

Initializing the MPC574xP Clock Generation Module and Progressive Clock Switching Feature

by: NXP Semiconductors

Contents

1 Introduction

The MPC574xP is a Power Architecture®-based MCU for Automotive and Industrial Applications.

This document details the MPC574xP specific PLL and clock divider settings to achieve 200 MHz Core. In addition, the document describes the Progressive Clock Switching feature, which supports a smooth ramp-up and ramp-down of device system clocks, for a 200 MHz clock example. The calculations for this example can be used for any source clock application by simply changing the inputs to the formulas.

2 Clock initialization

At Power on Reset the MCU is clocked from the on chip 16 MHz Internal RC Oscillator (IRCOSC). This section describes how to configure the clock related modules to run the MCU from the configurable and higher speed PLL's. It will also cover the setup of the clock trees to distribute and divide the clock sources to the buses and peripherals on the MCU.

1	Introduction.....	1
2	Clock initialization.....	1
2.1	Clock tree	1
3	Introduction to Progressive Clock Switching (PCS)	3
3.1	PCS operation.....	3
3.2	PCS control of current.....	4
3.3	PCS registers.....	4
3.4	PCS calculation example.....	5
3.5	PCS calculator.....	6
3.6	PCS sample oscilloscope plots....	10
4	Appendix A.....	11
4.1	C code listing.....	11
5	Revision history.....	14



The following diagram shows the clock tree configuration for the MPC574xP. In the software example used in this application note, PLL0 is configured to output a 160 MHz clock and PLL1 is configured to output a 200 MHz clock. Software configures both PLL0 and PLL1 to use the external oscillator (40 MHz) as an input.

Note: All dividers shown in the diagram (FCD not included) are integer dividers with a range of 1, 2, 3,..., n. All clock dividers are 50% duty cycle.

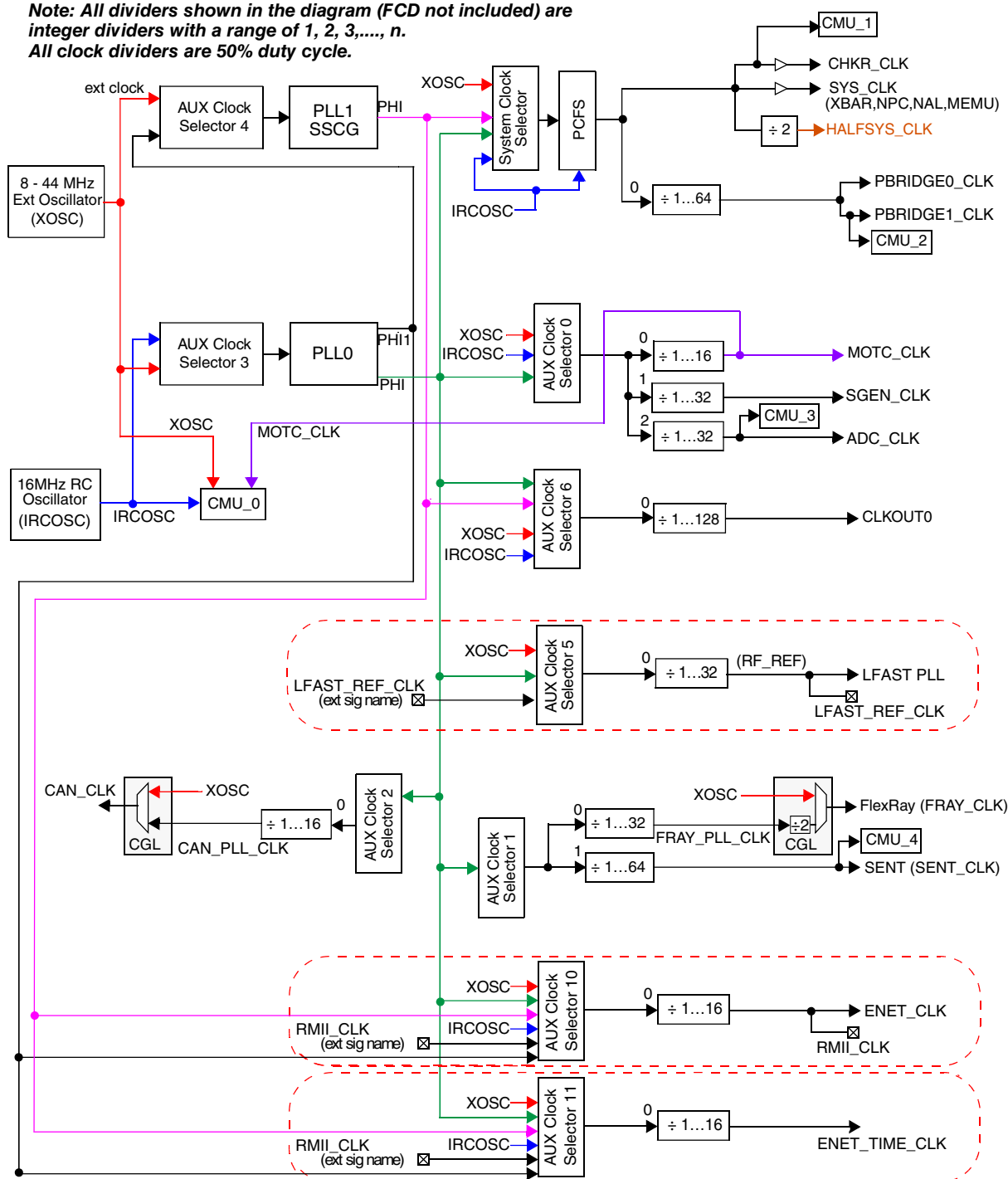


Figure 1. Block diagram of MPC574xP clock tree

A summary of the clock tree settings that are configured in the example software is shown in the following table.

Table 1. Example of clock settings

Clock(s)	Aux Selector and Divider	Source Clock	Divide/Multiply Factor	Frequency
XOSC	N/A	N/A	N/A	40 MHz
PLL0	Aux 3	XOSC	4	160 MHz
PLL1	Aux 4	XOSC	5	200 MHz
CHKR_CORE SYS_CLOCK	System	PLL1	1	200 MHz
PBRIDGE_0_CLK PBRIDGE_1_CLK	System	PLL1	4	50 MHz
HALF_SYS_CLOCK	System	PLL1	2	100 MHz
MOTC_CLK	Aux 0-0	PLL0	1	160 MHz
SGEN_CLK	Aux 0-1	PLL0	8	20 MHz
ADC_CLK	Aux 0-2	PLL0	2	80 MHz
CLKOUT0	Aux 6-0	PLL1	10	20 MHz
LFAST_PLL	Aux 5-0	PLL0	16	10 MHz
FRAY_CLK	Aux 1-0	PLL0	2	80 MHz
SENT_CLK	Aux 1-1	PLL0	4	40 MHz
CAN_CLK	Aux 2-0	PLL0	4	40 MHz
ENET_CLK	Aux 10-0	PLL1	4	50 MHz
ENET_TIME_CLK	Aux 11-0	PLL1	4	50 MHz

3 Introduction to Progressive Clock Switching (PCS)

Changing device operating modes and/or clock frequency in the MPC574xP typically results in instantaneous changes in current (IDD). These changes in current can cause undesired fluctuations in power supply voltage, causing the supply voltage at the device to operate at a higher or lower voltage than desired and causing possible triggering of a Low Voltage Detect (LVD) or High Voltage Detect (HVD). The PCS feature supports smooth transitions for frequency and mode changes, greatly reducing voltage overshoot, undershoot, and undesired LVDs / HVDs due to rapidly changing power supply load.

3.1 PCS operation

PCS is a feature of the clock generation module (MC_CGM) and is triggered by the mode entry module (MC_ME). To change the system clock frequency, a mode entry change must occur by writing the MC_ME mode control register (MC_ME_MCTL). If the power level (PWRLVL field within the various mode configuration registers) is different between the current and target modes, the PCS mechanism is enabled causing the system clock frequency to ramp down and/or up in multiple steps. For example, consider two modes with two different PWRLVL settings in respective Mode Configuration Register: MC_ME_RUN0_MC[PWRLVL] = 1, and MC_ME_RUN1_MC[PWRLVL] = 2. If RUN0 is the current mode, and a mode change occurs to switch to mode RUN1, then PCS will be invoked since the PWRLVL settings are different between RUN0 and RUN1. PCS may also be activated by doing a mode entry change that remains in the same mode (e.g. DRUN -> DRUN) so long as the PWRLVL field is changed before the mode change.

In order to prevent sudden voltage drops and overshoots due to frequency and load changes, the MC_ME requests the MC_CGM to ramp the system clock frequency down and/or up based on the power level values of the current and target modes. During ramp-down, the rate of the frequency change is based on the PCS Switch Duration Register

(MC_CGM_PCS_SDUR), PCS Divider Change Rate (MC_CGM_PCS_DIVCn), and PCS Divider End Value (MC_CGM_PCS_DIVEn) registers, where n corresponds to the current system clock source selection. During ramp-up, the rate of the frequency change is based on the MC_CGM_PCS_SDUR, MC_CGM_PCS_DIVCn, and PCS Divider Start Value (MC_CGM_PCS_DIVSn) registers, where n corresponds to the target system clock source selection.

3.2 PCS control of current

The following paragraphs describe the control of major components (CPU Cores, Cache, Peripherals). Components consuming larger amounts of power such as CPUs and cache, should be turned on at lower clock frequencies as detailed below.

Initializing Core e200z CPU – This can either be done at the low frequency (before ramp up or after ramp down) or it can be done during the same mode entry change along with PCS. The NXP example initialization software included in this application note contains CPU initialization as part of the same mode entry change where System Clock Dividers, Auxiliary Clock Dividers, PLL0, PLL1, and PCS are configured. The Mode Entry (MC_ME) technology manages all clock dividers, PCS, and CPU core initialization for the user. The MC_ME module, turns on the Core and Checker Core at the lower PCS frequency and then progressively switches clocks faster and faster, thereby minimizing any sharp changes in current consumption.

Enabling cache – Do at low frequencies before ramp up. Caches should be enabled as part of the start_core<n>.s code which executes during CPU core initialization.

Enabling Peripherals – Device peripherals consume a smaller fraction of overall device current. Peripherals can be initialized after the PCS completes when system clocks and peripheral clocks are running at their full application speed.

3.3 PCS registers

PCS is achieved by programming registers in the Clock Generation Module (MC_CGM). The user programs values for initial divider change, divider start (for ramp-up from slow to fast), divider end (for ramp-down from fast clock to slow clock), and the switch RATE. The following table lists the required configuration steps and associated registers for PCS.

Table 2. PCS registers

Step	Procedure description	Register : Comments
1	Configure Switch Duration	MC_CGM.PCS_SDUR: Defines the duration of one system clock switch step. Switch Duration = Number of 16 MHz clocks * SDUR * k ¹ steps
2	Configure DRUN power level	MC_ME.DRUN_MC[PWRLVL] : Power level must be set to different value than the power level setting for the mode currently in operation
3	Set the Divider Change Initial Value	MC_CGM.PCS_DIVCn.B.INIT : Defines the initial clock divider value.
4	Set the Divider Change Rate	MC_CGM.PCS_DIVCn.B.RATE : PCS Rate. See details in the PCS calculation example in this application note
5	Set the Divider Start Value (for clock ramp-up from slow to fast)	MC_CGM.PCS_DIVSn : Clock ramp-up start divider value. DIVSn is used for switching from slow clock to fast clock.
6	Set the Divider End Value (for clock ramp-down from fast to slow)	MC_CGM_PCS_DIVEn : Clock ramp-down end divider value. DIVEn is used for switching from fast clock to slow clock.

1. See PCS calculation example in this application note

It is important that the MC_CGM.PCS_SDUR, MC_CGM.PCS_DIVCn, MC_CGM.DIVSn, and MC_CGM.PCS_DIVEn registers are programmed while the IRCOSC is still selected as the system clock source. Once these registers have been programmed and the system clock source is operating off any clock other than the IRCOSC if there is any reset issued other than a POR the MPC574xP will automatically trigger a PCS sequence as it transitions to the IRCOSC as part of the reset sequence. So if the system clock frequency was PLL1_PHI and these PCS registers were being programmed for a future PCS sequence and a reset occurred before the register writes finished the MPC574xP would attempt a PCS sequence with possible invalid values in these registers which could result in the part getting stuck in reset.

3.4 PCS calculation example

The application designer can determine the register settings required for PCS, by first defining the following system design parameters:

- Normal Operating frequency (f_{src})
- Low Current Operating frequency, typically 16 MHz (f_{tgt})
- Maximum allowable IDD change in mA (assume 1 μ s PCS step duration)

To determine the maximum frequency change allowed, the change in current due to the change in device operating frequency must be known. For the MPC574xP, bench testing shows the device dynamic IDD change is 1.39 mA/MHz. Using this number, we can determine the maximum frequency change allowed, f_{chg} , using the following equation:

$$f_{chg} = \frac{\text{max allowable IDD change}}{\text{device dynamic IDD change}}$$

$$f_{chg} = \frac{50 \text{ mA}}{1.39 \text{ mA/MHz}}$$

$$f_{chg} = 36.0$$

where max allowable IDD change is determined by the user's power supply design.

The example below shows the steps required to calculate the needed register settings based on the Reference Manual formulas:

Given:

Normal operating frequency (f_{src}) = 200 MHz

Low current operating frequency (f_{tgt}) = 16 MHz

Maximum allowable IDD change in mA (assume 1 μ s PCS step duration) = 50 mA

Maximum frequency change (f_{chg}) = 36.0. MHz (calculated in steps above)

It is first necessary to determine the rate of frequency change. This is done by calculating:

$$a_{max} = \frac{f_{chg}}{f_{src}}$$

$$a_{max} = \frac{36.0 \text{ MHz}}{200 \text{ MHz}}$$

$$a_{max} = 0.18$$

$$a_{max} = 0.15 \text{ (rounded down to nearest 0.05)}$$

Next, based on $a_{max} = 0.15$, determine MC_CGM_PCS_DIVCn[RATE] (also called 'd') by looking up a_{max} in the MC_CGM_PCS_DIVCn[RATE] values table in the Reference Manual. From this table, we find the RATE is 0.112 for $a_{max} = 0.15$.

$$\text{RATE} = d = 0.112$$

Introduction to Progressive Clock Switching (PCS)

To calculate k, number of steps, use the following equation:

$$k = 0.5 + \sqrt{0.25 - \frac{2 \left(1 - \frac{f_{src}}{f_{tgt}}\right)}{d}}$$

$$k = 0.5 + \sqrt{0.25 - \frac{2 \left(1 - \frac{200 \text{ MHz}}{16 \text{ MHz}}\right)}{0.112}}$$

$$k = 0.5 + 14.3$$

$$k = 14.8 \text{ steps}$$

Given the number of steps and the Switch Duration Register (SDUR) setting, the user can calculate the duration, t_{pcs} , of the PCS switching. In this formula, we shall round k up to the nearest whole number. 14.8 rounds up to the next nearest whole number of 15.

$$t_{pcs} = \frac{1}{16 \text{ MHz}} \times \text{SDUR} \times k$$

$$t_{pcs} = \frac{1}{16 \text{ MHz}} \times 16 \times 15$$

$$t_{pcs} = 15 \mu\text{s}$$

At this point, we can calculate the initial divider increment change value, MC_CGM_PCS_DIVCn[INIT], by using the following equation:

$$\text{INIT} = d \times k \times 1000$$

$$\text{INIT} = 112 \times 15$$

$$\text{INIT} = 1,680$$

Next, let's calculate the divider start value (DIVSn) that determines the divide value required for the start of PCS switching. The following equation is used:

$$\text{DIVSn} = \left(1 + d^{\frac{k(k+1)}{2}}\right) \times 1000 - 1$$

$$\text{DIVSn} = \left(1 + 0.112 \times \frac{15(15+1)}{2}\right) \times 1000 - 1$$

$$\text{DIVSn} = 14,439$$

Finally, to determine DIVEn, the following equation applies:

$$\text{DIVEn} = 1000 \times \frac{f_{src}}{16} - 1$$

$$\text{DIVEn} = 1000 \times \frac{200}{16} - 1$$

$$\text{DIVEn} = 12,499$$

As a result of these calculations, we now have the final list of values to be programmed into the PCS registers. To summarize:

MC_CGM.PCS_SDUR.R = 16;
MC_CGM.PCS_DIVCn.B.INIT = 1680;
MC_CGM.PCS_DIVCn.B.RATE = 112;
MC_CGM.PCS_DIVSn.R = 14439;
MC_CGM.PCS_DIVEEn.R = 12499;

3.5 PCS calculator

To ease the process of determining PCS register settings, NXP provides an Excel based calculator. The steps shown below detail how the user can enter their application parameters into the calculator and receive a set of register settings as a result. The following five diagrams illustrate each step in the process:

- Step 1: enter the source frequency (for example, 180 MHz)
- Step 2: enter IDD change per MHz for the MCU
- Step 3: enter max allowable IDD change for the application
- Step 4: round down the calculated a_{max} to the nearest 0.05
- Step 5: enter PCS step duration in microseconds

Register settings are in Bold text $f_{target} = 16$ MHz

Parameter / Register	Example 1	Example 2	User
source (MHz)	160	200	180
dynamic IDD change (mA/MHz)	1.39	1.39	
max allowable IDD change (mA/usec)	50.0	50.0	
max Frequency step (MHz)	36.0	36.0	#DIV/0!
calculated a_{max} (f_{chg} / f_{src})	0.225	0.180	#DIV/0!
rounded a_{max} (f_{chg} / f_{src}) ^[1]	0.2	0.15	
step duration (μ sec)	1	1	
k steps	11.00	15.00	#N/A
duration (μ sec)	11	15	#N/A
PCS_SDUR[SDUR]	16	16	0
PCS_DIVC1[RATE]	184	112	#N/A
PCS_DIVCn[INIT]	2024	1680	#N/A
PCS_DIVEn[DIVE]	9999	12499	11249
PCS_DIVSn[DIVS]	13143	14439	#N/A

Step 1: enter source frequency

Figure 2. Enter source frequency into the calculator

Register settings are in **Bold text**

$f_{\text{target}} = 16 \text{ MHz}$

Parameter / Register	Example 1	Example 2	User
source (MHz)	160	200	180
dynamic IDD change (mA/MHz)	1.39	1.39	1.39
max allowable IDD change (mA/usec)	50.0	50.0	
max Frequency step (MHz)	36.0	36.0	0.0
calculated $a_{\text{max}} (f_{\text{chg}} / f_{\text{src}})$	0.225	0.180	0.000
rounded $a_{\text{max}} (f_{\text{chg}} / f_{\text{src}})^{[1]}$	0.2	0.15	
step duration (μsec)	1	1	
k steps	11.00	15.00	#N/A
duration (μsec)	11	15	#N/A
PCS_SDUR[SDUR]	16	16	0
PCS_DIVC1[RATE]	184	112	#N/A
PCS_DIVCn[INIT]	2024	1680	#N/A
PCS_DIVEn[DIVE]	9999	12499	11249
PCS_DIVSn[DIVS]	13143	14439	#N/A

Step 2: enter IDD change per MHz

Figure 3. Enter IDD change per MHz for the MCU

Register settings are in **Bold text**

$f_{\text{target}} = 16 \text{ MHz}$

Parameter / Register	Example 1	Example 2	User
source (MHz)	160	200	180
dynamic IDD change (mA/MHz)	1.39	1.39	1.39
max allowable IDD change (mA/usec)	50.0	50.0	20.0
max Frequency step (MHz)	36.0	36.0	14.4
calculated $a_{\text{max}} (f_{\text{chg}} / f_{\text{src}})$	0.225	0.180	0.080
rounded $a_{\text{max}} (f_{\text{chg}} / f_{\text{src}})^{[1]}$	0.2	0.15	
step duration (μsec)	1	1	
k steps	11.00	15.00	#N/A
duration (μsec)	11	15	#N/A
PCS_SDUR[SDUR]	16	16	0
PCS_DIVC1[RATE]	184	112	#N/A
PCS_DIVCn[INIT]	2024	1680	#N/A
PCS_DIVEn[DIVE]	9999	12499	11249
PCS_DIVSn[DIVS]	13143	14439	#N/A

Step 3: enter max allowable IDD change

Figure 4. Enter maximum allowable IDD change

Register settings are in **Bold text** $f_{\text{target}} = 16$ MHz

Parameter / Register	Example 1	Example 2	User
source (MHz)	160	200	180
dynamic IDD change (mA/MHz)	1.39	1.39	1.39
max allowable IDD change (mA/usec)	50.0	50.0	20.0
max Frequency step (MHz)	36.0	36.0	14.4
calculated $a_{\text{max}} (f_{\text{chg}} / f_{\text{src}})$	0.225	0.180	0.080
rounded $a_{\text{max}} (f_{\text{chg}} / f_{\text{src}})^{[1]}$	0.2	0.15	0.05
step duration (μsec)	1	1	
k steps	11.00	15.00	42.00
duration (μsec)	11	15	0
PCS_SDUR[SDUR]	16	16	0
PCS_DIVC1[RATE]	184	112	12
PCS_DIVCn[INIT]	2024	1680	504
PCS_DIVEn[DIVE]	9999	12499	11249
PCS_DIVSn[DIVS]	13143	14439	11835

Step 4: Round down to nearest 0.05

Figure 5. Round down the result to nearest 0.05

Register settings are in **Bold text** $f_{\text{target}} = 16$ MHz

Parameter / Register	Example 1	Example 2	User
source (MHz)	160	200	180
dynamic IDD change (mA/MHz)	1.39	1.39	1.39
max allowable IDD change (mA/usec)	50.0	50.0	20.0
max Frequency step (MHz)	36.0	36.0	14.4
calculated $a_{\text{max}} (f_{\text{chg}} / f_{\text{src}})$	0.225	0.180	0.080
rounded $a_{\text{max}} (f_{\text{chg}} / f_{\text{src}})^{[1]}$	0.2	0.15	0.05
step duration (μsec)	1	1	5
k steps	11.00	15.00	42.00
duration (μsec)	11	15	210
PCS_SDUR[SDUR]	16	16	80
PCS_DIVC1[RATE]	184	112	12
PCS_DIVCn[INIT]	2024	1680	504
PCS_DIVEn[DIVE]	9999	12499	11249
PCS_DIVSn[DIVS]	13143	14439	11835

Step 5: enter PCS step duration

Figure 6. Enter PCS step duration (number of microseconds)

At the end of these steps, the PCS Calculator produces all register values.

3.6 PCS sample oscilloscope plots

The plots below show an MPC574xP device in a non-PCS operation and PCS enabled operation. IDD_LV current is initially around 60 mA running from 16 MHz IRCOSC. When the Mode Entry switch occurs, the system clock is switched to run at full speed using PLL1 at 200 MHz which brings the current up to approximately 300 mA.

In the following figure, PCS is not enabled. The current changes by approximately 240 mA in 34 microseconds and the mode switch is brief as indicated by a single high speed toggle of the port pin.

The oscilloscope channel definition is:

YELLOW ch1 – IDD_LV current at 100 mA/div.

BLUE ch2 – VDD_LV at 200 mV/div.

PURPLE ch3 – software controlled toggling pin. Toggle rate is proportional to system clock frequency and pin toggles only during the mode transition.

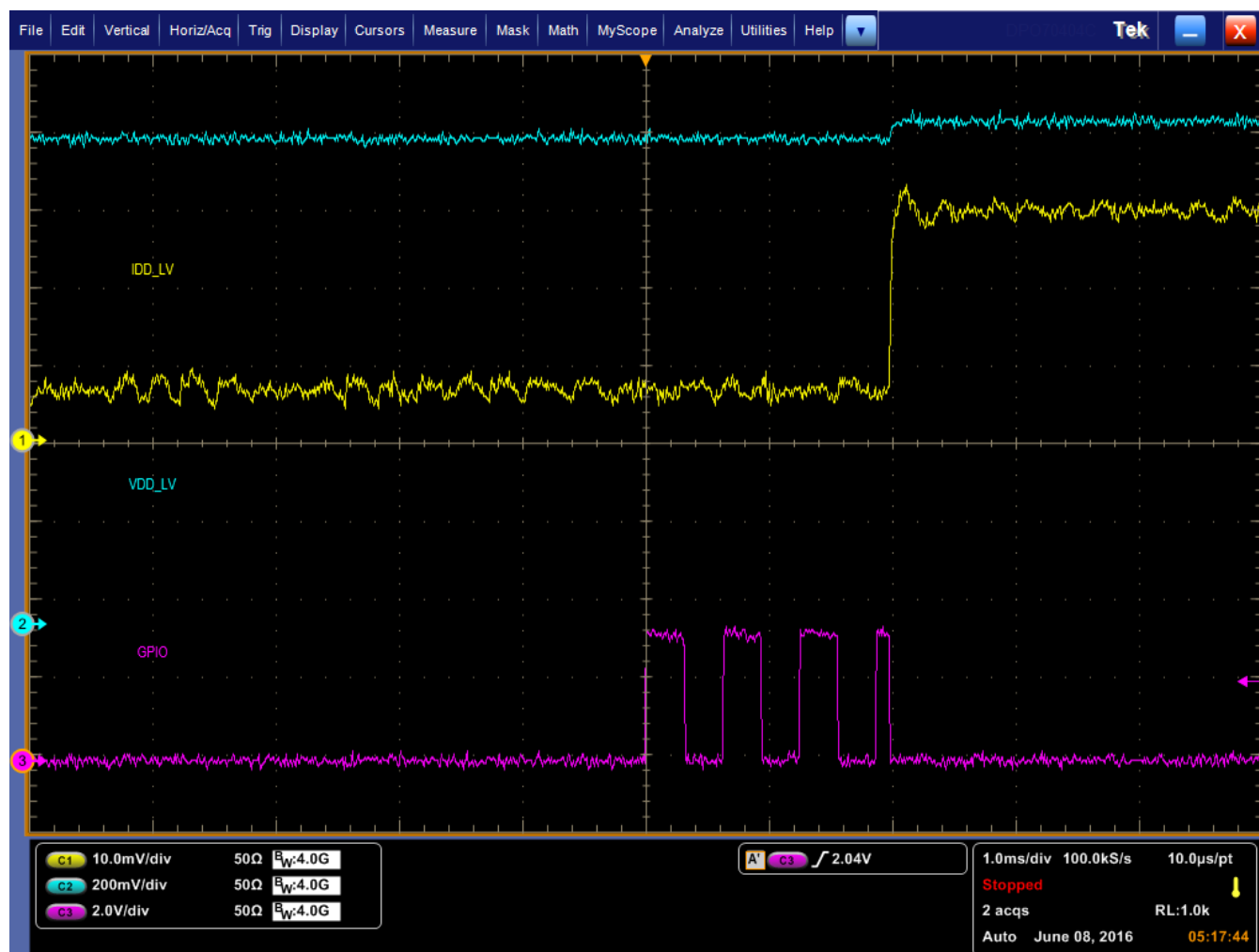


Figure 7. IDD_LV, VDD_LV, and I/O toggle rate with PCS disabled

In the next figure, PCS has been enabled with a rate equal to 0.15 as per the software example listed in this document. The MC_CGM.PCS_SDUR value was increased to 240 to make the change easier to see on the scope plots. PCS causes the system clock frequency to ramp up over a longer time period as can be seen by the increasing toggling pin rate. Now the current ramp time is around 225 μ s.

YELLOW ch1 – IDD_LV current at 100 mA/div.

BLUE ch2 – VDD_LV at 200 mV/div.

PURPLE ch3 – software controlled toggling pin. Toggle rate is proportional to system clock frequency and pin toggles only during the mode transition.

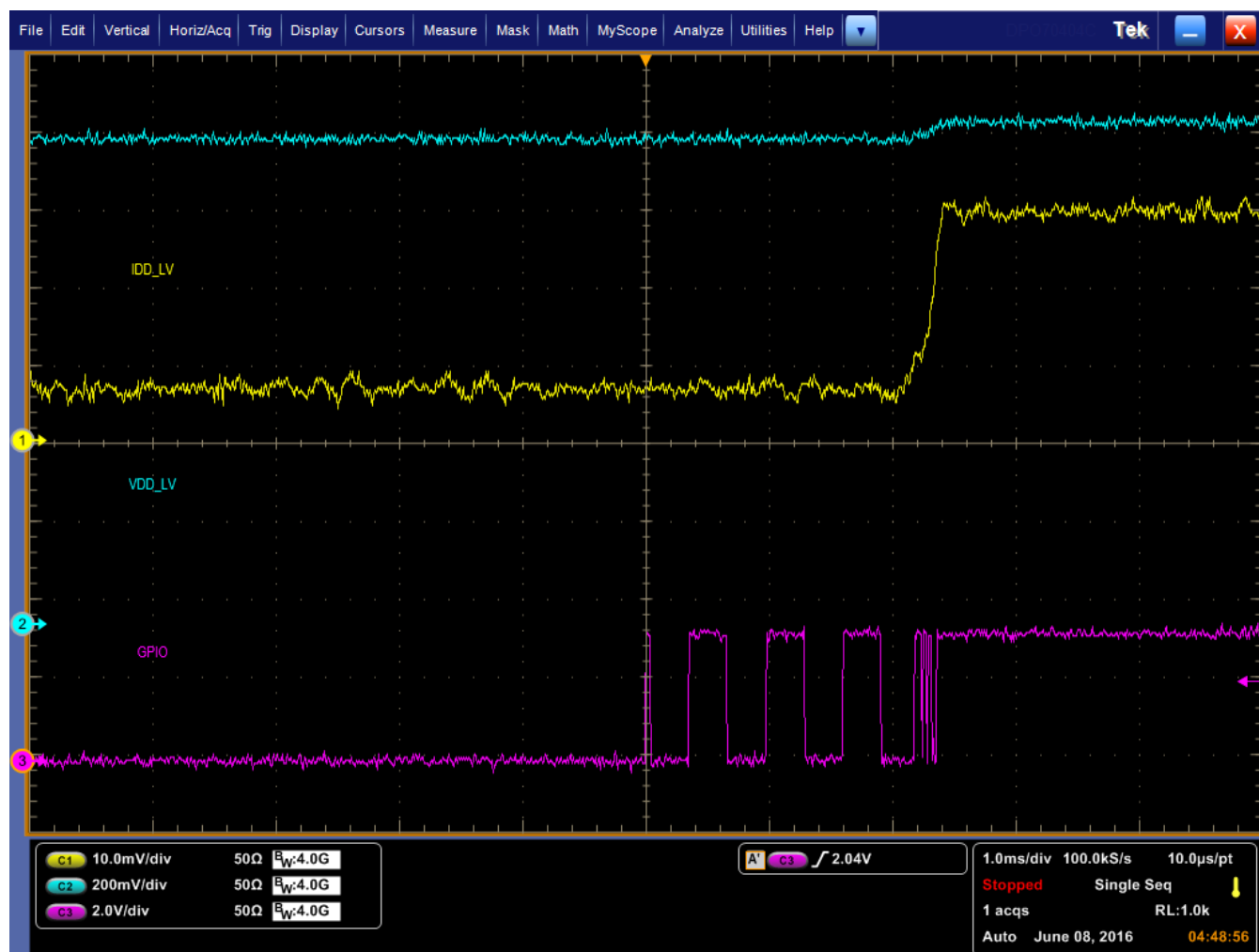


Figure 8. IDD_LV, VDD_LV, and I/O toggle rate with PCS enabled

4 Appendix A

This appendix contains sample software for Run Mode, Clocks, and PCS initialization.

4.1 C code listing

For the purposes of this document, we shall consider a typical application consisting of a main function and a function call to MC_MODE_INIT().

Sample software is shown below for main(). The main() function shall be target to run either from SRAM (0x4000_0000) or Flash (see MPC574xP RM for possible Flash target addresses).

Appendix A

```
*=====*
* COPYRIGHT:
* NXP Semiconductor, INC. All Rights Reserved. You are hereby
* granted a copyright license to use, modify, and distribute the
* SOFTWARE so long as this entire notice is retained without alteration
* in any modified and/or redistributed versions, and that such modified
* versions are clearly identified as such. No licenses are granted by
* implication, estoppel or otherwise under any patents or trademarks
* of NXP Semiconductor, Inc. This software is provided on an
* "AS IS" basis and without warranty.
*
* To the maximum extent permitted by applicable law, NXP
* Semiconductor DISCLAIMS ALL WARRANTIES WHETHER EXPRESS OR IMPLIED,
* INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A
* PARTICULAR PURPOSE AND ANY WARRANTY AGAINST INFRINGEMENT WITH REGARD
* TO THE SOFTWARE (INCLUDING ANY MODIFIED VERSIONS THEREOF) AND ANY
* ACCOMPANYING WRITTEN MATERIALS.
*
* To the maximum extent permitted by applicable law, IN NO EVENT
* SHALL NXP Semiconductor BE LIABLE FOR ANY DAMAGES WHATSOEVER
* (INCLUDING WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS,
* BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR OTHER
* PECUNIARY LOSS) ARISING OF THE USE OR INABILITY TO USE THE SOFTWARE.
*
* NXP Semiconductor assumes no responsibility for the
* maintenance and support of this software
*
*****/

/*****/
/* FILE NAME: main.c                                COPYRIGHT (c) NXP      2016    */
/*                                                    All Rights Reserved */
/* DESCRIPTION: Main C program                                */
/*                                                    */
/*****/
/* REV      AUTHOR      DATE      DESCRIPTION OF CHANGE */
/* ---      - - - - -    - - - - -    - - - - - */
/* 1.0 D.McMenamin      Sep 13, 2012  Initial Public Release */
/* 1.1 J.Fraser          Jun 08, 2016  Adapted for Panther */
/*****/
#include "../headers/project.h"
extern void MC_MODE_INIT(void);

/***** Main *****/
int main(){

    MC_MODE_INIT( ); // Setup the MCU clocks and modes
    /* Insert User code here */

}
```

MC_MODE_INIT() sample code is shown below.

```
*=====*
* COPYRIGHT:
* NXP Semiconductor, INC. All Rights Reserved. You are hereby
* granted a copyright license to use, modify, and distribute the
* SOFTWARE so long as this entire notice is retained without alteration
* in any modified and/or redistributed versions, and that such modified
* versions are clearly identified as such. No licenses are granted by
* implication, estoppel or otherwise under any patents or trademarks
* of NXP Semiconductor, Inc. This software is provided on an
* "AS IS" basis and without warranty.
*
* To the maximum extent permitted by applicable law, NXP
* Semiconductor DISCLAIMS ALL WARRANTIES WHETHER EXPRESS OR IMPLIED,
* INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A
* PARTICULAR PURPOSE AND ANY WARRANTY AGAINST INFRINGEMENT WITH REGARD
```

```

* TO THE SOFTWARE (INCLUDING ANY MODIFIED VERSIONS THEREOF) AND ANY
* ACCOMPANYING WRITTEN MATERIALS.
*
* To the maximum extent permitted by applicable law, IN NO EVENT
* SHALL NXP Semiconductor BE LIABLE FOR ANY DAMAGES WHATSOEVER
* (INCLUDING WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS,
* BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR OTHER
* PECUNIARY LOSS) ARISING OF THE USE OR INABILITY TO USE THE SOFTWARE.
*
* NXP Semiconductor assumes no responsibility for the
* maintenance and support of this software
*
*****/

/*****/
/* FILE NAME: mcu_init.c                                COPYRIGHT (c) NXP          2016 */
/*                                     All Rights Reserved */
/* DESCRIPTION: Provides functions to Initialize MPC5777M MCU */
/* */
/*! *****/
/*! REV      AUTHOR      DATE      DESCRIPTION OF CHANGE */
/*! ---      - - - - -      - - - - -      - - - - - */
/*! 1.0      C.Hillier     May 17, 2013  Initial Public Release */
/*! 2.0      J.Fraser      Jun 08, 2016  Updated for MPC574xP */
/*! *****/

#include "project.h"

void MC_MODE_INIT(void)
{
    /* Enable All Modes */
    MC_ME.ME.R = 0x000005E2;

    /* Peripheral ON in every run mode */
    MC_ME.RUN_PC[0].R = 0x000000FE;

    /* Enable EXT OSC First */
    XOSC.CTL.B.OSCM = 0x1;          /* Change OSC mode to LCP (Loop Controlled Pierce Mode) */
    XOSC.CTL.B.EOCV = 0x80;        /* Set the End of Count Value for when to check
                                   stabilization. */

    /****** PLL0, PLL1 *****/

    /* Route XOSC to PLL1 */
    MC_CGM.AC4_SC.B.SELCTL = 1;

    /* Route XOSC PLL0 */
    MC_CGM.AC3_SC.B.SELCTL = 1;

    /*
    Configure PLL0 Dividers for 160 MHz
    fPLL0_VCO = (fPLL0_ref x PLL0DV[MFD] x 2) / PLL0DV[PREDIV]
               = 40MHz x 8 x 2 / 1 = 640 MHz

    fPLL0_PHI  = fPLL0_ref x PLL0DV[MFD] / (PLL0DV[PREDIV] x PLL0DV[RFDPHI])
               = 40MHz x 8 / (1 x 2) = 160 MHz

    fPLL0_PHI1 = fPLL0_ref x PLL0DV[MFD] / (PLL0DV[PREDIV] x PLL0DV[RFDPHI1])
               = 40MHz x 8 / (1 x 8) = 40 MHz
    */
    PLLDIG.PLL0DV.B.RFDPHI1 = 8;
    PLLDIG.PLL0DV.B.RFDPHI  = 2;
    PLLDIG.PLL0DV.B.PREDIV  = 1;
    PLLDIG.PLL0DV.B.MFD     = 8;

    /*
    Configure PLL1 Dividers for 200 MHz
    fPLL1_VCO = fPLL1_REF x (PLL1DV[MFD] + PLL1FD[FRCDIV] / 2^12)
               = 40MHz x 20 + 0 = 800 MHz
    */

```

Revision history

```

fPLL1_PHI = fPLL1_REF * ( (PLL1DV[MFD] + PLL1FD[FRCDIV]/2^12) / (2 x PLL1DV[RFDPHI]) )
              = 40MHz x (20 + 0) / (2 x 2) = 200 MHz
*/
PLLDIG.PLL1DV.B.RFDPHI = 2;
PLLDIG.PLL1DV.B.MFD = 20;

/***** Configure Progressive Clock Switching *****/
MC_CGM.PCS_SDUR.R = 240;          /* set Switch Duration */

/* Configure PLL1 PCS switch */
MC_CGM.PCS_DIVC4.B.INIT = 1680;   /*! Set the Divider Change Initial Value */
MC_CGM.PCS_DIVC4.B.RATE = 112;   /*! Set the Divider Change Rate */
MC_CGM.PCS_DIVS4.R = 14439;      /*! Set the Divider Start Value */
MC_CGM.PCS_DIVE4.R = 12499;      /*! Set the Divider End Value */

/* Enable PLL0/PLL1 in DRUN mode, change PWRLVL to 1, and set PLL1 as SYS_CLK */
MC_ME.DRUN_MC.R = 0x101300F4;

/***** Configure Clock Dividers *****/
SIUL2.MSCR[22].R = 0x22800001;   /* Configure CLK_OUT (B6) */
MC_CGM.AC6_SC.B.SELCTL = 4;      /* source AC6 is PLL1 PHI */
MC_CGM.AC6_DC0.R = 0x80090000;   /* AC6 divider 0 --> div by 10 (CLK_OUT) */

MC_CGM.AC0_SC.B.SELCTL = 2;      /* source AC0 is PLL0 PHI */
MC_CGM.AC0_DC0.R = 0x80000000;   /* AC0 divider 0 --> div by 1 (MOTC_CLK) */
MC_CGM.AC0_DC1.R = 0x80070000;   /* AC0 divider 1 --> div by 8 (SWG_CLK) */
MC_CGM.AC0_DC2.R = 0x80010000;   /* AC0 divider 2 --> div by 2 (ADC_CLK) */

MC_CGM.AC1_DC0.R = 0x80010000;   /* AC1 divider 0 --> div by 2 (FRAY_PLL_CLK) */
MC_CGM.AC1_DC1.R = 0x80030000;   /* AC1 divider 1 --> div by 4 (SENT_CLK) */

MC_CGM.AC2_DC0.R = 0x80030000;   /* AC2 divider 0 --> div by 4 (CAN_PLL_CLK) */

MC_CGM.SC_DC0.R = 0x80030000;    /* SC divider 0 --> div by 4 (PBRIDGEEx_CLK) */

/***** Start the core *****/
/* Main and checker cores running in RUN3:0, DRUN, SAFE, TEST modes */
MC_ME.CCTL0.R = 0x00FE;

/***** Perform mode change *****/
/* Mode change re-enter the DRUN mode, to start cores, clock tree & PLL1 */
MC_ME.MCTL.R = 0x30005AF0;       /* Mode & Key */
MC_ME.MCTL.R = 0x3000A50F;       /* Mode & Key inverted */

while(MC_ME.GS.B.S_MTRANS == 1); /* Wait for mode entry complete */
while(MC_ME.GS.B.S_CURRENT_MODE != 0x3); /* Check DRUN mode entered */

}

```

5 Revision history

Table 3. Revision history table

Rev. No.	Date	Substantive Change(s)
0	June 2016	Initial release
1	November 2017	Editorial updates.
2	November 2017	Added PCS Calculator sheet in pdf attachment.

How to Reach Us:**Home Page:**nxp.com**Web Support:**nxp.com/support

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.

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, Vybrid, Airfast, 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. ARM, AMBA, ARM Powered, Artisan, Cortex, Jazelle, Keil, SecurCore, Thumb, TrustZone, and μ Vision are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. ARM7, ARM9, ARM11, big.LITTLE, CoreLink, CoreSight, DesignStart, Mali, mbed, NEON, POP, Sensinode, Socrates, ULINK and Versatile are trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. 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.

© 2017 NXP B.V.