APPLICATION NOTE

PAC52XX Clock Control Firmware Design

Power Application Controller^{TM}

Marc Sousa
Senior Manager, Systems and Firmware

TABLE OF CONTENTS

OVERVIEW

The PAC52XX family of devices have a sophisticated clock control system that allow applications different clocking options that can allow for high-efficiency, low power and accurate time-base features.

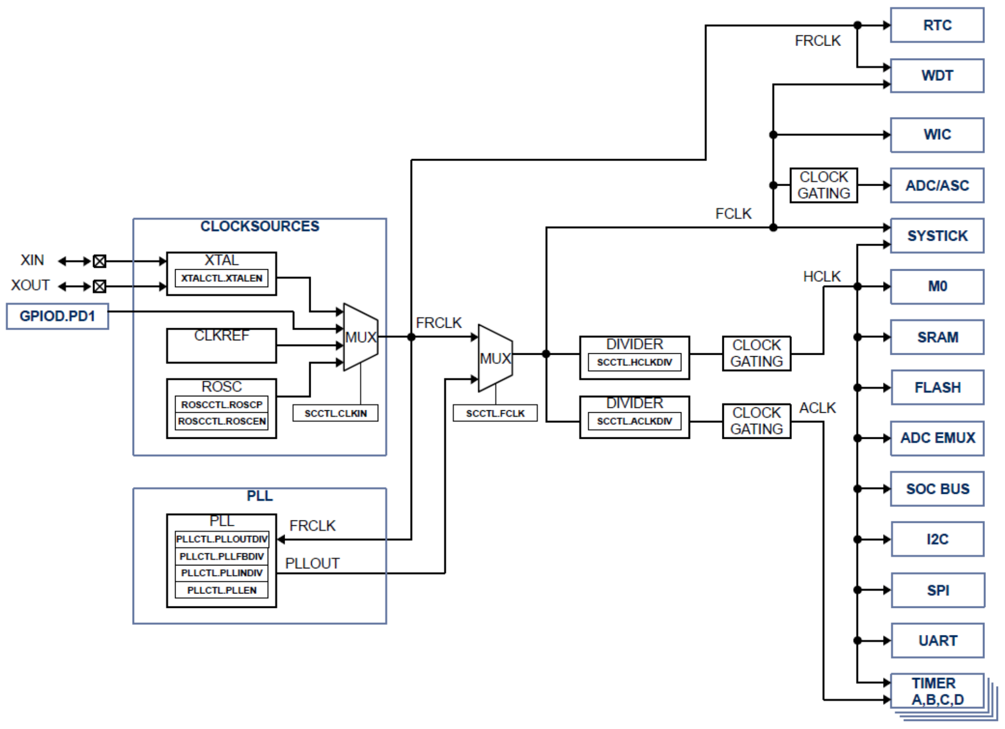
There are multiple types of system clock inputs ranging from high-accuracy RC oscillators to low-power ring oscillators, crystal drivers and external clocks.

For high-frequency applications, there is a PLL that can be used to drive high-speed peripherals such as the ADC and timers, as well as flexibly configured clock dividers that allow different peripherals different clock source frequencies.

For more detailed information on peripheral configuration of the clock control system, see the PAC52XX User Guide.

Clock Control System Block Diagram

The clock control system in the PAC52XX is shown in the diagram below.



Clock Sources

The clock control system has four clock input sources:

- CLKREF – 4MHz 1% clock reference
- ROSC – Internally generated ring-oscillator
- XTAL – Crystal driver
- EXTCLK – External clock input

A clock source MUX is used to select the system clock source that is used from a register. The output clock from this MUX is referred to as **FRCLK**.

CLKREF is a 1% trimmed 4MHz clock. For most applications using the PAC52XX, this clock would be the source. It is ideal for use with the on-chip PLL, because it is a highly accurate time-base.

ROSC is an internally generated ring oscillator that can be configured from 8.3MHz to 28.7MHz in four steps. Because of the error in the ROSC, it is generally not recommended for applications that have high-accuracy clock needs, and not recommended for a PLL source clock.

The **XTAL** crystal driver works with external crystal oscillators from 2MHz to 10MHz.

EXTCLK allows an external clock to be supplied on PD1 as the system clock source.

PLL

The PAC52XX family of controllers contains a Phase Lock Loop (PLL) that can be used to multiply an accurate input clock (CLKREF) to a much higher output clock (PLLOUT). This circuit would be used for a high-frequency clock input for the Cortex-M0 core, as well as the other system peripherals such as timers, ADC, communications peripherals, etc.

The PLL configuration is set via a set of registers that control the output, feedback and input divider that generate the output frequency.

The clock control system allows the system clock to be selected between FRCLK and PLLCLK via another MUX. The output of this MUX is the FCLK system clock.

Peripheral Clock Selection

FCLK is the clock input for the two main peripheral clocks in the PAC52XX: HCLK and ACLK.

All peripherals in the system are clocked from one or more of the following clock inputs:

- FRCLK
- FCLK
- HCLK
- ACLK

FRCLK is the selected system clock controller clock input, and is used to clock peripherals that need to operate in the device sleep modes. The RCLK, HCLK and ACLK clocks are turned off when in some of the sleep modes of the PAC52XX.

ACLK and HCLK allow dividers to divide their input clock (FCLK) to a frequency where the peripherals can efficiently operate.

The table below shows all the system peripherals and which clocks are available for those peripherals during operation.

Peripheral	Description	FRCLK	FCLK	HCLK	ACLK
RTC	Real-time Clock (GP Timer)	X			
WDT	Watchdog Timer	X	X		
WIC	Wakeup Interrupt Controller		X		
ADC	Analog to Digital Converter		X		
SysTick	SysTick Timer		X	X	
M0	ARM Cortex-M0 Core			X	
SRAM	SRAM			X	
FLASH	FLASH program memory			X	
ADC EMUX	Auto ADC mux selector			X	
SOC BUS	System on Chip bus			X	
I2C	I2C communications peripheral			X	
SPI	Synchronous Peripheral Interface			X	
UART	Universal Asynchronous Receive Transmit			X	
Timers	Timers A, B, C, D			X	X

USING THE INTERNAL OSCILLATOR

To use the internal oscillator (ROSC) as the system clock, the user needs to perform a few steps. The critical need is to make sure that the system clock to the ARM Cortex-M0 remains active at all times. If the MUX is switched to a clock that is not running, then the ARM Cortex-M0 will stop working and cannot be recovered until after a power cycle.

To configure the internal oscillator, the user should follow these steps in this order:

1. Select the internal oscillator frequency [*optional*: if not planning on using the default]
2. Enable the internal oscillator
3. Switch the FCLK MUX select to FRCLK [*optional*: if the PLL is FRCLK]
4. Switch the FRCLK MUX select to ROSC

At this point, the system will be running using the internal oscillator. At this point the user may change the HCLK and ACLK dividers, and also disable other clock sources if not needed (like the PLL).

Below is the code using the PAC52XX SDK that will enable these changes.

```
pac5xxx_sys_osc_config(OSCCTL_OSCP_28P17MHZ);        // Set Frosc to desired frequency
pac5xxx_sys_osc_enable();                            // Enable ROSC
pac5xxx_sys_ccs_config(CCSCTL_CLKIN_INTOSC,          // Select FRCLK = ROSC,
                       CCSCTL_HCLKDIV_DIV1,          // Set HCLK/ACLK dividers
                       CCSCTL_ACLKDIV_DIV1);         // to be /1
pac5xxx_sys_ccs_frclk_select();                      // Select FCLK = FRCLK
```

In addition, if the PLL was running and you want to save energy, you could make sure that it is disabled:

```
pac5xxx_sys_pll_disable();
```

USING THE CRYSTAL DRIVER

To use the crystal driver (XTAL) with an external crystal, the user needs to perform a few steps. The The critical need is to make sure that the system clock to the ARM Cortex-M0 remains active at all times. If the MUX is switched to a clock that is not running, then the ARM Cortex-M0 will stop working and cannot be recovered until after a power cycle.

To configure the crystal driver, the user should follow these steps in this order:

1. Enable the crystal driver
2. Switch the FCLK MUX select to FRCLK [*optional*: if the PLL is FRCLK]
3. Switch the FRCLK MUX select to XTAL

At this point, the system will be running using the crystal driver. The user may change the HCLK and ACLK dividers, and also disable other clock sources if not needed (like the PLL).

Below is the code using the PAC52XX SDK that will enable these changes.

```
pac5xxx_sys_xtal_enable();                           // Enable XTAL
pac5xxx_sys_ccs_config(CCSCTL_CLKIN_XTAL,            // Select FRCLK = XTAL,
                       CCSCTL_HCLKDIV_DIV1,          // Set HCLK/ACLK dividers
                       CCSCTL_ACLKDIV_DIV1);         // to be /1
pac5xxx_sys_ccs_frclk_select();                      // Select FCLK = FRCLK
```

In addition, if the PLL was running and you want to save energy, you could make sure that it is disabled:

```
pac5xxx_sys_pll_disable();
```

USING THE 4MHZ REFERENCE CLOCK

The 4MHz Reference Clock (CLKREF) is a 1% trimmed oscillator that is always running, and cannot be disabled. This clock source can be used as-is, or as input to the PLL for a much higher frequency clock. This section discusses how to use the 4MHz clock as-is.

If the PLL is already running, and the user wants to change to the 4MHz CLKREF, the user should follow these:

- Switch the FCLK MUX select to FRCLK [*optional*: if FRCLK = PLLCLK]

At this point, the system will be running using the CLKREF. The user may change the HCLK and ACLK dividers at any time, and also disable other clock sources if not needed (like the PLL).

Below is the code using the PAC52XX SDK that will enable these changes.

```
pac5xxx_sys_ccs_config(CCSCTL_CLKIN_CLKREF,          // Select FRCLK = CLKREF,
                       CCSCTL_HCLKDIV_DIV1,          // Set HCLK/ACLK dividers
                       CCSCTL_ACLKDIV_DIV1);         // to be /1
pac5xxx_sys_ccs_frclk_select();                      // Select FCLK = FRCLK
```

In addition, if the PLL was running and you want to save energy, you could make sure that it is disabled:

```
pac5xxx_sys_pll_disable();
```

USING THE PLL

Some of the PAC52XX peripherals that require a high-frequency, accurate time-base must use the PLL. For example, the UART and Timers (A, B, C, D) and other peripherals require a high-accuracy time-base and the ARM Cortex-M0 often needs to a high-frequency clock that can only be supplied from the on-chip PLL.

The PLL requires the CLKREF as its input clock. The output of the PLL (PLLCLK) is derived from the input clock and the values of the three dividers that are a part of the PLL:

- Feedback divider
- Input divider
- Output divider

The PLLOUT frequency is calculated by the following equation:

$$PLLOUT = \frac{PLLIN * PLLFBDIV}{PLLINDIV * PLLOUTDIV * 2}$$

Where the following are true:

- PLLOUT is the PLL output frequency in MHz
- PLLIN is the PLL input frequency in MHz (4MHz for CLKREF)
- PLLINDIV is the PLL input divider (2 to 33)
- PLLFBDIV is the PLL feedback divider (2 to 513)
- PLLOUTDIV is the PLL output divider (1 to 16)

In addition, the PLL configuration has constraints for the input and output divider selection, as shown below.

The input clock frequency and input clock divider selection must follow the formula below:

$$1\text{MHz} \le \frac{PLLIN}{PLLINDIV} \le 25\text{MHz}$$

The output clock frequency and output clock divider selection must follow the formula below:

$$100\text{MHz} \le PLLOUT * PLLOUTDIV \le 250\text{MHz}$$
$$PLLOUTDIV \ge 1$$

The table below shows some pre-calculated PLL output frequency settings using the 4MHz CLKREF as the PLL input clock.

PLL output	PLLOUTDIV	PLLFBDIV	PLLINDIV
10MHz	0x01 (/[1*2])	0x008 (*10)	0x0 (/2)
16.8MHz	0x05 (/[5*2])	0x052 (*84)	0x0 (/2)
20MHz	0x01 (/[1*2])	0x012 (*20)	0x0 (/2)
25MHz	0x01 (/[1*2])	0x019 (*25)	0x0 (/2)
30MHz	0x01 (/[1*2])	0x01C (*30)	0x0 (/2)
33.333MHz	0x01 (/[1*2])	0x030 (*50)	0x1 (/3)
40MHz	0x01 (/[1*2])	0x026 (*40)	0x0 (/2)
50MHz	0x01 (/[1*2])	0x030 (*50)	0x0 (/2)
60MHz	0x01 (/[1*2])	0x03A (*60)	0x0 (/2)
70MHz	0x01 (/[1*2])	0x044 (*70)	0x0 (/2)
80MHz	0x01 (/[1*2])	0x04E (*80)	0x0 (/2)
90MHz	0x01 (/[1*2])	0x058 (*90)	0x0 (/2)
100MHz	0x01 (/[1*2])	0x062 (*100)	0x0 (/2)

After the PLL dividers are configured and **before** the PLL output can be enabled, there has to be a mandatory delay of 500us. This allows the PLL time to *lock* for the clock to become stabilized before the system starts using it as it's time-base.

In order to simplify configuration of the PLL, the PAC52XX SDK provides some functions that allow the user to configure the PLL by calling a single function with the output frequency specified as the only argument. The function is below.

```
pac5xxx_sys_pll_config(80);    // Configure PLLOUT for 80MHz
```

Calling this function performs the following steps:

- Set FRCLK to CLKREF
- Disable the internal oscillator (done to save energy)
- Configure the PLL input, feedback and output dividers
- Delay 500us
- Switch FCLK to PLLOUT

It is also possible for the user to individually call each of these functions themselves by the PAC52XX SDK functions.

The relevant SDK functions are shown in the table below.

Function	Description
`void pac5xxx_sys_ccs_pll_select(void)`	Sets FCLK to be PLLOUT

`void pac5xxx_sys_ccs_config_dividers(`
`  uint8_t input_div,`
`  uint8_t feedback_div,`
`  uint8_t output_div);`
	Configures the PLL dividers.
	
	Note that this function will **not** check to see if the dividers are within correct range, so the user must take care to do this according to the guidelines above.

`void pac5xxx_sys_ccs_pll_enable(int delay);`	Delays for 500us. After returning from this function, the user must change the FCLK to

	PLLOUT by calling the `pac5xxx_sys_ccs_pll_select()` function.
`void pac5xxx_sys_ccs_pll_disable(void)`	Immediately disables the PLL. Before calling this, the user must make sure that the FCLK is set to FRCLK with a valid clock source, so that the system continues to have a valid clock input.

Clocking in Low-Power Modes

The PAC52XX has low-power modes that allow the system to go to sleep, to consume a minimum amount of current for applications that require low quiescent current.

The ARM Cortex-M0 has two low-power modes:

- Sleep Mode
- Deep Sleep Mode

In Sleep Mode, the CPU is put to sleep and can be woken up by any interrupt (timers, communications, etc.).

To save even more energy, deep sleep mode gates the ACLK, HCLK and some of the FCLK clocks. This effectively puts to sleep the ARM Cortex-M0 and most of the system peripherals. In this mode, only a few other peripherals are available to wake up the system:

- RTC (GP Timer)
- WDT
- GPIO

When the system enters deep sleep mode either the RTC (GP Timer) or WDT can be configured to wake the system up by enabling the disabled clocks (ACLK, HCLK and FCLK) and interrupting the Cortex-M0.

As an example, to change to deep sleep mode to save the maximum amount of energy, the firmware could implement the following:

```
void mcu_clock_config_low_power(void)
{
  pac5xxx_sys_ccs_config(CCSCTL_CLKIN_CLKREF, CCSCTL_ACLKDIV_DIV1, CCSCTL_HCLKDIV_DIV1);
  pac5xxx_sys_pll_disable();
  pac5xxx_arm_sleep_deep_enable();         // Configure system to use deep sleep mode
  pac5xxx_arm_sleep_wfi();                 // Go to sleep
}
```

This function changes to the REFCLK, without the PLL, disables the PLL and then sends the system to sleep. The RTC (GP Timer) or WDT could then wake up the system via an interrupt.

When this interrupt is serviced, it could then call a function like shown below that could re-configure the system for a high-speed clock:

```
void mcu_clock_config_high_power(void)
{
  pac5xxx_sys_ccs_config(CCSCTL_CLKIN_CLKREF, CCSCTL_ACLKDIV_DIV1, CCSCTL_HCLKDIV_DIV4);
  pac5xxx_sys_pll_config(80);
  pac5xxx_sys_osc_disable();
}
```

ABOUT ACTIVE-SEMI

Active-Semi, Inc. headquartered in Dallas, TX is a leading innovative semiconductor company with proven power management, analog and mixed-signal products for end-applications that require power conversion (AC/DC, DC/DC, DC/AC, PFC, etc.), motor drivers and control and LED drivers and control along with ARM microcontroller for system development.

Active-Semi's latest family of Power Application Controller (PAC)™ ICs offer high-level of integration with 32-bit ARM Cortex M0, along with configurable power management peripherals, configurable analog front-end with high-precision, high-speed data converters, single-ended and differential PGAs, integrated low-voltage and high-voltage gate drives. PAC IC offers unprecedented flexibility and ease in the systems design of various end-applications such as Wireless Power Transmitters, Motor drives, UPS, Solar Inverters and LED lighting, etc. that require a microcontroller, power conversion, analog sensing, high-voltage gate drives, open-drain outputs, analog & digital general purpose IO, as well as support for wired and wireless communication. More information and samples can be obtained from http://www.active-semi.com or by emailing marketing@active-semi.com

Active-Semi shipped its 1 Billionth IC in 2012, and has over 120 in patents awarded and pending approval.