Freescale Semiconductor
Application Note

Document Number:AN4734
Rev 0, 07/2013

Low-Power Pin Sampling Techniques Using Kinetis L Series

by: Chris Brown

Contents

1 Introduction

This application note, based on the Kinetis L Series 48 MHz devices, demonstrates analog low-power pin sampling techniques targeted for applications that interface with analog sensors such as photocells, thermistors, linear output accelerometers, linear output hall effect sensors, or other comparable devices. In addition, the techniques discussed in this application note can also be used in low-power zero crossing detection circuits.

Applications that employ these types of sensors can range from remote monitoring/control system nodes (such as security system nodes or smoke detectors) to position sensing applications to battery monitoring applications. These applications potentially require that the microcontroller perform some action when the output from the analog sensor to which they are monitoring is above or below some threshold level. For example, a security system node may want to alert a control unit when the ambient light in a room rises above a threshold level. In this case, the threshold level may be a fraction of VDD and the MCU should account for this to ensure that the correct signal is sent.

This application note discusses techniques for single-channel, multichannel, and high resolution low-power sampling. Example applications are provided that implement a simple program to showcase the low-power features of the Kinetis L series by setting an output pin high when the input is above a threshold and low when the input is below a threshold.

2 Using the on-chip comparator for low-power pin sampling techniques

The following three options are used when implementing the applications discussed in the Introduction :

- Add hardware (such as an external comparator)
- Use a smart sensor that communicates to the microcontroller via a digital communication protocol such as IIC or SPI (which is relatively expensive in both cost and power consumption)
- Use an on-chip analog peripheral such as an analog-to-digital converter or comparator

Employing an on-chip peripheral for this task is the best approach. However, choosing and implementing the appropriate peripheral within the power consumption constraints of your application requires careful consideration. The on-chip analog-to-digital converter can provide a high-resolution option capable of operating in high noise environments. However, the on-chip comparator can be a lower power and faster solution when monitoring analog sensors in a threshold detection application.

This application note assists in the development of an application which uses the comparator and other features of the Kinetis L Series to achieve the lowest possible power in your application. It is assumed that the application is performing no other tasks while monitoring the sensor output (as it would be in a fall detection application or a security system node application).

Example applications are provided for IAR Embedded Workbench 6.40 which uses the comparator in single channel, multiple channel, and high-resolution configurations.

2.1 Application design considerations

When developing a pin sampling algorithm or application, propagation delay and power consumption are the design parameters to be considered. The L Series comparator has two different modes that are user selectable to allow flexibility in your application. If propagation delay is critical in your application, then the high-speed mode should be selected (CMPx_CR1[PMODE] = 1). High-speed mode configures the module for short propagation delay at the expense of higher current draw. For details on the minimum propagation delay, refer to your device specific datasheet.

If propagation delay is not a critical parameter of your design, the low-speed mode should be selected (CMPx_CR1[PMODE] = 0). The low-speed mode significantly reduces the power consumption of the module when making comparisons at the expense of longer propagation delay.

While developing your application, selecting the appropriate low-power mode is a critical design consideration. The device will consume less power if the sleep mode is deeper. However, if the sleep mode is deeper it will require more time to wake-up the part. Refer to your device specific datasheet for more details on specific power consumption and wake-up time figures.

The provided example applications demonstrate LLS mode. For the target device (KL05), this is an excellent mode to use when the sampling period is on the order of tens of milliseconds. The comparator module (CMP) is operational in all power modes down to VLLS1 mode. Likewise, the Low-Power Timer, and Low-Power Oscillator are also available in all power modes down to VLLS1. Therefore, the provided example applications are capable of operating down to VLLS1 but for the purposes of these examples, LLS has been chosen. LLS mode was chosen for its balance of low current consumption and fast wake-up time.

3 Applications

The low-power applications discussed in this application note follow the same general scheme. There is a startup phase that includes initialization of the necessary modules, followed by a phase that configures and executes the low-power entry phase. Then the major part of the sampling is performed in an interrupt service routine and the part will return to the low-power mode immediately upon finishing the sampling process. Achieving the lowest power in this type of application requires the use of VLPR, if applicable, and the sleep-on-exit feature of the Cortex®-M core

NOTE

Some applications may not be able to use these features due to application specific requirements, but the example applications will make use of these features.

NOTE

The example applications utilize the low-power oscillator to establish a sampling frequency of 111.11 Hz. Therefore, the output waveforms will likely incur a significant amount of jitter and will not be an exact duplication of the input waveform. If your application requires low jitter and a more precise duplication of the input waveform, you will want to select an appropriate input clock and frequency for the LPTMR.

3.1 Single-Channel implementation

When implementing a single-channel, it is advantageous to use the triggered mode function of the comparator. In this mode, a trigger event initiates a two-stage compare sequence. The first stage of the compare sequence wakes the part to a wait mode where the CMP and associated 6-bit DAC will also be enabled. The second stage triggers the CMP to capture the result of the compare operation. Using this feature, the core need only be awakened if the comparator senses that the desired threshold has been crossed. The initialization for this option is as follows:

1. Clear control register 1 (CMPx_CR1 = 0x00).
2. Clear control register 0 (CMPx_CR0 = 0x00).
3. Set the Trigger Mode Enable bit in control register 1 (CMPx_CR1[TRIGM] = 1).
4. Set the appropriate interrupt enable bit in the Status and Control Register (SCR). If your application needs to detect values greater than a specific value, set the Comparator Interrupt Enable Rising bit (CMPx_SCR[IER] = 1); else, set the Comparator Interrupt Enable Falling bit (CMPx_SCR[IEF] = 1).

   NOTE

   It is possible to set, the IER and IEF bits at the same time and this is a valid configuration. However, exercise caution when setting both bits as unwanted continuous interrupt requests can occur when using low-power modes.

5. Configure the 6-bit DAC Control Register according to your specific application. This calls for setting the DAC Enable bit (CMPx_DACCR[DACEN]) and setting the VOSEL bits to the appropriate value (CMPx_DACCR[VOSEL 5:0]). Remember that DACO = (Vin/64) * (VOSEL[5:0] + 1), where Vin may either be VREFH or VDD, depending on the value of CMPx_DACCR[VRSEL]. Refer to your part specific reference manual for more information.
6. Configure the MUX Control Register (CMPx_MUXCR) to select the appropriate pins for the plus and minus inputs of the comparator.
7. Enable the comparator by setting the Comparator Module Enable in Control Register 1 (CMPx_CR1[EN] = 1).

Once the preceding algorithm has been executed, the trigger source, which is the Low-Power Timer (LPTMR), must be configured correctly. The trigger mode is a two-stage trigger. Therefore, the LPTMR must be configured such that the second trigger is not issued before the analog comparator initialization delay has elapsed. The analog comparator initialization delay can be found in the device specific datasheet. The required compare value (LPTMRx_CNR) will therefore, change depending on the LPTMR configuration. If the prescaler is enabled, the delay in the second trigger event is one-half of the prescaler output period. If the prescaler is not enabled, the delay will be one-half of the prescaler clock period.

The algorithm for configuring the LPTMR is as follows:

1. First disable the LPTMR by clearing the Control Status Register (LPTMRx_CSR = 0x00).
2. Write the LPTMR Prescale Register (LPTMRx_PSR) appropriately (this register selects the LPTMR clock source and sets the prescale enable as well as the prescale value).

NOTE

The LPTMR directly controls how often the part is in the desired low-power mode, which has a direct effect on power consumption. Therefore, you should choose the lowest power clock and configure this clock for the longest period possible, within the constraints of your application, to achieve the lowest possible power consumption.

3. Set the LPTMR period by writing the Compare Value to the LPTMR Compare Register (LPTMRx_CNR = xx).

Once the LPTMR is configured, enable the comparator interrupt, enable the timer by setting the Timer Enable bit in the Control and Status Register (LPTMRx_CSR[TEN] = 1), and enter the desired low-power mode.

3.2 Multichannel implementation

When implementing more than one channel, the triggered mode is a viable option that provides useful power savings. However, the triggered mode may limit your ability to independently control multiple channels if there is only one comparator on your device. Likewise, if there are only two comparators on the device and application calls for more than two channels to be independently sampled and controlled, issues may again arise. Therefore, the LPTMR should be used to periodically wake the core, obtain the comparator results, and then reenter low-power mode. In this multichannel implementation method, all necessary application activities (enabling the comparator/DAC, obtaining the comparator results, and so on) are performed in the interrupt service routine. The initialization code before the initial low-power mode entry initializes only the LPTMR. The LPTMR initialization is as follows:

1. First disable the LPTMR by clearing the Control Status Register (LPTMRx_CSR = 0x00).
2. Write the LPTMR Prescale Register (LPTMRx_PSR) appropriately (this register selects the LPTMR clock source and sets the prescale enable as well as the prescale value).

NOTE

The LPTMR directly controls how often the part is in the desired low-power mode, which has a direct effect on power consumption. Therefore, you should choose the lowest power clock and configure this clock for the longest period possible, within the constraints of your application, to achieve the lowest possible power consumption.

3. Set the LPTMR period by writing the Compare Value to the LPTMR Compare Register (LPTMRx_CNR = xx).
4. Enable the LPTMR interrupt in the NVIC, if your application is utilizing power modes that are not low-leakage modes (VLPS and above). If you are entering a low-leakage power mode, the LPTMR must be enabled as a wake-up source in the LLWU. Consult your device specific reference manual for the correct module wake-up slot in the LLWU.
5. Enable the timer interrupt and timer itself by setting the Timer Interrupt Enable and Timer Enable bits in the CSR (LPTMRx_CSR[TIE] = LPTMRx_CSR[TEN] = 1)
6. Enter the desired low-power mode.

NOTE

To save power, ensure that the CMP and DAC are disabled before entering low-power mode.

The actions that should be performed in the interrupt service routine are as follows:

1. Enable the comparator and DAC.
2. (Optional) Write the PMC control register to enable VLPR.

NOTE

If the application utilizes VLPS and VLPR mode, it will not be necessary to perform this step. If you enter VLPS mode from VLPR, the part will exit VLPS mode directly into VLPR mode.

3. Wait for the analog comparator to initialize. This time will be the analog comparator initialization (refer to the device specific data sheet for this value) minus the time it has taken to write to the PMCG control registers and the time it will take to write to the CMP MUX.

NOTE

This step is required. It is advantageous to insert pin toggles when developing your application such that you can tune this delay to the exact wait time in the device specific datasheet. In doing this, the least amount of time will be wasted waiting for the CMP and DAC to settle.

4. Configure the CMP MUX for the appropriate channel.
5. Test the COUT bit in the Status and Control Register (CMPx_SCR[COUT]) and perform the appropriate action.
6. Repeat steps 4 and 5 as necessary until all channels have sampled as required by your application.
7. Disable the CMP and DAC by clearing the appropriate control registers (CMPx_CR1 = CMPx_DACCR = 0x00).
8. Service all necessary interrupt flags.

NOTE

Alternatively, you can allow the interrupt service routine to exit and place this algorithm elsewhere in your application. However, be aware that this will be less than optimal as the part will have to use valuable clock cycles stacking and un-stacking registers as it enters and exits the interrupt service routines.

3.3 12-Bit DAC / comparator implementation

The 12-bit DAC implementation is identical to the multichannel 6-bit DAC implementation except the 12-bit DAC implementation uses the 12-bit DAC, which is not internal to the CMP, and a comparator initialization function is used due to a slightly more complex CMP/DAC configuration.

NOTE

The 12-bit DAC is not present on all Kinetis L Series parts. Refer to your device specific reference manual to determine if this module is available.

All of the same parameters considered in the multichannel example (DAC settling time, DAC power consumption, and so on) must be taken into account when implementing this option. Refer to the following algorithm when initializing the 12-bit DAC for your application:

1. Ensure that the DAC control register 0 is in its reset state by clearing this register (DAC_C0 = 0x00).
2. Select the desired reference voltage by setting the DAC Reference Select bit (DAC_C0[DACRFS] = 1) or clearing the DAC Reference Select bit (DAC_C0[DACRFS] = 0)

NOTE

Each part may have different options for reference voltages. Please refer to the device specific reference manual for voltage reference options.

3. Enable High Power mode by clearing the Low Power Control bit (DAC_C0[LPEN] = 0).

NOTE

This will result in a higher power consumption from the DAC but will significantly reduce the settling time of the DAC, resulting in a lower average current for your application. Refer to the device specific datasheet for DAC settling time.

4. Ensure that the DAC buffer and DMA modes are disabled by clearing the DAC Control Register 1 (DAC_C1 = 0x00).
5. Write the desired DAC conversion value to the Data 0 Low and Data 0 High registers (DAC_DAT0L = <Lower 8 bits of conversion value>, and DAC_DAT0H = <Upper 4 bits of conversion value>).

6. Leave the DAC disabled to save power during the low-power modes (DAC_C0[DACEN} = 0).

To insert the algorithm into the interrupt service routine, follow the outline described in the Multichannel Implementation. In this algorithm, replace all 6-bit DAC enable/disable commands with the appropriate 12-bit DAC commands and select the 12-bit DAC as the inverted input reference (instead of the 6-bit DAC) when selecting the comparator channels. Consult the device specific reference manual for the value to write to the comparator channel mux.

4 Example applications

4.1 Common structures

Each example provided implements a common structure that will be discussed here. The general program outline is as follows:

1. Enable the desired low-power mode in the Power Mode Protection Register.
2. Configure the device for a VLPR capable clock mode (BLPI in this case).
3. Configure the I/O pins.
4. Initialize the CMP (if necessary).
5. Initialize the LPTMR.
6. Run a common code execution loop in RAM.

Note that VLPR is not necessarily a requirement of the applications but is required if you expect to meet the lowest possible current. The following code is executed in the main loop:

```
// Configure the part to allow VLLS3 and VLPR modes.
  SMC_PMPROT = (SMC_PMPROT_AVLP_MASK
                | SMC_PMPROT_ALLS_MASK
                | SMC_PMPROT_AVLLS_MASK);

  // Move to FBI mode using the 4 MHz internal clock (FIRC)
  tempint = fei_fbi(4000000, 1);

  // If the clock frequency is not 2 MHz, stop the program as there is an error.
  if(tempint != 2000000)
      while(1);

  // Turn off the clock monitors
  MCG_C6 &= ~MCG_C6_CME0_MASK;
  // Move to BLPx mode
  MCG_C2 |= MCG_C2_LP_MASK;
```

The LPTMR initialization function configures the LPTMR to use the LPO clock with the pre-scale function bypassed. Hence, the LPTMR clock will operate at a 1 kHz frequency. The compare register sets the interrupt interval for the LPTMR, and is written to 9 for the purposes of these example applications to create a 9 ms sampling period. The multichannel CMP application enables the LLWU IRQ, as this application uses the LPTMR to trigger the LLS recovery sequence. In the other example applications, the CMP is used to trigger the LLS recovery sequence. Please refer to the following code:

```
// Disable the LPTMR first. For this application, the default control values are acceptable.
LPTMR0_CSR = 0x00;

// Set the PSR to a known state
LPTMR0_PSR = 0x00;

// Setup the PSR to use the prescaler, and select the appropriate clock.
LPTMR0_PSR |= (LPTMR_PSR_PCS(1) | LPTMR_PSR_PBYP_MASK);  // If using the LPO.

// Set the Compare register for a 9 ms interval.
LPTMR0_CMR = LPTMR_CMR_COMPARE(9);
```

```
// Write to the TCF bit to clear the TCF just in case.
LPTMR0_CSR = LPTMR_CSR_TCF_MASK;
```

The Run_RAM_Loop function is the function where the device is placed into the low-power mode. The first action of this function is to disable the flash. This provides a significant current savings.

```
// Turn off the Flash!!
SIM_FCFG1 |= SIM_FCFG1_FLASHDIS_MASK;
```

Subsequently, the Power Mode Control Register and System Control Registers are configured for the desired low-power modes. The single-channel triggered mode example will utilize LLS mode, while the other examples utilize VLPS and VLPR modes. The code for the single-channel triggered mode is as follows:

```
// Write the PMC control register to enter LLS
SMC_PMCTRL |= (SMC_PMCTRL_STOPM(0x3));
```

The Multichannel and 12-bit DAC examples utilize VLPS mode so that the VLPS exit to VLPR functionality can be employed. This functionality occurs automatically when VLPS mode is exited and VLPS was entered from VLPR. This provides significant power savings when trying to achieve the optimal current in this application. See the following code for these examples:

```
// Write the PMC control register to enter VLPS
SMC_PMCTRL = ((SMC_PMCTRL & ~SMC_PMCTRL_STOPM_MASK) | (SMC_PMCTRL_STOPM(0x2)));

// Move to VLPR mode
SMC_PMCTRL = ((SMC_PMCTRL & ~SMC_PMCTRL_RUNM_MASK) | SMC_PMCTRL_RUNM(2));
```

After executing the preceding code, the LPTMR and the LPTMR interrupt are enabled, if necessary (this is not necessary when using the comparator trigger mode capability). The sleep-on-exit bit in the System Control Register is also set. This bit configures the part to immediately reenter Deep Sleep once the interrupt routine has completed, which reduces the number of cycles executed when trying to achieve the lowest power possible. Finally, the WFI instruction is executed to initiate the LLS entry process. Refer to the following code:

```
    // Set the SLEEPDEEP bit and the SLEEPONEXIT bit.  The sleep-on-exit feature
    //  will ensure that the part is always in LLS except for when a comparison
    //  is being made and will save cycles by eliminating the need to stack
    //  and unstack registers between comparisons.
    SCB_SCR = (SCB_SCR_SLEEPDEEP_MASK | SCB_SCR_SLEEPONEXIT_MASK);

    // Finally, enable the LPTMR
    LPTMR0_CSR |= LPTMR_CSR_TEN_MASK;  // Use this setting if using the LPTMR interrupt
method

    CMP0_SCR |= (CMP_SCR_CFR_MASK | CMP_SCR_CFF_MASK);
    // Clear pending interrupts in the NVIC by writing a 1!
    NVIC_ICPR = 0xFFFFFFFF;

    // Execute the WFI instruction to enter LLS mode

#ifndef KEIL
    asm("WFI");
#else
    __wfi();
#endif

     // Inifinite while loop just in case the part returns to Main from the
     //  ISR.  However, this loop should never be entered.
     while(1)
     {}
```

4.2 Single-Channel, 6-bit DAC CMP configuration and operation

This example takes advantage of the trigger mode feature of the Kinetis L Family comparator. Thus, the CMP is configured in the initialize_CMP function before entering the Run_RAM_Loop function. The first step in the initialize_CMP function is to clear the CMP control registers 1 and 2. This places the CMP in a known (disabled) state.

```
// Set CMP0_CR1 to a known state
CMP0_CR1 = 0x00;

// Set CMP Filter Count and Hysteresis control to 0.
//  Filter should be disabled to allow for low lag time.
CMP0_CR0 = 0x00;
```

Subsequently, the Trigger Mode Enablebit is set (CMP_CR1[TRIGM] = 1).

```
// Now configure CMP0_CR1 (but do not enable it!)
CMP0_CR1 |= (CMP_CR1_TRIGM_MASK);   // Enable the CMP Trigger mode
```

After executing the preceding code, the rising edge interrupt enable is enabled. If preferred, the falling edge interrupt, or both, the rising and falling edge interrupts can be enabled instead. However, it is recommended that only one interrupt be enabled because the interrupt enable bits are altered in the interrupt service routine to detect a pulse. It is very important that only one interrupt enable bit is set upon exiting the interrupt to prevent oscillations from occurring.

```
// Configure the rising-edge interrupt enable bit.
CMP0_SCR |= CMP_SCR_IER_MASK;  // Look for a rising edge first!
```

Using the following code, the 6-Bit DAC is enabled and configured to output ½ * Vin. Immediately following the write to the DAC Control Register is the write to the CMP multiplexer register to select the external pin as the non-inverted input and the DAC output as the inverted input.

```
// Configure the 6-bit DAC
//  We want to use the Vin1in supply and to set the output voltage to 1/2 Vin
//  DACout = (Vin/64) * (VOSEL[5:0] + 1), therefore 31 will produce DACout = 1/2 * Vin
CMP0_DACCR = (CMP_DACCR_DACEN_MASK
              | CMP_DACCR_VOSEL(31));

// Select the appropriate mux control
//
//  We want to know when V0 > 50% Vcc, therefore, set the inverted input to
//  be the DAC output.  Select the non-inverted input to be V0.

CMP0_MUXCR = (CMP_MUXCR_PSEL(CMP_CHANNEL_V0) | CMP_MUXCR_MSEL(CMP_DACIN_6B));
```

At this stage in the configuration process, the CMP will be enabled as an LLWU interrupt source and the LLWU interrupt will be enabled in the NVIC. The code for these steps are as follows:

```
// Configure the LLWU to wake-up from the CMP
LLWU_ME = LLWU_ME_WUME1_MASK ;     // Enable the CMP as an LLWU Wake-up source

// Enable LLWU IRQ
enable_irq(7);
```

Finally, the CMP is enabled just before exiting the initialize_CMP function.

```
// Enable the CMP
CMP0_CR1 |= CMP_CR1_EN_MASK;
```

4.3 Multichannel, 6-bit DAC CMP configuration and operation

When implementing multiple channels, the trigger mode functionality should not be used as it limits the ability to control multiple channels independently. Instead, the LPTMR wakes the part from the low-power mode (VLPS in the case of this example application) and all of the real work of the application is performed in the interrupt service routine. Therefore, there is no CMP initialization function.

Since this example application utilizes VLPS and VLPR, the RAM loop must be changed slightly from the other examples. In the RAM loop, VLPS is entered instead of LLS, and then the power mode is immediately changed to VLPR as shown in the next code block.

```
// Write the PMC control register to enter VLPS
SMC_PMCTRL = ((SMC_PMCTRL & ~SMC_PMCTRL_STOPM_MASK) | (SMC_PMCTRL_STOPM(0x2)));

// Move to VLPR mode
SMC_PMCTRL = ((SMC_PMCTRL & ~SMC_PMCTRL_RUNM_MASK) | SMC_PMCTRL_RUNM(2));
```

Then the device will enter the low-power mode. The LPTMR will generate an interrupt once the TCF bit is set, and the part will then wake-up and execute the LPTMR ISR. The first action taken in the ISR is to enable the CMP and the 6-bit DAC. For this example application, the DAC is configured to output ½ * Vin. When enabling the DAC, remember to write the appropriate value to the output voltage select register. You should also take care to store this register in a dummy variable to ensure that the write has taken place before continuing code execution.

```
// Turn on the comparator
CMP0_CR1 = CMP_CR1_EN_MASK;

// Turn on the DAC and set the voltage reference voltage appropriately.
CMP0_DACCR = (CMP_DACCR_DACEN_MASK
              | CMP_DACCR_VOSEL(31));
```

Now, the CMP multiplexer selections are made. This example application selects channel IN1 as the non-inverted input and the 6-bit DAC output as the inverted input.

```
// Select V0 as the plus input and the reference voltage as the minu input.
CMP0_MUXCR = (CMP_MUXCR_PSEL(CMP_CHANNEL_V0) | CMP_MUXCR_MSEL(CMP_DACIN_6B));
```

Subsequently, the LPTMR interrupt flags will be cleared and the timer will be restarted. This interrupt flag will also be cleared in the NVIC displayed as follows:

```
LPTMR0_CSR =  (LPTMR_CSR_TCF_MASK
               | LPTMR_CSR_TEN_MASK
               | LPTMR_CSR_TIE_MASK);   // write 1 to TCF to clear the LPT timer compare
flagint

NVIC_ICPR = 0x10010000;  // Clear LPTMR interrupt in NVIC
```

Then, a simple while loop is used to ensure that the minimum CMP settling time is achieved. When developing the application, it is recommended to use pin toggles to measure the delay time and ensure that the minimum CMP settling time has been achieved and no more. This will ensure that you meet the data sheet requirements and maintain maximum efficiency in your application.

```
// Reset semaphore i
i = 0;

// WAIT!!  Must meet the settling time of 40 us here (decimal 17)!!
while(i<8)
    i++;
```

The application continues by reading the comparator output and performing the appropriate actions. This application simply makes an output pin follow the input pin. Therefore, a GPIO pin is set if the comparator output is high, and is cleared GPIO pin if the comparator output is low.

```
// Read the output and make decision
if (CMP0_SCR & CMP_SCR_COUT_MASK)
```

```
{
    FGPIOA_PSOR = 0x1000;
}
else
{
    FGPIOA_PCOR = 0x1000;
}
```

The next comparator channel to be sampled is selected in the comparator mux and the comparison result is polled. Notice that there is no need to disable the comparator when performing this action.

```
// Now select V1
CMP0_MUXCR = (CMP_MUXCR_PSEL(CMP_CHANNEL_V1) | CMP_MUXCR_MSEL(CMP_DACIN_6B));

// Read the output and make decision
if (CMP0_SCR & CMP_SCR_COUT_MASK)
{
    FGPIOB_PSOR = 0x40;
}
else
{
    FGPIOB_PCOR = 0x40;
}
```

Finally, the comparator and 6-bit DAC are disabled before exiting the ISR. This action must be performed to achieve the desired power savings.

```
// Ensure DAC is off before leaving ISR
CMP0_DACCR = 0x00;

// Ensure CMP is off before leaving ISR
CMP0_CR1 = 0x00;
```

4.4 Single-Channel, 12-bit DAC CMP configuration and operation

This example uses the 12-bit DAC (which is not internal to the CMP module) and a program structure similar to the Multichannel implementation. The LPTMR is used to periodically wake the part from low power mode and the core will perform the necessary tasks for the conversion. However, this application differs from the Multichannel implementation in that the CMP and DAC are configured in the initialize_CMP() function before entering the Run_RAM_Loop() function. The first step in this function is to clear the CMP control registers 1 and 2. This places the CMP in a disabled and known state.

```
// Set CMP0_CR1 to a known state
CMP0_CR1 = 0x00;

// Set CMP Filter Count and Hysteresis control to 0.
//  Filter should be disabled to allow for low lag time.
CMP0_CR0 = 0x00;
```

The 12-Bit DAC is now enabled and configured to output ½ * Vin.

```
// Configure the 12-Bit DAC before selecting it as a reference

// Ensure that the DAC is disabled
DAC0_C0 = 0x00;

// Select DACREF_2 (VDDA) as the reference voltage and set up for High Power
//  Mode (setting this register to the DAC_C0_DACRFS_MASK will clear the LPEN
//  bit, enabling high power mode).
DAC0_C0 = DAC_C0_DACRFS_MASK;

// Ensure that the DAC buffer and DMA mode are disabled.
DAC0_C1 = 0x00;

// Set the Data Low and Data High registers
// The DAC output voltage is defined by the formula:
//  Vout = Vin * (1 + DACDAT0[11:0]/4096.  The DAC value definitions used
//  should set the DAC voltage to 1/2 * Vin.
DAC0_DAT0L = DAC_V_LOW_VAL;

DAC0_DAT0H = DAC_V_HIGH_VAL;
```

Afterwards, the appropriate CMP pins should be selected through the CMP MUX. For this application, CMP IN0 is selected as the non-inverted input and the DAC output is selected as the inverted input.

NOTE

In the Kinetis L Family, the 12-bit DAC output is also multiplexed with input pin, IN3.

```
// Select the appropriate Mux control
//
//  We want to know when V0 > 50% Vcc, therefore, set the inverted input to
//  be the 12-bit DAC output.  Select the non-inverted input to be V0.

CMP0_MUXCR = (CMP_MUXCR_PSEL(CMP_CHANNEL_V0) | CMP_MUXCR_MSEL(CMP_DACIN_12B));
```

Finally, the LPTMR interrupt will be enabled in the NVIC. The code for these steps are as follows:

```
// Enable LPTMR IRQ
enable_irq(28);
```

NOTE

Observe that neither the CMP nor the DAC are enabled or disabled at the end of this routine. It is guaranteed that the CMP and DAC will be in the disabled state before exiting the routine because the appropriate control registers are written with 0x00 at the beginning of the routine. This is a desired state.

Since this application is based on the Multichannel implementation, the RAM loop is modified to match. The VLPS/VLPR combination will utilized here to optimize the power consumption. Notice that after disabling the flash, the power mode is changed to VLPR and then the STOPM bits are changed to setup the SMC for VLPS entry. These actions are displayed as follows:

```
// Write the PMC control register to enter VLPS
SMC_PMCTRL = ((SMC_PMCTRL & ~SMC_PMCTRL_STOPM_MASK) | (SMC_PMCTRL_STOPM(0x2)));

// Move to VLPR mode
SMC_PMCTRL = ((SMC_PMCTRL & ~SMC_PMCTRL_RUNM_MASK) | SMC_PMCTRL_RUNM(2));
```

Following steps are performed in the LPTMR interrupt handler:

1. The CMP and DAC are enabled.
2. The LPTMR interrupt flags are serviced.
3. The comparison will be performed.
4. The CMP and DAC are disabled to save power when returning to low power mode.

This routine is presented in the following code block:

```
// Turn on the comparator
CMP0_CR1 = CMP_CR1_EN_MASK;

// Turn on the DAC and set the voltage reference voltage appropriately.
DAC0_C0 = (DAC_C0_DACEN_MASK | DAC_C0_DACRFS_MASK);

// Clear the ISR flags while waiting on the comparator to settle!!
LPTMR0_CSR =  (LPTMR_CSR_TCF_MASK
               | LPTMR_CSR_TEN_MASK
               | LPTMR_CSR_TIE_MASK);   // write 1 to TCF to clear the LPT timer compare flag

NVIC_ICPR = 0x10010000;  // Clear LPTMR interrupt in NVIC

// Reset semaphore i
i = 0;
```

```
// WAIT!!  Must meet the settling time of 30 µs here (decimal 8)!!
while(i<8)
  i++;

// Read the output and make decision
if (CMP0_SCR & CMP_SCR_COUT_MASK)
{
    FGPIOA_PSOR = 0x1000;
}
else
{
    FGPIOA_PCOR = 0x1000;
}

// Ensure DAC is off before leaving ISR (ONLY the reference select should be enabled)
DAC0_C0 = DAC_C0_DACRFS_MASK;

// Ensure CMP is off before leaving ISR
CMP0_CR1 = 0x00;
```

5 Conclusion

In this application note, three methods of performing low-power pin sampling are discussed. Each method targeted a different scenario and obtained the optimal power consumption for that scenario. Each scenario uses a moderate sampling frequency of approximately 100 Hz. However, the concepts covered could easily be scaled to lower or higher sampling frequencies. The scenarios explored are:

- single-channel application that requires only 6-bit DAC resolution
- multichannel application that requires only 6-bit DAC resolution
- single-channel application that requires 12-bit DAC resolution

Under these scenarios and their conditions, the following results are obtained:

- For the single-channel 6-bit DAC scenario, the comparator trigger mode capability should be used. This capability can be utilized with any power mode in which the comparator and LPTMR are clocked.
- For the multichannel 6-bit DAC scenario, the comparator trigger mode capability can be utilized, but channel independence will be lost if the number of necessary channels exceeds the number of comparators on the part. A better solution that can yield comparable current consumption is to use the LPTMR to wake-up at the desired interval and initialize the comparator and DAC in the interrupt routine. With the sampling frequency used, taking advantage of the automatic VLPS to VLPR mode transition provides significant power savings.
- For the single-channel application with 12-bit DAC scenario, it is recommended that the multichannel framework should be followed. The comparator trigger mode could be utilized in this scenario. However, the 12-bit DAC would have to remain enabled during the low-power phase of the application, which considerably increases the current consumption considerably. For optimal power consumption, it is best to use periodic timer wake-ups, as used in the multichannel solution, and disable the DAC during the low-power phase of the application.

6 References

- Honeywell, Sensing and Control, Ch. 7, Application Examples at www.honeywell-sensing.com.cn
- Kinetis resources are available at www.freescale.com/kinetislseries
- KL05 Sub-Family Data Sheet
- KL05 Sub-Family Reference Manual

7 Revision history

Revision number	Date	Substantial changes
0	07/2013	Initial release

Document Number AN4734
Revision 0, 07/2013