AN02022: xcore.ai Clock Frequency Control

Publication Date: 2024/9/23
Document Number: XM-014200-AN v1.1.0

IN THIS DOCUMENT

1 PLL and Clock Divider Overview

xcore.ai devices have two internal phase locked loops (PLLs): the System PLL (sometimes known as the Core PLL or Primary PLL) and the Secondary PLL (sometimes known as the Application PLL). Both PLLs have the same internal workings.

The System PLL is driven from a low frequency external clock and it's output is used to obtain a high frequency System Clock. A set of clock dividers are then used on the system clock to derive the clocks for the *xcore* tiles, the switch and the reference clock. The Secondary PLL can be used for application purposes, and is driven either from the same external clock source, or from the System PLL.

The System PLL has fixed initial settings. They allow an 8-30MHz external clock to be used and operate the *xcore* tiles at 133-500MHz respectively. Typically, however, the PLL is reprogrammed on boot in order to set the PLL to the desired output frequency. The *XMOS* tools can be used to reprogram the PLL automatically by specifying the application's static clock configuration in a configuration file called the *XN file* that allows the designer to specify the components immediately around the *xcore* processor(s).

Note: Throughout this document it is assumed that a clock is supplied on the *XIN* pin. Instead of supplying a clock directly on *XIN*, one can instead use a crystal on *XIN*/*XOUT*, as specified in the datasheet.

1.1 PLL Constraints

There are a number of constraints on the frequencies of clocks at different points on the *xcore.ai* devices. These constraints must be met for the initial boot sequence, and if either PLL is reprogrammed, for the reprogrammed values too:

- The input clock on *XIN* must be between 8 and 30 MHz
  If using a crystal, it is suggested that a 24 or 25 MHz crystal is used as they are readily available, and are supported by the USB PHY.
- VCO frequency must be between 360 and 1800 MHz.
- The output frequency must be less than 800 MHz
- The comparator frequency must be between 220 kHz and 1800 MHz

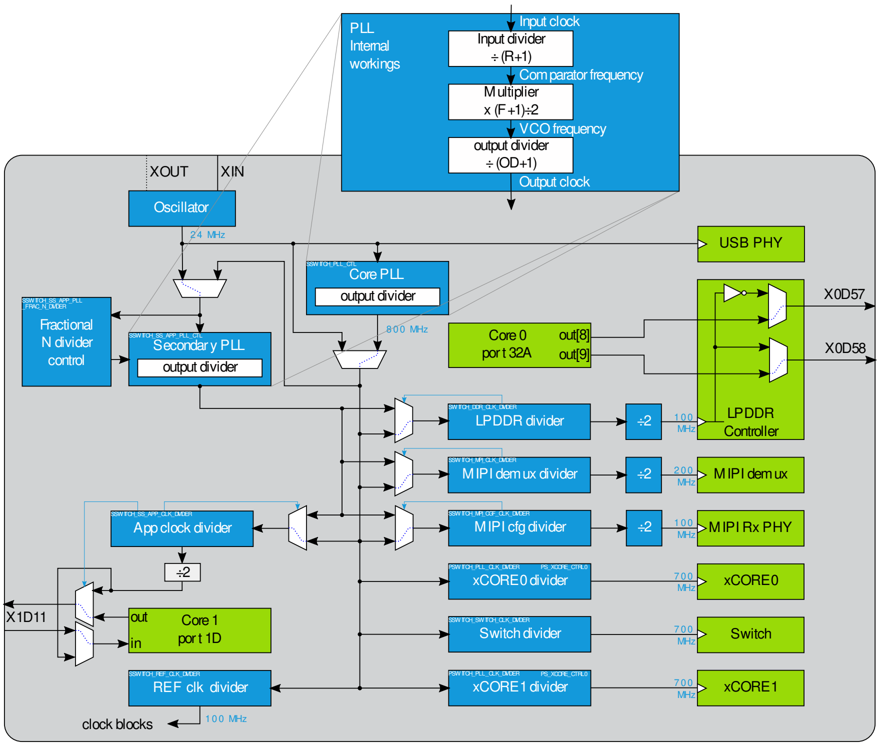

Fig. 1: PLL and Clock Dividers

- The device has a maximum operating frequency, which is specified in the datasheet for the device.
- The tile frequency should not be less than the reference frequency.

1.2 Fundamentals of the PLL

There are three dividers within the PLL. *R* divides the input clock down. The next divider, *F*, divides the output of the voltage controlled oscillator (VCO) stage down to the same frequency as the output of the *R* divider; hence *F* acts as a multiplier. *R* and *F* together determine the ratio between the VCO frequency and the input frequency. The output divider *OD* divides the VCO to form the output clock of the PLL.

- $f_{vco} = (F + 1)/2 \times 1 / (R+1) \times f_{in}$
- $f_{out} = f_{vco} / (OD + 1)$

Note that in this document we use *F*, *R*, and *OD* to mean the value that is stored in the register that controls the PLL. As such the value *1* is added to each of *F*, *R*, and *OD* before they are being used as a multiplier or divider. The divide value is the register value plus 1.

R must be in the range 0..63 (enabling a divider of between 1 and 64 inclusive), *OD* must be in the range 0..7 (enabling a divider of between 1 and 8 inclusive), and *F* must be in the range 0..8191 (enabling a divider of between 1 and 8192 inclusive)

2 Static configuration of the PLLs through the XN File

For the vast majority of applications, the tools can be used to set up the PLLs. The application's input oscillator frequency, system frequency, reference frequency and secondary frequency can be specified in the XN file as shown in table below . When the application code is written to a flash device with XFLASH, the code to reprogram the PLL to the desired system and reference frequencies will be added automatically into the boot sequence. When run with XRUN or XGDB the PLL is automatically reprogrammed via JTAG.

Attribute	Default	Description
Oscillator	none	Input frequency on the *XIN* pin. If this attribute is specified, the system frequency and the reference frequency are programmed using their specified (or default) values. If this attribute is not specified, the boot configuration for the system and reference frequencies are used for the application.
SystemFrequency	500 Mhz	The desired system frequency. The *Oscillator* attribute must be specified if this attribute is specified.
ReferenceFrequency	100 Mhz	The desired reference frequency. The *Oscillator* attribute must be specified if this attribute is specified.
SecondaryFrequency	none	If present, will cause Secondary PLL to be configured by the tools. It will not configure the fractional divider. The tools will choose the closest frequency possible without using the fractional divider. The *Oscillator* attribute must be specified if this attribute is specified.

The frequency control attributes should be added to the XML-element `<Node>` within the XN file. Frequencies should be specified with their unit of MHz, kHz or Hz, (for example 500MHz, 24576kHz or 6745800Hz). If the frequency control attributes are not specified in the XN file, then the XTC tools will not modify the frequency control registers.

If the target frequency specified in the XN file for the system, reference or secondary frequency cannot be met exactly for the application's input frequency, a frequency close to the target frequency will be selected by the XTC tools and a warning will be Issued. The XFLASH tool always issues the warning when it occurs, as does XGDB. XRUN only issues the warning if it has been run with the `--verbose` switch. XGDB issues the warning when the `connect` command is issued.

An Example XN file using frequency control attributes is listed below:

```
<?xml version="1.0" encoding="UTF-8"?>
<Network xmlns="http://www.xmos.com"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.xmos.com http://www.xmos.com">
  <Type>Board</Type>
  <Name>xcore.ai Explorer Kit</Name>

  <Declarations>
    <Declaration>tileref tile[2]</Declaration>
  </Declarations>

  <Packages>
    <Package id="0" Type="XS3-UnA-1024-FB265">
      <Nodes>
        <Node Id="0" InPackageId="0" Type="XS3-L16A-1024"
              SystemFrequency="500MHz"
              ReferenceFrequency="100MHz"
              Oscillator="20Mhz">
          <Tile Number="0" Reference="tile[0]"/>
```

(continues on next page)

(continued from previous page)

```
          <Tile Number="1" Reference="tile[1]"/>
        </Node>
      </Nodes>
    </Package>
  </Packages>
  <JTAGChain>
    <JTAGDevice NodeId="0"/>
  </JTAGChain>
</Network>
```

2.1 Configuration of LPDDR clock frequency

If an LPDDR memory is used, then its clock frequency can be configured by adding a Frequency attribute to the *ExtMem* external memory specification. The tools automatically decide how to set the LPDDR divider and its mux to achieve the desired frequency:

- If neither the System PLL (nor the Secondary PLL if supplied) can be used to perfectly achieve the desired ExtMem Frequency, a warning and advice message will be raised and an approximate frequency will be used.
- If the ExtMem Frequency can be perfectly achieved from either the SystemPLL or the SecondaryPLL, the System PLL will be chosen. This decision "frees up" the SecondaryPLL for application, rather than system usage.

See the application note on adding external memory, AN02021: Using external memory on xcore.ai, for details.

3 Configuring the system PLL of xcore.ai manually

The initial System PLL settings used after reset are:

- *R = 0*, no input divider
- *F = 99*, multiply by *(99+1)/2 = 50*
- *OD = 2*, output division by *(2+1) = 3*

This results in a multiplier of *50/3 = 16.667*. The minimum clock frequency allowed on the *XIN* pin is 8 MHz, which multiplies up to a 133 MHz clock. The maximum clock frequency allowed on *XIN* is 30 MHz, which multiplies up to a system clock of 500 MHz. This is a safe frequency for booting, typically the application program would override the settings and run the device at, for example, 600 or 800 Mhz.

If a different PLL configuration is required, and *the developer wants to configure the PLL manually*, then the new settings should be written to the *PLL_CTRL* register. (Please note that the rest of this subsection can be ignored if specifying the clock through an *XN* file.) The *PLL_CTRL* register has two bits that control how the new values are applied: an *nreset* bit and an *nlock* bit, at bit numbers *XS1_SS_PLL_CTL_NRESET_SHIFT* (31) and *XS1_SS_PLL_CTL_NLOCK_SHIFT* (30). Both bits are active low:

- By default, with both bits low, the chip will perform a soft reset with the new PLL values and the same boot code will execute again. It is important, therefore, that the boot code should read the value of the *PLL_CTRL* register and compare it to the reconfigured value. If there is a difference, then this is the first time the boot code has executed and the new PLL settings should be written to *PLL_CTRL*, causing a reset. The second time the boot code executes, the value read back from the *PLL_CTRL* register will be the reconfigured value and the boot process can continue without setting the PLL.
- If the *nreset* bit is set high and the *nlock* bit is set low, then the PLL clock will be suspended whilst the PLL adapts to the new settings. During this time, the chip will not have a clock, and it will be inactive for a short period of time (500 input clocks; typically a few microseconds).
- If both *nreset* and *nlock* bits are high, then the PLL will adapt whilst the chip is running. This means that the clock will be imprecise for a short period of time.

The last mode shall only be used if a small change is made to *F*; any change of more than 5% requires at least one of the *nlock* or *nreset* bits to be cleared.

The easiest way to reprogram the PLL is to specify the application's frequency requirements in the *XN* file and use the *XMOS* XTC tools to reprogram the PLL, as previously discussed.

3.1 Frequency Control Registers for the System Frequency

To access the frequency control registers on the *SSwitch* and *PSwitch*, packets of data must be constructed and communicated to the switches through a channel end.

Global PLL settings are controlled through the *Node Configuration Registers*. From *C*, use the *write_sswitch_reg()* and *read_sswitch_reg()* functions defined in *xs1.h* header file.

The fields that can be set in the *Node Configuration Registers* that control the clocks are shown in the tables below. Node control register 6, *XS1_SSWITCH_PLL_CTL_NUM*, controls the system PLL settings:

Field	Bits	Reset	Description
NRESET	31	0	Do not reset on write
NLOCK	30	0	Do not wait for PLL to be locked
JTAG	29	0	Boot from JTAG - keep this bit low
BYPASS	28	0	Bypass output clock from input clock
OD	25:23	2	PLL output divider stage = OD+1
F	20:8	99	Multiplier stage of the PLL = (F+1)/2
R	5:0	0	PLL input divider stage = R+1

Node control register 7, *XS1_SSWITCH_CLK_DIVIDER_NUM*, controls the divider for the xCONNECT switch:

Field	Bits	Reset	Description
SSDIV	15:0	0	System switch clock divider = SSDIV+1. The reset value produces 500MHz for a 500MHz system clock

Node control register 8, *XS1_SSWITCH_REF_CLK_DIVIDER_NUM*, controls the divider for the reference clock:

Field	Bits	Reset	Description
REFDIV	15:0	4	Reference clock divider = REFDIV+1. The reset value produces 100MHz for a 500MHz system clock

Warning: Writing to the *PLL_CTRL* register (0x6) may reset the xCORE tile. To reset a multi-tile device, make sure that tile 0 is reset last after any other tiles.

Settings on an individual tile basis are controlled through *Tile Configuration Registers*. From *C* use the *write_pswitch_reg()* and *read_pswitch_reg()* functions defined in *xs1.h*. The field that can be set to control the clock in the *Tile Configuration Registers* is shown in the table below. Tile control regiser 6, *XS1_PSWITCH_PLL_CLK_DIVIDER_NUM*, controls the divider for the tile:

Field	Bits	Reset	Description
XCDIV	15:0	0	xcore.ai clock divider = XCDIV+1. The reset value produces 500MHz for a 500MHz system clock

Setting the tile clock divider does not take effect until the tile itself allows the divider to be used, which is achieved by calling:

```
setps(XS1_PS_XCORE_CTRL0, 0x10)
```

The following sections show examples of setting the PLL to different values.

3.2 24MHz oscillator

For the initial boot, the system clock will be 400MHz with an 80 MHz reference clock. To get the xCORE to 500MHz the following are required: *R* = 0, *F* = 124, *OD* = 2. Write *0x01007C00* to *SSCTRL*, *PLL_CTRL* (0x6) register to bring the PLL output up to, for example, 500MHz, with code similar to the following:

```
    // This code should be called early on in main once
    #define PLL_VAL_24MHz 0x01007C00
    unsigned oldValue;
    read_sswitch_reg(get_local_tile_id(), XS1_SSWITCH_PLL_CTL_NUM, &oldValue);
    if (oldValue != PLL_VAL_24MHz) {
        write_sswitch_reg(get_local_tile_id(), XS1_SSWITCH_PLL_CTL_NUM, PLL_VAL_24MHz);
    }
```

3.3 12MHz oscillator

For the initial boot, the system clock will be 200MHz, with a 40 MHz reference clock. The following are required: *R* = 0, *F* = 249, *OD* = 2. Write *0x0100F700* to *SSCTRL*, *PLL_CTRL* (0x6) register to bring the PLL output up to 400MHz, with code similar to the following:

```
    // This code should be called early on in main once
    #define PLL_VAL_12MHz 0x0100F700
    unsigned oldValue;
    read_sswitch_reg(get_local_tile_id(), XS1_SSWITCH_PLL_CTL_NUM, &oldValue);
    if (oldValue != PLL_VAL_12MHz) {
        write_sswitch_reg(get_local_tile_id(), XS1_SSWITCH_PLL_CTL_NUM, PLL_VAL_12MHz);
    }
```

3.4 Fast Reference Clock

There may be situations where you want to increase the reference clock. The highest reference clock supported is half the system clock frequency. You can achieve this by setting *REFDIV* to *1* using the following code:

```
    write_sswitch_reg(get_local_tile_id(), XS1_SSWITCH_REF_CLK_DIVIDER_NUM, 0x01);
```

This will adjust all timers and clock-blocks (clocked from the reference clock) to run at half the system frequency. For example, if your system frequency is 600 MHz, then timers and ports will now run on a 300MHz clock that can be divided down to 150MHz, 75MHz and so on.

3.5 Slow switch clock

For applications where only a single *xcore.ai* tile is used, the SSwitch is only used for configuration purposes. Once the system is configured, the SSwitch clock can be substantially reduced to save on dynamic power. 1MHz is a good option for a low power SSwitch clock because the SSwitch power is dominated by the static power at this frequency.

To reduce the SSwitch clock to 1MHz with a system clock of 500MHz, set *SSDIV* to *499* using the following code:

```
    write_sswitch_reg(get_local_tile_id(), XS1_SSWITCH_CLK_DIVIDER_NUM, 499);
```

3.6 xcore Tile Clock 200MHz

If your application does not need to run the *xcore* tile at full speed to work, dynamic power can be saved by running the tile at a slower rate and activating the local divider.

For example, if you wish to run a tile at 250MHz from a system frequency of 500MHz, set *XCDIV* to *1* and enable the local divider by writing *0x10* to *XCORE_CTRL0*:

```
    write_pswitch_reg(get_local_tile_id(), XS1_PSWITCH_PLL_CLK_DIVIDER_NUM, 1);
    setps(XS1_PS_XCORE_CTRL0,0x10);
```

Note that without the SETPS the divider value will be ignored.

3.7 Low power configurations

For low power configurations, the PLLs need to be switched to a very low frequency. In order to keep a system clock running, the system PLL must first be set into a bypass mode, feeding the *XIN* clock directly into the system clock. After that, the System PLL can be set to have a very low *F* value and it will enter a low power mode. The code to perform this is shown in two functions below:

```
unsigned pllCtrlVal;

void pll_bypass_on(void) {
    #define PLL_VAL_BYPASS   0xE0000000
    setps(XS1_PS_XCORE_CTRL0,0x10);
    read_sswitch_reg(get_local_tile_id(), XS1_SSWITCH_PLL_CTL_NUM, &pllCtrlVal);
    write_pswitch_reg(get_local_tile_id(), XS1_PSWITCH_PLL_CLK_DIVIDER_NUM, 5);
    write_sswitch_reg(get_local_tile_id(), XS1_SSWITCH_PLL_CTL_NUM,
                      pllCtrlVal          | PLL_VAL_BYPASS);
    write_sswitch_reg(get_local_tile_id(), XS1_SSWITCH_PLL_CTL_NUM,
                      (pllCtrlVal& ~0xFFF)| PLL_VAL_BYPASS);
}

void pll_bypass_off(void) {
    write_pswitch_reg(get_local_tile_id(), XS1_PSWITCH_PLL_CLK_DIVIDER_NUM, 0);
    write_sswitch_reg(get_local_tile_id(), XS1_SSWITCH_PLL_CTL_NUM,
                      pllCtrlVal | PLL_VAL_BYPASS);
    write_sswitch_reg(get_local_tile_id(), XS1_SSWITCH_PLL_CTL_NUM,
                      pllCtrlVal | 0xC0000000    );
}
```

This will reduce the power consumption on the *PLL_AVDD* pin to a negligible value, and the dominant power will be leakage and any dynamic power used. This mode enables the device to comply with USB suspend.

More details on low-power operation are in the application note on power estimation, AN02023: xcore.ai Power Consumption Estimation.

4 Configuring the Secondary PLL

The Secondary PLL (or Application PLL) is a PLL with an optional software controlled fractional divider. The clock output can be divided down to as low as 171 Hz. It can be used for generating clocks inside the device, or to create an *application clock* out of the device. The Secondary PLL allows for one extra frequency to be created. Hence, the secondary PLL can be used for any one of the examples below.

When the secondary PLL is enabled, and the divider is enabled, the output is routed to pin X1D11 and port 1D on core 1 as is shown in the figure below. This makes the secondary PLL visible on a pin on the outside, and on tile 1 on port 1D. When enabled, tile 1 can input the clock on port 1D. If the clock is required on other tiles, then the clock should be routed to one-bit ports on those tiles externally.

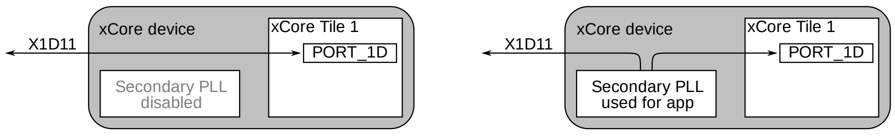

Fig. 2: Using the secondary PLL as an application clock.

The setup of the Secondary PLL is almost identical to the setup of the primary PLL. The main differences are:

- The output divider always has an extra divide by 2 included, to guarantee a 50/50 duty cycle.
- The Secondary PLL must be enabled before it can be configured.
- The Secondary PLL has an an optional software controlled fractional divider.

The XTC tools can set up the Secondary PLL as discussed before, but the fractional control has to be enabled manually.

4.1 Setting up the Secondary PLL manually

The secondary PLL is set-up in a similar manner to the System PLL, using register 15 rather than register 6. The differences are:

- The secondary PLL must be enabled by clearing bit 27 before writing to the *F* and *R* parts of the Secondary PLL.
- The *nreset* bit has no meaning, but similar to setting up the System PLL, if the frequency is changed a large amount, the output should not be used until the PLL has locked.

The output frequency of the Secondary PLL is the input frequency divided by *(R+1)* and *(OD+1)* and multiplied by *(F+1)/2*. *OD*, *F* and *R* must be chosen so that *R* is less than 64, *F* is less than 8192, and *OD* is less than 8. Also, the VCO frequency should be between 360 and 1800 MHz.

A flag allows the user to choose between two input frequencies: choices are either using the clock input to the device as the input to the secondary PLL, or the output of the primary PLL. This enables a large slew of frequencies to be created.

For example, *R* and *OD* can be set to 4 (dividing by 25), and *F* set to 60 (multiply by 61/2) by writing the following code:

```
#define PLL_VAL 0x02003C04
write_sswitch_reg(get_local_tile_id(), XS1_SSWITCH_SS_APP_PLL_CTL_NUM, 0); // Enable
write_sswitch_reg(get_local_tile_id(), XS1_SSWITCH_SS_APP_PLL_CTL_NUM, PLL_VAL);
```

With a 24 MHz clock, this will give a 24/25*61 = 29.28 MHz output clock.

4.2 Using the fractional divider

The Secondary PLL has an optional fractional divider, register *SSWITCH_APP_PLL_FRAC_N_DIVIDER*. When enabled, the fractional divider will count a period of divided input clocks, and over part of this period it will cause the secondary PLL to use a divider *F+1* rather than *F*.

The period *p* and fraction *f* are set through the control register for the fractional divider, and will result in an output frequency that is multiplied by *(F+1+f/p)/2* rather than *(F+1)/2*. *f* must be less than *p* at all times.

For example, supposing a crystal of 20 MHz, and the desired output frequency of 24.576 MHz. A multiplier of 768 (3 * 2^8), and a divider of 625 is required. A division by 625 cannot be achieved given the limitations of the PLL.

Instead, a division of 25 can be created, and a multiply by 30.72. That is, set both *R* and *OD* to 4, as *(4+1)*(4+1) = 25*, and then setting *F* as follows: *(F+1+f/p)/2 = 30.72*, hence *F+1+f/p = 61.44*, hence *F=60*, *f=44* and *p=100*. Although this will work, this can be improved further as large values of *p* lead to a lot of jitter. The common factor of four can be divided out in both *p* and *f*, and use *f=11*, *p=25*.

This is written to the registers with the following code:

```
#define PLL_VAL_2 0x02003C04
#define PLL_FRACT_VAL_2 0x80000B19
write_sswitch_reg(get_local_tile_id(), XS1_SSWITCH_SS_APP_PLL_CTL_NUM, 0); // Enable
write_sswitch_reg(get_local_tile_id(), XS1_SSWITCH_SS_APP_PLL_CTL_NUM, PLL_VAL_2);
write_sswitch_reg(get_local_tile_id(), XS1_SSWITCH_SS_APP_PLL_FRAC_N_DIVIDER_NUM, PLL_FRACT_VAL_2);
```

Fractional control will create jitter - the smaller the value of *p*, the less jitter there will be. Another solution with the same constraints would be: to divide by 20, and multiply by 38.4. That is, *(F+1+f/p)/2 = 38.4*, hence *F+1+f/p = 76.8*, hence *F=75*, *f=4* and *p=5*:

```
#define PLL_VAL_1 0x02004B03
#define PLL_FRACT_VAL_1 0x80000405
write_sswitch_reg(get_local_tile_id(), XS1_SSWITCH_SS_APP_PLL_CTL_NUM, 0); // Enable
write_sswitch_reg(get_local_tile_id(), XS1_SSWITCH_SS_APP_PLL_CTL_NUM, PLL_VAL_1);
write_sswitch_reg(get_local_tile_id(), XS1_SSWITCH_SS_APP_PLL_FRAC_N_DIVIDER_NUM, PLL_FRACT_VAL_1);
```

4.3 Keeping jitter low

In order to keep jitter low:

- Keep reference(input) divider as low as possible (to keep comprator frequency as high as possible)
- Keep feedback divider as low as possible.
- Keep VCO freq as high as possible.
- If you use the fractional-divider, keep *p* as low as possible.

4.4 Using the Secondary PLL for a PHY

One use of the Secondary PLL is to create a clock for one of the integrated PHYs or controllers in the device. Both the MIPI PHY and the LPDDR controller can be clocked from the Secondary PLL. This enables the external memory interface to be run at a high speed (eg, 166 MHz), whilst running the core at a lower speed (say, 150 MHz).

To use the Secondary PLL as an LPDDR clock with a divide of 4 use the following code:

```
#define DDR_CLK_VAL 0x80000001
write_sswitch_reg(get_local_tile_id(), XS1_SSWITCH_DDR_CLK_DIVIDER_NUM, DDR_CLK_VAL);
```

To use the Secondary PLL as a MIPI clock with a divide of 2 use the following code:

```
#define MIPI_CLK_VAL 0x80000000
write_sswitch_reg(get_local_tile_id(), XS1_SSWITCH_MIPI_CLK_DIVIDER_NUM, MIPI_CLK_VAL);
```

If the secondary PLL is used to generate a specific clock for one of these PHYs, the other PHY (if used) will need to be clocked using the core clock.

4.5 Using the Secondary PLL as an Application Clock

One use of the Secondary PLL is to create a clock for an external device, for example an audio CODEC or a microphone. To use the Secondary PLL as an application clock with a divide of 4 use the following code:

```
#define APP_CLK_VAL 0x80000003
write_sswitch_reg(get_local_tile_id(), XS1_SSWITCH_SS_APP_CLK_DIVIDER_NUM, APP_CLK_VAL);
```

Note: Setting the top bit selects the application clock to be derived from the Secondary PLL (otherwise the primary PLL will be divided), and that clearing bit 16 will enable the application clock divider, and also change the function of pin X1D11.

Normally pin X1D11 is connected to port 1D on core 1, but when the application clock divider is configured as shown above, the application clock will be driven on this pin, and on port 1D of tile 1. By connecting port 1D on tile 1 to a clock block on tile 1, that clock block can be used as a clock source for other ports, and that clock is also available to external devices.

5 Further reading

- The XS3 architecture manual
- XU316-1024-QF60A datasheet
- XU316-1024-QF60B datasheet
- XU316-1024-TQ128 datasheet
- XU316-1024-FB265 datasheet
- xcore.ai Power Consumption Estimation
- lib_sw_pll: A software library that, using the *xcore.ai* Application PLL, provides a PLL that will generate a clock that is phase-locked to an input clock.

6 Document history

6.1 AN02003 Changelog

1.1.0

- ADDED: Clarification of Secondary PLL differences
- ADDED: Missing constraints regarding PLL configuration
- ADDED: Clarification on F, R, OD values vs register values
- ADDED: Further emphasis on valid range for on F, R, OD values
- ADDED: Section with guidance on keeping PLL output jitter low
- CHANGED: Various minor documentation improvements
- FIXED: PLL adaption period corrected from 50 to 500 input clocks
- FIXED: Updated diagram in Fig 1 to represent a valid configuration

1.0.0

- Initial public release