PLL Configuration for the EK-TM4C1294XL

July 15, 2018

After several months of working with the Tiva C EK-TM4C123GXL board, I finally had an opportunity to try out its more advanced cousin, the EK-TM4C1294XL. For the most part, programming the TM4C1294 is similar to the TM4C123, but there are some key differences. One of them is the way in which the PLL is configured to drive the system clock (SysClk). SysClk is the main "ticker" that controls the CPU frequency and the on-chip peripherals.

Figure 1. Texas Instruments Tiva C Series EK-TM4C1294XL

The TM4C1294 has a different set of registers for clock control compared to the TM4C123, and different driver code is required to enable the PLL to run from the board's crystal oscillator. The datasheet has a section on Clock Control in Chapter 5, and a step-by-step procedure explaining how to do this. In general, I prefer to follow the datasheet, and program the Tiva C boards using direct register modification. However, the datasheet steps didn't work for me this time, and I could not get the PLL to initialize properly. At a certain point after initialization, the board would fault. So, I decided to look into the TivaWare API implementation of the SysCtlClockFreqSet function, located in the sysctl.c module, to see what it was doing. What I found in the API source was surprising, and pointed to some potential problems in both the datasheet procedure and in the TivaWare implementation itself. Let me explain.

The TM4C1294 datasheet says that to initialize SysClk to use the PLL from the main oscillator, the OSCSRC field in the RSCLKCFG register must be set to MOSC (0x3), which means to use the main oscillator as the clock source. This is stated in Step 4:

4. Set the `OSCSRC` field to 0x3 in the **RSCLKCFG** register at offset 0x0B0.

The TivaWare API does this, but it also sets another field called, PLLSRC, to use the main oscillator. The datasheet doesn't mention PLLSRC in the step-by-step. However, it's clear from its description and how it's referred to in other parts of the datasheet, that PLLSRC *must* be set to MOSC. Essentially, if the PLLSRC isn't set to MOSC, then the PLL is not going to be driven by the main oscillator, and this wouldn't achieve the objective. So at a minimum, it seems that a key step was left out of the datasheet procedure. Or was it?

The strange thing is that although the TivaWare's SysCtlClockFreqSet function correctly sets the PLLSRC field, it still sets the OSCSRC field, which controls a completely different clock source, namely the OSCCLK. The diagram below shows the clock path from the external crystal (OSC0 and OSC1) through the MOSC multiplexer, and down to the OSCCLK and PLL branches. The green line shows the preferred path for the PLL and the red line shows the OSCCLK branch. At the end of the path is the SysClk output, which is connected to the CPU.

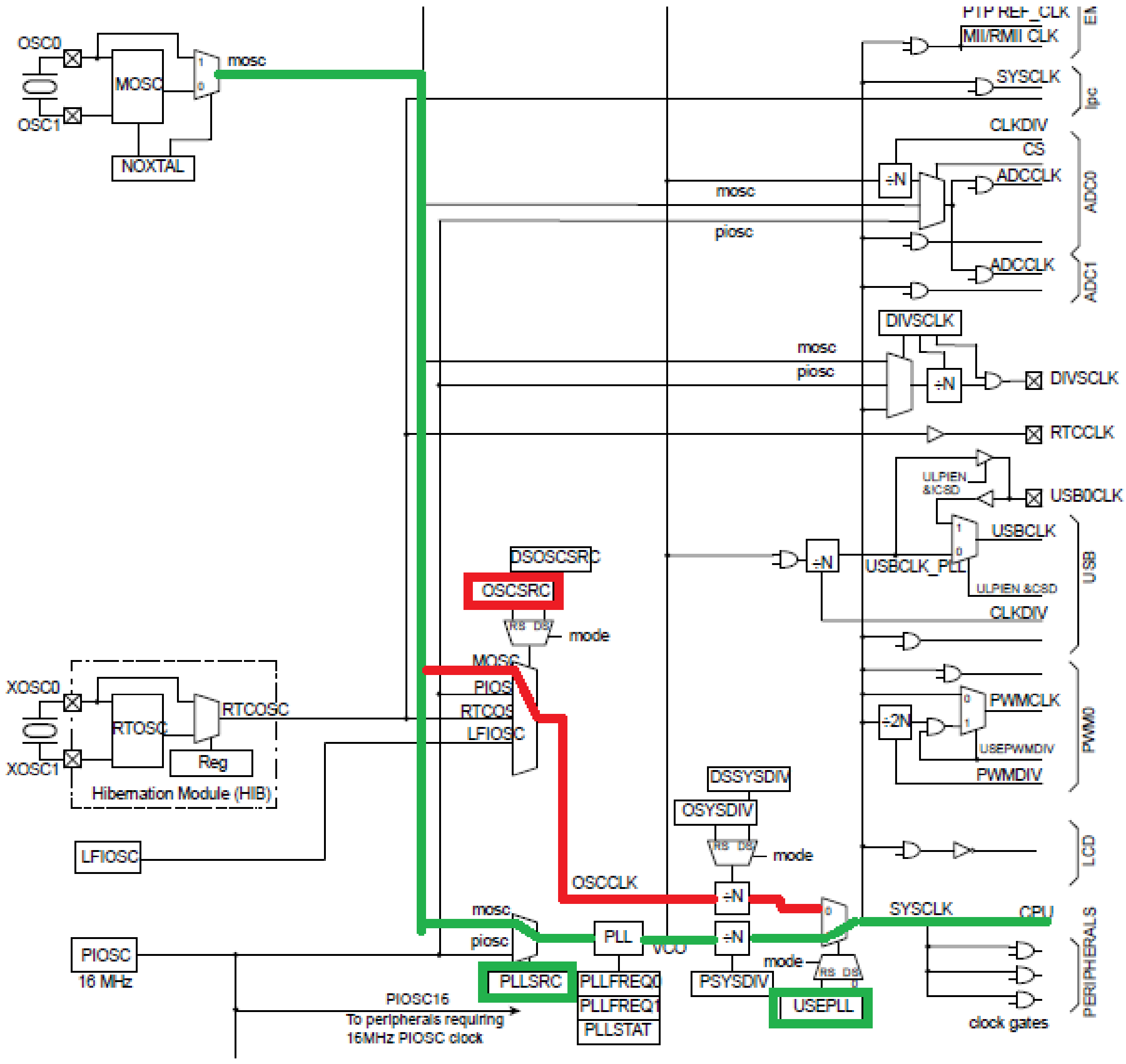


Figure 2. TM4C1294 Clock Tree Diagram

Basically, the clock tree diagram shows that SysClk can be driven by either the PLL or OSCCLK, and that these are two mutually exclusive clock sources. The source that gets used depends on the setting of the USEPLL bit at the end of the path. This is another configuration field in the RSCLKCFG system register. If the USEPLL bit is set, then the PLL is used as the clock source for the CPU and the red line from the OSCCLK source will be cut off. In other words, OSCCLK and the setting of OSCSRC will have no effect. If that's true, then why does the datasheet say to set OSCSRC in a procedure for initializing/configuring the PLL? Furthermore, why does the TivaWare API set *both* OSCSRC and PLLSRC?

I also looked into Daniel Valvano's implementation (http://users.ece.utexas.edu/~valvano/arm/), and found that this code is doing the same thing, i.e. setting both OSCSRC and PLLSRC to use the main oscillator. There is a comment above the

OSCSRC assignment which says, it's to "temporarily get run/sleep clock from the 25MHz main oscillator", but no reason indicating why a temporary *main* oscillator source is needed. Presumably, the run/sleep sleep clock could continue to come from the internal 16MHz oscillator (PIOSC) before the PLL locks and is enabled, just as it does when the board first powers up.

Seeking a definitive answer, I posted the question on the TM4C Microcontrollers Forum (https://e2e.ti.com/support/microcontrollers/tiva_arm/). This was the first time I'd posted anything to official TI support and, being unaware, I received an admonishment for using direct register modifications (DRM) instead of the TivaWare API. They have a rule in the forum not to support DRM questions and, more than that, to discourage you from using anything but the API. Since I had been perusing the SysCtlClockFreqSet function, however, I could at least relate a question to the API's implementation, which was "why does the API set the OSCSRC field to use the main oscillator when configuring the PLL as the clock source?".

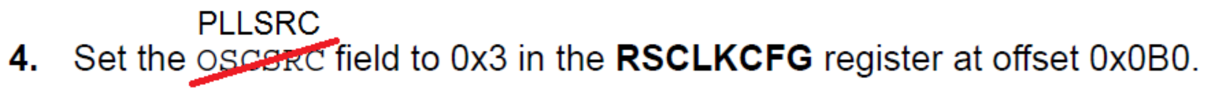
Unfortunately, the answer I received wasn't definitive enough. The TI support representative acknowledged that there was probably a *missing step* in the datasheet concerning the setting of PLLSRC, but could not provide an answer as to whether setting OSCSRC was correct and necessary. He did, however, point me to an errata that confirmed my sucspicion that setting OSCSRC to MOSC might be a mistake. The errata is entitled, *SYSCTL#23 MOSC as the Source to OSCCLK may Cause a bus Fault on Reset.* It says:

In TivaWare 2.1.2 or earlier, the SysCtlClockFreqSet sets the PLL source as MOSC when the API is called with the parameter SYSCTL_OSC_MAIN. It sets the OSCCLK as MOSC **which is not required** and may cause this issue.

In the case of the ROM version of the API, this may cause the system to fault. Now, I wasn't using the ROM version, but a fault warning is an obvious concern. The errata goes on to say that in order to workaround the issue, the SysCtlClockFreqSet function in the TivaWare API should be modified to *restore* the OCSSRC value to its original, default value (PIOSC). In other words, after OSCSRC is set to MOSC, it should then be undone!

This brings me to another detail I haven't mentioned so far. While looking at the TivaWare API source, I noticed that in the SysCtlClockFreqSet function, after setting both OSCSRC and PLLSRC to use the main oscillator, OSCSRC is set back to it's original, default value in the very last line of the function. All code paths in the function run through this. This didn't make any sense until after reading the errata: what I was observing was probably the prescribed workaround to prevent the system fault.

OK, but why is OSCSRC ever being set in the first place, especially if the objective is only to use the PLL? On this issue, I never received a clear answer on the TM4C forum. So, I've had to draw my own conclusion. Until a better explanation is put forth, I think that Step 4 in the datasheet has a typo. Let me be so bold as to *unofficially* correct it now:



PLLSRC

4. Set the ~~OSCSRC~~ field to 0x3 in the **RSCLKCFG** register at offset 0x0B0.

In my opinion, setting OSCSRC is just plain wrong. As the clock tree and errata show, the thing to do is leave the OSCCLK alone, if driving the PLL from the main oscillator is your only goal. Furthermore, while the OSCCLK may be used in deep sleep (DS), it's the DSOSCSRC field and not the OSCSRC field that's used to set the clock source in that mode, according to the clock tree. So I don't think deep sleep mode is the reason for setting OSCSRC either.

Conclusion

In writing my own PLL initialization function, I set *only* the PLLSRC field. I've also added a line to set the OSCRNG flag in the MOSCTL register to increase the drive strength of the main oscillator, since the crystal frequency of the EK-TM4C1294XL is 25MHz. This isn't mentioned in the datasheet's procedure, but the TivaWare API likewise performs this step. The code below has been working successfully for me at both 120MHz and 80MHz, and can be modified to support additional frequencies.

```
// PLL_HAL.h
#ifndef PLL_HAL_H
#define PLL_HAL_H

typedef enum {
         SYSCLK_80,
         SYSCLK_120
} SysClkFreq_t;

//------------------------- PLL_Init ---------------------------
// Initializes the PLL to generate the requested SysClk frequency.
// Inputs:  freq - the requested frequency.
// Outputs:  none.
void PLL_Init(SysClkFreq_t freq);

#endif
Copy


//------------------------------------ PLL_Init --------------------------------------
// Initializes the PLL to generate the requested SysClk frequency.  Assumes this will
// be called immediately after POR and not more than once.
//
// Inputs:  freq - the requested SysClk frequency
// Outputs: none.
// Notes:The steps here were derived from information in the 1294 data sheet,
//                          in particular section 5.3 Initialization and Configuration, and also
//                          the source code for SysCtlClockFreqSet in sysctl.c and the PLL example from
//                          Valvano's examples at http://users.ece.utexas.edu/~valvano/arm/PLL_4C1294.zip.
//                          There were differences in each source.  The implementation here attempts
//                          fix an apparent typo in the data sheet, remove what seems to be unncessary steps,
//                          and reorder them where it's feasible and makes sense to do so.
//
//                          After power-on reset (POR) the CPU will be using the precision internal
//                          oscillator (PIOSC) which runs at 16MHz.  There are three main steps to
//                          initializing/using the PLL:
//                                   1) Enable the main oscillator (MOSC).
//                                   2) Configure and enable the PLL.
//                                   3) Reconfigure SysClk to use the PLL as its source.
//                          Each main step has one or more substeps involving direct register access.
//
void PLL_Init(SysClkFreq_t freq)
{
         //
```

```
// STEP 1:  Enable the MOSC
//

// Apply sub-steps 1-3 together to preserve the current MOSCCTL bit values.
uint32_t mosc = SYSCTL_MOSCCTL_R;

// 1) Power up the MOSC (main oscillator) by clearing the "no crystal" bit.
// The crystal on the EK-TM4C1294XL board is 25MHz and connected to OSC0 and OSC1.
mosc &= ~SYSCTL_MOSCCTL_NOXTAL;

// 2) Enable power to the main oscillator by clearing the power down bit.
mosc &= ~SYSCTL_MOSCCTL_PWRDN;

// 3) Specify high oscillator range (>= 10 MHz).
// This substep is peformed by SysCtlClockFreqSet, which says is to "Increase
// the drive strength for MOSC of 10MHz and above".  It is not used by Valvano
// or mentioned in the data sheet configuration, but should in theory be set.
mosc |= SYSCTL_MOSCCTL_OSCRNG;

// 3) Set and wait until sufficient time has passed for the MOSC to reach the expected 25MHz frequency
// of the crystal.  The resulting MOSCCTL value will be 0x10.
SYSCTL_MOSCCTL_R = mosc;

//This is checking the power up masked interrupt status bit.
while ((SYSCTL_RIS_R & SYSCTL_RIS_MOSCPUPRIS) == 0) {}

//
// STEP 2:  Configure and enable the PLL
//

// 5) Clear and set the PLL input clock source to be the MOSC.
// It appears that this substep is not mentioned in the data sheet.  It says
// to set the OSCSRC field instead.  SysCtlClockFreqSet and Valvano both set PLLSRC,
// but then they also set OSCSRC as well. SysCtlClockFreqSet restores OSCSRC after
// enabling the PLL due errata #23, but Valvano does not.  It seems unnecessary to ever
// set OSCSRC given that the PLL will be used.  There is no explanation in the
// datasheet as to why OSCCRC must be configured to use MOSC temporarily while the PLL
// is being configured, so leaving it out here.
SYSCTL_RSCLKCFG_R = (SYSCTL_RSCLKCFG_R & ~SYSCTL_RSCLKCFG_PLLSRC_M) | SYSCTL_RSCLKCFG_PLLSRC_MOSC;

// Steps 6-8 are described in the data sheet tables (pgs. 237-238)
// fin = fxtal / ((Q+1)(N+1))
// MDIV = MINT + (MFRAC / 1024)
// fvco = fin * MDIV
// SysClk = fvco / (PSYSDIV + 1)

// 6) For fxtal= 25MHz, set Q = 0, N = 4 => fin = 25MHz/((0+1)(4+1)) = 5MHz
SYSCTL_PLLFREQ1_R = 0x4;

// 7) Set MFRAC = 0
SYSCTL_PLLFREQ0_R &= ~SYSCTL_PLLFREQ0_MFRAC_M;

// 8) Set MINT = 96 = 0x60 => fvco = (5MHz * 96) = 480MHz
SYSCTL_PLLFREQ0_R = (SYSCTL_PLLFREQ0_R & ~SYSCTL_PLLFREQ0_MINT_M) | 0x60;

// 9) Power up the PLL.  It will take some time to settle and lock the requested frequency.
SYSCTL_PLLFREQ0_R |= SYSCTL_PLLFREQ0_PLLPWR;

// 10) Wait until the PLL is powered and locked.
while ((SYSCTL_PLLSTAT_R & SYSCTL_PLLSTAT_LOCK) == 0) {}

//
// STEP 3:  Reconfigure SysClk to use the PLL.
//

// 11) Set the timing parameters for the main Flash and EEPROM memories.
// These settings will take effect once the MEMTIMU bit is set below.
// Clear the relevant field bits making sure to leave the reserved bits unchanged.
uint32_t memtim0 = SYSCTL_MEMTIM0_R;
memtim0 &= ~(SYSCTL_MEMTIM0_EBCHT_M | SYSCTL_MEMTIM0_EBCE | SYSCTL_MEMTIM0_EWS_M |
         SYSCTL_MEMTIM0_FBCHT_M | SYSCTL_MEMTIM0_FBCE | SYSCTL_MEMTIM0_FWS_M);

uint32_t physdiv;

switch (freq) {

case SYSCLK_80:
         // For CPU frequency 60MHz < f <= 80MHz:
         //       EBCHT = FBCHT = 0x4
         //       EBCE = FBCE = 0
         //       FWS = EWS = 0x3;
         // Since the reserved bits 20 and 4 are showing 0x1, the resulting MEMTIM0 value should be:  0x01130113
         memtim0 |= (SYSCTL_MEMTIM0_EBCHT_2_5 | SYSCTL_MEMTIM0_FBCHT_2_5 | (0x3 << SYSCTL_MEMTIM0_EWS_S) | 0x3);
         physdiv = 0x5;
         break;

case SYSCLK_120:
default:
```

```
                // For CPU frequency 100MHz < f <= 120MHz:
                //        EBCHT = FBCHT = 0x6
                //        EBCE = FBCE = 0
                //        FWS = EWS = 0x5;
                // Since the reserved bits 20 and 4 are showing 0x1, the resulting MEMTIM0 value should be:  0x01950195
                memtim0 |= (SYSCTL_MEMTIM0_EBCHT_3_5 | SYSCTL_MEMTIM0_FBCHT_3_5 | (0x5 << SYSCTL_MEMTIM0_EWS_S) | 0x5);
                physdiv = 0x3;
                break;
        }

        SYSCTL_MEMTIM0_R = memtim0;

        // Apply steps 12-14 together.
        uint32_t rsclkcfg = SYSCTL_RSCLKCFG_R;

        // 12) Set the PLL System Clock divisor, e.g. 3 => SysClk = 480MHz / (1+3) = 120MHz.
        rsclkcfg = (rsclkcfg & ~SYSCTL_RSCLKCFG_PSYSDIV_M) | physdiv;

        // 13) Use PLL as the system clock source
        rsclkcfg |= SYSCTL_RSCLKCFG_USEPLL;

        // 14) Apply the MEMTIMU register value and upate the memory timings set in MEMTIM0.
        rsclkcfg |= SYSCTL_RSCLKCFG_MEMTIMU;

        SYSCTL_RSCLKCFG_R = rsclkcfg;

}
```