S32V234: Clock Programming Considerations

by: NXP Semiconductors

1 Introduction

This application note highlights the basic programming model for S32V234 clocking architecture. The primary intent is to help the user setup the clocks to the maximum frequencies as defined in the reference manual, and reconfigure them to a new frequency, as per requirement. The document is primarily divided into the following sections:

1. Clock architecture blocks: A brief overview
2. Clock initialization sequence
3. Clock modification sequence

2 Terminologies and abbreviations used

Some of the terms and abbreviations used in this document are tabulated below with their meanings.

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>RM</td>
<td>Reference Manual</td>
</tr>
<tr>
<td>MC_CGM/CGM</td>
<td>Clock Generation Module</td>
</tr>
<tr>
<td>MC_ME</td>
<td>Mode Entry</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphics Processing Unit</td>
</tr>
<tr>
<td>DFS</td>
<td>Digital Frequency Synthesizer</td>
</tr>
<tr>
<td>PLL</td>
<td>Phase Lock Loop</td>
</tr>
<tr>
<td>FM</td>
<td>Frequency Modulation</td>
</tr>
<tr>
<td>PCFS/PCS</td>
<td>Progressive Clock Frequency Switching</td>
</tr>
<tr>
<td>FXOSC</td>
<td>Fast External Oscillator Clock</td>
</tr>
<tr>
<td>FIRC</td>
<td>Fast Internal RC clock</td>
</tr>
</tbody>
</table>

Table 1. Acronyms and abbreviations

Table continues on the next page...
Table 1. Acronyms and abbreviations (continued)

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;TargetMode_MC&gt;</td>
<td>Mode configuration register for desired target mode. For ex, if target mode is DRUN, &lt;TargetMode_MC&gt; is DRUN_MC register.</td>
</tr>
<tr>
<td>PCTL</td>
<td>Peripheral Control Register</td>
</tr>
<tr>
<td>MTRANS</td>
<td>Mode Transition</td>
</tr>
<tr>
<td>AC_SC</td>
<td>Auxiliary Clock Source Control</td>
</tr>
<tr>
<td>SRC</td>
<td>System Reset Controller</td>
</tr>
</tbody>
</table>

3 Clocking architecture blocks: A brief overview

3.1 What is new

Clocking on S32V234 is taken care of by the MC_ME, MC_CGM blocks. There have been quite some architectural changes, which have led to a change in the programming model. The major changes include:

- Multiple CGMs: The necessity of Progressive Clock Switching on the GPU clock domains and multiple high power consuming masters (including cortex A53s) led to introduction of four CGM instances. This would now mean multiple system clock selection options from MC_ME. Also, there is introduction of the concept of Secondary System clock sources for CGM 1, 2 and 3.

Digital Frequency Synthesizer (DFS): This block sits at the output of some of the PLLs (PLL-DFS relative placement) and it is the DFS output clocks that are used for most of the major operations in the chip. It provides wider granularity of
3.1.1 Clock sources and block diagram

Following are the seven clock sources in S32V234, out of which five are PLLs. Out of these PLLs, only ARM®-PLL, ENET-PLL and DDR-PLL have DFS block sitting at the PHI1 output.

- Fast Internal RC Oscillator 48 MHz (FIRC)
- Fast External Crystal Oscillator 40 MHz (FXOSC)
- FM PLL 1000 MHz (ARM-PLL)\(^1\)
- Non-FM PLL 1200 MHz (PERIPH-PLL)
- Non-FM PLL 1000 MHz (ENET-PLL)
- Non-FM PLL 1066 MHz (DDR-PLL)
- Non-FM PLL 1200 MHz (VIDEO-PLL)

<table>
<thead>
<tr>
<th>PLL</th>
<th>PHI0</th>
<th>PHI1</th>
<th>DFS1</th>
<th>DFS2</th>
<th>DFS3</th>
<th>DFS4</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM-PLL</td>
<td>1000</td>
<td>1000</td>
<td>266</td>
<td>600</td>
<td>600</td>
<td>--</td>
</tr>
<tr>
<td>PERIPH-PLL</td>
<td>400</td>
<td>100</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>ENET-PLL</td>
<td>500</td>
<td>1000</td>
<td>350</td>
<td>350</td>
<td>320/400/416</td>
<td>20/25/50/52/10</td>
</tr>
<tr>
<td>DDR-PLL</td>
<td>533</td>
<td>1066</td>
<td>500</td>
<td>500</td>
<td>350</td>
<td>--</td>
</tr>
<tr>
<td>VIDEO-PLL</td>
<td>600</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

4 Clock initialization sequence

This section illustrates the sequence in which the various clocks should be configured, enabled, and then selected as sources for various peripherals. The assumption here is that all clocks are OFF (except FIRC, which is the default system clock) when system boots up from Core M4. Following sections illustrate the step by step process of clock enablement.

\(^1\) For some of the chips from S32V family like S32V232 supports up to 800 MHz FM PLL frequency. This AppNote will assume 1000 MHz (S32V234). Other chips from S32V family can be similarly treated.
4.1 Wait for previous mode transition

Make sure that the previous mode transition is complete, by polling the MC_ME_GS[MTRANS] bit (Previous mode in MC_ME_DMTS may be RESET or DRUN depending on whether a mode transition has been executed or not by the boot ROM). See Clock Initialization Code.

4.2 PCFS settings

While system clock is still FIRC, PCFS configuration needs to be done to facilitate smooth transitions of system clock. This will allow ramp up and ramp down of system clock frequencies in multiple steps. For more details and how to configure this, see Progressive System Clock Switching section in MC_CGM chapter of S32V234 RM. An example of settings is provided in the initialization code attached. See Clock Initialization Code.

4.3 FXOSC setup

FXOSC can be enabled in any of the following two modes:

1. Normal Mode: Configure the FXOSC_CTL[EOCV] and FXOSC_CTL[GMSEL] field. The first parameter depends upon stabilization time of crystal and 2nd parameter depends upon trans-conductance applied by FXOSC amplifier. We will be using 0x9E and 0x7 respectively for these two parameters for our example case.

2. Oscillator Bypass Mode: In this mode, the on-board crystal is not used and clock is provided from an external source. To switch the XOOSC configuration to Bypass mode, configure FXOSC_CTL[OSCBYP] bit to 1. Also, configure the FXOSC_CTL[MISC_IN1] bit to 0 to disable the comparator.

Enable the FXOSCON bit for the <TargetMode_MC> register in MC_ME and give a mode transition to enable XOSC. See Clock Initialization Code.

4.4 PLL and DFS configuration

Following are the steps to enable PLL and DFS:

1. PLL Source Clock selection: The Source Clock for PLL can be either IRC or FXOSC clock. The PLLs must be locked on FXOSC as reference clock. The selection can be done using System Reset Controller module - SRC_GPR1[31:27] register bits, where each bit is dedicated for source selection of each of the five PLLs.

2. PLLDIG Configuration: Next step involves configuring the PLLDIG block to lock the PLL to a required clock frequency. For user reference, we have attached a PLL configuration calculator in this App Note. User can generate his own configuration based on the required output frequency. Also, the configurations for all PLL blocks for maximum frequency spec are given in Initialization code.

3. Enable PLL from MC_ME: Configure the PLLON bit (Corresponding to the PLL that is being enabled) in the <TargetMode_MC> register and give a mode transition to enable PLL from MC_ME. Wait for MTRANS to go low, indicating completion of mode transition.

4. DFS Configuration: Configure the DLLPRG1 to the recommended values (mentioned in the Init code Clock Initialization Code). Configure the DVPORT value to generate required output DFS frequency. For user reference, we have attached a DFS configuration calculator in this application note. User can generate his own configuration based on the required output frequency. De-assert the DLL_RESET by writing to DFS_CTRL register. Next, de-assert the PORTRESET for the DFS ports to enable the DFS CLKOUT. Poll the PORTSTAT bit to check if the DLL is locked.
5. ARM_PLL_DFS as System Clock: Configure <TargetMode_MC>[SYSCLK] as ARM_PLL_DFS1 and execute a mode transition to select ARM_PLL_DFS1 as system clock for CGM0.

6. Mode Transition: Configure <TargetMode>_SEC_CC_I_SYSCLK1 to 0x2, which corresponds to the slot for ARM_PLL_PHI0 for CGM1. Again, execute a mode transition, and wait for MTRANS to be de-asserted

Repeat the step five to select ARM_PLL_DFS_2, ARM_PLL_DFS3 as system clock for CGM2 & CGM3 respectively.

4.5 Setup auxiliary clocks

Although all the dividers are enabled by default on POR, there is a recommended sequence of clock source selection and divider value change. The sequence is as follows:

1. Disable the Divider by setting the DE bit to 0.
2. Reset the Source Select (AC_SC) to 0.
3. Select the source (AC_SC) to the actual clock source to be selected.
4. Enable the divider by enabling the DE bit and configure the DIV bit to divide the clock as per requirement.

See Clock Initialization Code.

4.6 Enable PCTLs

By default, peripheral clocks are gated from MC_ME. Once the Auxiliary MUX are appropriately configured to the correct clock frequencies, configure PCTLs and RUN_PCn registers from MC_ME to ungate the clock reaching the individual peripherals. Do not forget to give a mode transition to bring the PCTL configuration into effect. For low power consumption, only the required Peripherals should be configured to be enabled. See Clock Initialization Code.

5 Clock modification sequence

This section explains how to change any clock without affecting the rest of the system. This is usually characterized by the following high level sequence:

1. Disable the peripherals operating on the clock
2. Disable, Reconfigure, and Enable the clocks
3. Enable the peripherals

The exact sequence depends on the clocking tree and this can be different for different clocks. For example- a change in system clock is always characterized by a mode re-entry. It may be required to change the DFS configuration or in some cases, PLL re-configuration may also be required. To simplify things, the sequence can be divided into 3 types, with increasing complexities.

1. Frequency change via MC_CGM
2. Frequency change via DFS module
3. Frequency change via PLL module

To understand it clearly, corresponding examples are explained below:

1. Change CAN_CLK from current clock (PERIPH_PLL/5) to XOSC Part B1
   a. Disable CAN PCTL from MC_ME.
b. As per clocking diagram shown in Figure 1. Top level clocking tree on page 2, CGM0_AC6_SC controls the CAN clock. Configure the SELCTL in this register to directly change the source of CAN clk to XOSC.

i. Disable the Divider by setting the DE bit to 0.

ii. Reset the Source Select(AC6_SC) to IRC (0).

iii. Select the source (AC6_SC) to XOSC (1).

iv. Enable the divider by enabling the DE bit and configure the DIV bit to divide the clock as per requirement.

c. Enable CAN PCTL from MC_ME.

2. Change ENET_PLL_DFS3 from current clock frequency (320) to 416 MHz Part B2

a. The Initialization sequence mentioned in previous section would set ENET_PLL_DFS3 to 320 MHz. CGM dividers alone can’t change the frequency from 320 to 416 MHz

b. The source ENET_PLL_DFS3 comes from ENET_PLL as shown in Figure 1. Top level clocking tree on page 2. The DFS3 is the 3rd port of ENETPLLDFS. The configuration for the new DFS frequency is computed with the following formula.

\[ f_{dfsclockoutn} = \frac{f_{dfsclockin}n}{(DFS_DVPORTn[MFI] + (DFS_DVPORTn[MFN] / 256)} \]

where, DFS_DVPORTn[MFI]: integer part of division [1:255], DFS_DVPORTn[MFN]: fractional part of division [0:255] and n: Analog DFS output port number.

3. Change CAN_CLK (Source -> DIV by five PERIPH_PLL PHI0) from current value (80) to 60 MHz Part B3

a. As per clocking diagram, CGM AC6_SC is the AUX_MUX supplying the CAN_CLK. The required output cannot be achieved by changing the divider value of AC6_DC from CGM, as target frequency is not an integral multiple.

b. PERIPH_PLL does not have any DFS. PERIPH_PLL divided by five is used directly as the source clock for CAN clock (see Aux clock 6 of CGM0). To reduce the clock frequency from 80 MHz to 60 MHz, we need to change the PERIPH_PLL clock frequency from 400 MHz to 300 MHz.

c. Disable All Peripherals working on PERIPH_PLL or switch them to other clock sources if required using MC_CGM.

d. Disable PERIPH_PLL from MC_ME and give a mode transition.

e. The PERIPH_PLL configuration needs to be changed. The configuration for the new PLL frequency is computed with the following formula:

\[ f_{pll\_phi0} = f_{pll\_ref} \times \frac{mfd + mfn/20480}{prediv \times rfdphi} \]

The exact value to be programmed in the PLLDV and possibly PLLFD register can be conveniently determined by using the attached tool. See examples here. To guarantee the correct functionality of PLLs, their configuration must follow the right programming sequence like shown in Part B3.

f. Enable PERIPH_PLL from MC_ME and give a mode transition. Now again configure peripherals on PERIPH_PLL to be ON and give a mode transition.
6 Tool usage example

1. DFS The syntax to use dfs_config is `dfs_config.exe <Input freq in MHz> <required freq in MHz>`

```
C:\App>dfs>dfs_config.exe 1000 416
Input Frequency is 1000
Output Frequency is 416
mfi 2
mfn 103
DFS_DUPORT register value: 0x267
C:\App\dfs>
```

2. PLL The syntax to use pll_config is `pll_config.exe <ref clk freq> <required phi0 freq> <required phi1 freq>`

   a. Example 1

   ```
   C:\App\pll>pll_config.exe 40 1297 1297
   Reference frequency 40
   Output frequency phi0 1297
   Output frequency phi1 1297
   Possible PLLDU & PLLFD register values are:
   PLLDU: 0x2012040
   PLLFD: 0x40004400
   PLLDU: 0x2011020
   PLLFD: 0x40004400
   C:\App\pll>
   ```

   b. Example 2

   ```
   C:\App>pll_config.exe 40 400 100
   Reference frequency 40
   Output frequency phi0 400
   Output frequency phi1 100
   Possible PLLDU register values are:
   0x1803203c
   0x1803101e
   0x1002303c
   0x10022028
   0x10021014
   ```
7 Clock Initialization Code

// ###################################################################### Sec A.1 - Wait for Mode transition ######################################################################
void wait_for_mtrans()
{
    while(MC_ME.GS.B.S_MTRANS != 0);       //Wait for default mode entry to complete
}

// ###################################################################### Sec A.2 - PCFS Settings ######################################################################
void PCFS_Settings()
{
    /* CGM0 -> IRC-> ARM_PLL_DFS1*/
    MC_CGM_0.SDUR.R = 0x20;
    MC_CGM_0.PCS_DIVE2.R = 0x15A4;
    MC_CGM_0.PCS_DIVC2.B.INIT = 0x150;
    MC_CGM_0.PCS_DIVC2.B.RATE =0xC;
    MC_CGM_0.PCS_DIVS2.R = 0x16f3;

    /* CGM1 -> IRC-> ARM_PLL_PHI0*/
    MC_CGM_1.SDUR.R = 0x20;
    MC_CGM_1.PCS_DIVE2.R = 0x5160;
    MC_CGM_1.PCS_DIVC2.B.INIT = 0x2B7;
    MC_CGM_1.PCS_DIVC2.B.RATE =0xC;
    MC_CGM_1.PCS_DIVS2.R = 0x5413;

    /* CGM2 -> IRC-> ARM_PLL_DFS2*/
    MC_CGM_2.SDUR.R = 0x20;
    MC_CGM_2.PCS_DIVE2.R = 0x30D3;
    MC_CGM_2.PCS_DIVC2.B.INIT = 0x0213;
    MC_CGM_2.PCS_DIVC2.B.RATE =0xC;
    MC_CGM_2.PCS_DIVS2.R = 0x32E0;

    /* CGM3 -> IRC-> ARM_PLL_DFS3*/
    MC_CGM_3.SDUR.R = 0x20;
    MC_CGM_3.PCS_DIVE2.R = 0x30D3;
    MC_CGM_3.PCS_DIVC2.B.INIT = 0x213;
    MC_CGM_3.PCS_DIVC2.B.RATE =0xC;
    MC_CGM_3.PCS_DIVS2.R = 0x32E0;
}

// ###################################################################### Sec A.3 - XOSC Enablement ######################################################################
void XOSC_setup()
{
    FXOSC.CTL.B.EOCV = 0x9E;              //EOCV value = 0x9E
    FXOSC.CTL.B.GM_SEL = 0x7;             //transconductance = 1x
    #ifdef XOSC_BYPASS_ENABLE
        FXOSC.CTL.B.OSCBYP = 1;
        FXOSC.CTL.B.MISC_IN1 = 0;
    #else
        FXOSC.CTL.B.OSCBYP = 0;
        FXOSC.CTL.B.MISC_IN1 = 1;
    #endif
    MC_ME.DRUN_MC.B.FXOSCON = 1;         //Enable FXOSCON from MC_ME

    // RE enter the drun mode, to update the configuration */
    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 to complete
    while(MC_ME.GS.B.S_CURRENT_MODE != 0x3); // Check DRUN mode has been entered
    FXOSC.CTL.R = FXOSC.CTL.R;            //Clear XOSC Stable Interrupt Flag
}

// ###################################################################### Sec A.4 - PLLs & DFS Enablement ######################################################################
// ###################################################################### Sec A.4.1 - ARMPPLL enablement & Sys clock Setup ######################################################################
void ARM_PLL_Setup()
{

DFS_0.CTRL.R = 0x2;  //Assert DFS DLL reset
DFS_0.PORTRESET.R = 0xffffffff;  //Output port reset is asserted
SRC.GPR1.R |= 0x08000000;  //set XOSC as source of ARMPPLL
PLLDIG_0.PLLCAL2.R = 0x2b;  //CAL2
PLLDIG_0.PLLCAL1.R = 0x44000000;  //CAL1
PLLDIG_0.PLLDV.R = 0x02011019;  //Configuration for PLL Frequency = 1000MHz
PLLDIG_0.PLLFD.R |= 0x40000000;
MC_ME.DRUN_MC.B.ARMPLLON = 1;  //Enable ARMPPLL from MC_ME

// Re enter the drun mode, to update the configuration
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 to complete
while(MC_ME.GS.B.S_CURRENT_MODE != 0x3);  // Check DRUN mode has been entered

DFS_0.DVPORT1.R = 0x0000003c2;  // Set ARMPPLL_DFS_1 to 266MHz
DFS_0.DVPORT2.R = 0x0000001aa;  // Set ARMPPLL_DFS_2 to 600MHz
DFS_0.DVPORT3.R = 0x0000001aa;  // Set ARMPPLL_DFS_3 to 600MHz

DFS_0.CTRL.R &= 0xffffffffD;  //Deassert DFS DLL reset
DFS_0.PORTRESET.R &= 0xffffffff0;  //Deassert Output port reset

while(DFS_0.PORTSR.B.PORTSTAT != 0xF);  //Wait for Output port to be locked

MC_ME.DRUN_MC.B.SYSCLK = 2;  //Set ARMPLLDFS1 as System Clock for CGM0
MC_ME.DRUN_MC.B.PWRLVL = 0x7;  // Enable PCFS on Mode Transition

/* Re enter the drun mode, to update the configuration */
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 to complete
while(MC_ME.GS.B.S_CURRENT_MODE != 0x3);  // Check DRUN mode has been entered

MC_ME.DRUN_SEC_CC_I.B.SYSCLK1 = 2;  //ARMPPLL_PHIO as system clock1
MC_ME.DRUN_SEC_CC_I.B.PWRLVL1 = 0x7;  // Enable PCFS on Mode Transition

/* Re enter the drun mode, to update the configuration */
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 to complete
while(MC_ME.GS.B.S_CURRENT_MODE != 0x3);  // Check DRUN mode has been entered

MC_ME.DRUN_SEC_CC_I.B.SYSCLK2 = 2;  //ARMPPLLDFS2 as system clock2
MC_ME.DRUN_SEC_CC_I.B.PWRLVL2 = 0x7;  // Enable PCFS on Mode Transition

/* Re enter the drun mode, to update the configuration */
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 to complete
while(MC_ME.GS.B.S_CURRENT_MODE != 0x3);  // Check DRUN mode has been entered

MC_ME.DRUN_SEC_CC_I.B.SYSCLK3 = 2;  //ARMPPLLDFS3 as system clock3
MC_ME.DRUN_SEC_CC_I.B.PWRLVL3 = 0x7;  // Enable PCFS on Mode Transition

/* Re enter the drun mode, to update the configuration */
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 to complete
while(MC_ME.GS.B.S_CURRENT_MODE != 0x3);  // Check DRUN mode has been entered

// ################################################################### Sec A.4.2 - PERIPHPLL enablement ###################################################################

void PERIPH_PLL_setup()
{
    SRC.GPR1.R |= 0x04000000;  //set XOSC as source of PERIPHPLL
}
PLLDIG_1.PLLDV.R=0x1803101E;  //Config for PLL_PHI0= 400MHz, PHI1=100
PLLDIG_1.PLLFD.R =0x40000000;

MC_ME.DRUN_MC.B.PERIPHPLLON = 1;  //Enable PERIPHPLL from MC_ME
/* Re enter the drun mode, to update the configuration */
MC_ME.MCTL.R = 0x30005AF0;  // Mode & Key
MC_ME.MCTL.R = 0x3000A5OF;  // Mode & Key inverted
while(MC_ME.GS.B.S_MTRANS == 1);  // Wait for mode entry to complete
while(MC_ME.GS.B.S_CURRENT_MODE != 0x3);  // Check DRUN mode has been entered
}

// ########################### Sec A.4.2 - ENETPLL enablement ###########################
void ENET_PLL_setup()
{
  SRC.GPR1.R |= 0x10000000;  //set XOSC as source of ENETPLL

PLLDIG_2.PLLDV.R= 0x02021019;  //Config for PLL_PHI0=500, PHI1=1000
PLLDIG_2.PLLFD.R=0x40000000;

MC_ME.DRUN_MC.B.ENETPLLON = 1;  //Enable ENETPLL from MC_ME
/* Re enter the drun mode, to update the configuration */
MC_ME.MCTL.R = 0x30005AF0;  // Mode & Key
MC_ME.MCTL.R = 0x3000A5OF;  // Mode & Key inverted
while(MC_ME.GS.B.S_MTRANS == 1);  // Wait for mode entry to complete
while(MC_ME.GS.B.S_CURRENT_MODE != 0x3);  // Check DRUN mode has been entered
DFS_1.DLLPRG1.R =0x00005445;  //Configure Parameters for DFS
DFS_1.DLLPRG2.R =0x0000;
DFS_1.DVPORT1.R =0x000002db;  // Set ENETPLL_DFS_1 to 350MHz
DFS_1.DVPORT2.R =0x000002db;  // Set ENETPLL_DFS_2 to 350MHz
DFS_1.DVPORT3.R =0x00000320;  // Set ENETPLL_DFS_3 to 320MHz
DFS_1.DVPORT4.R =0x0000099c;  // Set ENETPLL_DFS_4 to 104MHz
DFS_1.CTRL.R &= 0xfffffffd;  //Deassert DFS DLL reset
DFS_1.PORTRESET.R &= 0xfffffff0;  //Deassert Output port reset
while(DFS_1.PORTSR.B.PORTSTAT != 0xF);  //Wait for Output port to be locked
/* Re enter the drun mode, to update the configuration */
MC_ME.MCTL.R = 0x30005AF0;  // Mode & Key
MC_ME.MCTL.R = 0x3000A5OF;  // Mode & Key inverted
while(MC_ME.GS.B.S_MTRANS == 1);  // Wait for mode entry to complete
while(MC_ME.GS.B.S_CURRENT_MODE != 0x3);  // Check DRUN mode has been entered
}

// ############################ Sec A.4.2 - DDRPLL enablement ##########################
void DDR_PLL_setup()
{
  SRC.GPR1.R |= 0x20000000;  //set XOSC as source of PERIPHPLL

PLLDIG_3.PLLDV.R=0x0102101A;  //Config for PLL_PHI0=533, PHI1=1066
PLLDIG_3.PLLFD.R=0x40000000;

MC_ME.DRUN_MC.B.DDRPLLON = 1;  //Enable DDRPLL from MC_ME
/* Re enter the drun mode, to update the configuration */
MC_ME.MCTL.R = 0x30005AF0;  // Mode & Key
MC_ME.MCTL.R = 0x3000A5OF;  // Mode & Key inverted
while(MC_ME.GS.B.S_MTRANS == 1);  // Wait for mode entry to complete
while(MC_ME.GS.B.S_CURRENT_MODE != 0x3);  // Check DRUN mode has been entered
DFS_2.DLLPRG1.R =0x000005445;  //Configure Parameters for DFS
DFS_2.DLLPRG2.R =0x0000;
DFS_2.DVPORT1.R =0x0000002db;  // Set DDRPLL_DFS_1 to 500MHz
DFS_2.DVPORT2.R =0x0000002db;  // Set DDRPLL_DFS_2 to 500MHz
DFS_2.DVPORT3.R =0x0000030b;  // Set DDRPLL_DFS_3 to 350MHz
DFS_2.DVPORT4.R =0x0000099c;  // Set DDRPLL_DFS_4 to 104MHz
DFS_2.CTRL.R &= 0xfffffffd;  //Deassert DFS DLL reset
DFS_2.PORTRESET.R &= 0xfffffff0;  //Deassert Output port reset
while(DFS_2.PORTSR.B.PORTSTAT != 0xF);  //Wait for Output port to be locked
/* Re enter the drun mode, to update the configuration */
MC_ME.MCTL.R = 0x30005AF0;  // Mode & Key
MC_ME.MCTL.R = 0x3000A5OF;  // Mode & Key inverted
while(MC_ME.GS.B.S_MTRANS == 1);  // Wait for mode entry to complete
while(MC_ME.GS.B.S_CURRENT_MODE != 0x3);  // Check DRUN mode has been entered
}
while(DFS_2.PORTSR.B.PORTSTAT != 0xF); //Wait for Output port to be locked
}

// ################################### Sec A.4.2 - VIDEOPLL enablement ###################################
void VIDEO_PLL_setup()
{
    SRC.GPR1.R |= 0x80000000; //set XOSC as source of PERIPHPLL
    PLLDIG_4.PLLDV.R =0x0002101e; //Config for PLL_Phi0=600
    PLLDIG_4.PLLFD.R=0x40000000;
    MC_ME.DRUN_MC.B.VIDEOPLLON = 1; //Enable VIDEOPLL from MC_ME
    /* Re enter the drun mode, to update the configuration */
    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 to complete
    while(MC_ME.GS.B.S_CURRENT_MODE != 0x3); // Check DRUN mode has been entered
}

// ################################### Sec A.5 - Auxiliary clock Setup ###################################
void setup_aux_clks()
{
    /* Example shown only for AC0. User needs to configure other ACs as shown below*/
    MC_CGM_0.AC0_DC0.B.DE = 0; //divider enable(1)/disable(0)
    MC_CGM_0.AC0_SC.B.SELCTL = 0; //AC0 clock source selection - DDR_PLL_DFS1
    MC_CGM_0.AC0_DC0.B.DE = 1; //divider enable(1)/disable(0)
    MC_CGM_0.AC0_DC0.B.DIV = 0; //divided by 1 (1 + DIV); ISP_CLK = 500 MHz
    /* Setting RUN Configuration Registers */
    MC_ME.RUN_PC[0].R=0x000000FE; // Peripheral ON in every mode
    MC_ME.RUN_PC[1].R=0x000000FE; // Peripheral ON in every mode
    /* Re enter the drun mode, to update the configuration */
    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 to complete
    while(MC_ME.GS.B.S_CURRENT_MODE != 0x3); // Check DRUN mode has been entered
}

// ################################### Sec A.6 - System Initialization (Complete Flow) ###################################
void sys_init(void)
{
    wait_for_mtrans();
    CMU_0.CSR.B.RCDIV = 0x3; //RCDIV max
    MC_ME.RUN_PC0.R=0x00000000; // Peripheral Off
    MC_ME.RUN_PC1.R=0x00000000; // Peripheral Off
    MC_ME.DRUN_MC.R=0x100010; //IRC as system clock0
    MC_ME.DRUN_SEC_CC_I.B.SYSCLK1 =0; //IRC as system clock1
    MC_ME.DRUN_SEC_CC_I.B.SYSCLK2 =0; //IRC as system clock2
    MC_ME.DRUN_SEC_CC_I.B.SYSCLK3 =0; //IRC as system clock3
    /* Re enter the drun mode, to update the configuration */
    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 to complete
    while(MC_ME.GS.B.S_CURRENT_MODE != 0x3); // Check DRUN mode has been entered
    MC_ME.ME.R = 0x80FF; //enable all modes from MC_ME
    XOSC_setup(); //Basic XOSC initialization
PCFS_Settings(); // Progressive Clock Switching settings for sysclk transition from IRC - ARMPLL

ARM_PLL_Setup(); // Settings for ARMPLLDFS
PERIPH_PLL_Setup(); // Settings for PERIPHLPLL
ENET_PLL_Setup(); // Settings for ENETPLLDFS
DDR_PLL_Setup(); // Settings for DDRPLLDFS
VIDEO_PLL_Setup(); // Settings for VIDEOPLL

8 Clock Modification Code

#Sec B.1. Change CAN_CLK from current clock(PERIPH_PLL/5) to XOSC

DISABLE_CAN_PCTL();
MC_CGM_0.AC6_DC0.B.DE = DISABLE;
MC_CGM_0.AC6_SC.B.SELCTL= XOSC_CLK;
while((MC_CGM_0.AC6_SS.B.SELSTAT)!=XOSC_CLK & & TIMEOUT--);
if(!(TIMEOUT))
{
    error("CLK Source not getting selected");
    return 1;
}
MC_CGM_0.AC6_DC0.R = ((ENABLE<<31) | ((DIV_VALUE_1-1)<<16));
ENABLE_CAN_PCTL();

#Sec B.2. Change ENET_PLL_DFS3 from current clock frequency(320) to 416 MHz

DFS_1.CTRL.B.DLL_RESET = 1; //Assert DFS DLL reset
DFS_1.PORTRESET.R |= 0x0000000f; //Assert Output port reset
DFS_1.DLLPRG1.R =0x00005445; //Configure Parameters for DFS
DFS_1.DLLPRG2.R =0x0000;
DFS_1.DVPORT1.R =0x0000002db; //Set ENETPLL_DFS_1 to 350MHz
DFS_1.DVPORT2.R =0x0000002db; //Set ENETPLL_DFS_2 to 350MHz
DFS_1.DVPORT3.R =0x000000267; //Set ENETPLL_DFS_3 to 416MHz
DFS_1.DVPORT4.R =0x00000099c; //Set ENETPLL_DFS_4 to 104MHz
DFS_1.CTRL.R &= 0xfffffffd; //Deassert DFS DLL reset
DFS_1.PORTRESET.R &=0xfffffff0; //Deassert Output port reset
while(DFS_1.PORTSR.B.PORTSTAT != 0xF); //Wait for Output port to be locked

#Sec B.3. Change CAN_CLK(Source -> DIV by 5 PERIPH_PLL PHI_0)from current value(80 MHz) to 60 MHz

MC_ME.DRUN_MC.B.PERIPHPLLON = 0; //Disable PERIPH_PLL from MC_ME

/* Re enter the drum mode, to update the configuration */
MC_ME.MCTL.R = 0x30005A50; // Mode & Key
MC_ME.MCTL.R = 0x3000A50F; // Mode & Key inverted
while(MC_ME.GS.B.S_MTRANS == 1); // Wait for mode entry to complete
while(MC_ME.GS.B.S_CURRENT_MODE != 0x3); // Check DRUN mode has been entered
PLLDBG_1.PLLDV.R=0x1804101E; //Config for PLL_PHI0= 300MHz,
PHI1=100
PLLDBG_1.PLLFD.R =0x400000000;
MC_ME.DRUN_MC.B.PERIPHPLLON = 1; //Enable PERIPHPLL from MC_ME

/* Re enter the drum mode, to update the configuration */
MC_ME.MCTL.R = 0x30005A50; // Mode & Key
MC_ME.MCTL.R = 0x3000A50F; // Mode & Key inverted
while(MC_ME.GS.B.S_MTRANS == 1); // Wait for mode entry to complete
while(MC_ME.GS.B.S_CURRENT_MODE != 0x3); // Check DRUN mode has been entered

/#*--------- Definitions of functions/Macros used in Section B source code --------*/

#define DISABLE 0
#define ENABLE 1
#define TIMEOUT 0x00FFFFFF
#define DIV_VALUE_1 1

typedef enum{
    IRC_CLK =0,
    XOSC_CLK =1,
    ARMPLL_CLK=2
}CLK_SRC;

typedef enum{
    RUN_PC0 = 0,
    RUN_PC1,
    RUN_PC2,
    RUN_PC3,
    RUN_PC4,
    RUN_PC5,
    RUN_PC6,
    RUN_PC7
};

void DISABLE_CAN_PCTL()
{
    MC_ME.RUN_PC2.R = 0x00; // Disable peripheral in all modes
    MC_ME.PCTL85.R = RUN_PC2; //CANFD_0 takes settings from RUN_PC2 which is
    configured for clock gated in all modes
    MC_ME.PCTL190.R = RUN_PC2; //CANFD_1 takes settings from RUN_PC2 which is
    configured for clock gated in all modes
    /* RE enter the drun mode, to update the configuration */
    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 to complete
    while(MC_ME.PS2.B.S_CANFD0 != 0); // Checking Peripheral status
    while(MC_ME.PS5.B.S_CANFD1 != 0);
}

void ENABLE_CAN_PCTL()
{
    MC_ME.RUN_PC0.R = 0xFE; // Enable peripheral in all modes
    MC_ME.PCTL85.R = RUN_PC0; //CANFD_0 takes settings from RUN_PC0 which is
    configured for clock enabled in all modes
    MC_ME.PCTL190.R = RUN_PC0; //CANFD_1 takes settings from RUN_PC0 which is
    configured for clock enabled in all modes
    /* RE enter the drun mode, to update the configuration */
    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 to complete
    while(MC_ME.PS2.B.S_CANFD0 != 1); // Checking Peripheral status
    while(MC_ME.PS5.B.S_CANFD1 != 1);
}
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, AtiVec, C-5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetics, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airlast, 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 μVision 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.