)** <br> LPUART6_RXD <br> LPUART6_TXD | GPIO_SD_B1_01 <br> GPIO_SD_B1_00 | **LPUART6_ON_EMC** <br> GPIO_EMC_13 <br> GPIO_EMC_12 | |
| **iMX_LPUART_7 (6)** <br> LPUART7_RXD <br> LPUART7_TXD | GPIO_SD_B0_05 <br> GPIO_SD_B0_04 | **LPUART7_ON_EMC** <br> GPIO_EMC_35 <br> GPIO_EMC_34 | |
| **iMX_LPUART_8 (7)** <br> LPUART8_RXD <br> LPUART8_TXD | GPIO_SD_B1_03 <br> GPIO_SD_B1_02 | **LPUART8_ON_EMC** <br> GPIO_EMC_27 <br> GPIO_EMC_26 | |

For example, if LPUART2 is used its default pin-out Tx/RX is on
GPIO_AD_B1_08/GPIO_AD_B1_09 [GPIO1-IO24 and GPIO1-IO25]
but can be set instead to

 GPIO_EMC_22/GPIO_EMC_23 [GPIO2-IO22 and GPIO2-IO23]
by enabling the define LPUART2_ON_EMC. 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 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 I$^2$C 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. GPT

The i.MX RT 1021 has two General Purpose Timers (GPT), which are 32 bit up-counters with 12-bit pre-scaler (1..4096) clocked from PERCLK but with programmable counter clock source. Drivers are active when the project define SUPPORT_TIMER is enabled in `app_hw_iMX.h`.

The GPTs supports capture mode on programmable edges of their associalted timer input pins (`GPT_CAPTUREx`) and can generate an event on their output compare pins (`GPT_COMPAREx`), as well as an interrupt when the counter reaches a programmed value. Capture mode support is active when SUPPORT_CAPTURE is enabled.

Programmable clock sources to the prescaler are:

- Crystal oscillator (with its own 1..16 prescaler)
- External `GPT_CLK` input
- Peripheral Clock
- Low frequency reference clock (32kHz)
- High frequency reference clock


Each GPT has two input capture channels and three output compare channels and operates either in Restart (based on compare channel 1) or Free-run mode of operation.


The GPT driver interface is highy compatible with the FlexTimer/TPM driver in the Kinetis project. Each GPT can be used to generate periodic or single shot interrupts, capture inputs or generate a square wave, low or high edge on match. See the Hardware Timer Guide at https://www.utasker.com/docs/uTasker/uTaskerHWTimers.PDF for more details.


Unfortunately, although the GPT can generate a square wave output, it can't generate PWM outputs. Also it can't trigger DMA operations.


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

| GPT | Default | Alternative 1 | Alternative 2 |
|---|---|---|---|
| **iMX_GPT_1 (0)**<br>GPT1_CLK | GPIO_B1_04 | **GTP1_CLK_ON_AD_B0**<br>GPIO_AD_B0_13 | |
| **iMX_GPT_2 (1)**<br>GPT2_CLK | GPIO_AD_B1_02 | **GTP2_CLK_ON_AD_B0**<br>GPIO_AD_B0_09 | |

## 9. 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.

## 10.    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=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 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.

 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

The i.MX RT 1064 has the option to use fast GPIOs when GPIO6, 7, 8 or 9 are used. These ports are connected to the AHB and so operate at the core clock speed, while the ports 1, 2, 3, 4 and 5 are clocked by the slower IPG clock.
Although the GPIO registers are always all available the fast GPIOs share pins with the slow GPIO and only one of the two can be used at a time.

For example GPIO1-00 could be used as an output on its physical pin at the normal GPIO speed. To use this pin as a fast GPIO it is instead controlled as GPIO6-00 and this is multiplex to the physical pin instead – GPIO1-00 is then no longer connected.
GPIO1 pins can be exchanged – on a pin for pin basis – for faster ones controlled by GPIO6.
GPIO2 pins can be exchanged – on a pin for pin basis – for faster ones controlled by GPIO7.
GPIO3 pins can be exchanged – on a pin for pin basis – for faster ones controlled by GPIO8.
GPIO4 pins can be exchanged – on a pin for pin basis – for faster ones controlled by GPIO9.
GPIO5 doesn't have an alternative fast GPIO multiplexing capability.

By default the normal speed GPIOs are connected to the GPIO physical pins, which allows code compatibility across the various devices. If GPIO6..9 ports are configured as input or output they will automatically be set to their fast pin connection, which means that using these GPIOs automatically selects them over the normal ones.
Example:

```
_CONFIG_DRIVE_PORT_OUTPUT_VALUE(1, (BLINK_LED | DEMO_LED_2), (BLINK_LED),
(PORT_SRE_SLOW | PORT_DSE_HIGH));
_TOGGLE_PORT(1, BLINK_LED);
```

shows how an output on GPIO1 is configure and used.

```
_CONFIG_DRIVE_PORT_OUTPUT_VALUE(6, (BLINK_LED | DEMO_LED_2), (BLINK_LED),
(PORT_SRE_SLOW | PORT_DSE_HIGH));
_TOGGLE_PORT(6, BLINK_LED);
```

Here the same pin is controlled as a fast output instead, whereby if the output was initially configured as a normal output it will subsequently be multiplexed to its fast connection and the original normal output no longer be connected to the pin.

If the normal mode were to be required again the reconfiguration

```
_CONFIG_DRIVE_PORT_OUTPUT_VALUE(1, (BLINK_LED | DEMO_LED_2), (BLINK_LED),
(PORT_SRE_SLOW | PORT_DSE_HIGH));
```

will switch it back to that connection.
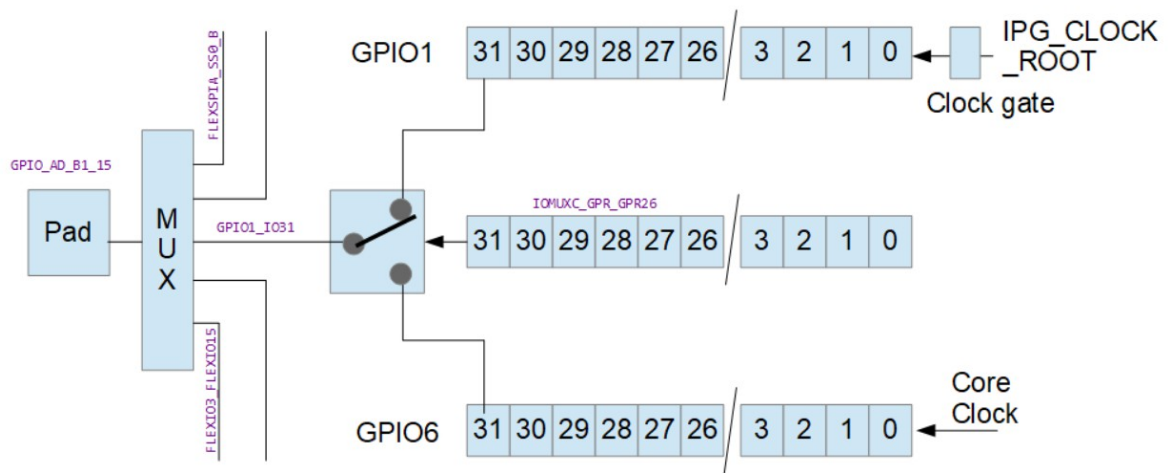
As shown in this illustration of GPIO1 and GPIO6 a pin can be connected on a GPIO bit basis to either GPIO1 or GPIO6, controlled by the corresponding bit in the register `IOMUX_CPR_CPR26`. Either the standard port (GPIO1) or the high-speed port (GPIO6) bit can be connected to the pin when no other of the pins multiplexed peripherals is selected instead; *both GPIO1-x and GPIO7_x cannot be used at the same time*.

The same is true for GPIO2/7, GPIO3/8 and GPIO4/9 port pairs.

The video https://youtu.be/nLInUIboLR0 shows the toggling rates that can be achieved on the i.MX RT 1064 standard and high-speed ports, as well as giving additional information concerting their control.

# 11.    RAM and Cache

he i.MX RT 117x contains 2M of internal RAM, whereby 1.5M 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 117x 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 0kB OCRAM, 256k DTCM and 256k ITCM; *the ROM loader may use the first 64k of the DTCM 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:
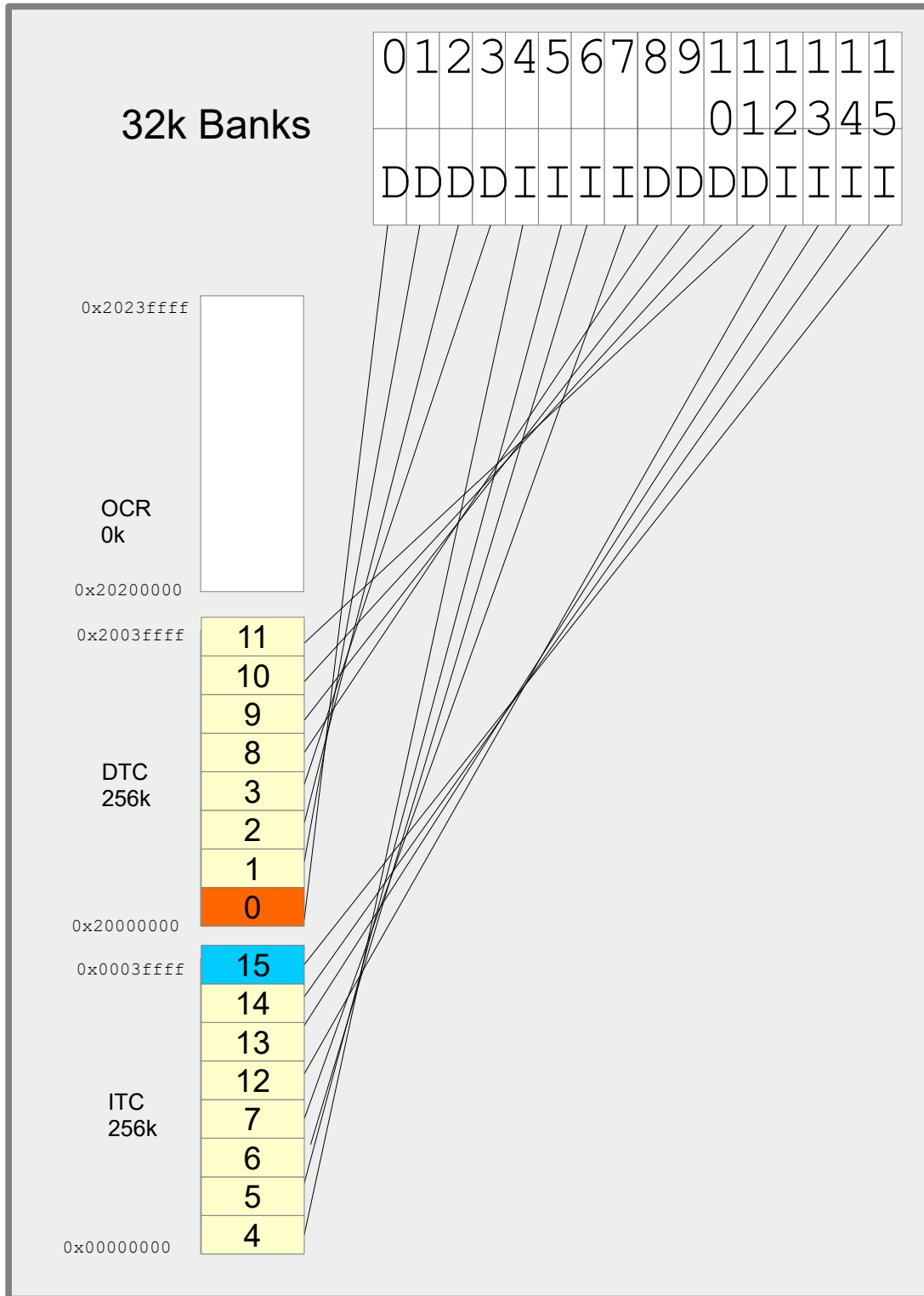
Assuming it is decided that an application were best configured to have 11 banks for ITCM (so that the code could be completely located there – 352k) and 5 banks for data (so that up to 160k of data could be accesses at optimal speed) and no OCRAM the 16 banks could be configured as follows:

## 32k Banks

```
0x2023ffff

OCR
0k

0x20200000
```

```
0x20027fff    [15]
              [14]
              [13]
DTC           [12]
160k          [11]
0x20000000
```

```
0x00057fff    [10]
              [9]
              [8]
              [7]
              [6]
ITC           [5]
352k          [4]
              [3]
              [2]
              [1]
0x00000000    [0]
```

I I I I I I I I I D D D D D
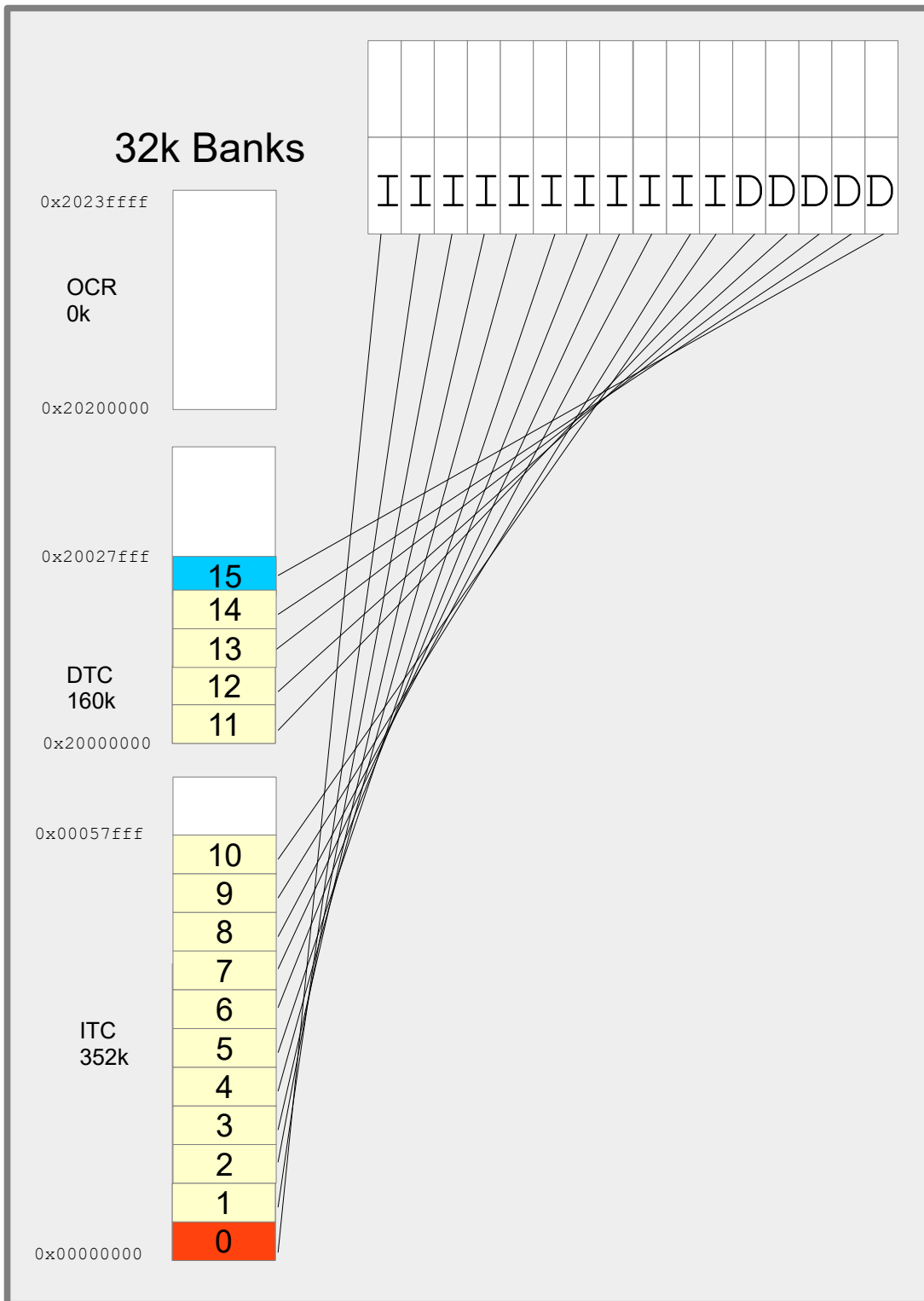
The µTasker FlexRAM driver always uses the bank order from instruction use to data use (if OCRAM were used it would be inserted between the two). The most important thing to understand is that when the bank use is modified the address range of the bank is also changed (it is moved in the memory map). This means that data that was in bank 15 (default in ITC starting at the address `0x00038000`) before the change still exists in the bank memory but is now addressed in the DTC memory space instead (in the case of the example starting at `0x20020000`).

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 15 is assigned to ITC 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.

*The µTasker primary loader sets its stack pointer to the top of ITC (default bank 15). When it jumps to either a µTasker serial loader or application running in internal memory it dynamically reconfigures the FlexRAM to locate bank 15 to the top of DTC memory instead. During the reconfiguration the stack pointer is moved from ITC to DTC and the processor stack thus preserved.*

Due to the nature of the memory operation of the i.MX RT 107X, 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) and integral part of every project (apart from when a stand-alone application is loaded in a debug environment for test purposes).

# 12.    Boot Mode

The address of the QSPI flash which is typically used as boot device is 0x30000000

# 13.     Ethernet

## 14.　　Low Power and Battery Backup

The i.MX RT 1064 supports three RU modes
- Overdrive Run
- Full Speed Run
- Low Power Run

and four low power modes:
- System IDLE
- Low Power IDLE
- SUSPEND
- SNVS

The i.MX RT 1064 has a coin cell input that can be used to supply power to parts of the device that require minimum operation and data retention when the main power is removed.

Specifically, the energy supplied by the input (VDD_SNVS_IN 2.4V..3.6V) is used to retain the state of the processors pads, supply power to its internal real time clock and retal data in its SNVS (secure non-volatile storage module) module. An internal low drop out regulator reduces the input voltage to a stable 1.1V for the internal operation and requires a 220nF capacitor on the VDD_SNVS_CAP output for stability. In addition a serier input resistor is recommended to reduce suge current when connecting the battery (ca. 1K is typically).

The register CCM_ANALOG_MISC0 contains a switch called DISCON_HIGH_SNVS which by default connects the VDD_HIGH_IN power input to the VDD_SNVS_IN input when the main supply is present, thus supplying this circuit's energy from the main power supply (and optionally charging the coin cell from the main power supply). This can to be set to '1' to move this connection

There are 128 bits of battery-backed memory in the SNVS which can be read/written as 4 long words as registers SNVS_LPGPR0, SNVS_LPGPR1, SNVS_LPGPR2 and SNVS_LPGPR3.

These registers are however reset to 0x00000000 when a tamper event is detected by the chip (in order to clear user keys stored here) and in systems not using the tamper detection circuity it is necessary to disable the clearing mechanism by setting bit 24 in SNVS_LPCR. This bit is referred to by user documentatio nbut is otherwise not defined by the literature. Since this control flag is also retained across software resets it is suitable for enabling general purpose storage to these registers. Read/write access do however need to be long word accesses for correction operation!.

# 15. Conclusion

Modifications:
V0.00 16.04.2021: Initial version in development

# Appendix A – Hardware Dependencies

### a) Space for first Appendix