Embedding it better...

µTasker Document

i.MX RT 1062 – the µTasker way

Table of Contents

1. Introduction

See also the i.MX RT 1021 document – only deviations relevent to the i.MX RT 106x may be incuded in this version.

The **MIMXRT1060-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		

Although the system PLL2 is intended to be used as a fixed 528MHz oscillator it can in fact be adjusted within the range 528MHz .. 252MHz be modifying its PLL loop settings. If the
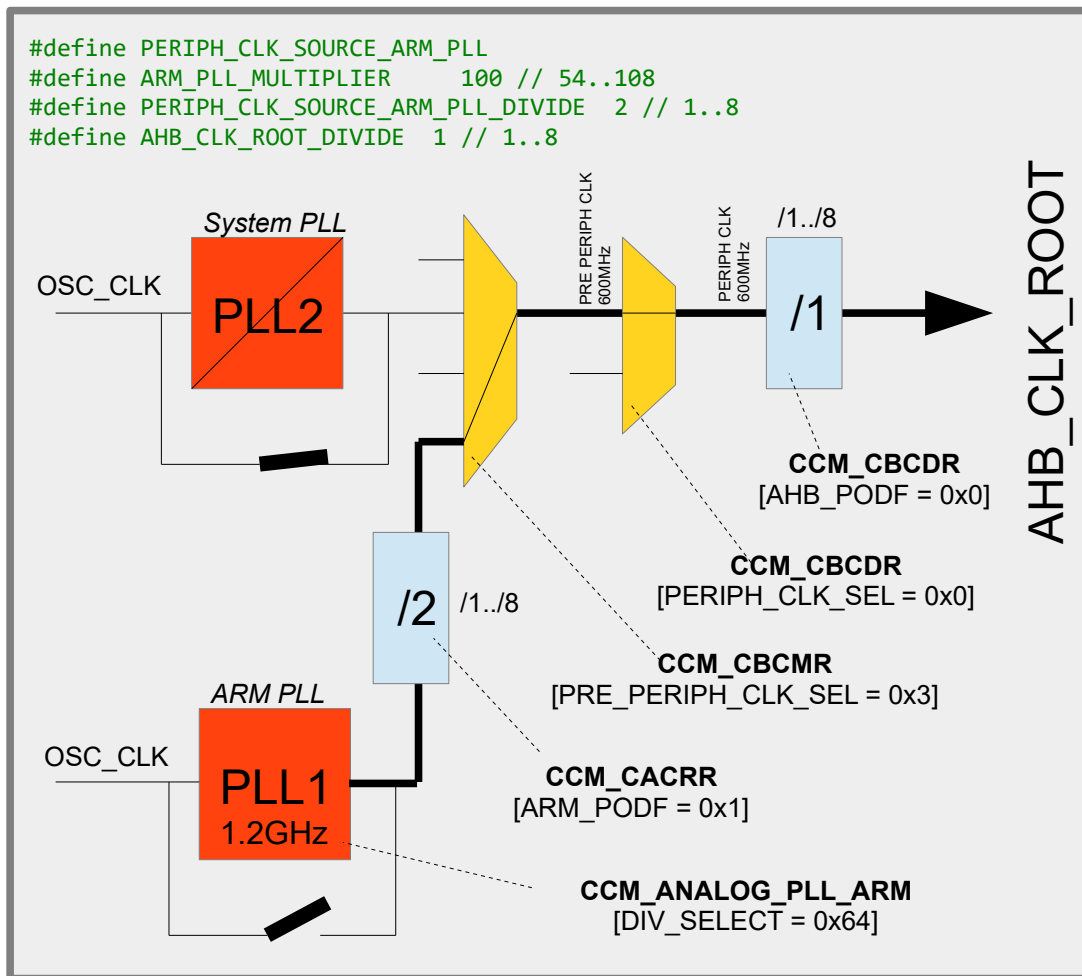
frequency is adjusted the values in the previous table are of course no longer exact and need to also be adjusted accordingly!

Very fine adjustments can be made and the optional define

```
#define PLL2_FREQUENCY                    528123456
```

causes the clock configuration to be performed to give the required frequency.

```
#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

IPG_CLK_ROOT is the internal clock that feeds the ADC and XBAR. It is derived exclusively from the AHB_CLK_ROOT, which was configured in the initial step. IPG_CLK_ROOT is equal to AHB_CLK_ROOT divided by 1 to 4.
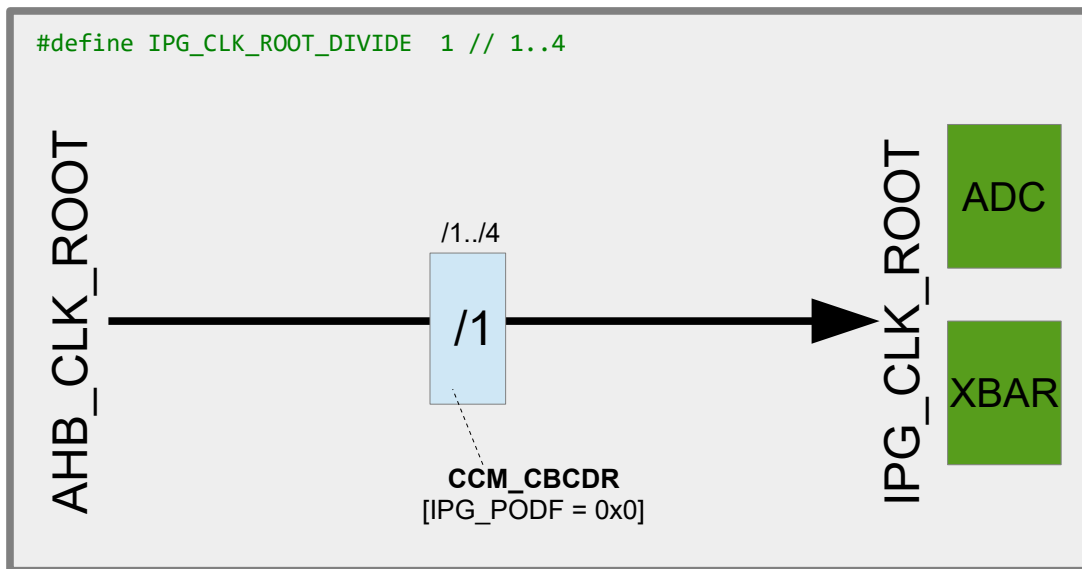

```
#define IPG_CLK_ROOT_DIVIDE 1                // 1..4
```

is used to configure this ratio, whereby if the define is not used it defaults to 1 (that is, IPG_CLK_ROOT is equal to AHB_CLK_ROOT).

Should nether the ADC nor the XBAR be used in the project this clock will automatically be disabled by the control code by disabling its clock gate.

The maximum IPG_CLK_ROOT frequency for the i.MX RT 1021 is 150MHz (max. for AHB_CLK_ROOT is 500MHz) and so it is important to ensure that the divider is set to ensure this speed is not exceeded.

The µTasker project driver code will signal a build error if it detects that and such a frequency has been exceeded. The µTasker simulator also checked run-time derived frequencies and will exception if it detects such violations.

1.3. PERCLK – used by PIT and GPT

PERCLK_CLK_ROOT is the internal clock that feeds the PIT and GPT. It is derived either from the IPG_CLK_ROOT frequency, which was configured in the previous step, or from the 24.0MHz OSC_CLK. In each case it has an optional pre-scaler of divide by 1..64.

```
#define PERCLK_CLK_ROOT_SOURCE_IPG_CLK
```



```
#define PERCLK_CLK_ROOT_DIVIDE 1 // 1..64
```

is used to configure this ratio, whereby if the define is not used it defaults to 1.

```
#define PERCLK_CLK_ROOT_SOURCE_OSC_CLK
```

Should neither the PIT nor the GPT be used in the project this clock will automatically be disabled by the control code by disabling its clock gate.
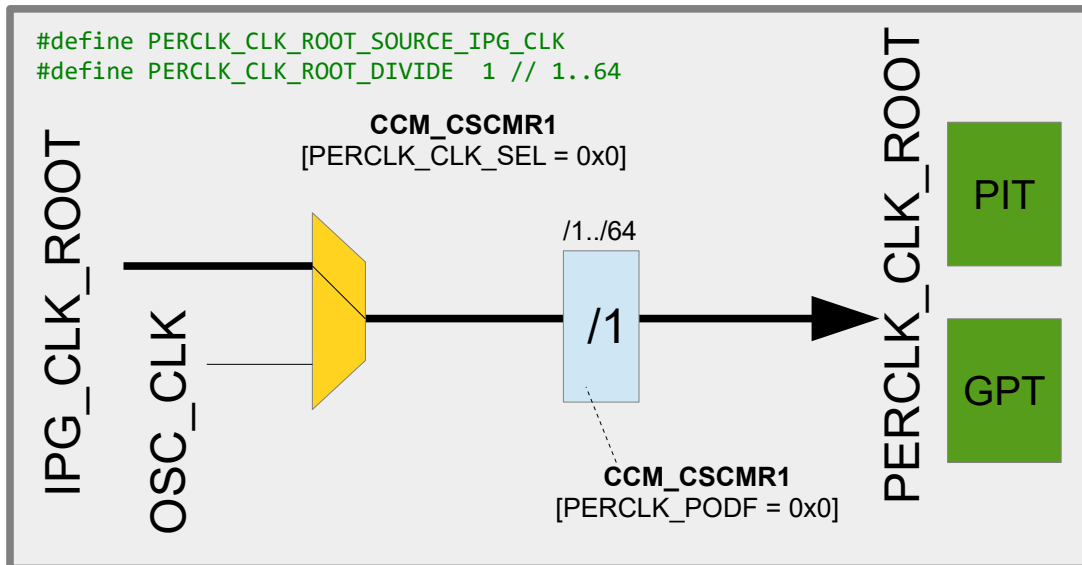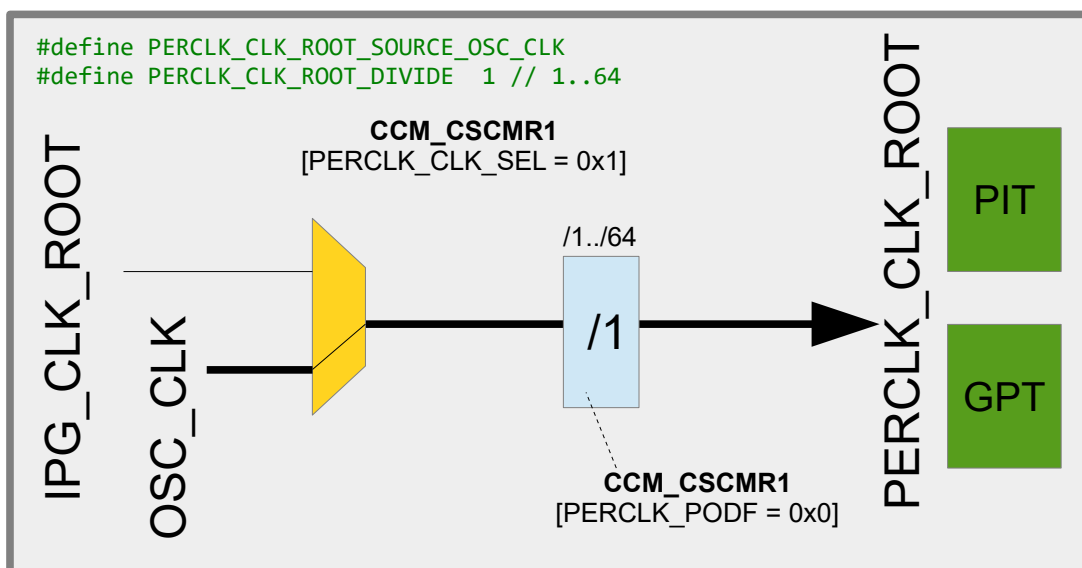
The maximum `PERCLK_CLK_ROOT` frequency for the i.MX RT 1021 is 75MHz and so it is important to ensure that the divider is set to ensure this speed is not exceeded.

The µTasker project driver code will signal a build error if it detects that and such a frequency has been exceeded. The µTasker simulator also checked run-time derived frequencies and will exception if it detects such violations.

1.4. UART_CLK_ROOT – used by all LPUARTs

The LPUARTs in the i.MX RT 1021have a common clock that can be derived from either `OSC_CLK` or from `pll_sw_clk/6`. A common pre-scaler allows the frequency to be divided by 1..64.

`#define UART_CLK_ROOT_FROM_PLL3_SW_CLK_6`
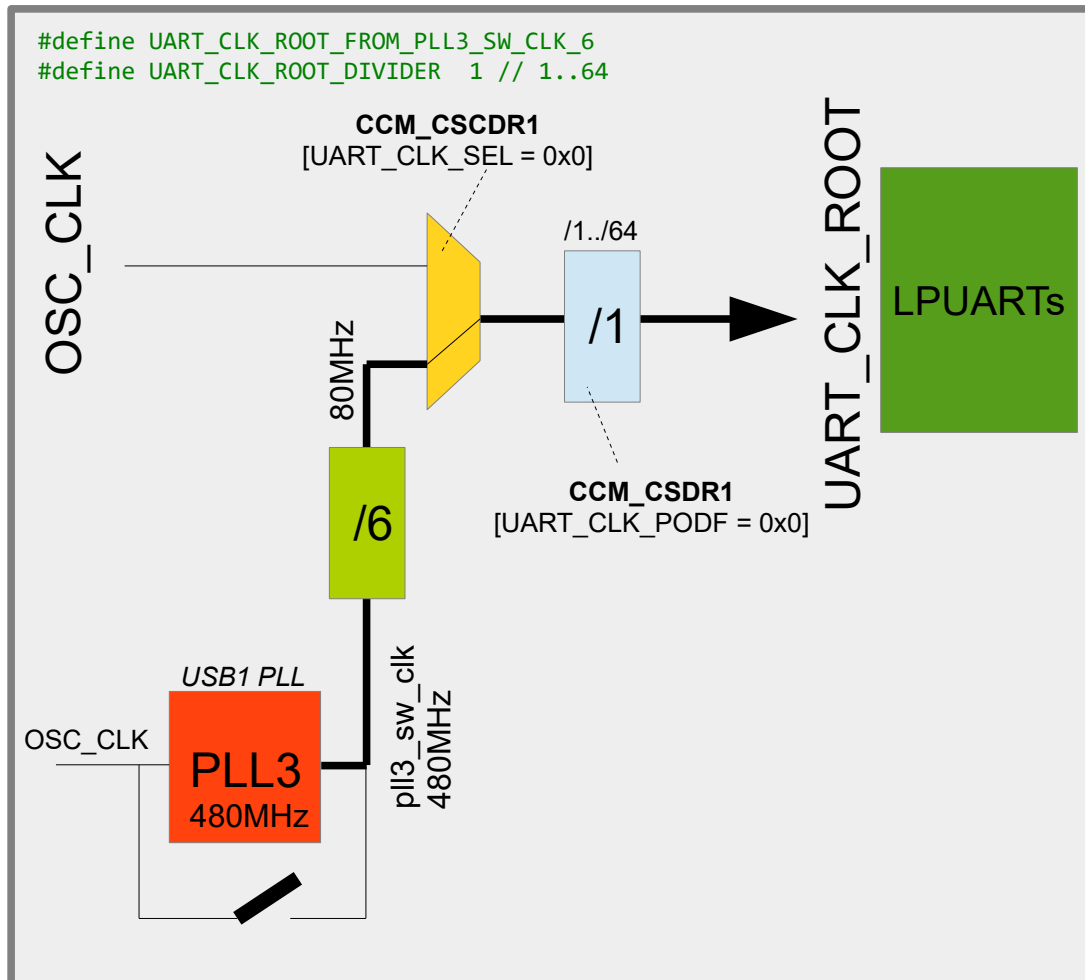
In this configuration the USB1 PLL is used as reference (called `pll_sw_clk`) and divided by a fixed value of 6, resulting in 80MHz. An optional pre-scaler defined by `UART_CLK_ROOT_DIVIDER` can divide this by 1 to 64 (when not defined, the default is 1)

```
#define UART_CLK_ROOT_FROM_OSC_CLK
```
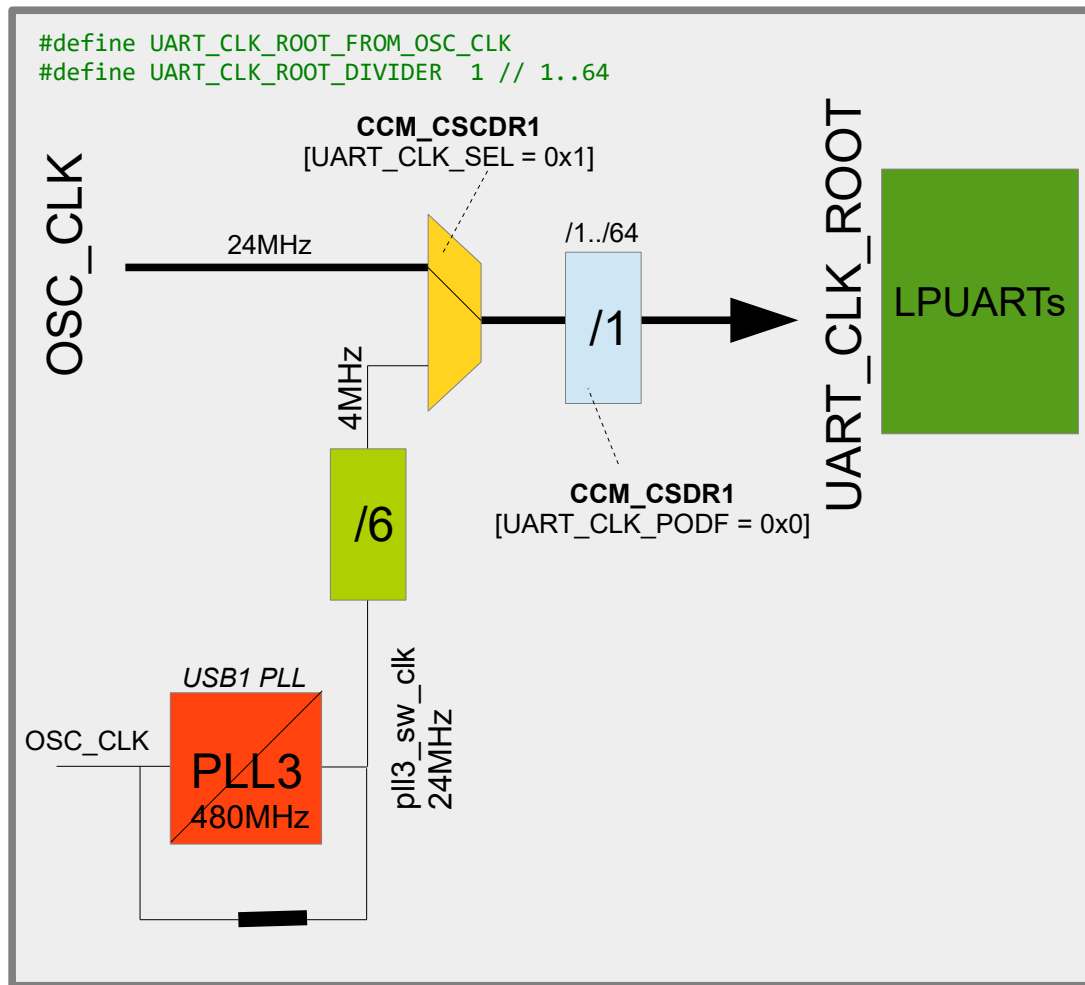

In this configuration the `OSC_CLK` (24MHz) is used as reference, with the optional pre-scaler of 1 to 64.

Note that when PLL3 is not enabled it is left in its powered down, bypassed mode and the alternative clock would be 4MHz instead of 80MHz. This does in fact correspond to the default for the LPUART clock out or reset but this is not used as configuration option since it has no advantage.

`UART_CLK_ROOT` supplies all LPUARTs but the clock is automatically disabled at each individual LPUART input when the corresponding LPUART is not used.

1.5. USDHC1_CLK_ROOT/USDHC2_CLK_ROOT

To add..

1.6. SEMC_CLK_ROOT

By default the SEMC_CLK_ROOT is sourced from PERIPH_CLK, which is the clock that supplies the core clock's pre-scaler. Out of reset it is divided by 3. This configuration is shown in the first image:
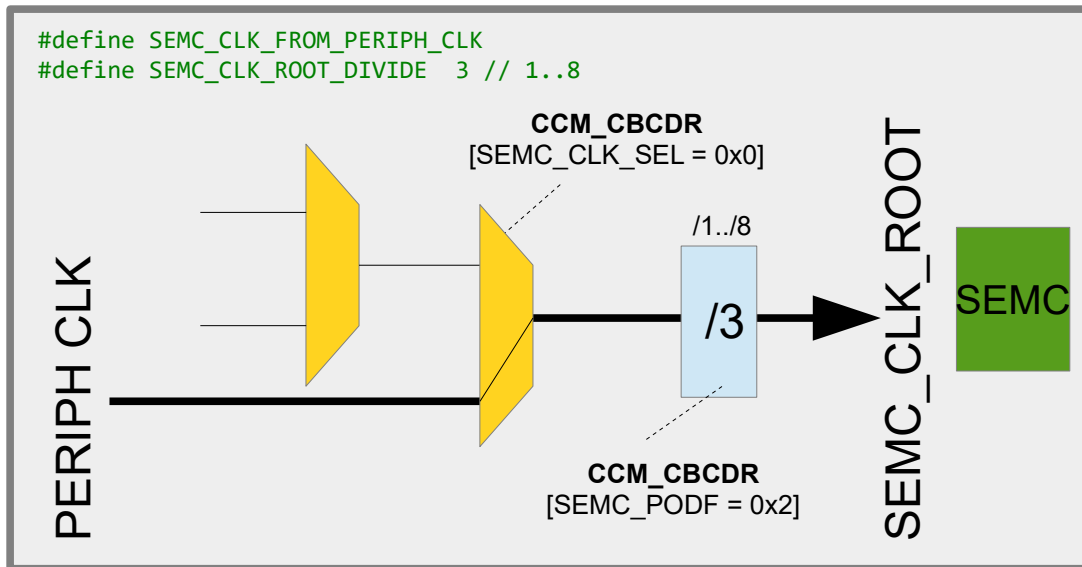
Other configuration possibilities are for PLL2-PFD2 or PLL3-PFD1 as shown in the further images.

```
#define SEMC_CLK_FROM_PLL3_PFD1
#define SEMC_CLK_ROOT_DIVIDE  3 // 1..8
```
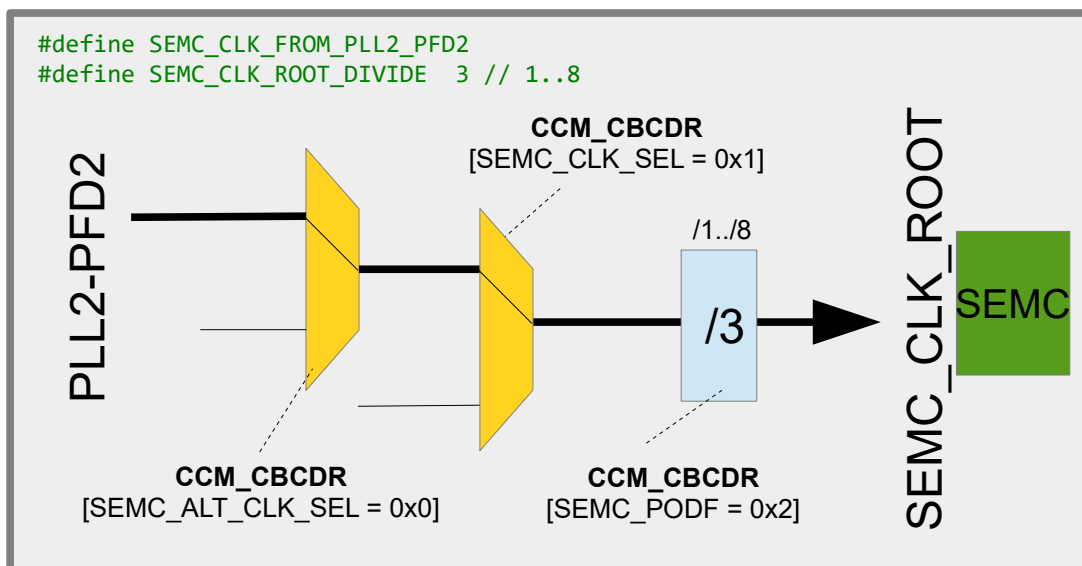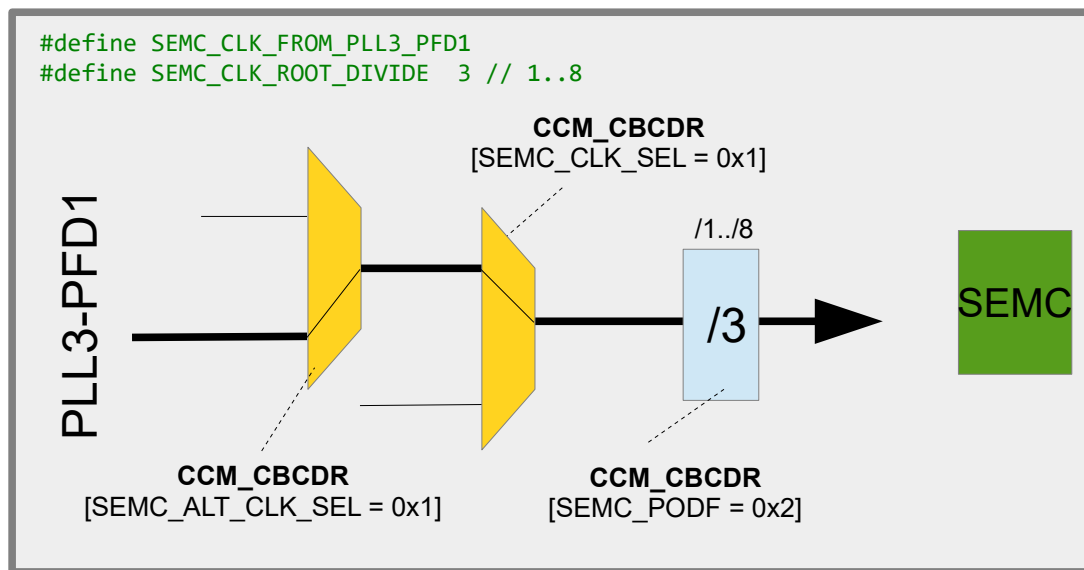
The SEMC_CLK_ROOT frequency must not exceed 166MHz.

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

CCM_CSCMR1
[FLEXSPI_CLK_SEL = 0x0]

SEMC_CLK_ROOT

/1../8

/2

Flex SPI

CCM_CSCMR1
[FLEXSPI_PODF = 0x1]

```
#define FLEXSPI_CLK_FROM_PLL3_SW_CLK
#define FLEXSPI_CLK_ROOT_DIVIDE  2 // 1..8
```

CCM_CSCMR1
[FLEXSPI_CLK_SEL = 0x1]

pll3_sw_clk

480MHz

/1../8

/2

Flex SPI

CCM_CSCMR1
[FLEXSPI_PODF = 0x1]

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

CCM_CSCMR1
[FLEXSPI_CLK_SEL = 0x2]

/1../8

PLL2-PFD2

/2

CCM_CSCMR1
[FLEXSPI_PODF = 0x1]

Flex SPI

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

CCM_CSCMR1
[FLEXSPI_CLK_SEL = 0x3]

/1../8

PLL3-PFD0

/2

CCM_CSCMR1
[FLEXSPI_PODF = 0x1]

FLEXSPI_CLK_ROOT

Flex SPI

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

The LPI2C controllers in the i.MX RT 1021have a common clock that can be derived from either `OSC_CLK` or from `pll_sw_clk/8`. A common pre-scaler allows the frequency to be divided by 1..64.

`#define LPI2C_CLK_ROOT_FROM_PLL3_SW_CLK_8`
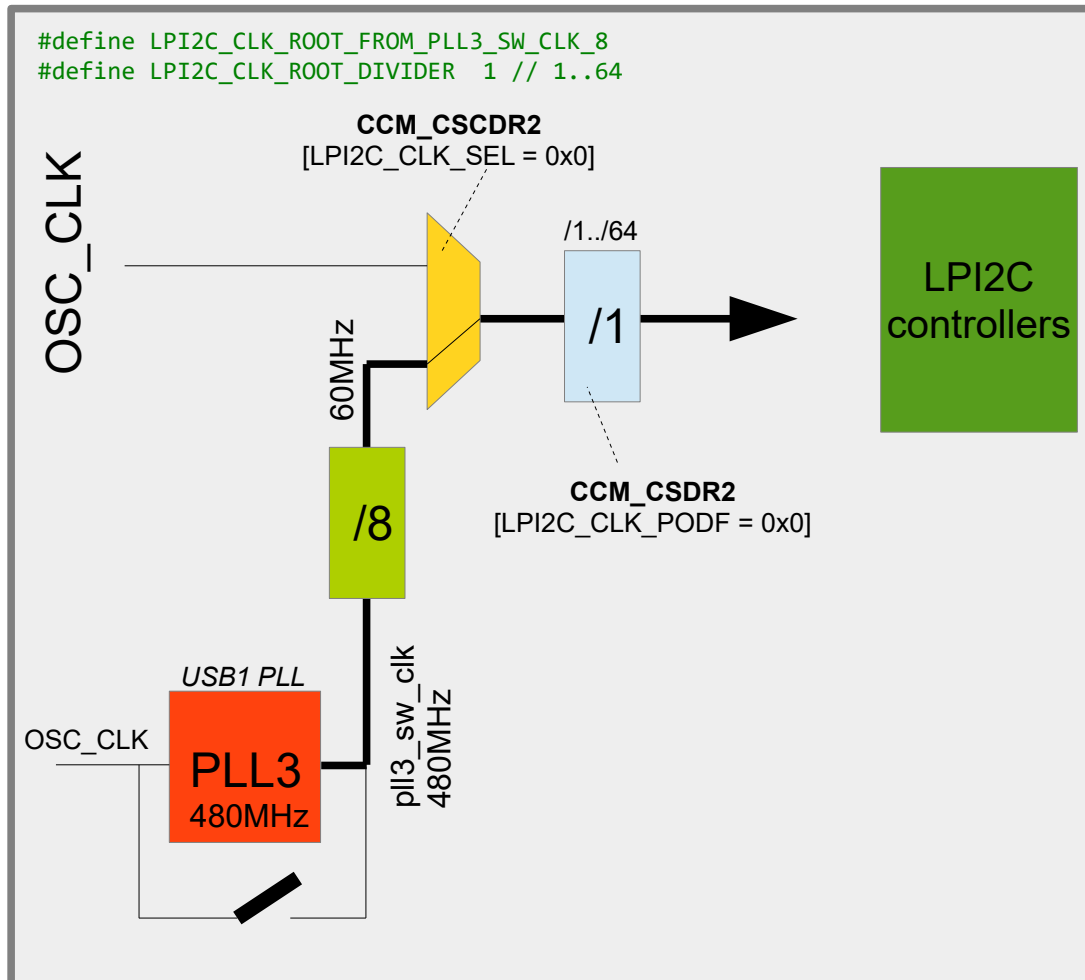
In this configuration the USB1 PLL is used as reference (called `pll_sw_clk`) and divided by a fixed value of 8, resulting in 60MHz. An optional pre-scaler defined by `LPI2C_CLK_ROOT_DIVIDER` can divide this by 1 to 64 (when not defined, the default is 1)

```
#define LPI2C_CLK_ROOT_FROM_OSC_CLK
```


In this configuration the `OSC_CLK` (24MHz) is used as reference, with the optional pre-scaler of 1 to 64.

Note that when PLL3 is not enabled it is left in its powered down, bypassed mode and the alternative clock would be 3MHz instead of 60MHz. This does in fact correspond to the default for the LPI2C clock out or reset but this is not used as configuration option since it has no advantage.

`LPI2C_CLK_ROOT` supplies all LPI2C controllers but the clock is automatically disabled at each individual LPI2C input when the corresponding LPI2C controller is not used.

1.12.   CAN_CLK_ROOT

The FLEXCAN controllers in the i.MX RT 1021have a common clock that can be derived from either `OSC_CLK,` `pll_sw_clk/6` or from `pll_sw_clk/8.` A common pre-scaler allows the frequency to be divided by 1..64.

```
#define CAN_CLK_ROOT_FROM_PLL3_SW_CLK_6
```
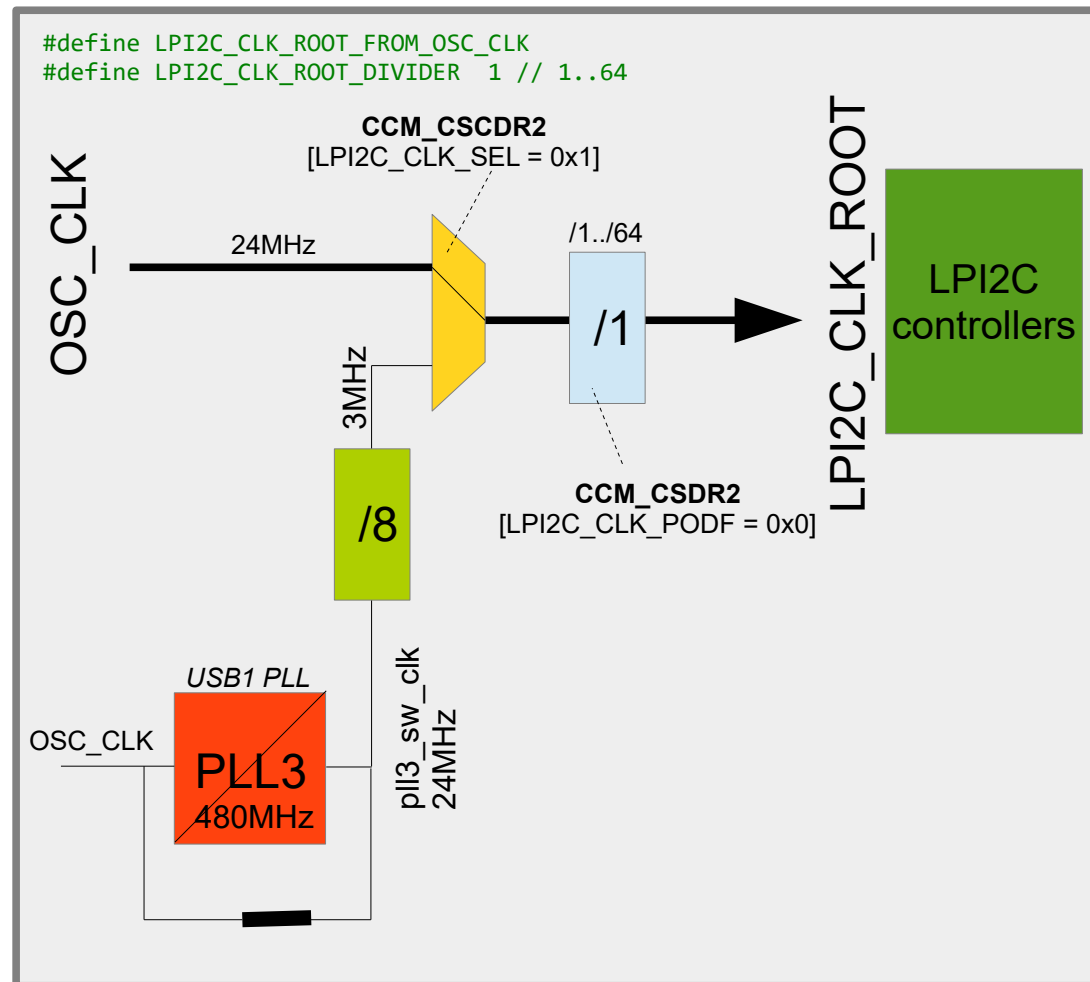

In this configuration the USB1 PLL is used as reference (called `pll_sw_clk`) and divided by a fixed value of 6, resulting in 80MHz. An optional pre-scaler defined by `CAN_CLK_ROOT_DIVIDER` can divide this by 1 to 64 (when not defined, the default is 2)

```
#define CAN_CLK_ROOT_FROM_PLL3_SW_CLK_8
```
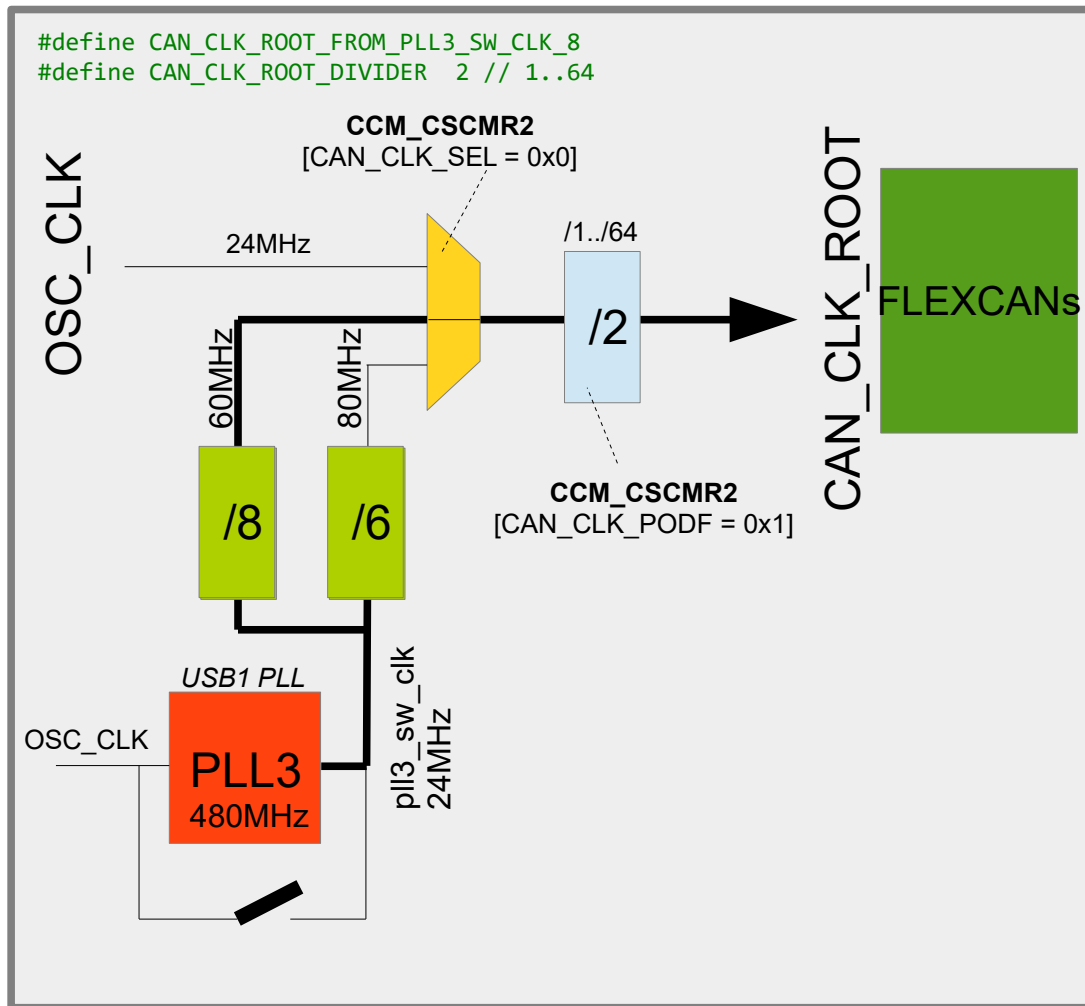

In this configuration the USB1 PLL is used as reference (called `pll_sw_clk`) and divided by a fixed value of 8, resulting in 60MHz. An optional pre-scaler defined by CAN`_CLK_ROOT_DIVIDER` can divide this by 1 to 64 (when not defined, the default is 2)

```
#define CAN_CLK_ROOT_FROM_OSC_CLK
```
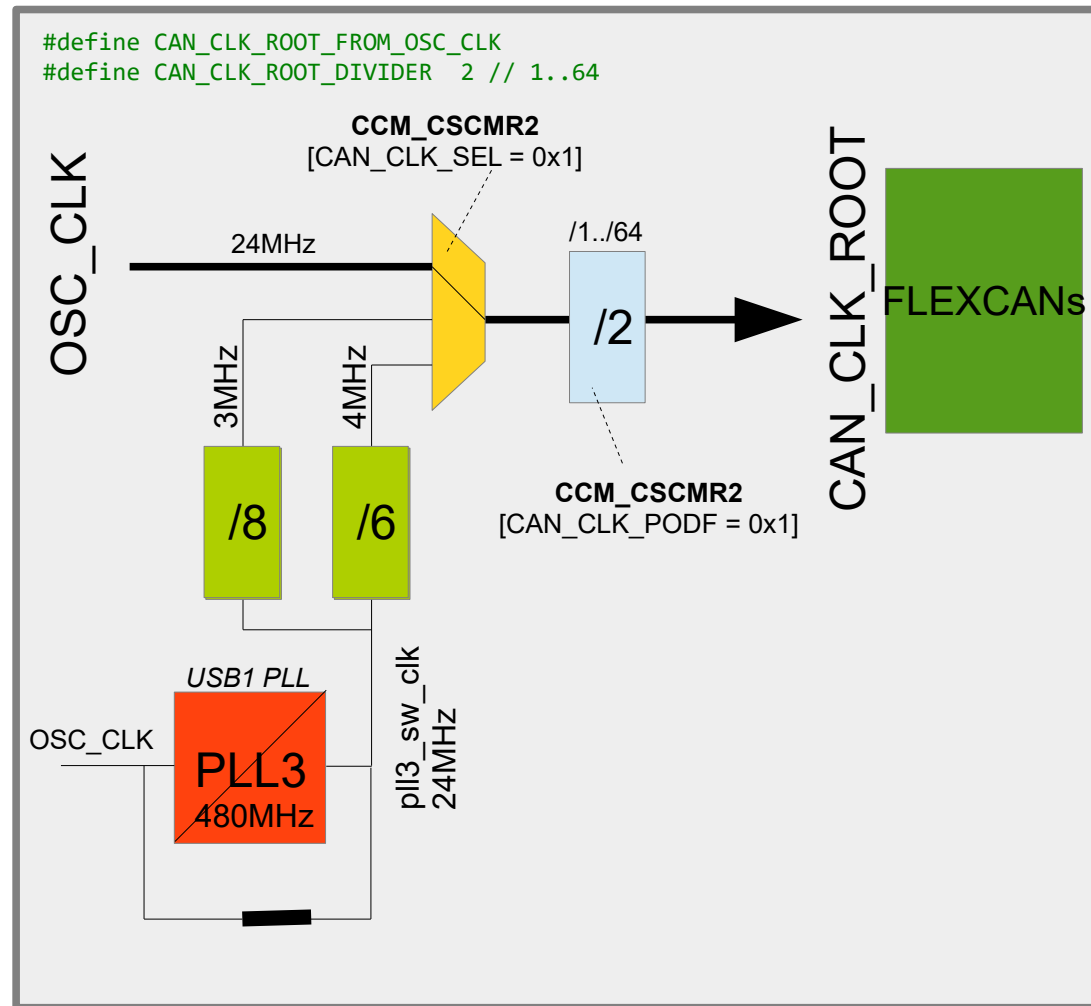

In this configuration the `OSC_CLK` (24MHz) is used as reference, with the optional pre-scaler of 1 to 64.
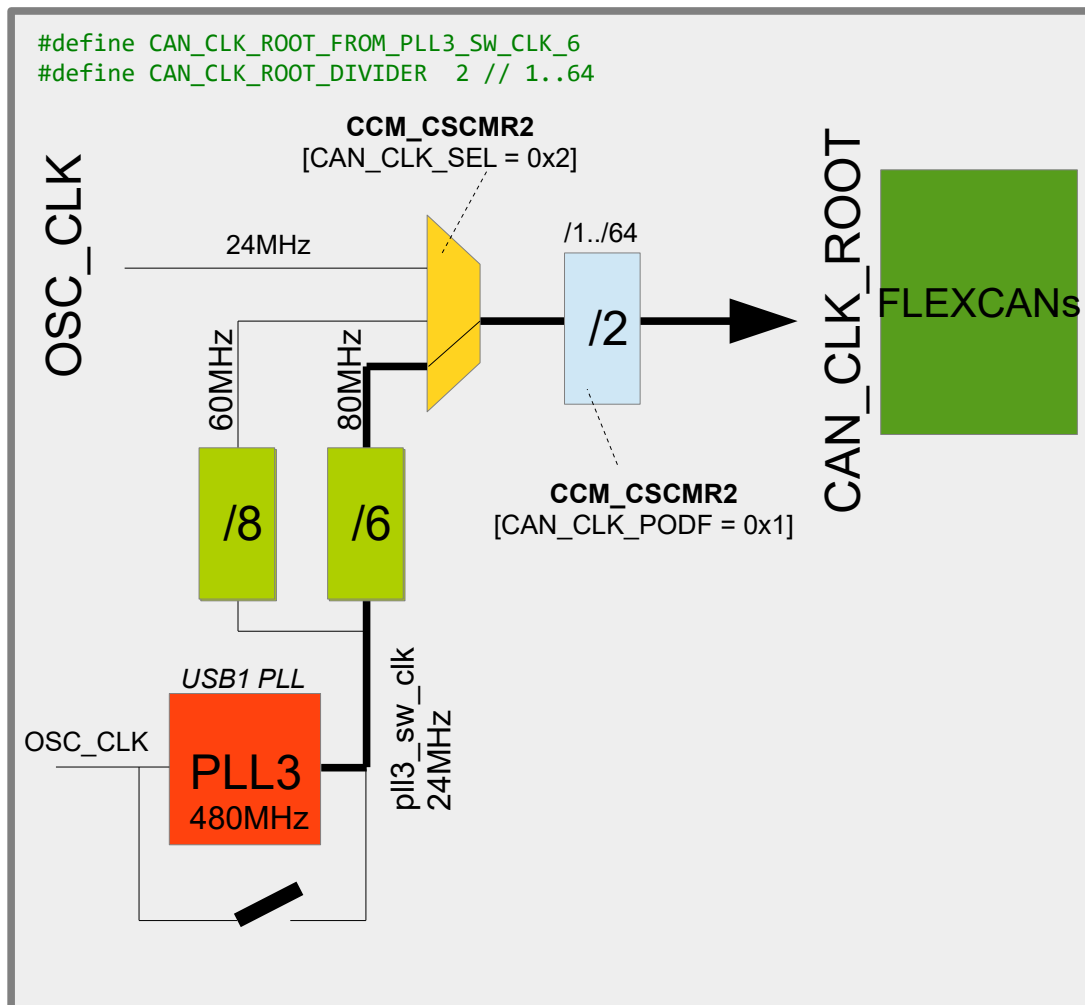
Note that when PLL3 is not enabled it is left in its powered down, bypassed mode and the alternative clocks would be 4/3MHz instead of 80/60MHz. This is not used as configuration option since it has no advantage.

`CAN_CLK_ROOT` supplies all FLEXCAN controllers but the clock is automatically disabled at each individual FLEXCAN input when the corresponding FLEXCAN controller is not used.

1.13. SPDIF0_CLK_ROOT

To add..

1.14. FLEXIO1_CLK_ROOT

To add..

3. Internal Clock Monitoring

The i.MX RT 1021 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 1062 LPUART driver is shared with the Kinetis LPUART driver and supports interrupt and DMA driven modes. Minor differences due to the i.MX RT 1062 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 1062 and improves maintenance since only one source needs to be managed and the i.MX RT 1062 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 1062 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 LPUART8_TXD	GPIO_EMC_39 GPIO_EMC_38	**LPUART8_ON_AD** GPIO_AD_B1_11 GPIO_AD_B1_10	**LPUART8_ON_SD** GPIO_SD_B0_05 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`

When HW flow control is used the following pin-mux are valid for the RTS and CTS lines of the LPUARTs:

Channel	Default	Alternative 1	Alternative 2
iMX_LPUART_1 (0) LPUART1_RTS LPUART1_CTS	GPIO_AD_B0_15 GPIO_AD_B0_14		
iMX_LPUART_2 (1) LPUART2_RTS LPUART2_CTS	GPIO_AD_B1_01 GPIO_AD_B1_00	**LPUART2_ON_SD** Same as default	
iMX_LPUART_3 (2) LPUART3_RTS LPUART3_CTS	GPIO_AD_B1_05 GPIO_AD_B1_04	**LPUART3_ON_B0** Same as default	**LPUART3_ON_EMC** GPIO_EMC_16 GPIO_EMC_15
iMX_LPUART_4 (3) LPUART4_RTS LPUART4_CTS	GPIO_EMC_18 GPIO_EMC_17	**LPUART4_ON_SD** Same as default	**LPUART4_ON_EMC** Same as default
iMX_LPUART_5 (4) LPUART5_RTS LPUART5_CTS	GPIO_EMC_27 GPIO_EMC_28	**LPUART5_ON_EMC** Same as default	
iMX_LPUART_6 (5) LPUART6_RTS LPUART6_CTS	GPIO_EMC_29 GPIO_EMC_30	**LPUART6_ON_AD** Same as default	
iMX_LPUART_7 (6) LPUART7_RTS LPUART7_CTS	GPIO_SD_B1_07 GPIO_SD_B1_06	**LPUART7_ON_SD** Same as default	
iMX_LPUART_8 (7) LPUART8_RTS LPUART8_CTS	GPIO_SD_B0_03 GPIO_SD_B0_02	**LPUART8_ON_AD** Same as default	**LPUART8_ON_SD** Same as default

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 1062 thus could immediately inherit this operation.

5. LPI2C

The i.MX RT 1062 LPI2C driver is shared with the Kinetis LPUART driver and supports interrupt and DMA driven modes. Minor differences due to the i.MX RT 1062 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 1062 and improves maintenance since only one source needs to be managed and the i.MX RT 1062 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 1062 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.

The LPI2Cs can be multiplexed onto various physical pins. When a specific LPI2C is used it defaults to a certain set of pins if nothing else is specified – the following gives a list of the LPI2C 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_LPI2C_1 (0) LPI2C1_SCL LPI2C1_SDA	GPIO_AD_B1_00 GPIO_AD_B1_01	**LPI2C1_ON_SD** GPIO_SD_B1_04 GPIO_SD_B1_05	
iMX_LPI2C_2 (1) LPI2C2_SCL LPI2C2_SDA	GPIO_B0_04 GPIO_B0_05	**LPI2C2_ON_SD** GPIO_SD_B1_11 GPIO_SD_B1_10	
iMX_LPI2C_3 (2) LPI2C3_SCL LPI2C3_SDA	GPIO_AD_B1_07 GPIO_AD_B1_06	**LPI2C3_ON_EMC** GPIO_EMC_22 GPIO_EMC_21	**LPI2C3_ON_SD** GPIO_SD_B0_00 GPIO_SD_B0_01
iMX_LPI2C_4 (3) LPI2C4_SCL LPI2C4_SDA	GPIO_AD_B0_12 GPIO_AD_B0_13	**LPI2C4_ON_EMC** GPIO_EMC_12 GPIO_EMC_11	

For example, if LPI2C2 is used its default pin-out SCL/SDA is on GPIO_B0_04/GPIO_B0_05 [GPIO2-IO04 and GPIO2-IO05]
but can be set instead to

GPIO_SD_B1_11/GPIO_SD_B1_10 [GPIO3-IO11 and GPIO3-IO10]
by enabling the define `LPI2C2_ON_SD`. This define is set in `app_hw_iMX.h`

6. FLEXCAN

The i.MX RT 1021 CAN driver is shared with the Kinetis CAN driver. Minor differences due to the i.MX RT 1021 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 1021 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 1021 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 1021 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.

These are the FlexCAN pin mux options:

Channel	Default	Alternative 1	Alternative 2	Alternative 3
iMX_FLEXCAN_1 **(0)** FLEXCAN1_TX FLEXCAN1_RX	GPIO_AD_B1_08 GPIO_AD_B1_09	**FLEXCAN1_ON_B0** GPIO_B0_02 GPIO_B0_03	**FLEXCAN1_ON_EMC** GPIO_EMC_17 GPIO_EMC_18	**FLEXCAN1_ON_SD** GPIO_SD_B1_02 GPIO_SD_B1_03
iMX_FLEXCAN_2 **(1)** FLEXCAN2_TX FLEXCAN2_RX	GPIO_AD_B0_02 GPIO_AD_B0_03	**FLEXCAN2_ON_ EMC** GPIO_EMC_09 GPIO_EMC_10	**FLEXCAN2_ON_ B1** GPIO_B1_08 GPIO_B1_09	**FLEXCAN2_ON_ AD_B0_HI** GPIO_AD_B0_14 GPIO_AD_B0_15
iMX_FLEXCAN_3 **(2)** FLEXCAN3_TX FLEXCAN3_RX	GPIO_AD_B0_10 GPIO_AD_B0_11	**FLEXCAN3_ON_ AD_B0_HI** GPIO_AD_B0_14 GPIO_AD_B0_15	**FLEXCAN3_ON_ EMC** GPIO_EMC_36 GPIO_EMC_37	

7. PIT

The i.MX RT 1021 PIT driver is shared with the Kinetis PIT driver. Minor differences due to the i.MX RT 1021 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 1021 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.

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 1021 DMA driver is shared with the Kinetis eDMA driver. Minor differences due to the i.MX RT 1021 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 1021 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.

9. TEMPMON

The i.MX RT 106x includes a temperature monitor module (TEMPMON) that allows interrupts to be generated if the core temperature exceeds two programmable high temperature levels or falls below a low programmable lower limit level.

Software can also read the present die temperature at any time.

In order for it to work correctly it needs the bandgap reference to be enabled, plus the 480MHz PLL and the 32kHz RTC modules to be operating.

The temperature monitor is factory calibrated and the calibration values can be read from the HW_OCOTP_ANA1 registers. These are used by software to extrapolate the temperature and also to correctly set temperature limits.

The support in the µTasker project is enabled with the define SUPPORT_TEMPMON which adds an interface to read the core temperature via the ADC API in a compatible manner for projects that also run on processors that use ADC based temperature reading.

An example of reading the temperature periodically can be activated in `ADC_Timers.h` by activating the define ADC_INTERNAL_TEMPERATURE when using the ADC reference. For compatibility the ADC API is used with the input set to

```
adc_setup.int_adc_bit = ADC_TEMP_SENSOR; // ADC internal temperature
```

which is how ADC based temperature reading is performed.

Although there is no interrupt generated when the measurement has completed such an interrupt is emulated so that applications remain compatible. The only difference is that the result returned when collecting the value is in °C x 100 (allowing hundredth of degree resolution) and not the raw ADC value itself. Therefore the only modification at the application level is to remove any HW specific conversion that may originally have been performed and use the result directly (or after modification to the desired form). The following shows retrieving and rounding to 1°C resolution:

```
ADC_SETUP adc_setup; // interrupt configuration parameters
ADC_RESULTS results;
adc_setup.int_type = ADC_INTERRUPT; // identifier
adc_setup.int_adc_mode = (ADC_READ_ONLY | ADC_GET_RESULT);
adc_setup.int_adc_controller = 0;
adc_setup.int_adc_result = &results;
fnConfigureInterrupt((void *)&adc_setup);
results.sADC_value[0] += 50;
results.sADC_value[0] /= 100; // the approximate temperature
rounded up/down to 1°C
fnDebugDec(results.sADC_value[0], DISPLAY_NEGATIVE);
fnDebugMsg(" degC\r\n");
```

In this particular case the conversion was started previously and it shows just the subsequent retrieval. The conversion was started using the standard API for the ADC with the ADC_TEMP_SENSOR defined as input. When multiple ADC controllers are implemented in the i.MX RT the same one should be referenced for the conversion and retrieval, although there is only one TEMPMON module shared by both and the ADC controller is not actually used.

10.   GPIO

The i.MX RT 1062 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=zNWlG-O7ZW0&feature=youtu.be

The i.MX RT 1062 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
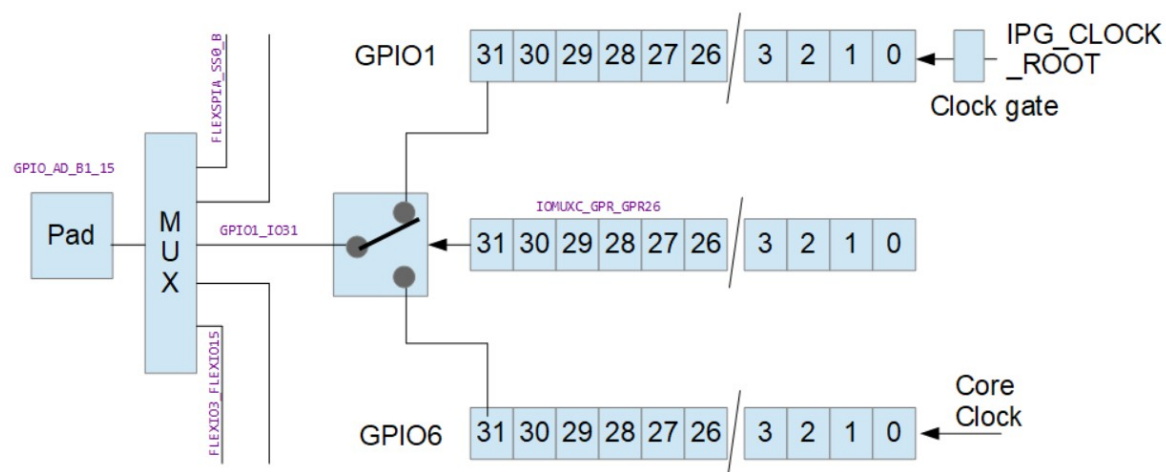
```
_CONFIG_DRIVE_PORT_OUTPUT_VALUE(1, (BLINK_LED | DEMO_LED_2), (BLINK_LED),
(PORT_SRE_SLOW | PORT_DSE_HIGH));
```

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 1062 standard and high-speed ports, as well as giving additional information concerting their control.

11.   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
```

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:
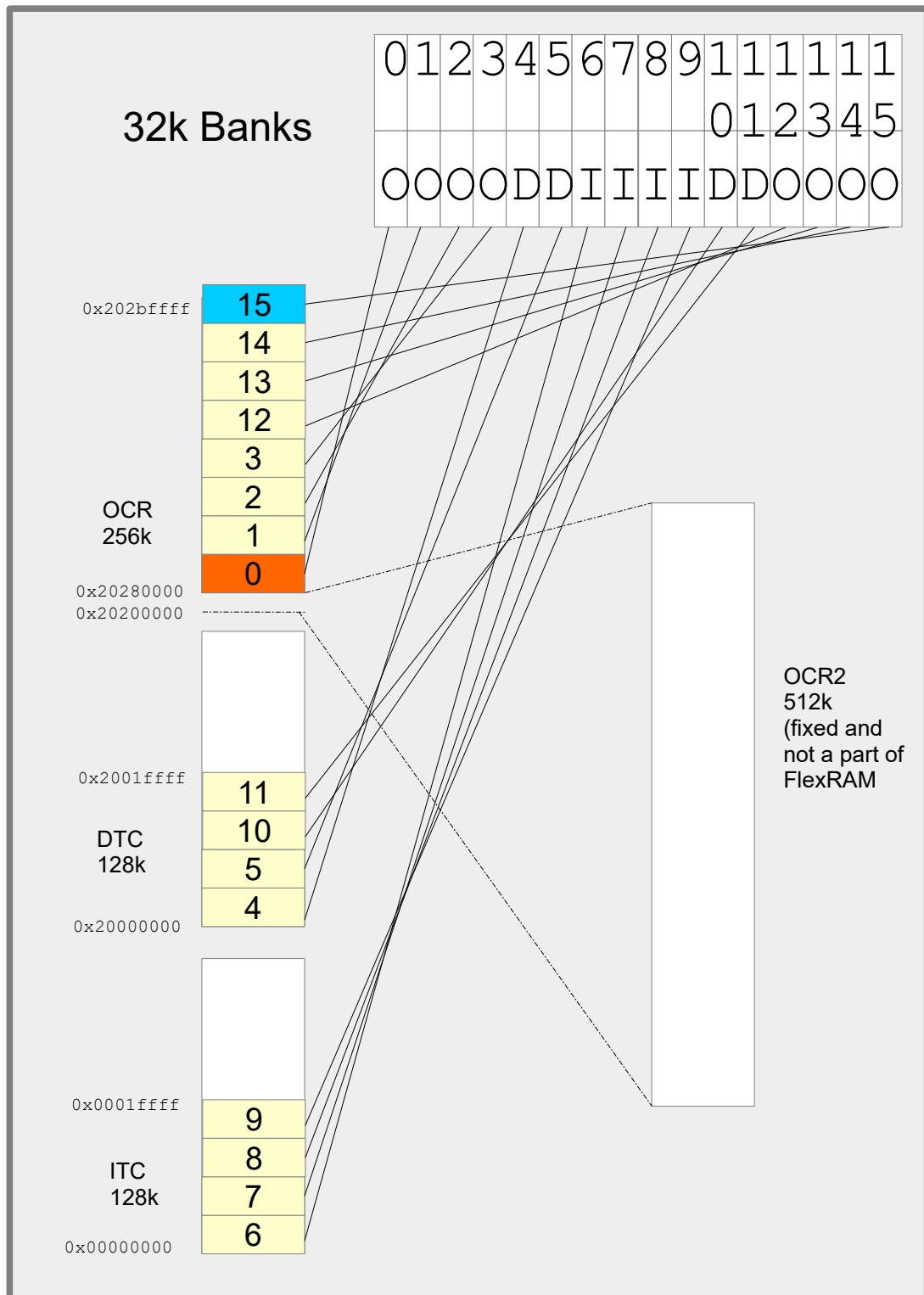
Assuming it is decided that an application were best configured to have 11 banks for ITCM (so that the code could be completely located there – 352k) and 5 banks for data (so that up to 160k of data could be accesses at optimal speed) and no OCRAM the 16 banks could be configured as follows:
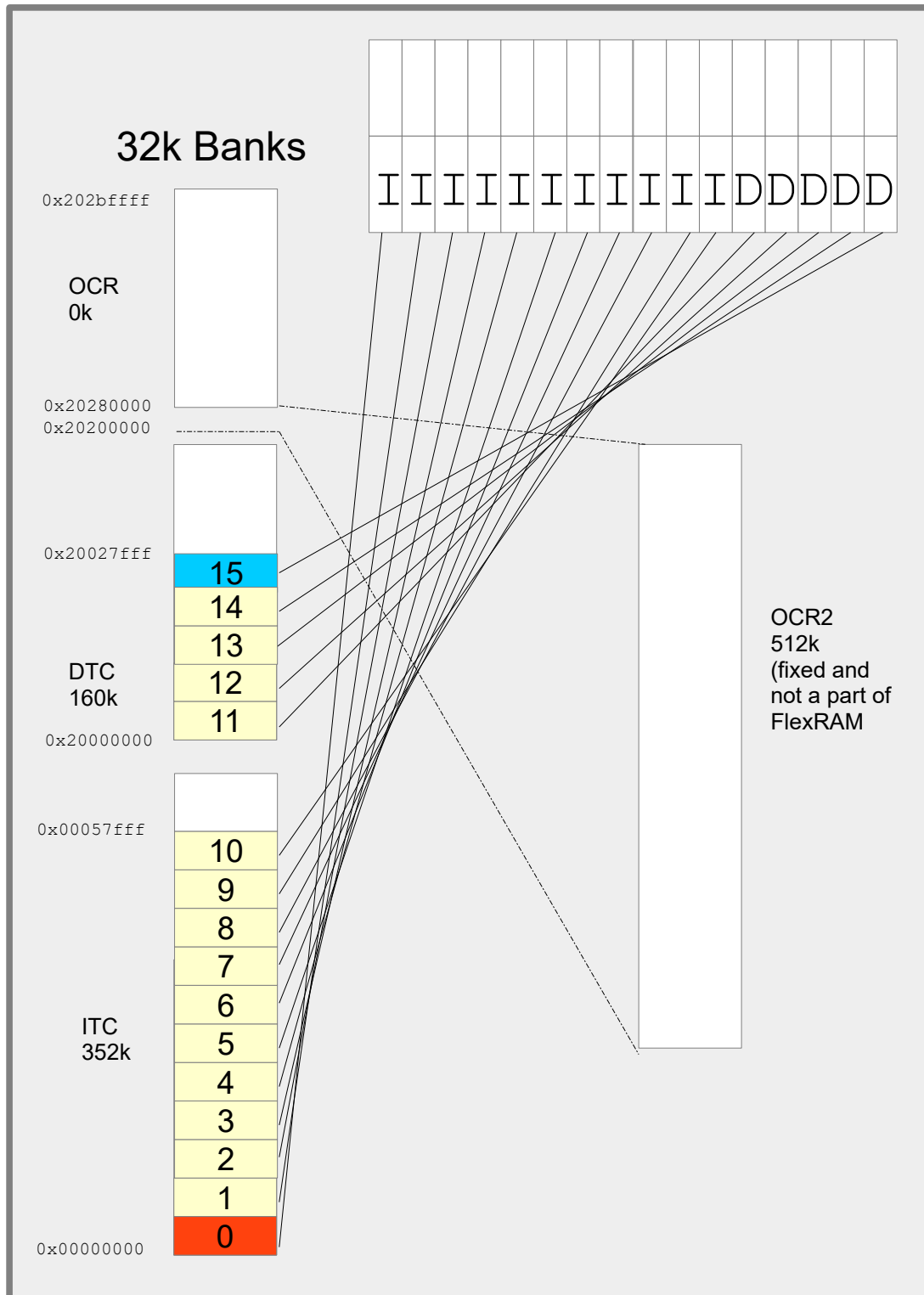
The µTasker FlexRAM driver always uses the bank order from instruction use to data use (if OCRAM were used it would be inserted between the two). The most important thing to understand is that when the bank use is modified the address range of the bank is also changed (it is moved in the memory map). 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 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).

12. Boot Mode

The i.MX RT 1021 doesn't have internal flash and needs to boot from an external program source. This is controlled by an internal ROM which offers various boot loader capabilities which are controlled by two inputs (SRC_BOOT_MODE00/GPIO2-IO16 and SRC_BOOT_MODE01/GPIO2-IO17) as well as some eFUSEs (or further pins). *The µTasker project avoids the use of eFUSES and instead controls the operation in its code so that chips can be used in their default configuration*.

The i.MX RT 1021 supports three main modes:

SRC_BOOT_MODE01/ GPIO2-IO17/GPIO_EMC_17	SRC_BOOT_MODE00/GPIO2-IO16/GPIO_EMC_16	
0	0	Boot from fuses
0	1	Serial downloader (LPUART1 or USB1)
1	**0**	**Internal boot**

For simplicity the µTasker project assumes that internal boot option is used as standard, meaning that the ROM loader runs and the exact configuration is taken from eFUSEs or pin overrides, whereby the µTasker assumes that serial (QPI) NOR-Flash is used: the MIMXRT1020-EVK has an IS25LP064A-JBLE to this effect, which is an 8 Mbyte part. It is connected in QSPI mode on the primary FlexSPI interface of the i.MX RT 1021.

To ensure that the NOR-Flash mode is used the processor pins
*GPIO_EMC_25/*GPIO2_IO25

GPIO_EMC_24/GPIO2_IO24

GPIO_EMC_23/GPIO2_IO23

GPIO_EMC_22/GPIO2_IO22

should be pulled down at reset.

SRC_BT_CFG07 /GPIO2-IO25/ GPIO_EMC_25	SRC_BT_CFG06/ GPIO2-IO24/ GPIO_EMC_24	SRC_BT_CFG05/ GPIO2-IO23/ GPIO_EMC_23	SRC_BT_CFG04/ GPIO2-IO22/ GPIO_EMC_22	
0	**0**	**0**	**0**	**FlexSPI1 (Serial NOR)**
0	0	1	x	SD card
1	0	x	x	MMC/eMMC
0	1	x	x	SEMC (NAND)
0	0	0	1	SEMC (NOR)
1	1	x	x	FlexSPI1 (Serial NAND)

The MIMXRT1020-EVK has all configuration inputs pulled to ground by default and supplies a DIP switch with 4 switches to allow configuring the main boot mode and whether the FlexSPI or SD card is booted from.

The following setting is used:

SW8

DIP-1 SRC_BT_CFG00 /GPIO2- IO18/ GPIO_EMC_18	DIP-2 SRC_BT_CFG05 /GPIO2- IO23/ GPIO_EMC_23	DIP-3 SRC_BOOT_MOD E01/GPIO2- IO17/ GPIO_EMC_17	DIP-4 SRC_BOOT_MOD E00/GPIO2- IO16/ GPIO_EMC_16	
OFF	**OFF**	**ON**	**OFF**	Internal boot from FlexSPI1 (serial NOR)
ON	*OFF*	*ON*	*OFF*	*Internal boot from FlexSPI1 (serial NOR)* ***Encrypted XIP***

The eFUSES in a new part are set to supply the following configuration options in NOR Flash boot mode:
BOOT_CFG1[0] = 0 = XIP is not encrypted
BOOT_CFG2[2] = b00 = 500us hold time before read from device
 BOOT_CFG1[7..4] = b0000 = serial NOR device
BOOT_CFG1[3..1] = b000 = NOR device supports 0x3b read by default (on primary pin-mux option)

The result is that when the processor starts the very first thing that it does (its internal ROM loader controlling it) is read a block of data from the start of SPI Flash (address 0x60000000 in the memory map) in 2 line mode at 30MHz. This block is expected to contain details about the NOR Flash that is connected (how many lines are connected, what speed is to be used to communicate with it – including setup times - what instruction sequences does it need, plus various other such information.
When the data read is valid (there is also a header to help identify valid content) the ROM loader sets up the FlexSPI interface accordingly, configures the SPI Flash further and begins communicating at the final speed.

The following diagram shows the µTasker boot loader strategy which allows for running applications in internal SRAM (for maximum efficiency), stand-alone code updating (via USB, UART, Ethernet, and with existing protocols such as KBOOT), as well as OTA (Over The Air) loading by the working application via an Upload Image Area in QSPI with the support of the µTasker Bare-Minimum loader for fail safe re-flashing (of new applications or of updated stand-alone Serial Loader).

8MByte QSPI Flash

```
0x2023ffff                          0x607fffff    ┌──────┐  uParameterSystem
                                    0x607f0000    │ 64k  │  2 x 32k fail-safe
                    OCR                           │      │  swap block
                                                  │      │
0x20200000                                        │ 7.52M│  uFileSystem
                                                  │      │
0x2003ffff                          0x60098000    │      │
                                                  ├──────┤
                                                  │      │  Upload Image Area
                    DTC                           │ 256k │  - Application or
0x2000ffff          64k             0x60058000    │      │  - Serial Loader
0x20000000                                        ├──────┤
0x0003ffff                                        │      │  Application (supporting
                                                  │ 256k │  OTA updates) and
0x0002ffff                          0x60018000    │      │  executed in RAM
                                                  ├──────┤
                    190k            0x60008000    │ 64k  │  uTasker Serial Loader
                    ITC             0x60000000    ├──────┤
0x00000000                                        │ 32k  │  uTasker "BM" Loader
                                                  └──────┘
```

The operation is as follows:

1. Each time the processor starts, the µTasker "BM" Loader is the first code to be run in QSPI-Flash. It checks to see whether there is a valid new Application or Serial loader in the Upload Image Area.

2. If there is no valid image found it jumps to the Serial Loader (the serial loader is always present)

3. If the serial loader doesn't detect that it should operate (usually via an input) and also detects a valid application in the application code area it calls the µTasker "BM" Loader again, signalling that it should now start this valid application.

4. The µTasker "BM" Loader copies the application code to internal RAM and allows this to start.

2a. If the µTasker "BM" Loader finds either a valid new Application or a valid new Serial loader in the Upload Image Area it deletes the original code and updates it with the new source before resetting so that the new versions can operate.

3a. If the serial loader operates (due to detecting that it should remain in its mode or because there is no valid application code available) it allows new code to be loaded directly to the application area (using the chosen µTaster Serial Loader strategy, such as USB-MSD) before resetting so that the new code can subsequently be executed.

Typically the remaining QSPI-Flash space is used by the application for storing parameters (µParameterSystem) and files (µFileSystem).

The application is responsible for allowing OTA uploads to the Upload Image Area by whatever means is appropriate for the project/product. It can support both OTA loading of new applications (to replace itself) or new serial loader, after which the physical swap of the code is performed by simply commanding a reset so that the µTasker "BM" Loader can complete the work.

Initially there is always a combined firmware consisting of the µTasker "BM" Loader and the µTasker Serial Loader programmed to the board so that the first application can be loaded via the serial loader. Alternatively an initial application can also be combined to immediately have a complete firmware set.

All internal operations performed by the µTasker "BM" Loader are fail safe to ensure that updating code from the Upload Image Area cannot fail and cause the product to subsequently not be operational. *The application's OTA procedure must however ensure that non-operational images are ever loaded as valid new images to ensure that these cannot fail after such an upgrade!*

The exact dimensions of the areas shown are configurable, as are the sizes of the OCR/ITC and DTC areas in RAM. Generally it is the "BM" loader that configures the RAM banks to optimally suit the application that is to be loaded and initialises the application's initial stack pointer suitably to the top of DTC memory (see the section on RAM and Cache for more details).

13.   **Ethernet**

The i.MX RT 1021 has an internal 10/100M Ethernet controller but requires an external PHY for the connection to the cable.

There is a dedicated 500MHz fixed frequency PLL (PLL6) which supplies the Ethernet controller clocks. It also has a fixed 25MHz output referred to as ref_enet_pll2 and a configurable output called ref_enet_pll1, programmable to 25MHz, 50MHz, 100MHz or 125MHz.

The MIMXRT1020-EVK uses a Micrel KSZ8081RNB as PHY, connected in RMII mode. This PHY accepts a 25MHz reference clock from the processor (Ethernet controller) which is supplied on `GPIO_AD_B0_08` (`GPIO1-08`) – peripheral function `ENET_REF_CLK1`

14.   Conclusion

The present state of development is that the i.MX RT 1021 (MIMXRT1020-EVK) can be operated from SRAM (with debugger) or from QSPI Flash (stand-alone) in various clocking configurations up to maximum speed. GPIO, port interrupts, LPUARTs (interrupt and DMA driven), Ethernet, PIT, dynamic low power (WAIT) mode and DMA operations (memory-to-memory) can be demonstrated. Instruction and dat cache can be optionally enabled/disabledAll such operation is simulated in visual studio.

Next immediate steps:
- Add QSPI loader to run exclusively in SRAM (for optimal speed efficiency)
- Demonstrate full Modbus serial and TCP operation.
- Verify dynamic low power operation

 Subsequent steps:
- Add (new) USB device driver
- Add FreeRTOS configurations
- Verify on cheapest i.MX RT 1010 system (without ENET)

Present problems/investigation:

   – memory to memory DMA transfer is not giving any speed benefits over CPU copy:
      https://community.nxp.com/thread/518925

   – Free-running LPUART DMA reception works correctly when cache is disabled but has disturbances when cache is enabled

   – Ethernet works reliably after a power cycle but not always after a push-button or a commanded reset

   – Push button resets are understood as power cycle resets by the reset monitor

Modifications:
V0.04 18.11.2019: Initial version in development
V0.05 11.07.2023: Added TEMPMON

Appendix A – Hardware Dependencies

a) Space for first Appendix