NXP Semiconductors
Application Note

Document Number: AN5393
Rev. 9, April 2018

MPC574xP Clock Calculator Guide

How to use MPC5744P tool to easily calculate device frequency domains

by: NXP Semiconductors

1 Introduction

NXP's MPC574xP is a lockstep, dual-core 32-bit microcontroller intended for safety and chassis applications. This application note will refer to any device in the MPC574xP family, MPC5741P, MPC5742P, MPC5743P, and MPC5744P, as simply "MPC5744P."

The MPC5744P supports an 8-44 MHz external oscillator (XOSC), a 16 MHz internal RC oscillator (IRCOSC), and two phase locked loops (PLL) for a maximum operating frequency of 200 MHz. The IRCOSC is selected out of reset so increasing the operating frequency from 16 MHz requires additional configuration. The MPC574xP clock calculator is meant to complement the reference manual. It seeks to simplify the clock configuration process by providing a graphical, interactive tool to help the user find the correct register settings in order to achieve the desired clock frequencies.

Accompanying this application note is the clock calculator. You can download it from MPC574xP_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*.

Contents

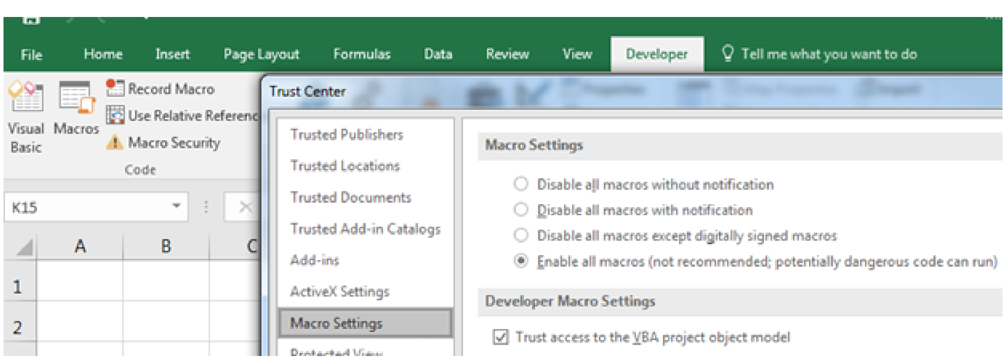


Figure 1. Enabling macros

2 Clock calculator design

The MPC574xP clock calculator takes the form of an interactive Microsoft Excel spreadsheet organized into multiple tabs as shown in the following figure.

Figure 2. MPC574xP clock calculator setup

Clock sources (e.g. oscillators and PLLs) 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 a pin. There are also input cells that set muxes and clock dividers. All cells that take entries have blue borders instead of black, shown below.

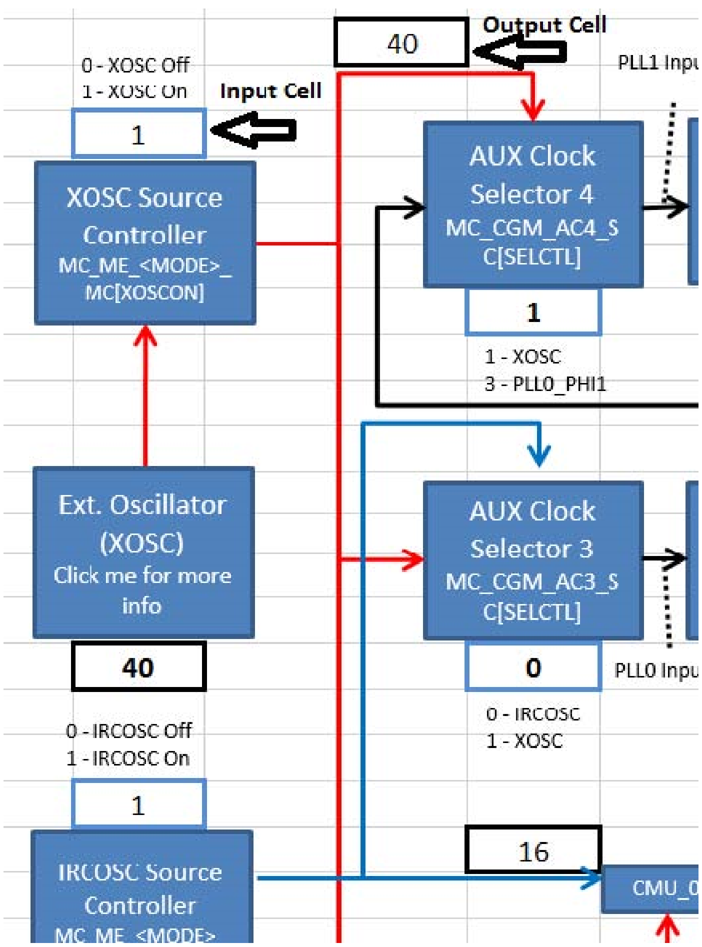


Figure 3. Input cells vs. output cells

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, as will be explained in greater depth later in this application note.

Frequency values are linked across tabs, so *FRAY_CLK* in the *Tree* tab will always be the same as *FRAY_CLK* in the *Peripheral Domains* tab. Hyperlinks are provided to duplicate domain names to link back to their points of origin. For example, *FRAY_CLK* originates in *Tree*. Therefore, clicking the *FRAY_CLK* textbox in *Peripheral Domains* will take the user to *FRAY_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.

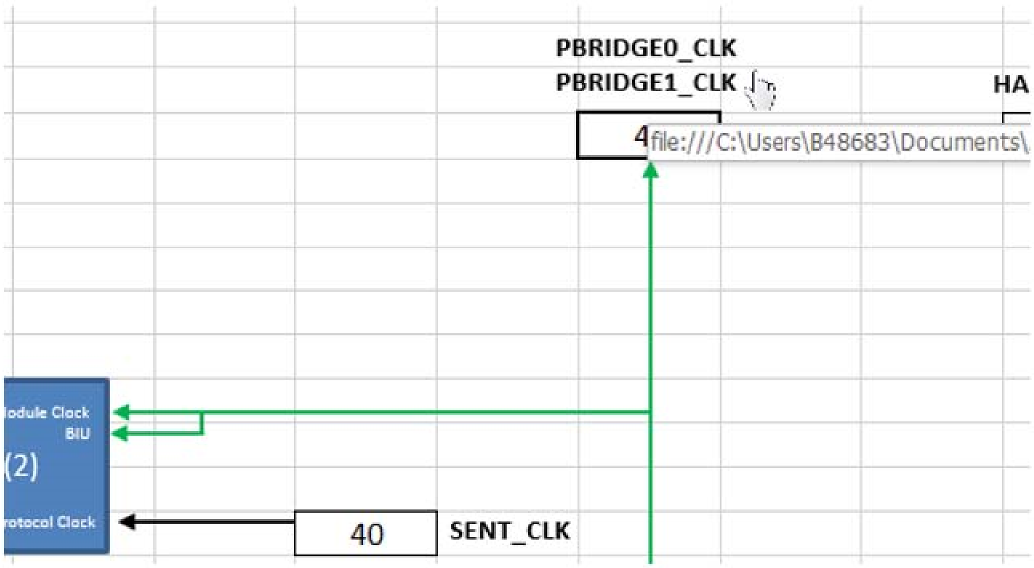


Figure 4. Clicking on a link

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 MPC5744P clock tree as presented in the following figure.

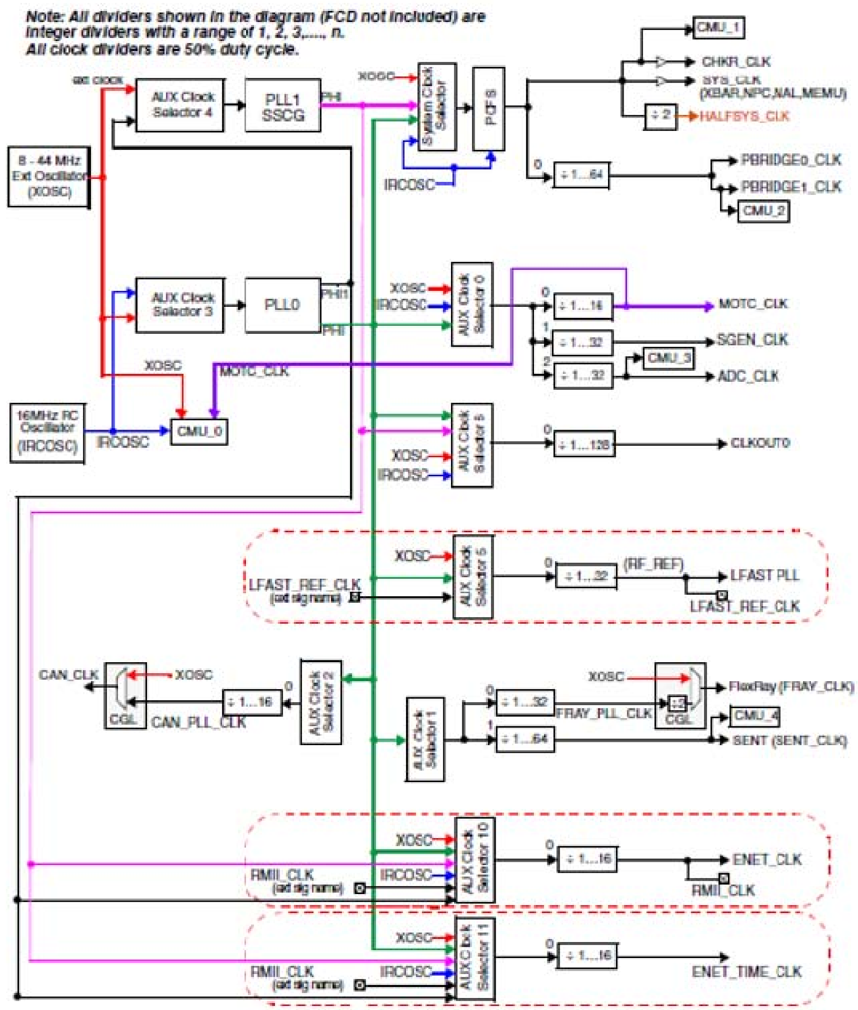


Figure 5. MPC5744P Reference Manual clock tree

The following figure shows, in part, the diagram's clock tool counterpart. The difference between the two is that the latter is interactive.

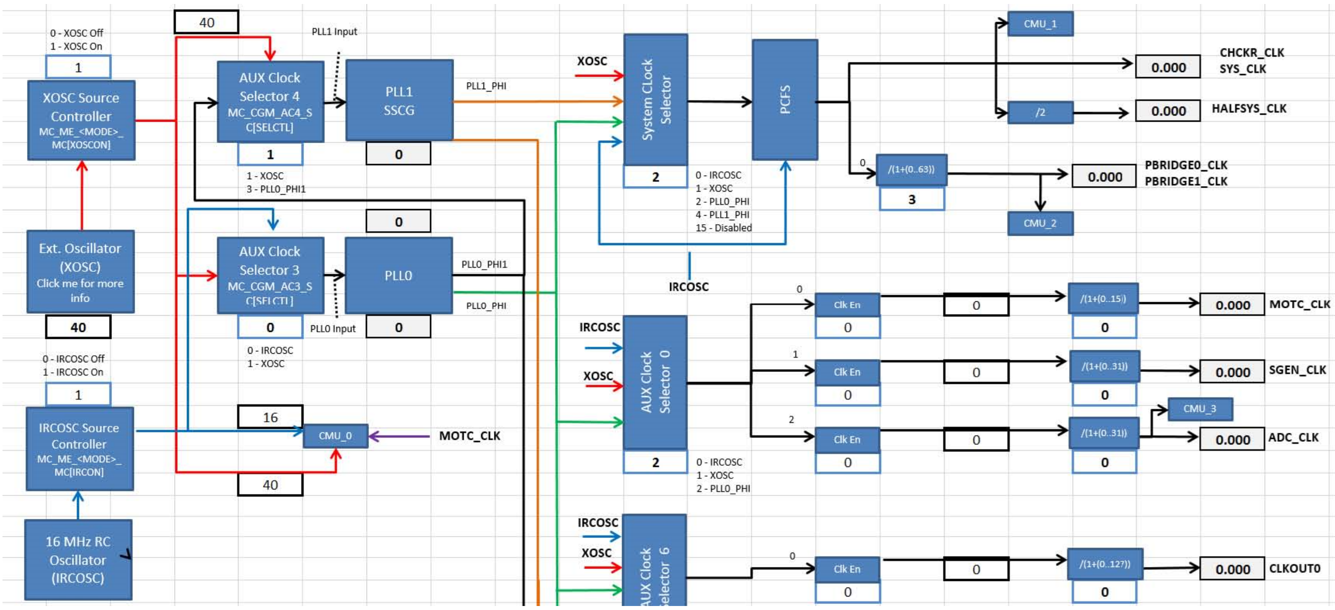


Figure 6. Clock calculator tree

The flow of the diagram generally goes from left to right. On the left are the MPC5744P 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 generated by preceding steps in the clock tree. Take *MOTC_CLK*, for example: its value is sourced from either the *IRCOSC*, *XOSC*, or *PLL0_PHI*. Now, look at the *IRCOSC* block. *IRCOSC* is at 16 MHz, but the frequency that propagates depends on the next block, *IRCOSC Source Controller*. Therefore, the actual input frequency received by blocks that take IRCOSC as a source is the IRCOSC frequency of 16 MHz, filtered by the *IRCOSC Source Controller* block. The same goes for XOSC. *PLL0_PHI* is configured in the *PLL0* tab. *MOTC_CLK* selects from these three clock sources by selecting the value of the *AUX Clock Selector 0* block. Then finally, the selected signal is divided by the *MOTC_CLK* prescaler value.

Each auxiliary clock and the system clock can feed into multiple domains that each have their own dividers. The number to the left of the prescaler shows the number of the divider that is associated with that clock. In the case of *MOTC_CLK*, the number “0” is shown next to the *MOTC_CLK* divider. This means that *MOTC_CLK* is configured by Divider 0 of Auxiliary Clock 0.

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.

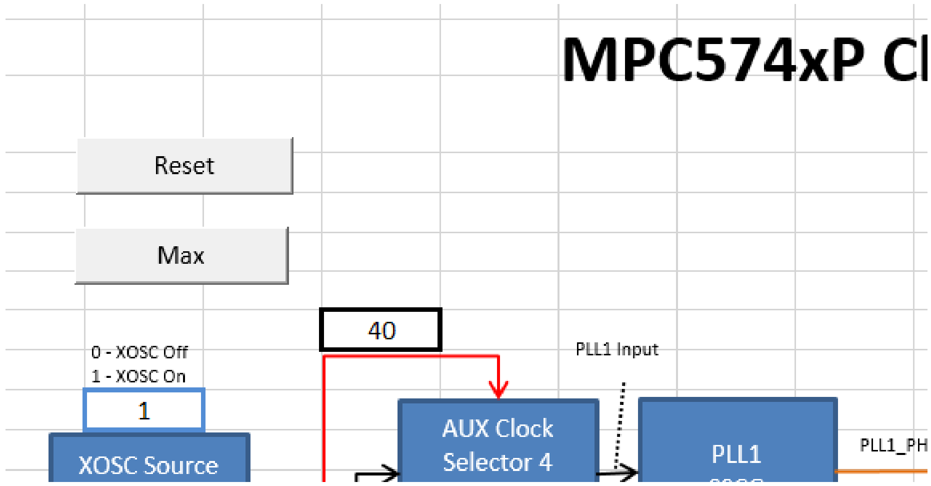


Figure 7. Buttons

2.2 Oscillator control

Oscillator Control controls the generation of the external oscillator (XOSC) frequency. MPC5744P supports two ways of XOSC generation. The chip has two external oscillator pins, *XTAL* and *EXTAL*. An 8-44 MHz external oscillator can be connected to both pins. This external oscillator is also referred to simply as the *XTAL*. If the *XOSC Select* block selects *XTAL*, XOSC will derive its frequency from the 8-44 MHz external oscillator (XTAL) block. Alternatively, a waveform can be driven directly to the *EXTAL* pin. This signal is also referred to simply as *EXTAL*. When the *XOSC Select* block selects *EXTAL*, XOSC will derive its frequency from the *EXTAL* pin. Shown below is a screenshot.

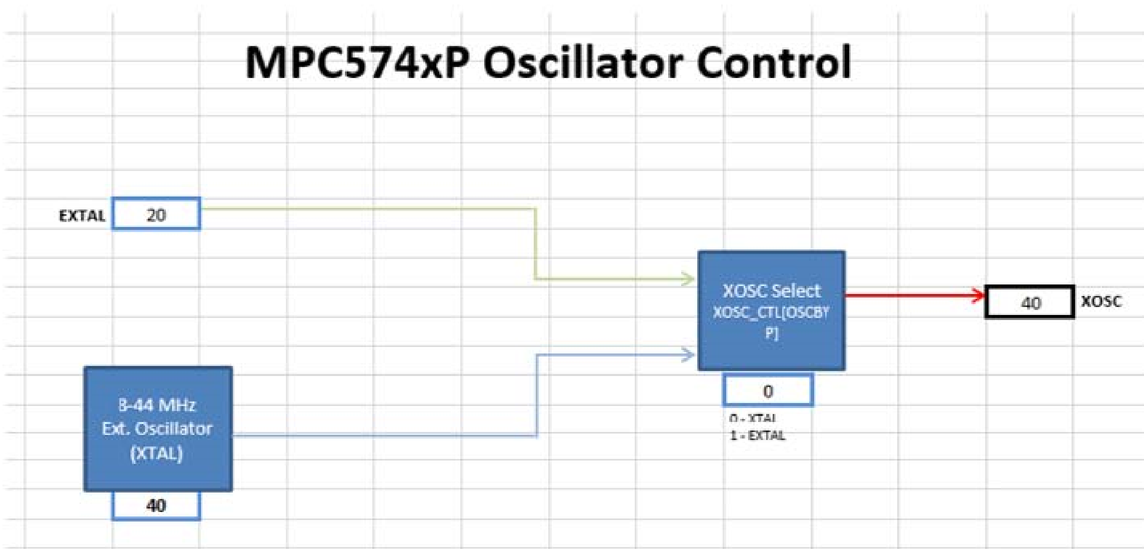


Figure 8. Oscillator control

2.3 Peripheral domains

Peripheral Domains is an in-depth diagram of MPC5744P modules. Where *Tree* leaves off at the clock domain level, *Peripheral Domains* picks up and progresses to the module level, shown below.

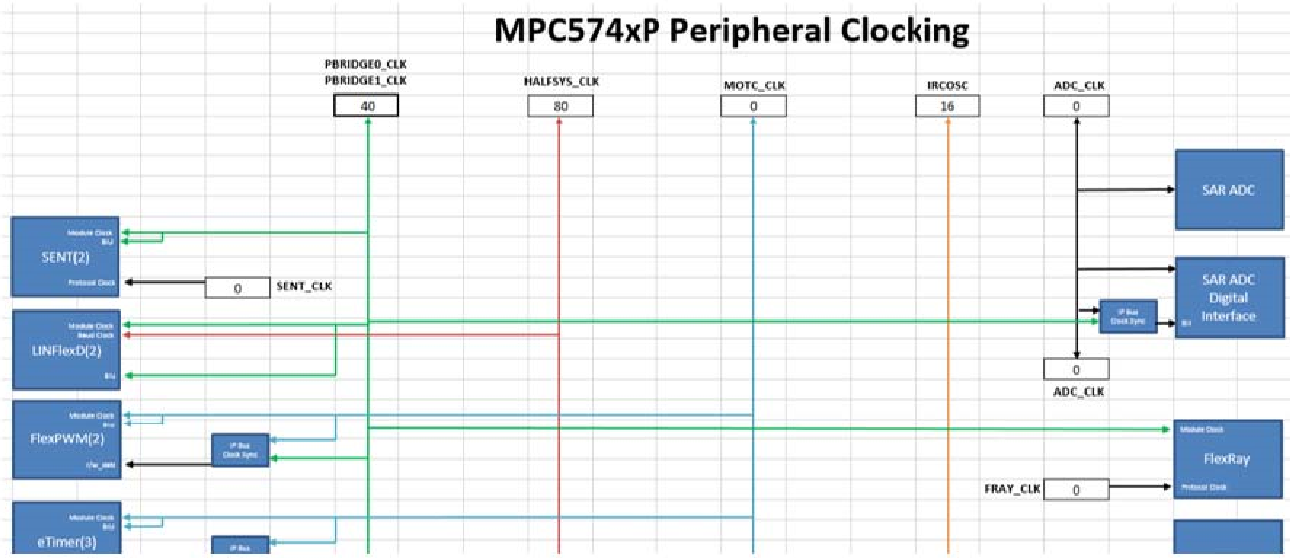


Figure 9. Peripheral domains

The clock domains are color-coded. Black lines are reserved for clock domains that only a few modules use. For example, the SAR ADC module takes both *PBRIDGEx_CLK* and *ADC_CLK. ADC_CLK* is black because only the ADC uses that clock. 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. The frequencies on this tab are not meant to be modified and are dependent on frequency values in the *Tree* tab.

2.4 LFAST clocking

The LFAST is a versatile, but intricate module. It supports its own PLL which generates multiple phases and generates a signal within specification only if its inputs are certain frequencies. These intricacies make it necessary to give LFAST is own dedicated tab. *Peripheral Domains* still hosts an LFAST block that shows its input clocks and is hyper-linked to *LFAST Clocking*, as shown in the following figure.

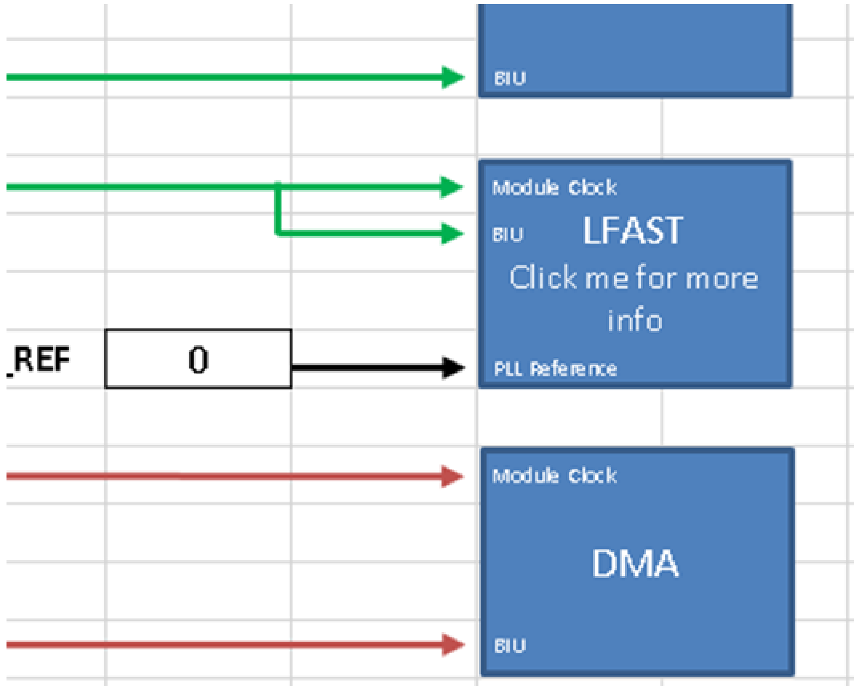


Figure 10. LFAST in peripheral domains

LFAST Clocking presents a block diagram of the module with various clocks going into it. It also supports LFAST_PLL configuration to increase the LFAST frequency up to 320 MHz. The LFAST also supports a low-speed mode as well as a high-speed mode. This tool allows the user to select between the two modes. Below is a screenshot of the sheet.

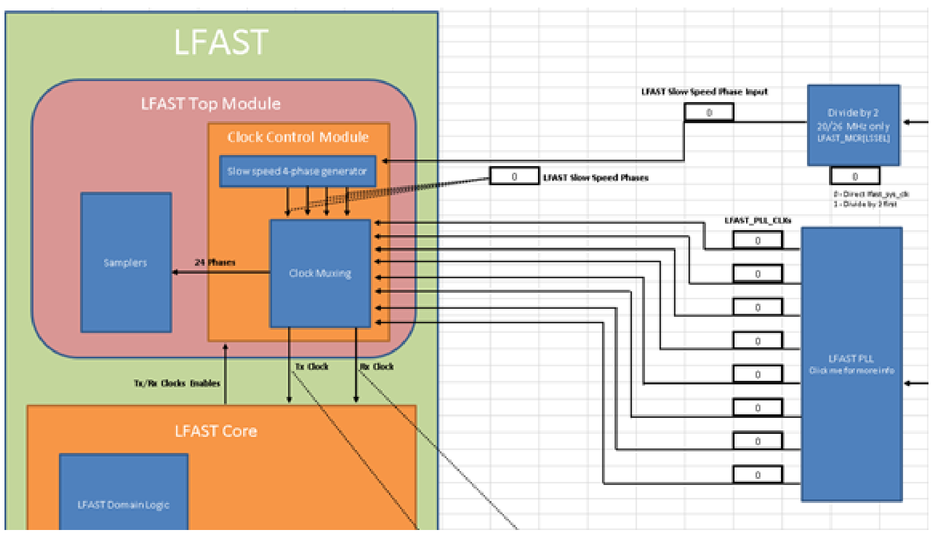


Figure 11. LFAST block diagram

Since the LFAST signal must be generated from an input clock of 10, 13, 20, or 26 MHz, this tool blocks any input from the signal *RF_REF* other than these four values. *RF_REF* can technically be set to any value, but any frequency that is not 10, 13, 20, or 26 MHz generates an invalid signal. Therefore, in this tool, *RF_REF* goes through the *LFAST Input Filter* block before becoming *lfast_sys_clk*, which in turn is the signal that gets fed into the LFAST_PLL and phase generators, as shown in the following figure.

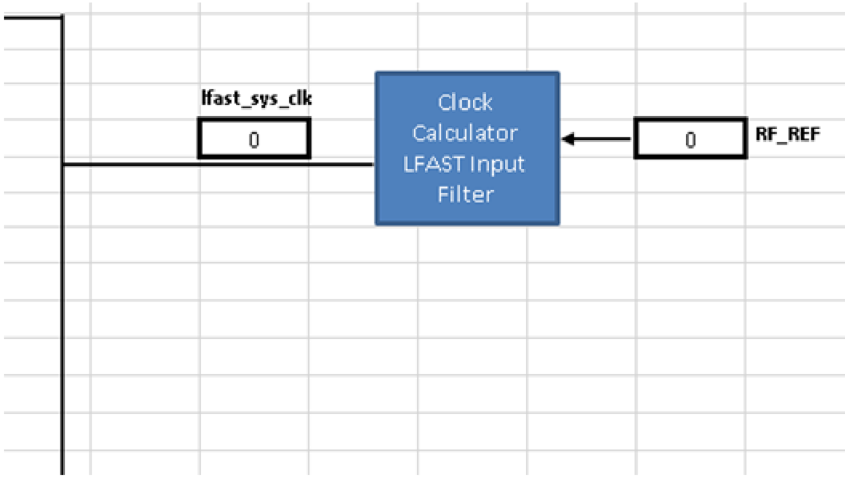


Figure 12. LFAST clocking input filter

If *RF_REF* is 10, 13, 20, or 26 MHz, lfast_sys_clk is also 10/13/20/26 MHz; otherwise, lfast_sys_clk is 0. MPC5744P does not actually filter *RF_REF* the way this tool does. The purpose of the *LFAST Input Filter* block is to simulate how the user can technically set *RF_REF* to any value, but the resulting LFAST output would be unusable. Therefore, if a user were to enter an invalid input frequency (i.e. not 10, 13, 20, or 26 MHz), all subsequent frequencies would be 0, and the user would know to change the input.

2.5 PLLx

PLL0 and PLL1 are visual abstractions of the PLL digital interface, as shown in the following figure.

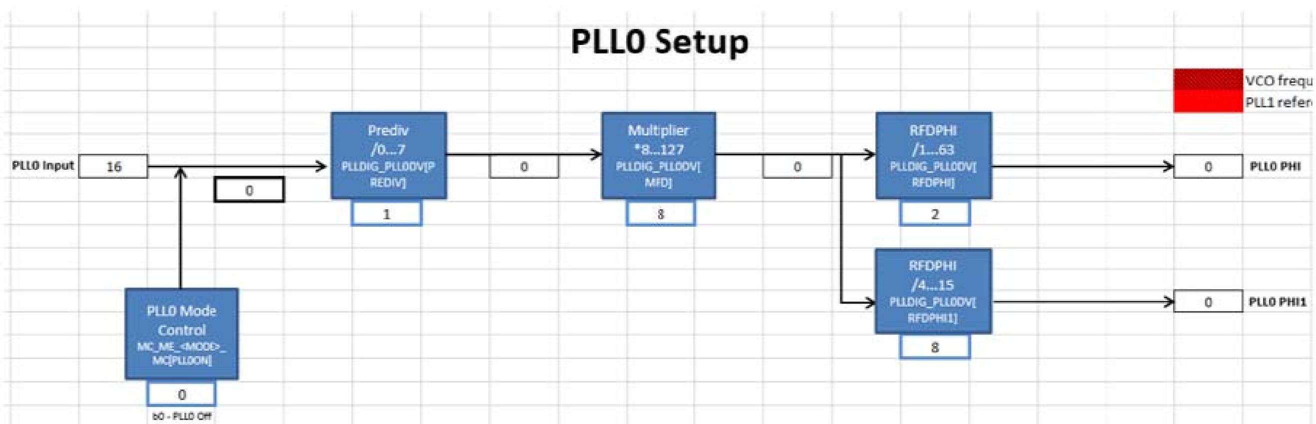


Figure 13. PLL0 control

The input source of PLL0 and PLL1 is selected by the auxiliary clock selectors, *AUX Clock Selector 3* and *AUX Clock Selector 4* in the *Tree* tab. Then, from the source, the dividers and multipliers located in the *PLL0* and *PLL1* tabs are set in order to achieve the PLL output frequencies. The PLL output frequencies are in turn propagated to the *PLLx_PHIn* clock domains in the *Tree* tab.

2.6 Reference Tables (pll0_phi, pll0_phi1, and pll1_phi)

The three tabs *pll0_phi*, *pll0_phi1*, and *pll1_phi* are reference tables for the user to find the appropriate PLL dividers and multipliers to achieve the desired PLL frequency. There is a tab for each PLL output because input frequencies and the range of acceptable divider/multiplier values differ between each other. However, they all follow the same setup. Note that Columns A, B, and C of these tabs are frozen so if the table looks cut off, just scroll left or right.

PLL frequencies are calculated from a reference frequency, a reference divider (RFD), a multiplier (MFD), and in PLL0, a prescaler (PREDIV). The PLL reference is not manually configurable because there are a finite number of input values the PLL can take. For example, PLL0 can only reference either the 16 MHz IRCOSC or the 8-44 MHz XOSC. PLL reference therefore comes from the *Tree* tab. Configure *AUX Clock Selector 3* and *AUX Clock Selector 4* in *Tree* for PLL0 and PLL1, respectively. Once the PLL reference frequency is configured, enter the desired PLL output frequency. Also, enter the PREDIV value when using *PLL0_PHI* or *PLL0_PHI1*. The reference table then calculates the output frequency for each MFD and RFD 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 PLL frequencies are calculated. MFD and RFD settings that achieve the exact desired frequency is marked in green; values that exceed the desired frequency, but are within MPC5744P hardware specifications are marked in yellow; and frequencies that exceed the MPC5744P hardware specification are colored red. Below is a screenshot of the reference table for *PLL0_PHI*.

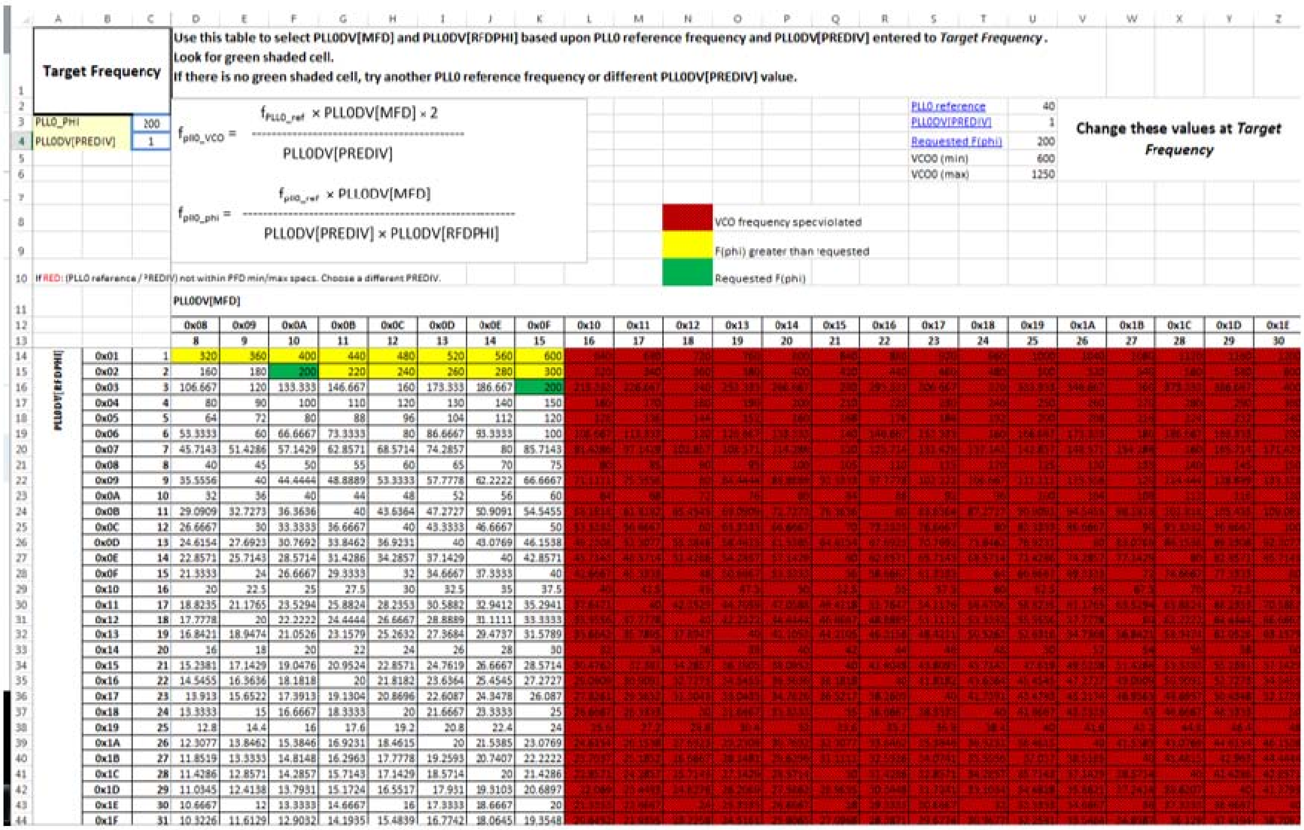


Figure 14. PLL0_PHI reference table 2.6

2.7 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.

MPC574xP S

Register Summary

Register	Value
MC_ME_<MODE>_MC	0xX0XX0070
XOSC_CTL	0b0XX00000XXXXXXXXX0000000X0000000
MC_CGM_SC_DC0	0x80000000
MC_CGM_AC0_SC	0x00000000
MC_CGM_AC0_DC0	0x00000000
MC_CGM_AC0_DC1	0x00000000
MC_CGM_AC0_DC2	0x00000000
MC_CGM_AC1_DC0	0x00000000
MC_CGM_AC1_DC1	0x00000000
MC_CGM_AC2_DC0	0x00000000
MC_CGM_AC3_SC	0x01000000
MC_CGM_AC4_SC	0x01000000
MC_CGM_AC5_SC	0x01000000
MC_CGM_AC5_DC0	0x00000000
MC_CGM_AC6_SC	0x00000000
MC_CGM_AC6_DC0	0x00000000
MC_CGM_AC10_SC	0x00000000
MC_CGM_AC10_DC0	0x00000000
MC_CGM_AC11_SC	0x00000000
MC_CGM_AC11_DC0	0x00000000
PLLDIG_PLL0DV	0x40022010
PLLDIG_PLL1DV	0x00020014
PLLDIG_PLL1FD	0x000X0000
LFAST_PLLCR	0bXXX0000000000XXXXX00XX001011100
LFAST_SCR	0x000X0000
LFAST_COCR	0bXXXXXXXX0000000000000000000000XXX0
LFAST_MCR	0xXX00X0XX

Figure 15. Register summary table

The register values are displayed in either hexadecimal or binary format, where a “0x” prefix represents hexadecimal and “0b” denotes binary. A capital “X” represents a “don’t care” bit/half-byte. These bits do not affect the clock frequency, so users can set these values to whatever suits 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 MC_ME_DRUN_MC (among the MC_ME_<MODE>_MC registers) should be set to 0xX0XX0070. Assuming the instances of “X” are “0”, the resulting S32DS C code would be: "MC_ME.DRUN_MC.R = 0x00000032;".

Summary also includes an overview of the clock domain frequencies. Since this tool consists of multiple interdependent spreadsheets, it might 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.

ummary

Clock Summary

Module	Clock Domain	Frequency (MHz)
System	IRCOSC	16
	XOSC	40
	PLL0_PHI	160
	PLL0_PHI1	40
	PLL1_PHI	0
	SYS_CLK	16
	CHKR_CLK	16
	HALFSYS_CLK	8
	PBRIDGE_0_CLK	16
	PBRIDGE_1_CLK	16
	MOTC_CLK	0
	SGEN_CLK	0
	ADC_CLK	0
	CLKOUT0	0
	LFAST_PLL	0
	LFAST_REF_CLK	0
	FRAY_CLK	0
	SENT_CLK	0
	ENET_CLK	0
	RMII_CLK	0
	ENET_TIME_CLK	0
	CAN_CLK	0
SENT0:1	Module Clock	16
	BIU	16
	Protocol Clock	0
LINFlexD0:1	Module Clock	16
	Baud Clock	8
	BIU	16
FlexPWM0:1	Module Clock	0
	BIU	0

Figure 16. Clock summary 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 copied and 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.

Sample Initialization Code Copy Code

```
//Enable XOSC, PLL0, PLL1, and enter RUN0 with PLL1_PHI as system clock (200 MHz).
void SysClk_Init(void)
{
    MC_CGM.AC3_SC.R = 0x01000000; //Connect XOSC to the PLL0 input.
    MC_CGM.AC4_SC.R = 0x03000000; //Connect PLL0_PHI1 to the PLL1 input.

    //Set PLL0 to 160 MHz with 40 MHz XOSC reference.
     PLLDIG.PLL0DV.R = 0x40024020; //PREDIV = 4, MFD = 32, RFDPHI = 2, RFDPHI1 = 8

    MC_ME.RUN0_MC.R = 0x00130070;   // RUN0 cfg: IRCON,OSC0ON,PLL0ON,syclk=IRC

    // Mode Transition to enter RUN0 mode:
    MC_ME.MCTL.R = 0x40005AF0;   // Enter RUN0 Mode & Key
    MC_ME.MCTL.R = 0x4000A50F;   // Enter RUN0 Mode & Inverted Key
    while (MC_ME.GS.B.S_MTRANS) {};   // Wait for mode transition to complete
    while(MC_ME.GS.B.S_CURRENT_MODE != 4) {};   // Verify RUN0 is the current mode

    //Set PLL1 to 200 MHz with 40 MHz PLL0_PHI1 input.
    PLLDIG.PLL1DV.R = 0x00020014; //MFD = 20 MHz, RFDPHI = 2 MHz
    PLLDIG.PLL1FD.R = 0x00000000; //EnableandconfigurationfractionalmultiplierforPLL1

    MC_ME.RUN_PC[0] = 0x000000FE; //Enable peripherals to run in all modes
    MC_ME.RUN0_MC.R = 0x001300F4;   // RUN0 cfg: IRCON, OSC0ON, PLL1ON, syclk=PLL1_PHI

    MC_CGM.SC_DC0.R = 0x80030000; //Divide system clock by 4 to achieve PBRIDGEx_CLK of 50 MHz

    // Mode Transition to enter RUN0 mode:
    MC_ME.MCTL.R = 0x40005AF0;   // Enter RUN0 Mode & Key
    MC_ME.MCTL.R = 0x4000A50F;   // Enter RUN0 Mode & Inverted Key
    while (MC_ME.GS.B.S_MTRANS) {};   // Wait for mode transition to complete
    while(MC_ME.GS.B.S_CURRENT_MODE != 4) {};   // Verify RUN0 is the current mode
}
```

Figure 17. Sample initialization code

2.8 Limits

Limits is the reference tab for all the color-coding and code generation rules. The values in its tables are based on the MPC5744P's datasheet and reference manual, and therefore should not be modified by the user. The following figure is a screenshot of the *Limits* tab.

	A	B
1		
2	Do not change these numbers	
3	PLL0 Input (min)	8
4	PLL0 Input (max)	40
5	PLL0_VCO (min)	600
6	PLL0_VCO (max)	1250
7	PLL0_PHI (min)	4.76
8	PLL0_PHI (max)	625
9	PLL0_PHI1 (min)	20
10	PLL0_PHI1 (max)	156
11	PLL1 Input (min)	38
12	PLL1 Input (max)	78
13	PLL1_VCO (min)	600
14	PLL1_VCO (max)	1250
15	PLL1_PHI (min)	4.76
16	PLL1_PHI (max)	625
17		
18		
19		
20		
21		
22	Clock Name	Max (MHz)
23	CHKR_CLK, SYS_CLK	200
24	PBRIDGE_0/1	50
25	MOTC_CLK	160
26	HALFSYS_CLK	100
27	ADC	80
28	SGEN	20
29	LFAST	320
30	FRAY_CLK	80
31	CAN_CLK	80
32	SENT_CLK	80
33	Ethernet	100
34	ENET_CLK	50
35	ENET_TIME_CLK	50

Figure 18. MPC5744P frequency limits

3 Clock tool example use case: configure ADC to PLL0 at 80 MHz

The following sections will present an example application of the MPC574xP clock calculator. This application note's example will configure the ADC to PLL0 at 80 MHz and 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 at *Peripheral Domains*. As shown in the next figure, SAR ADC follows two clock domains, *PBRIDGEx_CLK* for the bus interface unit and *ADC_CLK* for the actual converter.

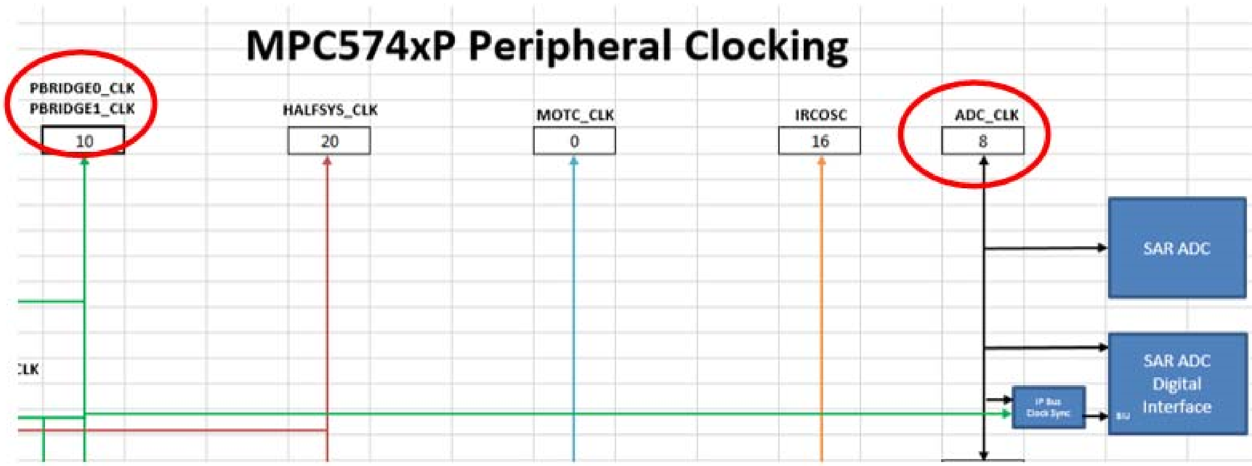


Figure 19. ADC clocks

PBRIDGEx_CLK and *ADC_CLK* are currently 10 MHz and 8 MHz, respectively. Configuring the clock calculator can be in any order; this example will start with *ADC_CLK*.

3.1 Configure ADC_CLK

Click on *ADC_CLK* to forward to the *ADC_CLK* cell of *Tree*, as shown in the following figure.

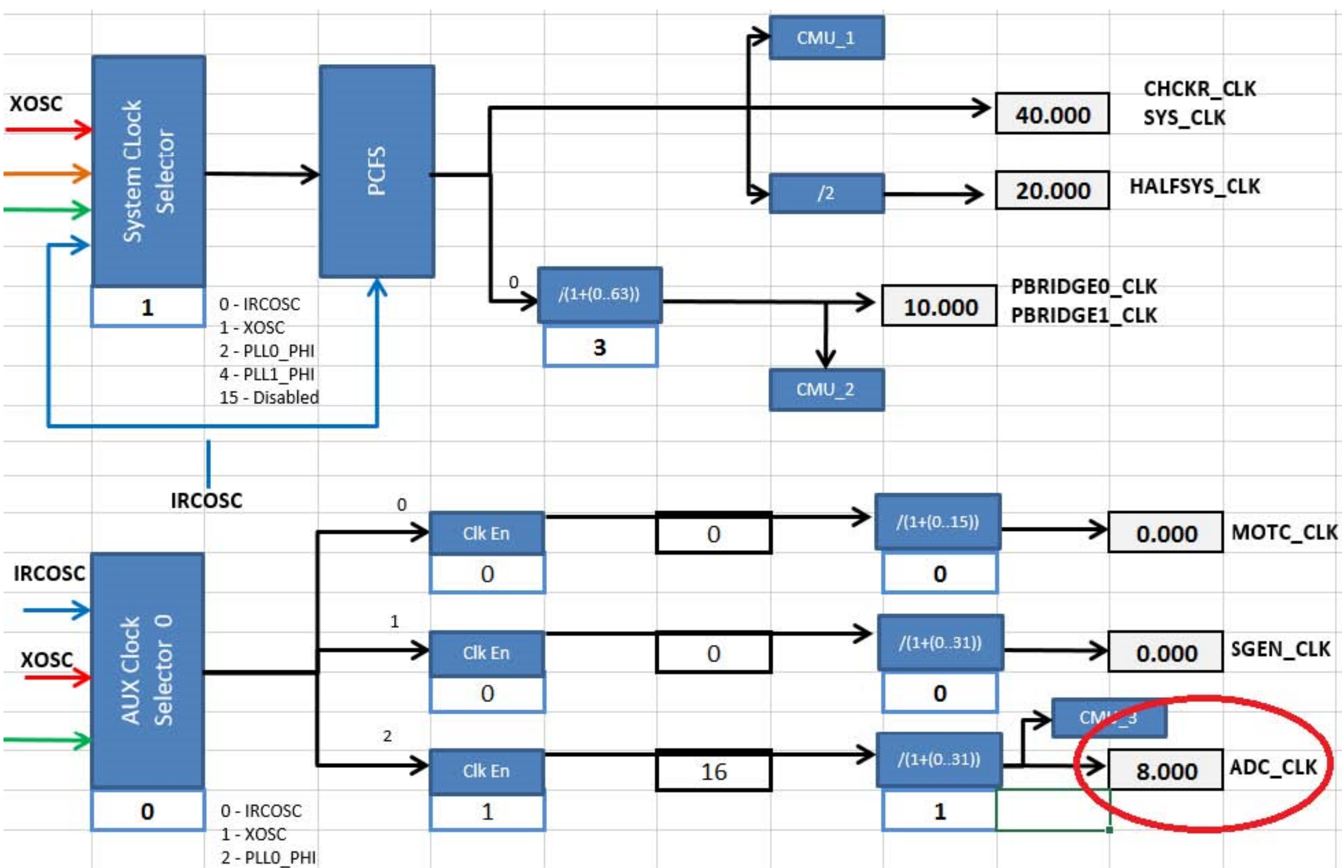


Figure 20. ADC_CLK, Tree tab

Trace *ADC_CLK* all the way back to its point of origin. As shown in the figure, *ADC_CLK* is enabled and is sourced from *AUX Clock Selector 0*, whose current value is 0. The cell is a drop-down menu and the accompanying textbox explains what each available value represents. Currently, *ADC_CLK* is enabled and is sourced from the 16 MHz IRCOSC, divided by 2, for a final frequency of 8 MHz.

Since the only way to achieve 80 MHz is through the PLL and PLL0 is the only PLL that goes to *ADC_CLK*, trace *PLL0_PHI* back to its own sources. PLL0 selects from either IRCOSC or XOSC via *AUX Clock Selector 3*. These oscillators are the point of origin for all clock domains. The following figure shows the ADC_CLK being traced back to PLL0 and then finally to the oscillators.

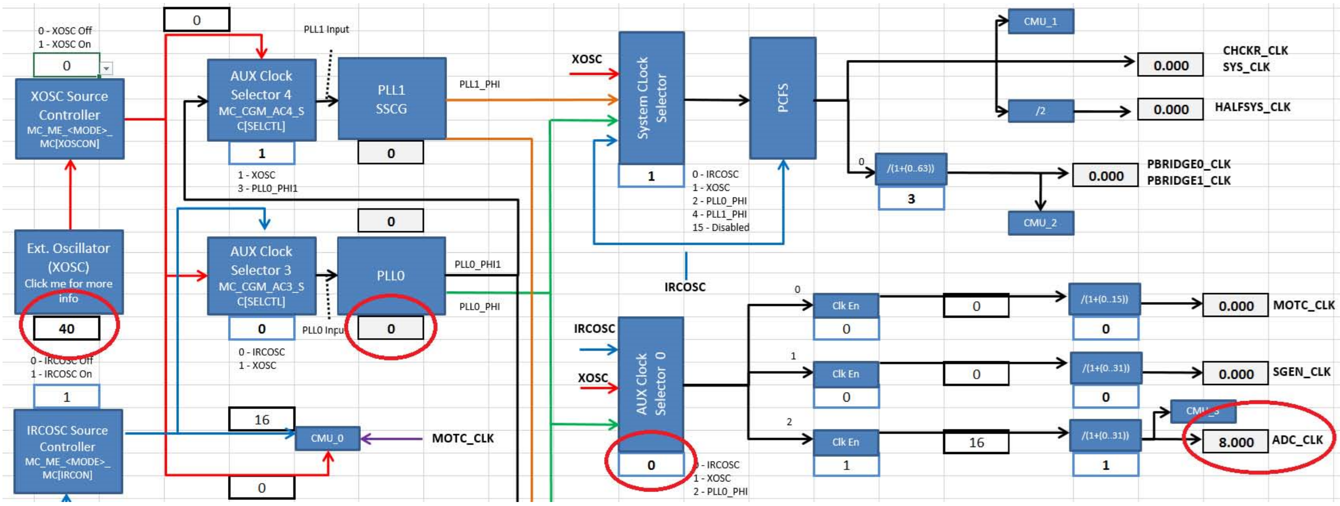


Figure 21. ADC_CLK, Tree tab

3.1.1 Configure the oscillator

Now start going downstream, configuring from the oscillator down to *ADC_CLK*. The external oscillator is configured by the *Oscillator Control* tab. Its frequency is application-dependent and can be any value between 8 MHz and 44 MHz. This tool has a safeguard to prevent invalid values from being entered. The following figure shows an attempt to enter 7 MHz to the XOSC 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.

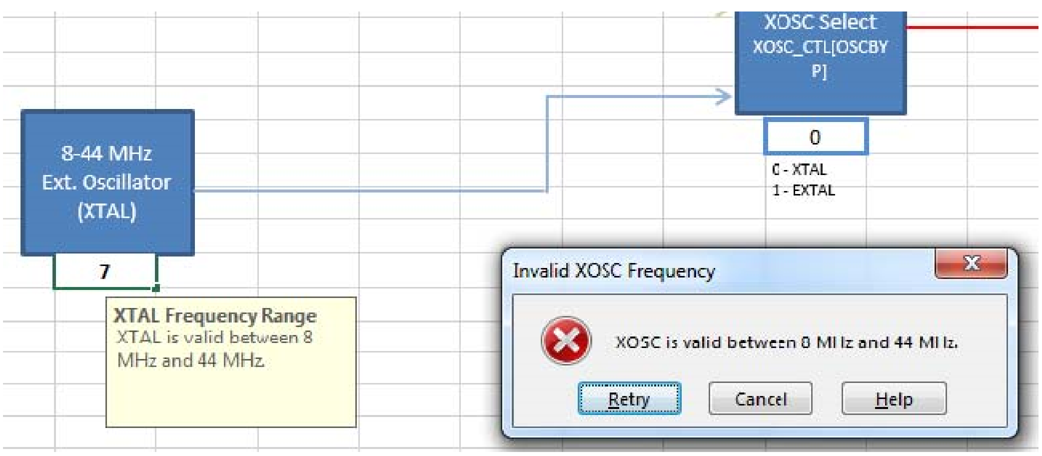


Figure 22. Invalid frequency input

Set the XOSC frequency back to 40 MHz. Set the value of the *XOSC Select* block to 0 to select the XTAL, the external oscillator, as shown in the following figure.

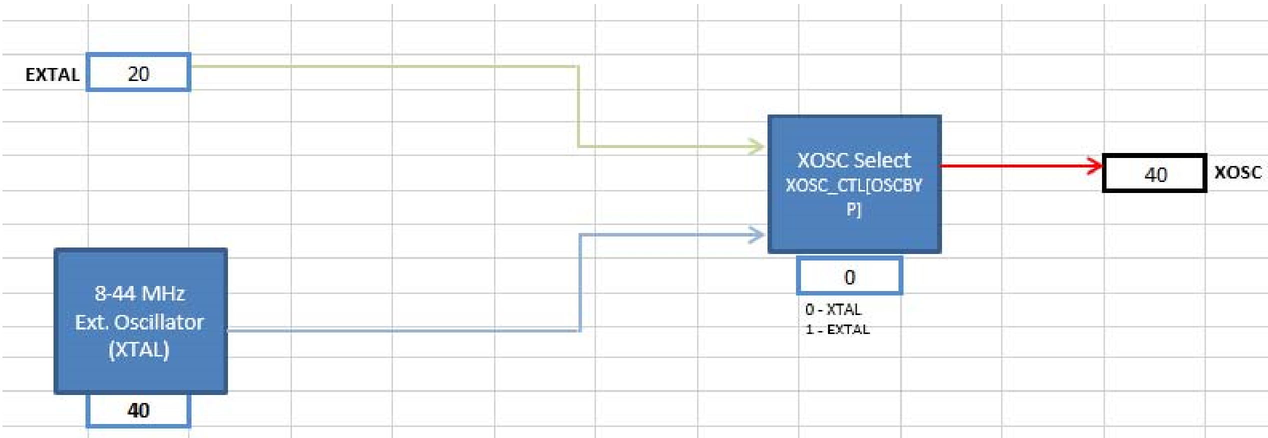


Figure 23. Oscillator configuration

Return to *Tree*. Trace forward from the XOSC block to *XOSC Source Controller*. The value of *XOSC Source Controller* is 0, meaning that the XOSC is turned off. The following figure circles the blocks that represent the XOSC crystal, the XOSC controller, and the effective frequency as sensed by *AUX Clock Selector 4* and *CMU_0*.

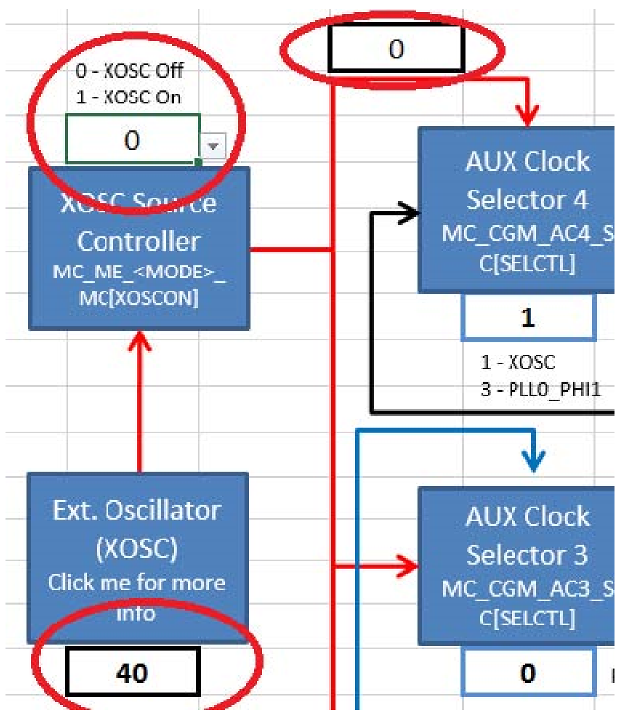


Figure 24. Actual XOSC frequency with source turned off

Switch the *XOSC Source Controller* value to 1 to turn on the XOSC. The output XOSC frequency is now 40 MHz, as shown in the following figure.

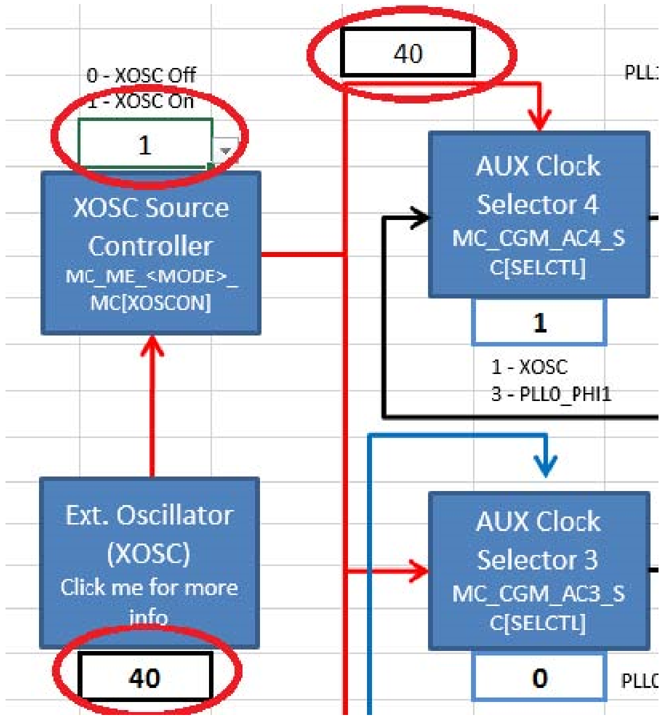


Figure 25. Figure 19. Actual XOSC frequency with source turned on

3.1.2 Configure PLL0

Follow the XOSC path to *AUX Clock Selector 3*. Change the *AUX Clock Selector 3* value to 1, so that PLL0 sources from XOSC, as shown in the following figure.

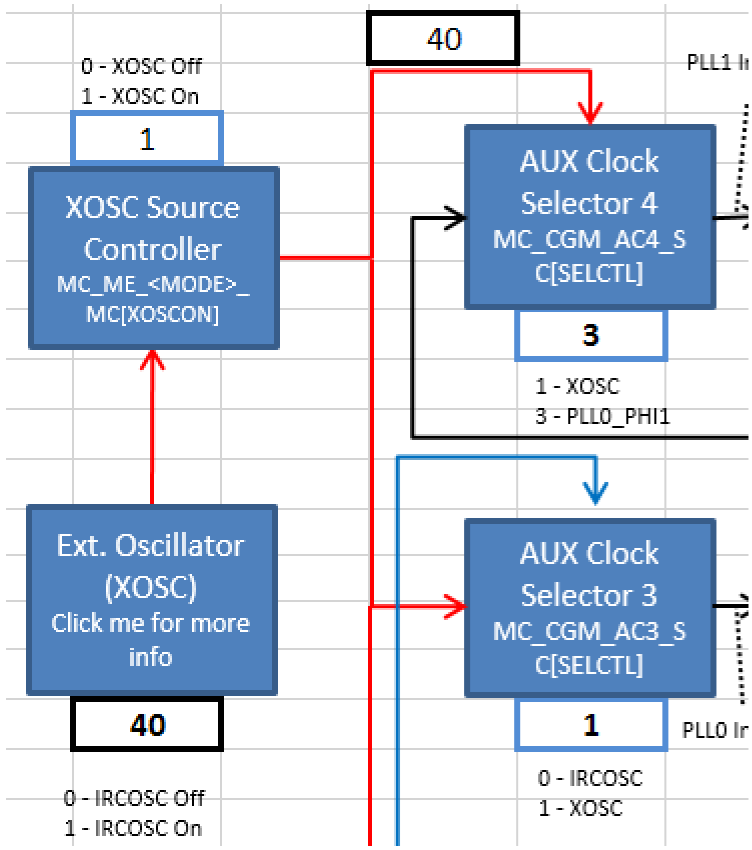


Figure 26. PLL0 source to XOSC

Next, configure PLL0. Click on the *PLL0* block to forward automatically to the *PLL0* tab. This is the tab that sets up the *PLL0_PHI* frequency. The *PLL0 Input* block of the figure below shows that PLL0 detects the 40 MHz XOSC as its source frequency.

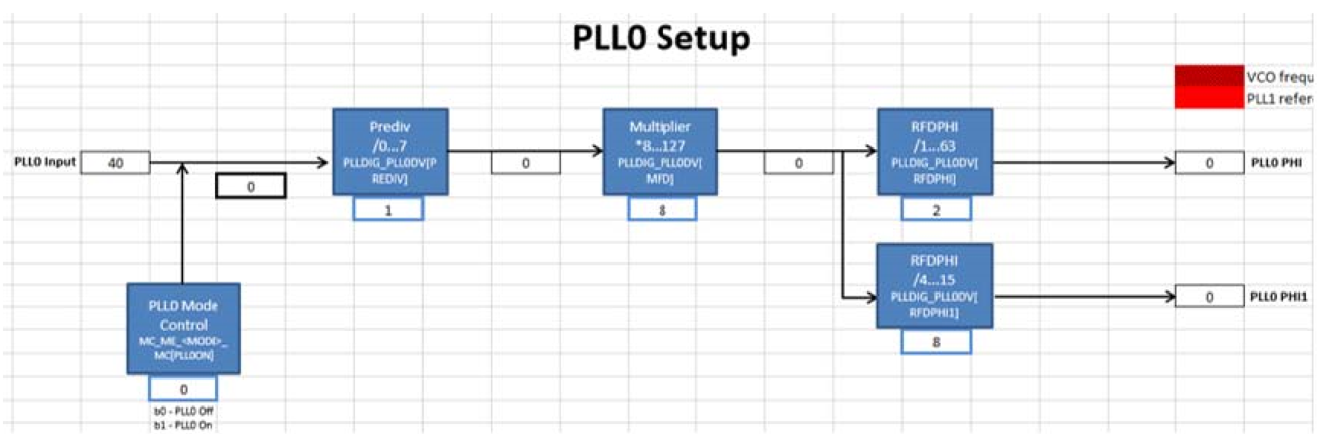
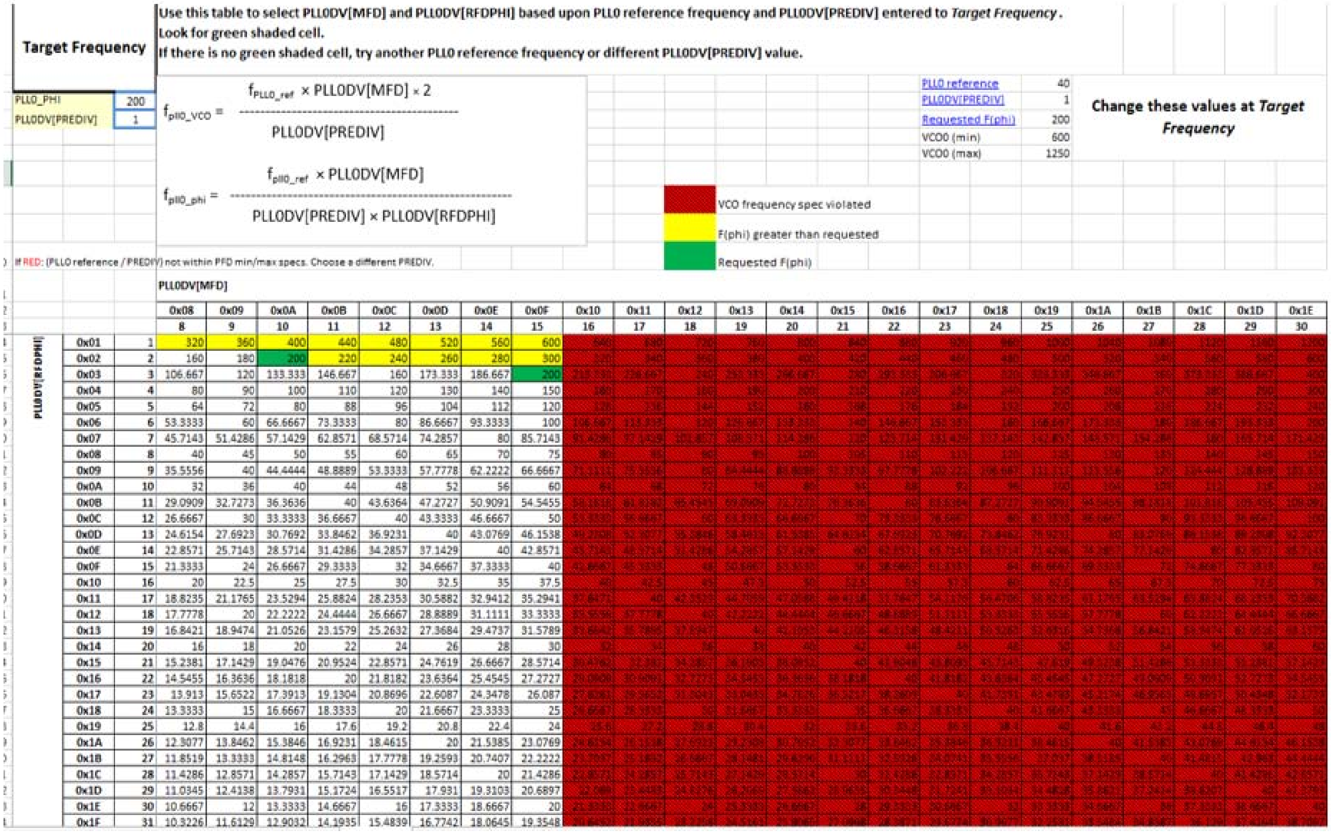


Figure 27. PLL0 calculator

Configure the dividers to achieve 160 MHz. The correct configuration can be achieved by trial and error, but the MPC574xP clock calculator provides a lookup table in the *pll0_phi* tab, as shown in the following figure.

Clock tool example use case: configure ADC to PLL0 at 80 MHz



PLL0DV[RFDPHI]		0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F
		8	9	10	11	12	13	14	15
0x01	1	320	360	400	440	480	520	560	600
0x02	2	160	180	200	220	240	260	280	300
0x03	3	106.667	120	133.333	146.667	160	173.333	186.667	200
0x04	4	80	90	100	110	120	130	140	150
0x05	5	64	72	80	88	96	104	112	120
0x06	6	53.3333	60	66.6667	73.3333	80	86.6667	93.3333	100
0x07	7	45.7143	51.4286	57.1429	62.8571	68.5714	74.2857	80	85.7143
0x08	8	40	45	50	55	60	65	70	75
0x09	9	35.5556	40	44.4444	48.8889	53.3333	57.7778	62.2222	66.6667
0x0A	10	32	36	40	44	48	52	56	60
0x0B	11	29.0909	32.7273	36.3636	40	43.6364	47.2727	50.9091	54.5455
0x0C	12	26.6667	30	33.3333	36.6667	40	43.3333	46.6667	50
0x0D	13	24.6154	27.6923	30.7692	33.8462	36.9231	40	43.0769	46.1538
0x0E	14	22.8571	25.7143	28.5714	31.4286	34.2857	37.1429	40	42.8571
0x0F	15	21.3333	24	26.6667	29.3333	32	34.6667	37.3333	40
0x10	16	20	22.5	25	27.5	30	32.5	35	37.5
0x11	17	18.8235	21.1765	23.5294	25.8824	28.2353	30.5882	32.9412	35.2941
0x12	18	17.7778	20	22.2222	24.4444	26.6667	28.8889	31.1111	33.3333
0x13	19	16.8421	18.9474	21.0526	23.1579	25.2632	27.3684	29.4737	31.5789
0x14	20	16	18	20	22	24	26	28	30
0x15	21	15.2381	17.1429	19.0476	20.9524	22.8571	24.7619	26.6667	28.5714
0x16	22	14.5455	16.3636	18.1818	20	21.8182	23.6364	25.4545	27.2727
0x17	23	13.913	15.6522	17.3913	19.1304	20.8696	22.6087	24.3478	26.087
0x18	24	13.3333	15	16.6667	18.3333	20	21.6667	23.3333	25
0x19	25	12.8	14.4	16	17.6	19.2	20.8	22.4	24
0x1A	26	12.3077	13.8462	15.3846	16.9231	18.4615	20	21.5385	23.0769
0x1B	27	11.8519	13.3333	14.8148	16.2963	17.7778	19.2593	20.7407	22.2222
0x1C	28	11.4286	12.8571	14.2857	15.7143	17.1429	18.5714	20	21.4286
0x1D	29	11.0345	12.4138	13.7931	15.1724	16.5517	17.931	19.3103	20.6897
0x1E	30	10.6667	12	13.3333	14.6667	16	17.3333	18.6667	20
0x1F	31	10.3226	11.6129	12.9032	14.1935	15.4839	16.7742	18.0645	19.3548

Figure 28. ***PLL0_PHI*** **reference table**

The *PLL0 reference* field is the frequency of the PLL0 input, in this case the 40 MHz XOSC. Set the target frequency and PREDIV values. This example will target 160 MHz and change PREDIV to 2. 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 are the modified settings.

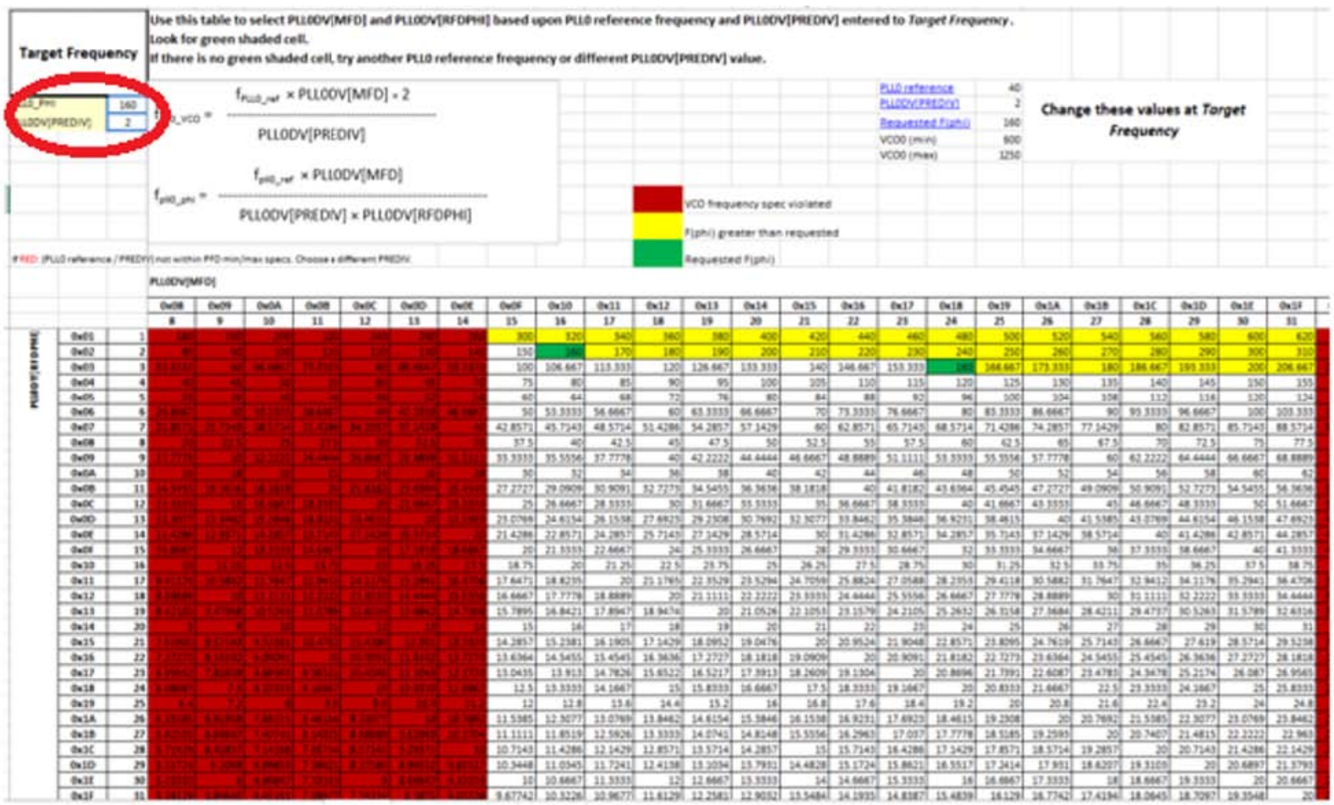


Figure 29. *PLL0_PHI* table with new settings

The cells shaded green means there are two divider combinations that can achieve exactly 160 MHz given an input frequency of 40 MHz and a PREDIV of 2. This example uses a MFD of 16 and a RFD of 2, but before configuring the *PLL0* tab, it is worth noting what happens if the output PLL frequency is out of range.

In the following figure, the PLL has been configured so that the output frequency is 5.08 GHz. This obviously exceeds the maximum hardware spec of 625 MHz. The associated voltage controlled oscillator (VCO) frequency, which can be back-calculated from *PLL0_PHI* also exceeds the maximum VCO spec of 1250 MHz. Therefore, the output is crosshatched and shaded red.

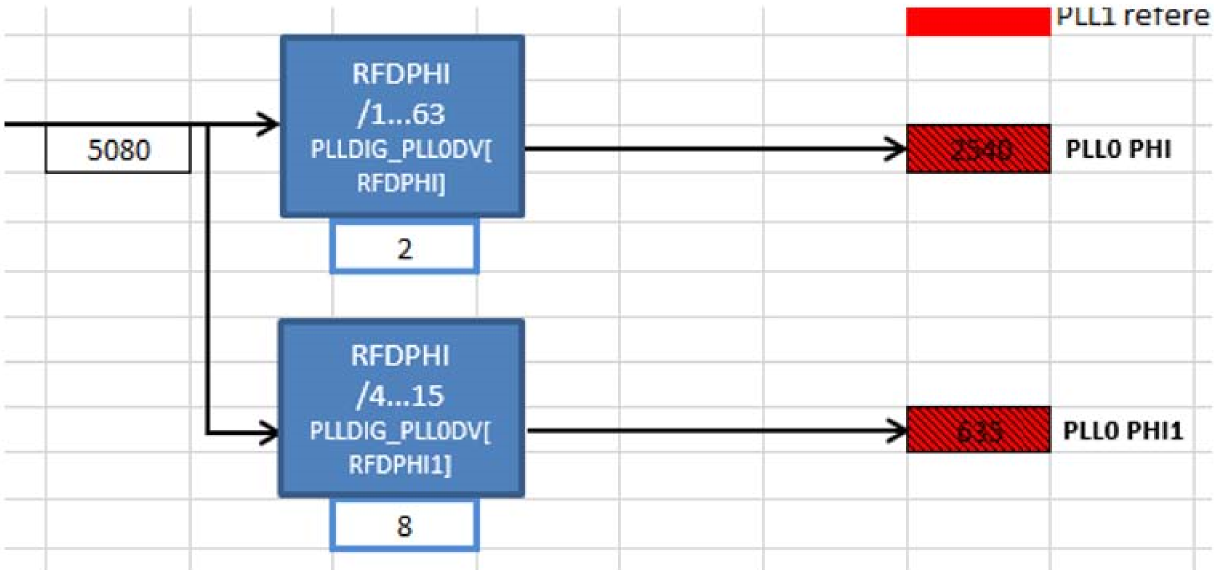


Figure 30. When *PLL0_PHI* exceeds VCO and PLL spec

Now, let's configure the PLL correctly. Turn on the PLL in the *PLL0* tab by setting the *PLL0 Mode Control block* to 1, set *Prediv* to 2, *Multiplier* to 16, and *RFDPHI* to 2. As shown in the next figure, the output *PLL0_PHI* is 160 MHz and the cell remains unshaded, meaning the configuration fits within spec.

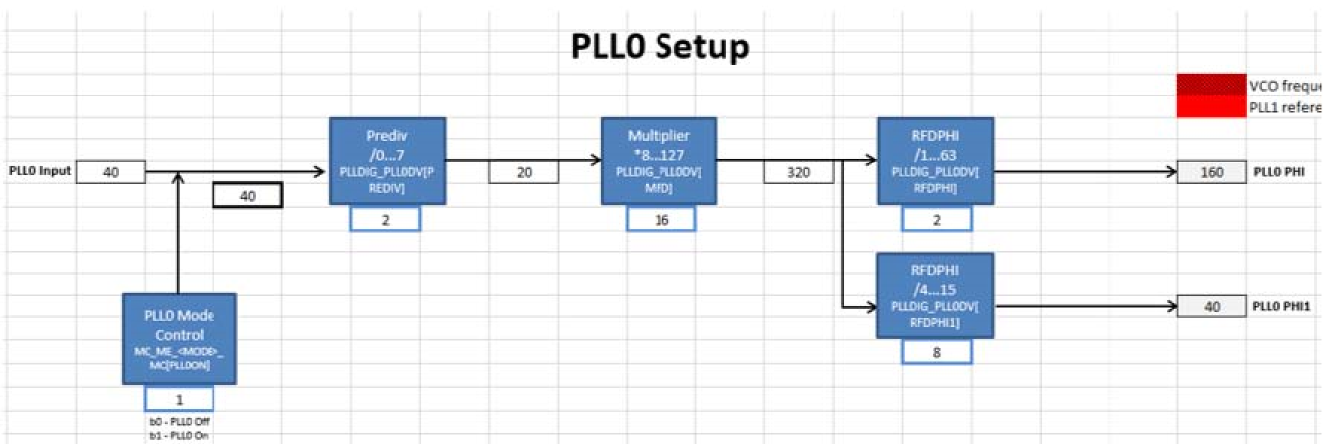


Figure 31. *PLL0_PHI* configured to 160 MHz

Go back to *Tree* to observe that the *PLL0_PHI* frequency is now 160 MHz.

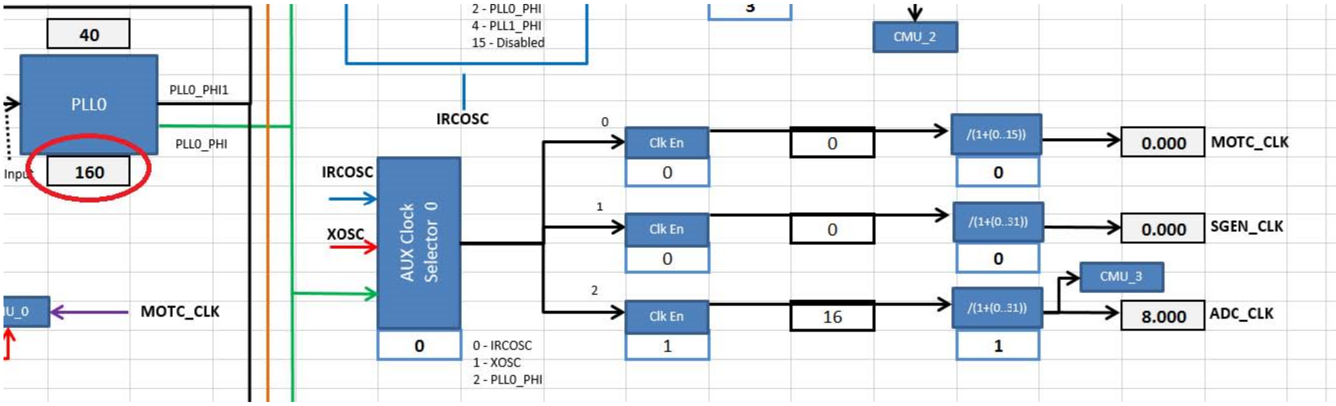


Figure 32. ***PLL0_PHI*** **propagated to Tree**

3.1.3 Finish setting ADC_CLK

Next, follow the *PLL0_PHI* signal down to *AUX Clock Selector 0*. IRCOSC is the current source of *MOTC_CLK*, *SGEN_CLK*, and *ADC_CLK*. Change the value of *AUX Clock Selector 0* to *2* to follow *PLL0_PHI*. After this, make sure the associated *Clk En* block is 1 and set the *ADC_CLK* divider, if necessary. The small number to the left of the divider block shows the divider number associated with that clock. Since a “2” is present next to the *ADC_CLK* divider, the *ADC_CLK* is configured by Divider 2 of Auxiliary Clock 0. The user input for the divider field is not the desired divider, but the bitfield value that one would have to enter to achieve the desired divider. That is why the divider block says "/ (1+(0...31)) rather than simply "/1...32". The user provides a value between 0 and 31, to which the hardware automatically adds 1 to calculate a divider that is between 1 and 32. In this case, the divide block is already set to 1: 160 MHz divided by (1+1) results in an *ADC_CLK* of 80 MHz. See the following figure.

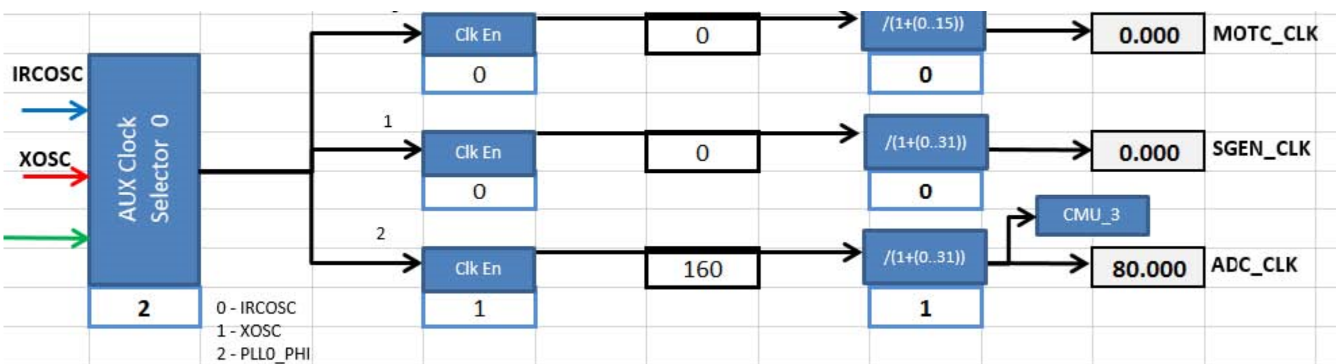


Figure 33. ***ADC_CLK*** **at 80 MHz PLL0**

If, for example, the *ADC_CLK* divider is 1, *ADC_CLK* would be 160 MHz, which would exceed the maximum allowable *ADC_CLK* frequency of 80 MHz. The tool will highlight the *ADC_CLK* cell red to signify that such a frequency is not allowed, as shown in the following figure.

Clock tool example use case: configure ADC to PLL0 at 80 MHz

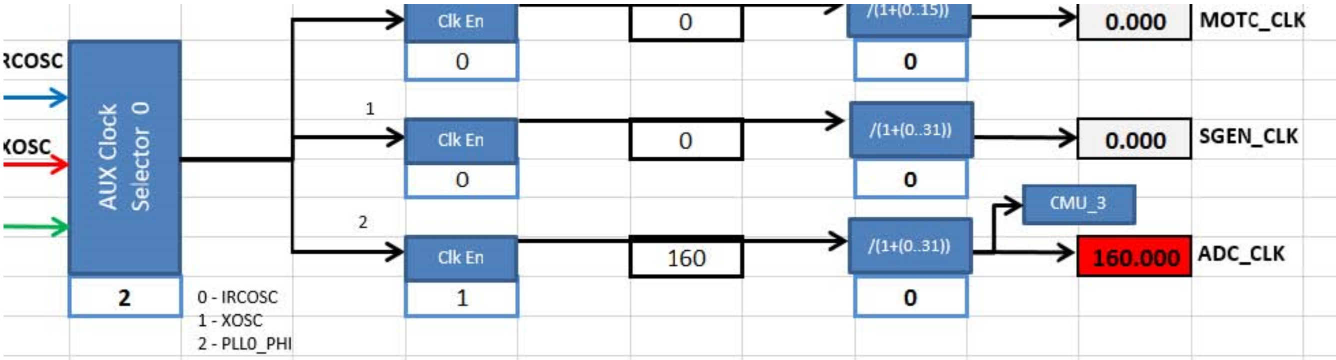


Figure 34. ***ADC_CLK*** **when frequency exceeds spec**

3.2 Configure ADC bus clock to 40 MHz PLL

The ADC's converter clock has been configured, but the ADC uses *PBRIDGEx_CLK* for its bus interface. Follow the same steps as *ADC_CLK* to trace *PBRIDGEx_CLK* back to the oscillators. Circled in the next figure is the location of *PBRIDGEx_CLK* in *Tree*.

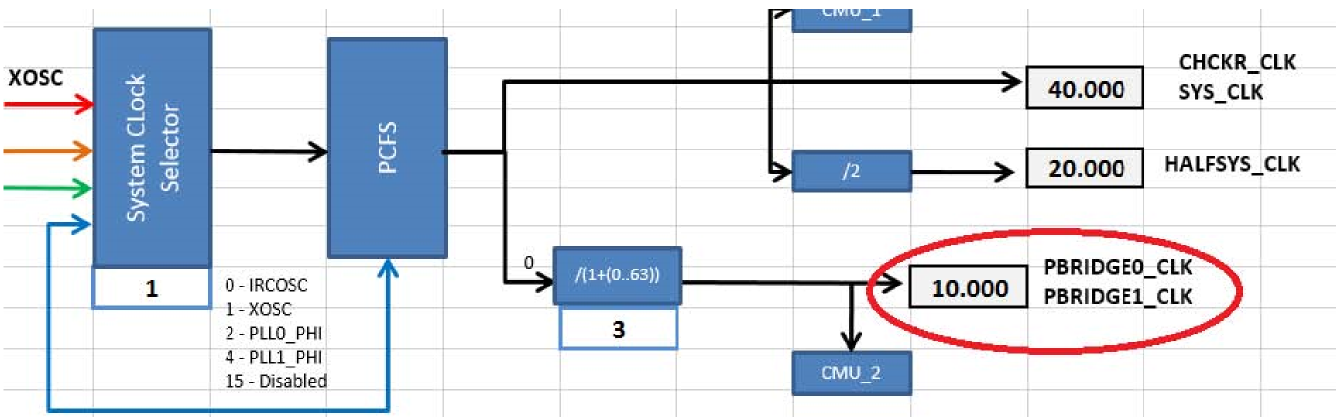


Figure 35. PBRIDGEx_CLK in Tree

XOSC and *PLL0_PHI* are already configured from the previous section, so there is no need to repeat those steps. *PBRIDGEx_CLK* traces back to the *System Clock Selector*, which currently follows XOSC. Change the *System Clock Selector* to follow *PLL0_PHI*.

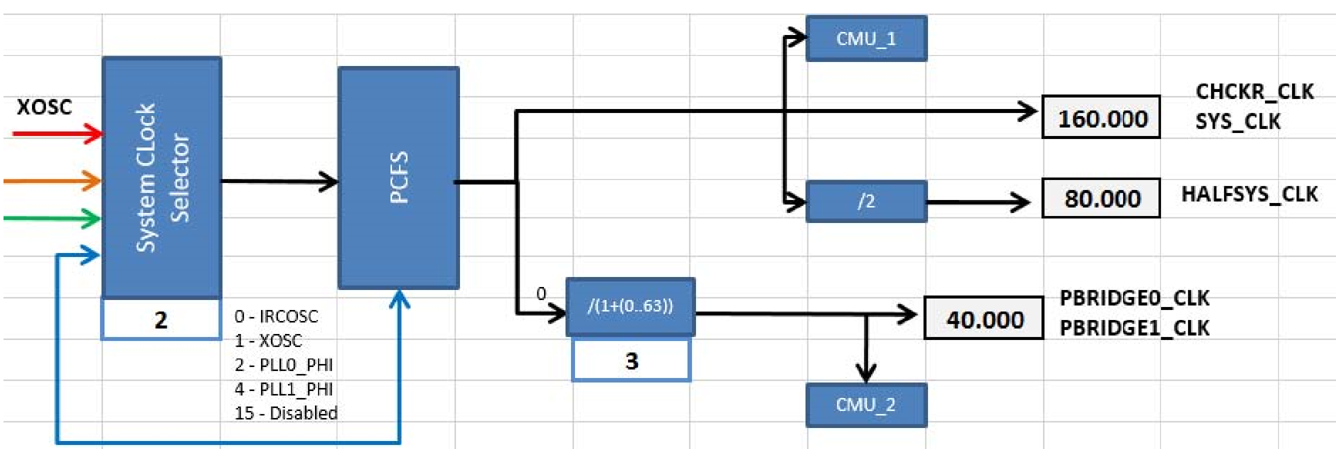


Figure 36. PBRIDGEx_CLK changed to follow PLL0_PHI

The *PCFS* block stands for *Progressive Clock Frequency Switch*. This is a feature supported in the MPC5744P to smooth the transition of the system clock from one clock source to another. The block here is just a visual representation for the user to know that the system clock filters through the progressive clock switch before propagating to the various system clock domains. *PCFS* takes IRCOSC in this diagram because its logic is organized in terms of IRCOSC cycle. You can find more information on the MPC5744P progressive clock switch in the application note "AN5304 – Initializing the MPC574xP Clock Generation Module and Progressive Clock Switching Feature".

Next set the *PBRIDGEx_CLK* divide value to 3: 160 MHz/(3+1) = 40 MHz. So, in closing, this example has achieved its goal: a 40 MHz XOSC driving a PLL that produces an output of 160 MHz, and from there the PLL running the *PBRIDGEx_CLK* at 40 MHz and the *ADC_CLK* at 80 MHz. Finally, the *PBRIDGEx_CLK* and *ADC_CLK* drive the ADC module.

3.3 Observe the registers

The final register summary table, as displayed in Summary, is shown in the following figure. 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 MC_CGM_AC11_DC0 would not have to be included, since Auxiliary Clock 11 was untouched. Registers that would have to be written would be ones like PLLDIG_PLL0DV and MC_CGM_AC0_DC2.

MPC574x

Register Summary

Register	Value
MC_ME_<MODE>_MC	0xX0XX0072
XOSC_CTL	0b0XX00000XXXXXXXXX0000000X0000000
MC_CGM_SC_DC0	0x80030000
MC_CGM_AC0_SC	0x02000000
MC_CGM_AC0_DC0	0x00000000
MC_CGM_AC0_DC1	0x00000000
MC_CGM_AC0_DC2	0x80010000
MC_CGM_AC1_DC0	0x00000000
MC_CGM_AC1_DC1	0x00000000
MC_CGM_AC2_DC0	0x00000000
MC_CGM_AC3_SC	0x01000000
MC_CGM_AC4_SC	0x01000000
MC_CGM_AC5_SC	0x01000000
MC_CGM_AC5_DC0	0x00000000
MC_CGM_AC6_SC	0x00000000
MC_CGM_AC6_DC0	0x00000000
MC_CGM_AC10_SC	0x00000000
MC_CGM_AC10_DC0	0x00000000
MC_CGM_AC11_SC	0x00000000
MC_CGM_AC11_DC0	0x00000000
PLLDIG_PLL0DV	0x40022010
PLLDIG_PLL1DV	0x00020014
PLLDIG_PLL1FD	0x000X0000
LFAST_PLLCR	0bXXX00000000000XXXXX00XX001011100
LFAST_SCR	0x000X0000
LFAST_COCR	0bXXXXXXXX00000000000000000000XXX0
LFAST_MCR	0xXX00X0XX

Figure 37. Register summary after configuration

3.4 Copy the code

SysClk_Init and *InitPeriClkGen* provide 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.

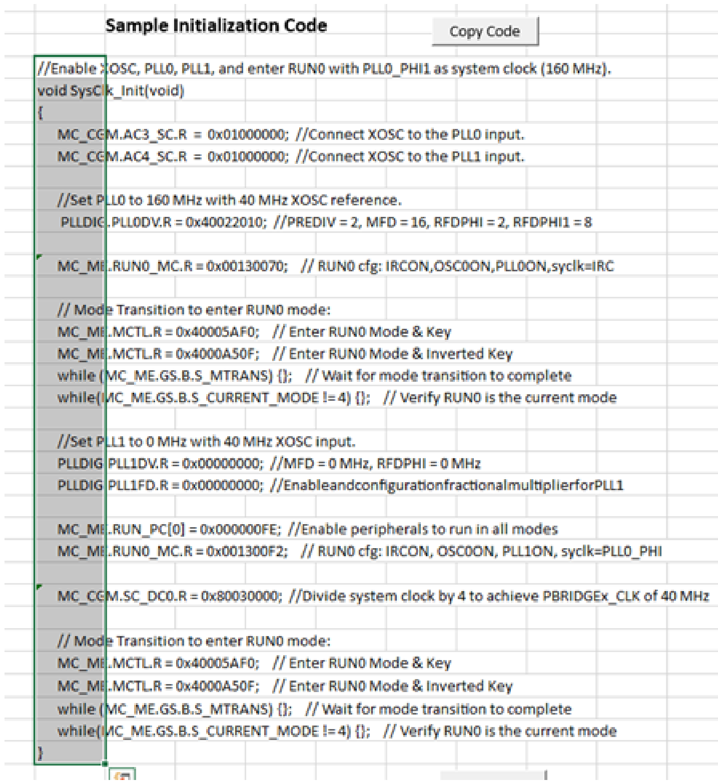


Figure 38. SysClk_Init after example

Therefore, to summarize, this example has achieved its goal: an ADC whose bus interface clock is driven by a PLL-sourced *PBRIDGEx_CLK* at 40 MHz. The 40 MHz *PBRIDGEx_CLK* is divided down from a 160 MHz PLL output; and the PLL output in turn is driven by the 40 MHz external oscillator. And finally, the ADC's engine clock is driven by a 80 MHz PLLsourced *ADC_CLK*.

4 Conclusion

This application note gives an overview of the MPC5744P interactive clock calculator. It seeks to simplify clock configurations 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 MPC574xG and S32K14x. Visit nxp.com to find more of these tools.

5 Revision history

Rev. No.	Date	Substantive Change(s)
0	January 2017	Initial version
1	February 2017	Updated the clock divider scheme and corrections to errors.

Table continues on the next page...

Table continued from the previous page...

Rev. No.	Date	Substantive Change(s)
2	May 2017	1. Updated the following section: • Summary on page 10 2. Added the following new section: • LFAST clocking on page 7 3. Replaced the following image: • Register summary after configuration 4. Updated the document from the editorial perspective.
3	May 2017	1. Updated the following sections: • Introduction on page 1 • Summary on page 10 • Tree on page 3 2. Added the following new section: • Copy the code on page 26
4	July 2017	Editorial updates.
5	July 2017	1. Editorial updates. 2. Updated Finding the tools 3. Updated PLL0 source to XOSC
6	August 2017	1. Updated the MPC574xP_Clock_Calculator_Rev4. 2. Removed the figure Finding tools. 3. Updated the Introduction on page 1 section.
7	October 2017	Updated the associated MPC574xP_Clock_Calculator file.
8	February 2018	Updated the associated MPC574xP_Clock_Calculator file.
9	April 2018	Updated the associated MPC574xP_Clock_Calculator file.

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Document Number: AN5393
Rev. 9, April 2018