S32K1xx Clock Calculator Guide
How to use S32K1xx tool to easily calculate device frequency domains

by: NXP Semiconductor

1 Introduction

The S32K1xx is NXP’s 32-bit general purpose MCU family for automotive and industrial applications. Our offer combines the latest 90nm technologies so that customers will not have to compromise performance in exchange for low power consumption. The S32K1xx is built upon the ARM Cortex-M4®, running at up to 112 MHz. This device family consists of two subfamilies: S32K14x and the S32K11x. The S32K14x series is the performance-grade line, comprising the devices S32K142, S32K144, S32K146, and S32K148; while the S32K11x (S32K116 and S32K118) is the low-cost sub-family for users who wish to operate at a lower price point but with a reduced feature set. For simplicity’s sake, this application note will refer to the S32K1xx family as “S32K”.

This device supports four clock oscillators and, in S32K14x, one system phase locked loop (SPLL) for a total of up to five clock sources. There are also multiple input pins through which external clock signals can be driven into the MCU. Of the four oscillators, there is a system oscillator (SOSC), a 48 MHz fast internal RC oscillator (FIRC), a 2-8 MHz slow internal RC oscillator (SIRC), and a 128 kHz low power oscillator (LPO). The SOSC can source from either a signal driven into the EXTAL pin or a crystal oscillator connected to the XTAL and EXTAL pins (henceforth referred as simply “XTAL”). EXTAL can support up to 50 MHz, while there are two ranges that are allowed for the XTAL depending on configuration: 4-8 MHz or 8-40 MHz; FIRC can be trimmed to 48 MHz; SIRC can be either 2 MHz or 8 MHz. In addition, the SPLL on S32K14x devices supports frequencies from 90 MHz to 160 MHz. See the following table for a summary.

<table>
<thead>
<tr>
<th>Clock Source</th>
<th>Allowed Frequencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIRC</td>
<td>48 MHz</td>
</tr>
<tr>
<td>SIRC</td>
<td>Selectable among 2 and 8 MHz</td>
</tr>
<tr>
<td>LPO</td>
<td>128 kHz</td>
</tr>
<tr>
<td>SPLL (S32K14x only)</td>
<td>90-160 MHz</td>
</tr>
<tr>
<td>SOSC</td>
<td>Selectable between XTAL and EXTAL</td>
</tr>
<tr>
<td>XTAL</td>
<td>Selectable ranges: 4-8 MHz and 8-40 MHz</td>
</tr>
<tr>
<td>EXTAL</td>
<td>Up to 50 MHz</td>
</tr>
</tbody>
</table>

Clock setup is a necessary step in almost all applications. The S32K clock calculator seeks to complement the configuration instructions in the reference manual by providing a graphical, interactive tool to help users find the correct register configuration in order to achieve their desired clock frequencies.

Accompanying this application note is the clock calculator. You can download it from S32K1xx_Clock_Calculator.
The clock calculator makes use of macros to perform functions like resetting the spreadsheet to initial values, configuring all clock frequencies to the maximum allowable settings, and copying generated code. Macros must be enabled in the user's MS Excel to access these features. If macros are turned off however, the tool will still be able to calculate clock frequencies, but the aforementioned features will be disabled. To turn on macros in MS Excel 2016, go to the Developer tab on the top toolbar and click on Macro Security. A popup window will appear. In it, select Enable all macros.

![Figure 1. Enabling macros](image)

2 Clock calculator design

The S32K clock calculator takes the form of an interactive Microsoft Excel spreadsheet, organized into multiple tabs as shown in the following figure.

![Figure 2. S32K1xx clock calculator setup](image)

Clock sources (i.e. oscillators, SPLL, external input pins) propagate to the various clock domains from which the MCU modules take their clocks. Most cells representing clock domain frequencies are not to be modified manually. The user is meant to enter frequencies to the few select clock sources and all clock domain frequencies derive from these sources. Several clock domain inputs are meant to be modified manually as they represent external clocks that are driven into the chip. There are also input cells that set muxes and clock dividers. All cells that take user inputs have blue borders instead of black, shown below. Blocks that require inputs also show the register fields that the blocks represent.
There are limits to what frequencies can be entered to the input frequency cells. Values that are out of range will be rejected and the user will receive an error message. Invalid clock domain frequencies that arise from valid input values and legal, but improper, dividers will be shaded in red. This is explained in greater depth later in this application note.

Frequency values are linked across tabs, so BUS_CLK in the Tree tab will always be the same as BUS_CLK in the Module Domains tab. Hyperlinks are provided to duplicate domain names to link back to their points of origin. For example, BUS_CLK originates in Tree. So clicking the BUS_CLK textbox in Module Domains will take the user to BUS_CLK in Tree. Textboxes that are links, when hovered over, will cause the mouse cursor to turn into a hand icon and a pop-up to appear, showing the address of the destination, as shown in the following figure.

Figure 3. Input cells vs. Output cells

Figure 4. Clicking on a link
The following subsections will explain in depth the purpose of each tab.

2.1 Tree

Tree is the centerpiece of the tool. This tab is the starting point for all clock frequency calculations. It is organized to resemble the S32K clock tree, as presented in the following figure.

![Clock Tree Diagram](image)

**Figure 5. S32K Reference Manual clock tree**

Figure 5 shows, in part, the diagram's clock tool counterpart. Additions were made to the Tree diagram to reflect the nuances that are not shown in the reference manual graphic. For the sake of simplicity, the reference manual graphic displays only the essential features. This tool consolidates all clocking options into a single platform.
Figure 6. Clock calculator tree

This tool's version is obviously a lot more complex than in the reference manual. In fact the screenshot could only reasonably display the top-left section of the diagram. The flow of the diagram generally goes from left to right. On the left are the S32K clock sources and on the right are the clock domains. MCU modules run on one or more of these clock domains.

Clock domain frequency values are displayed in the outlined cells next to their labels. Most cells are not meant to be written to, their values are dependent on the frequencies of preceding steps in the clock tree. Take BUS_CLK, for example: its value depends on the system power mode, the core clock divider, the system clock selector, and the controller of the source selected by the system clock selector. The system clock selector can choose either the SOSC, SIRC, FIRC, or the output of the SPLL. Now look at one of the sources, the FIRC block. FIRC is trimmed to 48 MHz but the frequency that propagates depends on the next block, FIRC Enable. Therefore the actual input frequency received by blocks that take the FIRC as a source is the FIRC frequency of 48 MHz, filtered by FIRC Enable. The same goes for SOSC, SIRC, and LPO. The SPLL output is configured in the SPLL tab.

BUS_CLK selects from these four clock sources by selecting the value of the System Clock Selector block. Then finally the selected signal is divided by the core clock prescaler value and filtered by the system mode.

This tab also features two buttons, Reset and Max. They only have function when macros are enabled. Clicking on these buttons with macros disabled will return an error. If macros are enabled, the Reset button will set all blocks to their reset value, as described in the reference manual. The Max button sets all blocks in this tool to values that configure the system and auxiliary clock domains to their respective maximum allowable frequencies. Below is a screenshot of the buttons.
2.2 Device select

The Device Select tab selects between the two S32K subfamilies. S32K11x lacks the SPL, the HSRUN power mode, and several modules compared to the S32K14x. Since this tool visualizes the fully featured S32K14x, when S32K11x is selected, S32K14x-only features are turned off. This means that the SPL output will be set to 0 and unavailable as a clock source, power mode blocks will be shaded red if the HSRUN power mode is selected, and S32K-only peripherals will have their clocks zeroed out as well.

2.3 Oscillator source control

S32K's external oscillators have a comprehensive set of options that warrants a separate tab. These features are reflected in the S32K clock calculator in the Oscillator Source Control tab. Oscillator Source Control contains the options for the SOSC and for the LPO. Below is a screenshot of the tab.
Figure 9. Oscillator source control

For the system oscillator, this tab provides options for choosing the frequency range, enable/disabling the oscillator, and selecting between XTAL and XTAL. The LPO control allows for frequency trimming, which is rated for 128 kHz, but can vary between 113 kHz and 139 kHz.

2.4 Power mode control

Since many clock domains are affected by the S32K system power mode, the power mode control options need its own tab. The figure below shows the power mode control sheet.
HSRUN and low power modes VLPR and VLPS need to be enabled in their own blocks, reflecting the S32K power management design. The list of options for S32K Power Mode will change, based on the setting of HSRUN Enable and VLPR/VLPS Enable. Note that S32K11x lacks an HSRUN mode, so if S32K11x is selected in the Device Select tab, the HSRUN Enable block has no effect, and HSRUN will not be available.

2.5 Module domains

The module domain tabs are an in-depth representation of the clocking for S32K modules. Where Tree leaves off at the clock domain level, the Module Domain tab picks up and progresses to the module level. A screenshot of Module Domains is shown in the figure below.
Figure 11. Module domains

The clock domains are color-coded. Black lines are reserved for local clock nodes. For example, BUS_CLK branches out to LPSPi, but is filtered through an LPSPi Clock Enable block. The arrow color after the block is changed to black to denote that the frequency value associated with that black line applies only to LPSPi. As a rule of thumb, clock domains are represented with black lines if all modules using it can fit within a single window without having to scroll.

2.6 SPLL

SPLL is a visual abstraction of the SPLL digital interface, as shown in the figure below.

![SPLL control diagram](image)

The input source of SPLL is the SOSC. Then, from the source, the dividers and multipliers located in the SPLL tab are set in order to achieve the SPLL output frequencies. The SPLL output frequencies are in turn propagated to the SPLL_CLK clock domain in the Tree tab. As mentioned in the previous sections, S32K11x lacks a PLL, so if S32K11x is selected in Device Select, SPLL_CLK will always be 0.
2.7 spll clk

The tab spll_clk is a reference table for the user to find the appropriate SPPLL dividers and multipliers to achieve the desired SPPLL frequency. Note that Columns A, B, and C of these tabs are frozen so if the table looks cut off, just scroll left or right.

SPPLL frequencies are calculated from a reference frequency, a multiplier (MFD), and a prescaler (PREDIV). The SPPLL reference is not manually configurable because there are a finite number of input values the SPPLL can take; the SPPLL will be whatever frequency SOSC is configured for. SPPLL reference therefore comes from the Tree tab. Once the SPPLL reference frequency is selected, enter the desired SPPLL output frequency. The reference table will then calculate the output frequency for each valid MFD and PREDIV setting. Like in the other sections, frequencies are color-coded to define which values are valid and which are not. Shading will change automatically once the output SPPLL frequencies are calculated. MFD and PREDIV settings that achieve the exact desired frequency will be shaded in green, values that exceed the desired frequency, but are within S32K hardware specifications are marked in yellow, and frequencies that exceed the S32K hardware specification are colored red. Below is a screenshot of the reference table.

![SPPLL_CLK reference table](image)

**Figure 13. SPPLL_CLK reference table**

2.8 Detailed module diagrams (RTC, SAI, QSPI, ENET, FlexCAN)

Some modules such as the FlexCAN and QSPI have additions clock configuration options, which can get too large to fit into the Module Domains tab. Therefore the modules RTC, SAI, QSPI, ENET, and FlexCAN each have their own dedicated sheet. The following section shows the RTC. Its concept can be extrapolated to the other aforementioned peripherals. The RTC block inside *Module Domains* is a hyperlink to the *RTC Clocking* tab, shown below.
The above figure shows that the module takes BUS_CLK, LPO1K_CLK, and RTC_CLK and outputs RTC_CLKOUT. RTC Clocking houses the actual RTC setup options that process these three inputs to produce RTC_CLKOUT. Below is a screenshot of the RTC Clocking tab.

### 2.9 Summary

Almost all blocks populating this clock calculator represent real register fields in silicon. The Summary tab collates all the information from the rest of the clock calculator into a list of register values, a screenshot of which is shown in the following figure. The values in the register summary are interactive, updating automatically when the associated block is changed. Registers listed within Summary are only the ones whose values are affected by clock configuration, not every single register available in the SoC.
The register values are displayed in either hexadecimal or binary format, where an “0x” header represents hexadecimal and “0b” denotes binary. A capital “X” represents a “don’t care” bit/half-byte. These bits do affect the clock frequency so users can set these values to the values that suit their purposes. Users can best utilize Summary by setting the configuration they want in the clock calculator and then copying the resulting register value into code. For example, taking from the figure above, the register SCG_SIRCCSR, should be set to 0x0XX00001. Assuming the “X” are “0”, the resulting S32DS C code would be `SCG->SIRCCSR = 0x00000001;`.

**Summary** also includes an overview of the clock domain frequencies. Since this tool consists of multiple interdependent spreadsheets, it may be cumbersome for users to weave through them all to find a clock domain. This table provides a place where all of them can be found. The table is organized by module, followed by the clock type (i.e. BIU clock, peripheral clock, protocol clock, etc.), and finally the frequency, as currently configured. Below is a screenshot.

**Figure 16. Register summary table**
<table>
<thead>
<tr>
<th>Module</th>
<th>Clock Domain</th>
<th>Frequency (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIRC</td>
<td></td>
<td>48</td>
</tr>
<tr>
<td>SIRC</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>SOSC</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>LPO128K_CLK</td>
<td></td>
<td>0.128</td>
</tr>
<tr>
<td>LPO32K_CLK</td>
<td></td>
<td>0.032</td>
</tr>
<tr>
<td>LPO1K_CLK</td>
<td></td>
<td>0.001</td>
</tr>
<tr>
<td>SPLL_CLK</td>
<td></td>
<td>96</td>
</tr>
<tr>
<td>CORE_CLK</td>
<td></td>
<td>48</td>
</tr>
<tr>
<td>SYS_CLK</td>
<td></td>
<td>48</td>
</tr>
<tr>
<td>BUS_CLK</td>
<td></td>
<td>48</td>
</tr>
<tr>
<td>FLASH_CLK</td>
<td></td>
<td>24</td>
</tr>
<tr>
<td>SPLLDIV1_CLK</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>SPLLDIV2_CLK</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>FIRCDIV1_CLK</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>FIRCDIV2_CLK</td>
<td></td>
<td>24</td>
</tr>
<tr>
<td>SIRCDIV1_CLK</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>SIRCDIV2_CLK</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>SOSCDIV1_CLK</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>SOSCDIV2_CLK</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>CLKOUT</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>LPO_CLK</td>
<td></td>
<td>0.128</td>
</tr>
<tr>
<td>RTC_CLKOUT</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>BIU</td>
<td></td>
<td>48</td>
</tr>
<tr>
<td>Peripheral Clock</td>
<td></td>
<td>24</td>
</tr>
</tbody>
</table>

This tool also supports a degree of code generation. *Summary* provides two sample clock initialization functions, `SysClk_Init` for configuring oscillators and PLLs and `InitPeriClkGen` for providing sources/dividers to auxiliary clocks. The dynamic C code in these functions depend on tool settings just like the register summary. These functions can be copy-pasted to a source file via Ctrl+C/Ctrl+V or by clicking on the associated *Copy Code* button if macros are enabled. The following figure shows `SysClk_Init` and its *Copy Code* button.
2.10 Limits

Limits is the reference tab for all the color-coding rules. The values in its tables are based on the S32K datasheet and reference manual and so should not be modified by the user. The following figure is a screenshot of the Limits tab.
3 Clock tool example use case: Configure LPSPi to SPLL BUS_CLK at 48 MHz and peripheral clock at 24MHz FIRC in RUN mode on S32K14x

The following sections will present an example application of the S32K clock calculator. This application note's example will configure the LPSPi bus interface clock to SPLL at 40 MHz and the LPSPi peripheral clock to FIRC at 24 MHz. It will not only show the correct configurations but also how the tool responds if improper configurations are attempted.

When configuring clocks for a module, start by looking at the module block. For this example, find LPSPi0:2 within Module Domains.
The module diagram shows that BUS_CLK drives the bus interface and either SOSCDIV2_CLK, SIRCDIV2_CLK, FIRCDIV2_CLK, or SPLLDIV2_CLK drives the LPSPI peripheral engine clock. The LPSPI bus interface clock, BUS_CLK, is currently 8 MHz; the LPSPI peripheral clock is 0 MHz, because the block LPSPI Per. Clk. Select contains the value 0, meaning no clock is selected. Configuring the clock calculator can be in any order, this example will start with BUS_CLK.

### 3.1 Set the device

First, make sure the correct S32K flavor is chosen. This example sets out to configure the S32K14x, so go to the Device Select tab and change the device to S32K14x.
Clock tool example use case: Configure LPSPI to PLL BUS_CLK at 48 MHz and peripheral clock at 24MHz FIRC in RUN mode on S32K14x

### 3.2 Set the power mode

Next make sure that the system is in Run mode. Go to the *Power Mode Control* tab and set the *S32K Power Mode* block to Run, as in the next figure.

![S32K in run mode](image)

**Figure 22.** S32K in run mode

### 3.3 Configure BUS_CLK

Return to the *Module Domains* tab and click on *BUS_CLK*; it will take you to the *BUS_CLK of Tree*, shown below.
Trace BUS_CLK all the way back to its point of origin. Start by tracing it to the Power Mode block, then the divider DIVBUS, onward to DIVCORE, and, finally, System Clock Selector, whose current value is 2. The cell is a dropdown menu and the textbox explains what each available value is associated with.

Since the goal is to configure BUS_CLK to SPLL, trace the SPLL back to its own source. SPLL sources from the SOSC. The oscillators FIRC, SIRC, SOSC, and LPO are the point of origin for all clock domains. The figure below shows the trace-back from SPLL back to the oscillators.

### 3.3.1 Configure the oscillator

Now start going downstream, configuring from the oscillator down to BUS_CLK. To give the SPLL a source, start with the SOSC. Click on the SOSC_CLK textbox to forward to the Oscillator Source Control sheet. SOSC_CLK can come from either the external

[NXP Semiconductors](https://www.nxp.com)
oscillator XTAL or a signal driven into a pin, EXTAL. XTAL is application-dependent and can be any value between 4 MHz and 8 MHz or 8 MHz and 40 MHz, depending on XTAL configuration. EXTAL must be under 50 MHz. Set the SOSC Range block to 3 to select the 8-40 MHz range, shown in the next figure. The 4-8/8-40 MHz SOSC block can now take any value between 8 and 40 MHz.

![Figure 25. SOSC set to high range](image)

This tool has a safeguard to prevent invalid values from being entered. The figure below shows an attempt to enter 7 MHz to the SOSC frequency cell. A dialog box appears notifying the user that the value is not accepted when he/she tries to click away from the cell.

![Figure 26. Invalid frequency input](image)

Set the SOSC frequency to 8 MHz. Trace forward from the 4-8/8-40 MHz SOSC block to SOSC Enable. Set SOSC Enable to 1 to enable the 8 MHz SOSC to propagate downstream, shown below.
Next, configure Ext. Ref. Select to 1 to select XTAL over EXTAL. SOSC_CLK will be sourced from the system oscillator at 8 MHz rather than the EXTAL pin. See below.

**3.3.2 Configure SPLL**

Now that SOSC_CLK is set to 8 MHz, go back to Tree and follow SOSC_CLK to the SPLL block, as seen in the next figure.
Click on the SPLL block to forward automatically to the SPLL tab. This is the tab that sets up the SPLL_CLK frequency. The Input Clock block of the figure below shows that SPLL detects the 8 MHz SOSC_CLK as its source frequency.

Configure the dividers to achieve 96 MHz; this frequency will be divided to 48 MHz later. The correct configuration can be achieved by trial and error, but the S32K clock calculator provides a lookup table in the spll_clk tab, shown below.
The **SPLL reference** field is the frequency of the SPLL input, in this case the 8 MHz SOSC. Set the target frequency. This example will target 96 MHz. The values and shading in the lookup table will automatically change to fit these new settings. In the figure below, the table has changed and circled is the modified field.

The cell shaded green means there is a divider combination that can achieve exactly 96 MHz given an input frequency of 8 MHz. In this case, a MFD of 8 and a PREDIV value of 0 will do the job. However, it is worth noting what happens if the output SPLL frequency is out of range.

In the following figure, the SPLL has been configured so that the output frequency is 188 MHz. This obviously exceeds the maximum hardware spec of 160 MHz. The associated voltage controlled oscillator (VCO) frequency, which can be back-calculated from SPLL_CLK also exceeds the maximum VCO spec of 320 MHz. Therefore, the output is crosshatched and shaded red.
Now let's configure the SPLL correctly. Turn on the SPLL in the SPLL tab by setting the SPLL Enable block to 1, and then set Prediv to 0 and Multiplier to 8. As shown in the next figure, the output SPLL_CLK is 96 MHz and the cell remains unshaded, meaning the configuration fits within spec.

Go back to Tree to observe that the SPLL_CLK frequency is now 96 MHz.
3.3.3 Finish Setting BUS_CLK

*BUS_CLK* is one of the system clocks. So, follow the *SPLL_CLK* signal down to *System Clock Selector*. *SIRC_CLK* is the current source of the system clocks. Change the value of *System Clock Selector* to 6 for the system clocks to follow *SPLL_CLK*, shown below.
After this, follow the system clock output to DIVCORE. The max frequency of CORE_CLK and SYS_CLK is 48 MHz in Run mode, so set DIVCORE from 0 to 1. This will divide the 96 Mhz signal by 2, thereby setting CORE_CLK and SYS_CLK to 48 MHz as well as the input to the DIVBUS block, whose output is BUS_CLK. See the figure below.

The user input for these fields is not the desired divider but the bitfield value that one would have to enter to achieve the desired divider. That is why the DIVCORE block description states “/(1+{0…15})” rather than simply “/1…16”. The user provides a value between 0 and 15, to which the hardware automatically adds 1 to calculate a divider that is between 1 and 16.

If, for example, DIVCORE is left at 0, which corresponds to a divider of 1, CORE_CLK and SYS_CLK would be 96 MHz, which would exceed their maximum allowable frequency of 48 MHz. The tool will highlight their cells red to signify that such a frequency is not allowed, shown below.
Set DIVCORE back to 1 and leave DIVBUS at 0 in order to keep BUS_CLK at 48 MHz. BUS_CLK has now been configured to 48 MHz SPLL, as seen in the figure below.

3.4 Configure LPSPI Peripheral Clock, FIRCDIV2_CLK

LPSPI follows BUS_CLK for its bus interface clock, but the peripheral clock can be SOSCDIV2_CLK, SIRCDIV2_CLK, FIRCDIV2_CLK, or SPLLDIV2_CLK. This example will set the peripheral clock to FIRCDIV2_CLK at 24 MHz. Go to the 48 MHz FIRC block in Tree. S32K’s FIRC can only be trimmed to 48 MHz, so leave the 48 MHz FIRC block value at 0 and set FIRC Enable to 1 to make the signal propagate, as shown below.
Figure 40. FIRC at 48 MHz

Trace the FIRC clock signal to the FIRC.Div block in Tree and set the block to 2. This enables FIRC.Div_CLK and divides the 60 MHz FIRC signal by 2, thus achieving an FIRC.Div_CLK domain of 24 MHz. See the following figure.

Figure 41. FIRC.Div_CLK set to 30 MHz

3.5 Configure LPSPI clocks

Go back to the Module Domains tab. Set the LPSPI Clock Enable block to 1 to enable the BUS_CLK signal. The LPSPI bus interface clock is now the 48 MHz BUS_CLK. Configure the LPSPI peripheral clock to FIRC.Div_CLK, setting the value of the LPSPI Per. Ck. Select block to 3. The LPSPI configuration will look like the following figure.
3.6 Observe the registers

The final register summary table, as displayed in *Summary*, is shown in the figure below. Note that most of these registers would not have to be written in code to achieve the setup that this example just configured. For example, the register PCC_FlexIO would not have to be included, since the FlexIO module was untouched. Registers that *would* have to be written would be ones like SCG_FIRCDIV and PCC_LPSPIx (the “x” means the LPSPI instance of your choice).
Copy the code

SysClk_Init and InitPeriClkGen provides dynamic clock generation C code. The code will configure the clocks to the settings as configured in this clock calculator. It can be copied and pasted to a source file. The following figure shows SysClk_Init as configured by this example. The solid-bordered highlight around the function means that the code has been copied with the Copy Code button; a regular Ctrl+C causes a dashed-bordered highlight. In both cases, the code can be pasted into a source with a regular Ctrl+V.
So, to summarize, this example has achieved its goal: a bus interface clock whose signal is driven by the BUS_CLK at 48 MHz. BUS_CLK comes from an 8 MHz SOSC driving an SPLL that produces an output of 96 MHz, and from there the SPLL driving BUS_CLK at 48 MHz. And finally a peripheral clock driven by a 24 MHz FIRCDIV2_CLK whose source is divided from a 48 MHz FIRC.

Figure 44. SysClk_Init after example

*/

VC

S32K1xx Clock Calculator Guide, Rev. 6, 09/2018

NXP Semiconductors
4 Conclusion

This application note gives an overview of the S32K interactive clock calculator. It seeks to aid clock configuration in the form of a graphical tool so that a user can more easily visualize the device's clock signals' propagation. There are similar clock calculators for other NXP products, including the MPC574xP and MPC574xG. Visit the NXP website to find more of these tools.

5 Revision history

<table>
<thead>
<tr>
<th>Revision Number</th>
<th>Date</th>
<th>Substantive changes</th>
</tr>
</thead>
</table>
| 1               | 05/2017| • In Summary on page 11 added the text "Summary also includes.......is a screenshot" and added Figure 17 on page 13.  
|                 |        | • Updated the S32K14x_Clock_Calculator sheet, please see the attachment. |
| 2               | 08/2017| • In Introduction added the texts "The clock calculator........Enable all macros" and "Attached to this........the attachment". Added figure Enabling macros and Finding the tool.  
|                 |        | • In Tree on page 4 added the text "This tab also.....of the buttons" and added figure Buttons.  
|                 |        | • Changed the section name from "RTC clocking" to "Detailed module diagrams (RTC, SAI, QSPI, ENET, FlexCAN)" and updated the section.  
|                 |        | • In Summary added the text "This tool also..... Copy code button" and added figure Sample initialization code  
|                 |        | • Added section Copy the Code.  
|                 |        | • Added the updated S32K14x_Clock_Calculator_Rev3 |
| 3               | 11/2017| Updated the associated S32K14x_Clock_Calculator file. |
| 4               | 01/2018| Updated the associated S32K14x_Clock_Calculator file. |

Table continues on the next page...
### Revision history

<table>
<thead>
<tr>
<th>Revision Number</th>
<th>Date</th>
<th>Substantive changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>08/2018</td>
<td>- Added information for S32K11x family.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Updated Introduction on page 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Added Device select on page 6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Added Set the device on page 12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Changed the name to Clock tool example use case: Configure LPSPI to SPLL BUS_CLK at 48 MHz and peripheral clock at 24MHz FIRC in RUN mode on S32K14x on page 15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Updated SPLL Calculator, When SPLL_CLK exceeds VCO and PLL spec, SPLL_CLK propagated to Tree and SPLL_CLK configured to 96 MHz.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Updated Register summary table.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Updated S32K RTC clocking.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Updated SPLL control.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Updated S32K power mode control.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Updated Clock calculator tree and Buttons.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Updated S32K1xx clock calculator setup.</td>
</tr>
<tr>
<td>6</td>
<td>09/2018</td>
<td>Updated the associated S32K14xClock_Calculator file.</td>
</tr>
</tbody>
</table>
Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, I2C BUS, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, Altivec, C-5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QoriQ, QoriQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vivib, Afast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, and UMEMS are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© 2018 NXP B.V.