Mode transition blocking by internal IPs clock busy signal

Introduction

This document presents a possible mode transition stuck in mode entry module (MC_ME) for SPC58xx devices.

The relationship between mode transition command and internal clock busy signal assertion will be discussed and in order to report a way to avoid the hang.

The phenomenon is generic to all IP auxiliary clock dividers. In this document, for describing the scenario, CAN_0 auxiliary clock is reported, also SPC58xC device is used for showing relevant signals and schematics.

1 Overview

Application software could observe a mode transition getting stuck, after several mode transitions requests. This behavior is not systematic it happens only under specific conditions, here described.

2 Steps for reproducing the phenomenon

2.1 Software

The phenomenon, described in this application note, can occur when the user's application attempts to switch from one running mode with clocks enabled to another in which clocks, are disabled after peripherals have been disabled via their PCTLx in mode entry (ME).

The following steps help for understanding the sequence.

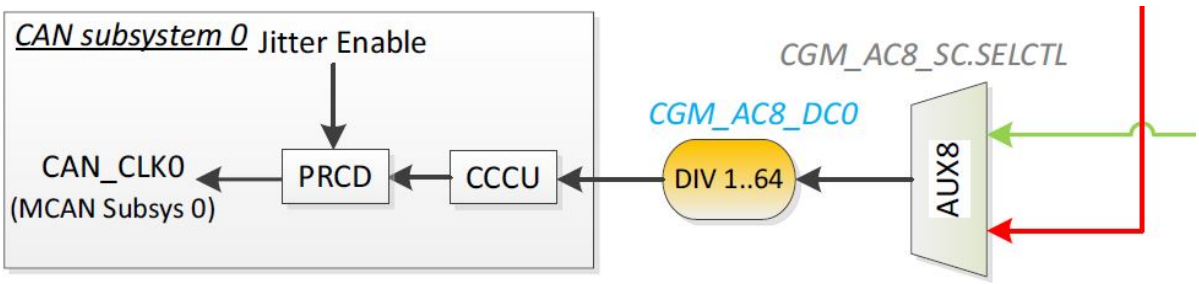
- A mode transition is performed for disabling the peripherals (ME.PCTLx -> 0)
  - No clock disabled instruction is programmed
- CAN auxiliary clock gets disabled (CGM.AC8_DC0 = 0x0)
- A mode transition is performed for disabling the clock sources (PLLx, XOSC)

2.2 CAN clock tree

The following figure shows an example of the peripheral clock tree (SPC58xC) with the dependence of PLL0:PHI (green) or XOSC (red) based on source clock selection in the CGM_AC8_SC.SELCTL register.

The full clock diagram can be found in the device's proper reference manual.

Figure 1. CAN clock tree on SPC58xC device



2.3 Block diagram

The following figure shows the block diagram of interconnection between mode entry (MC_ME) and clock generation module (MC_CGM) that are involved in completing a mode transition. PLL0 drives auxiliary AC8 clock divider here.

Figure 2. Block diagram when performing a transition

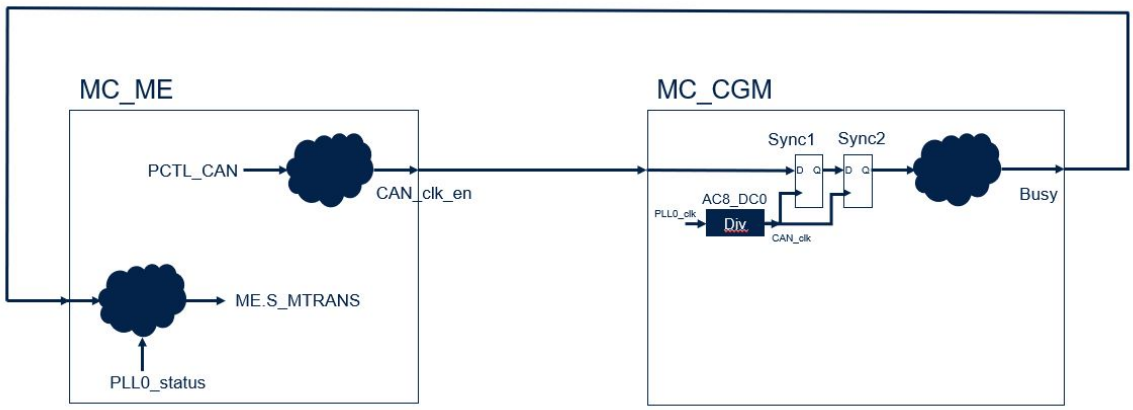


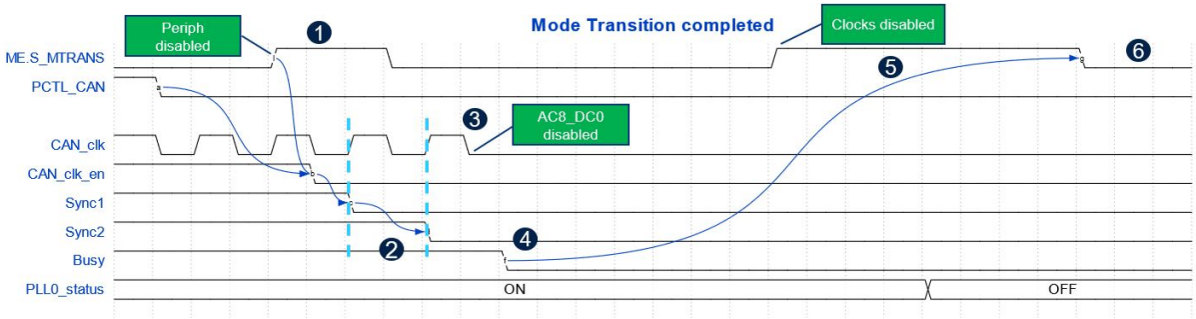
Diagram logical steps are described below:

1. PCTL_CAN: for enabling/disabling the peripheral.
2. CAN_clk_en: this is the signal used for evaluating if the peripheral is enabled or disabled.
3. MC_CGM.AC8_DC0: the auxiliary clock divider is used for configuring the peripheral, driven by PLL0:PHI.
4. PLL0_status: this is the signal used for evaluating if the PLL0 is switched on or off.
5. MC_ME.S_MTRANS: this is the mode transition status bit.

3 Expected behavior

The following figure shows the working scenario on which a transition for disabling the peripheral, the next peripheral clock disable and the final mode transition for disabling all the clocks in the current user mode are successfully completed.

Figure 3. Mode entry expected behavior



The steps to be described are summarized here below (details are in the following sections):

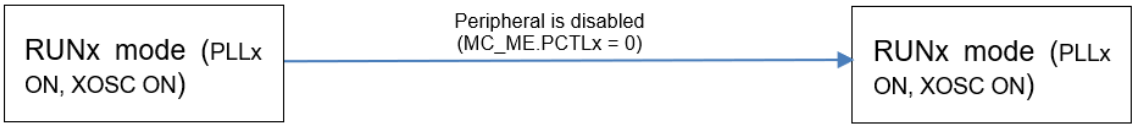
1. First mode transition to disable CAN_PCTL.
2. CAN clock propagation and synchronization.
3. CAN clock disabled via auxiliary clock divider.
4. Busy signal released.
5. Second mode transition for disabling clock sources (PLLx, XOSC).
6. Mode transition finished.

3.1 Peripheral disabling (1)

In this step, the MC_ME.PCTL related to CAN is set to zero for allowing the device to disable the peripheral and a mode transition is performed for activating the peripheral disabling.

Mode transition ends with success, mode entry does not check the clock sources because they are not powered-off in the current mode.

Figure 4. Transition clock status when disabling peripheral



3.2 CAN clock propagation (2) and busy signal (4)

CAN_clk_en signal is propagated through *CAN_clk* until the busy signal goes low (point 4).

This is because the synchronization steps complete before point (3).

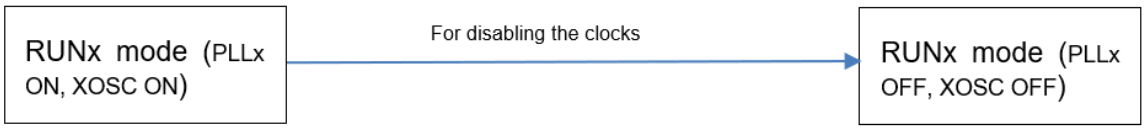
3.3 Auxiliary CAN clock disabling (3)

CAN_clk is driven by the MC_CGM.AUX8_DC0 auxiliary clock register and, based on the frequency of the IP, it remains in synch with the synchronization signals and allows the busy signal to go low.

3.4 Second mode transition, clocks disabling (5) and mode transition completion (6)

When the second mode transition is performed, for disabling the clocks in the final run mode, the busy signal is already low and so no problem to complete the transition, MC_ME.S_MTRANS goes low accordingly.

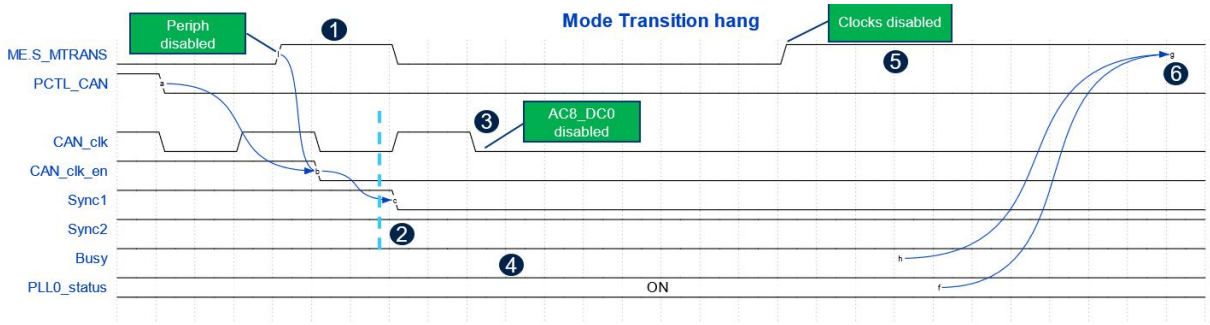
Figure 5. Mode transition for disabling the clocks



4 Mode transition stuck behavior

The following figure shows the scenario on which the final transition for disabling the clocks hangs.

Figure 6. Mode entry stuck behavior



The steps to be described are summarized here below:

1. First mode transition to disable CAN_PCTL.
2. CAN clock propagation and synchronization.
3. Slow CAN clock disable via auxiliary clock divider–during synchronization process.
4. Busy signal not released.
5. Second mode transition for disabling clock sources (PLLx, XOSC).
6. Mode transition pending–waiting for the busy signal release. It is not possible to disable clock source (PLL_0) to a periphery, which is using the clock source–signaled by the busy signal here. Similarly: XOSC cannot be disabled when used for PLL_0 as a source clock.

4.1 Differences across good behavior

The wrong behavior is due to the CAN_clk, which is disabled via MC_CGM.AC8_DC0 (3) during the synchronization process. At this point, the busy signal (4) does not go low and the second transition (5) loops indefinitely, because (6) the PLL0 clock is expected to be switched off while busy signal is active.

5 Workarounds

Two workarounds can be put in place for avoiding the reported phenomenon, see them in details.

5.1 Disable auxiliary clock

Disabling the AC8_DC0 auxiliary clock divider at the beginning of the application completely avoid the phenomenon. No clocks remain active and the busy signal works properly.

5.2 Use the right divider for the auxiliary clock

Using a proper value in auxiliary clock dividers is the first mandatory step for avoiding the phenomenon.

For each SPC58xx device there is a proper RM paragraph in the clocking chapter reporting the maximum auxiliary level clock frequencies.

Please refer to the relative chapter for every detailed information (see Reference documents).

Appendix A Reference documents

Doc Name	Title
RM0403	*SPC58 2B Line - 32 bit Power Architecture automotive MCU z2 core 80 MHz, 1 MByte Flash, ASIL-B.*
RM0449	*SPC58 4B Line - 32 bit Power Architecture automotive MCU z4 core 120 MHz, 2 MBytes Flash, HSM, ASIL-B.*
RM0407	*SPC58 C Line - 32 bit Power Architecture automotive MCU Dual z4 cores 180 MHz, 4 MBytes Flash, HSM, ASIL-B – Reference Manual.*
RM0391	*SPC58 E/G Line - 32 bit Power Architecture automotive MCU Triple z4 cores 180 MHz, 6 MBytes Flash, HSM, ASIL-D.*
RM0452	*SPC58 H Line - 32 bit Power Architecture automotive MCU Triple z4 cores 200 MHz, 10 MBytes Flash, HSM, ASIL-D.*
RM0421	*SPC58xNx 32-bit Power Architecture microcontroller for automotive ASILD applications.*

Revision history

Table 1. **Document revision history**

Date	Version	Changes
12-Dec-2022	1	Initial release.

Contents

List of tables

List of figures