

Hi, I am trying to setup the clocks/PLL on the SPC570S40E1 microcontroller. I would like use the PLL0 as the system clock.

Here is my configuration:

AC3 SC -> 16Mhz IRSOC

AC2 SC -> PLL1

PLL0 Settings:

PREDIV	-> 2
MFD	-> 80
RFDPHI	-> 10
RFDPHI1	-> 10

PLL1 Settings:

MFD	-> 18
RFDPHI	-> 8

With the above settings, I get the following clk rates.

Calculated Clock Points		
XOSC	8000000	IRC
PLL0-IN	16000000	PLL0-VCO
PLL0-PHI	64000000	PLL0-PHI1
PLL1-IN	64000000	PLL1-VCO
PLL1-PHI	72000000	
SYSCLK	64000000	IPS_CLK
CLKOUT	16000000	
PIT_CLK	12800000	SARADC_CLK
CTU_CLK	2560000	DSPI_CLK
LIN_CLK	64000000	ETIMER_CLK
CAN_CLK	8000000	

Here is my initialization code. Its getting stuck in the while loop waiting for PLL0 to lock. I am following the PLLs initialization procedure that is recommended in the reference manual.

```

void spc_clock_init(void) {
    /* Waiting for IRC stabilization before attempting anything else.*/
    while (MC_ME.GS.B.S_IRC == 0U) {
    }

    #if (SPC5_NO_INIT == FALSE)

    #if SPC5_DISABLE_WATCHDOG
        /* SWTs disabled.*/
        SWT_2.SR.R = 0xC520U;
        SWT_2.SR.R = 0xD928U;
        SWT_2.CR.R = 0xFF000002UL;
    #endif

    /* The system must be in DRUN mode on entry, if this is not the case then
     * it is considered a serious anomaly.*/
    if (MC_ME.GS.B.S_CURRENT_MODE != (uint8_t)SPC5_RUNMODE_DRUN) {
        SPC5_CLOCK_FAILURE_HOOK();
    }

    /* Memory-mapped register definitions incompatible with MISRA rule. */
    /*lint -e10 -e40 -e63*/
    /* Setting the system dividers to their final values.*/
    MC_CGM.SC_DC0.R = SPC5_CGM_SC_DC0_BITS;
    MC_CGM.SC_DC1.R = SPC5_CGM_SC_DC1_BITS;
    MC_CGM.SC_DC2.R = SPC5_CGM_SC_DC2_BITS;

    /* Setting the auxiliary dividers to their final values.*/
    MC_CGM.AC0_DC0.R = SPC5_CGM_AC0_DC0_BITS;
    MC_CGM.AC0_DC1.R = SPC5_CGM_AC0_DC1_BITS;
    MC_CGM.AC0_DC2.R = SPC5_CGM_AC0_DC2_BITS;
    MC_CGM.AC0_DC3.R = SPC5_CGM_AC0_DC3_BITS;
    MC_CGM.AC0_DC4.R = SPC5_CGM_AC0_DC4_BITS;
    MC_CGM.AC0_DC5.R = SPC5_CGM_AC0_DC5_BITS;
    MC_CGM.AC1_DC0.R = SPC5_CGM_AC1_DC0_BITS;

    /* Setting the clock selectors to their final sources.*/
    MC_CGM.AC0_SC.R = SPC5_CGM_AC0_SC_BITS;
    MC_CGM.AC1_SC.R = SPC5_CGM_AC1_SC_BITS;
    MC_CGM.AC2_SC.R = SPC5_CGM_AC2_SC_BITS;
    MC_CGM.AC3_SC.R = SPC5_CGM_AC3_SC_BITS;
    /*lint +e10 +e40 +e63*/

    /* Enables the XOSC in order to check its functionality before proceeding
     * with the initialization.*/
    MC_ME.DRUN_MC.R = SPC5_ME_MC_SYSCLK_IRC | SPC5_ME_MC_IRCON |
        SPC5_ME_MC_XOSCON | SPC5_ME_MC_FLAON_NORMAL |
        SPC5_ME_MC_MVRON;
    if (SPC5SetRunMode(SPC5_RUNMODE_DRUN) == 0) {
        SPC5_CLOCK_FAILURE_HOOK();
    }
}

```

```

// PLL0 and PLL1 initialization sequence

// Program PLL0 Divider Register
PLLDIG.PLL0DV.R = SPC5_PLL0_DV_RFDPHI1(SPC5_PLL0_RFDPHI1_VALUE) |
                   SPC5_PLL0_DV_RFDPHI(SPC5_PLL0_RFDPHI_VALUE) |
                   SPC5_PLL0_DV_PREDIV(SPC5_PLL0_PREDIV_VALUE) |
                   SPC5_PLL0_DV_MFD(SPC5_PLL0_MFD_VALUE);

// Power up PLL0
MC_ME.DRUN_MC.B.PLL0ON = 1;

// Poll PLL0SR[LOCK] until it asserts
while (PLLDIG.PLL0SR.B.LOCK == 0)
{
}

// PLLDIG.PLL1CR.R = 0U;
// Program PLL1 Divider Register
PLLDIG.PLL1DV.R = SPC5_PLL1_DV_RFDPHI(SPC5_PLL1_RFDPHI_VALUE) |
                   SPC5_PLL1_DV_MFD(SPC5_PLL1_MFD_VALUE);

// Power up PLL1
MC_ME.DRUN_MC.B.PLL1ON = 1;
// Poll PLL1SR[LOCK] until it asserts
while (PLLDIG.PLL1SR.B.LOCK == 0)
{
}

// Engage Normal mode
PLLDIG.PLL0CR.B.CLKCFG = 3U;
PLLDIG.PLL1CR.B.CLKCFG = 3U;

/* Switches to DRUN mode (current mode) in order to update the
   settings.*/
if (SPC5SetRunMode(SPC5_RUNMODE_DRUN) == 0) {
    SPC5_CLOCK_FAILURE_HOOK();
}

/* Run modes initialization, note writes to the MC registers are verified
   by a protection mechanism, the operation success is verified at the
   end of the sequence.*/
MC_ME.IS.R = 8U; /* Resetting I_ICONF */
MC_ME.ME.R = SPC5_ME_ME_BITS;
MC_ME.SAFE_MC.R = SPC5_ME_SAFE_MC_BITS;
MC_ME.DRUN_MC.R = SPC5_ME_DRUN_MC_BITS;
MC_ME.RUN_MC[0].R = SPC5_ME_RUN0_MC_BITS;
MC_ME.RUN_MC[1].R = SPC5_ME_RUN1_MC_BITS;
MC_ME.RUN_MC[2].R = SPC5_ME_RUN2_MC_BITS;
MC_ME.RUN_MC[3].R = SPC5_ME_RUN3_MC_BITS;
MC_ME.HALT_MC.R = SPC5_ME_HALTO_MC_BITS;
MC_ME.STOP_MC.R = SPC5_ME_STOP0_MC_BITS;

if ((MC_ME.IS.B.I_ICONF & 1U) == 1U) {
    /* Configuration rejected.*/
    SPC5_CLOCK_FAILURE_HOOK();
}

/* Peripherals run and low power modes initialization.*/
MC_ME.RUN_PC[0].R = SPC5_ME_RUN_PC0_BITS;
MC_ME.RUN_PC[1].R = SPC5_ME_RUN_PC1_BITS;
MC_ME.RUN_PC[2].R = SPC5_ME_RUN_PC2_BITS;
MC_ME.RUN_PC[3].R = SPC5_ME_RUN_PC3_BITS;
MC_ME.RUN_PC[4].R = SPC5_ME_RUN_PC4_BITS;
MC_ME.RUN_PC[5].R = SPC5_ME_RUN_PC5_BITS;
MC_ME.RUN_PC[6].R = SPC5_ME_RUN_PC6_BITS;
MC_ME.RUN_PC[7].R = SPC5_ME_RUN_PC7_BITS;
MC_ME.LP_PC[0].R = SPC5_ME_LP_PC0_BITS;
MC_ME.LP_PC[1].R = SPC5_ME_LP_PC1_BITS;
MC_ME.LP_PC[2].R = SPC5_ME_LP_PC2_BITS;
MC_ME.LP_PC[3].R = SPC5_ME_LP_PC3_BITS;
MC_ME.LP_PC[4].R = SPC5_ME_LP_PC4_BITS;
MC_ME.LP_PC[5].R = SPC5_ME_LP_PC5_BITS;
MC_ME.LP_PC[6].R = SPC5_ME_LP_PC6_BITS;
MC_ME.LP_PC[7].R = SPC5_ME_LP_PC7_BITS;

/* Switches again to DRUN mode (current mode) in order to update the
   settings.*/
if (SPC5SetRunMode(SPC5_RUNMODE_DRUN) == 0) {
    SPC5_CLOCK_FAILURE_HOOK();
}

```