White Paper – RL78 Microcontroller Family

Using the Snooze Mode Feature to Dramatically Reduce Power Consumption

Renesas Electronics America Inc.
August 2011

Abstract

Many electronic equipment designs are being driven by customer requirements for reduced size, improved portability and additional features. To support these end-user demands, system engineers want MCU devices that have improved low-power characteristics. One way that Renesas is meeting this need is by incorporating advanced power-management capabilities into the MCUs in the RL78/G12 and RL78/G13 groups.

An important new feature is the Snooze Mode, which can dramatically reduce power consumption. The Snooze Mode saves power by allowing common peripheral functions to operate independently, while the CPU and other MCU functions are disabled. This operational flexibility is a significant advantage over other low-power modes in which the CPU must remain active and assist with common peripheral functions.

This paper illustrates the advantages of Snooze Mode using a simplified design example: a system that periodically measures an analog signal and issues an alert if the signal wanders outside of the normal operating range. In this example design, the Snooze Mode enables an RL78/G13 MCU to achieve an over 30% reduction in the system’s average power dissipation compared to an implementation without this Mode.

Introduction

Renesas introduced the RL78/G12 and RL78/G13 MCU groups to satisfy increasing application demands for MCUs with enhanced low-power capabilities. In addition to excellent general low-power characteristics, these 16-bit devices incorporate special functions to minimize operating current. Major sections of the MCUs can be turned off, while key peripheral blocks of the device continue to function. In particular, the chips have a Snooze Mode capability that can be used to dramatically reduce the power consumption of many typical MCU functions. The power savings are accomplished by allowing common data-acquisition or data-transmission functions to operate without the need to ‘wake-up’ the CPU.

This white paper explains the use of the Snooze feature, in the context of the overall low-power capabilities of the RL78, by using an example application with two different implementations: one that doesn’t use Snooze Mode operation and one that does. Datasheet numbers from the RL78/G13 will be used for the comparisons.
Also, a Power Profile methodology will be applied to develop estimates of power consumption. This methodology can be helpful in developing estimates for a wide range of applications; it's a useful tool for any low-power design engineer.

**Low Power Operation – a Growing Application Requirement**

The increasing need for low-power operation is driven by market demands for increased mobility, portability, smaller size and new features in many electronic products for consumer, home, industrial, office, and medical applications. For example, many industrial firms now use handheld meters for everything from inventory control to the remote measurement of power or fluid or gas flow. These meters must operate from batteries for hours at a time between periodic recharges. The function-per-mA system specification is critical so that the meters can deliver the required functions and performance between recharges. Power-efficient operation and the ability to drain as little current as possible when not operating are important considerations in these applications.

Some types of battery operated equipment must run off a battery for their entire operating lifetime. Such designs require the lowest operating current possible. They might have to operate for several years off battery power alone, without any recharges whatsoever. In this type of application, it is absolutely critical to ‘turn-off’ functions in the system whenever they aren’t needed and ‘wake-up’ functions only when they’re required. The ability to wait in a very-low-power state until action is required and then wake up to take necessary action – and do so while using as little current as possible—is one that can dramatically extend useful lifetimes.

**The Renesas Approach to the Growing Need for Low Power Operation**

Renesas’ RL78/G12 and RL78/G13 MCUs are recommended solutions for embedded systems that mandate low-power operation requirements because they provide advanced power-management capabilities. These functions allow the MCU to operate as follows:

1) Run with exceptional power efficiency in the normal Run Mode

2) Disable CPU operation to save power in the Halt Mode (while allowing a fast CPU-enable time)

3) Disable more of the MCU functions in the Stop Mode to save the most power (at the expense of a longer CPU wake-up time)

4) Use a new low-power mode – the Snooze Mode – to deliver even greater power savings.

The Snooze Mode lets some peripheral functions ‘wake-up’ and execute simple operations while the rest of the MCU is stopped. This saves a significant amount of power compared to the Run or Halt Modes because in Snooze the CPU is off and only the peripherals that must operate are enabled. For example, in the RL78/G13 Snooze Mode, the A/D converter can complete a conversion without waking up the CPU. Thus, an A/D conversion can be performed in Snooze Mode using only 0.5mA, 90 percent less than the 5mA required to make a conversion in the Run Mode.

Besides Snooze Mode, RL78/G13 MCUs have other important low-power characteristics that are valuable for power-constrained designs. Their wide operating range, from 1.6V to 5.5V, suits battery-based applications where the voltage (Vcc) drops over time as the battery gradually discharges.

To enable excellent application performance, in the Run Mode the RL78/G13 delivers 41 DMIPs at 32MHz while requiring as low as 66μA/MHz operating current. This processing capability allows important functions to be completed rapidly, reducing current drain during key operations.

RL78/G13 devices also have exceptional low-power characteristics in standby operation; i.e., in the Halt or Stop Modes. For example, they have a low 0.57μA current requirement in the Halt Mode when using the 32kHz internal oscillator with the Real-Time Clock (RTC) and Low-Voltage Detector (VLD) functions running. This performance keeps current drain low while the chip waits between periodic operations, a common scenario in battery-based designs.
Using Snooze Mode in Low-Power Designs

The Snooze Mode provides a new approach to saving power. By enabling Snooze operation during common peripheral functions, the CPU doesn't have to operate; it can sleep, saving power. In this mode, when a data reception event is triggered by specific peripherals, the applicable peripheral clock is enabled and data is received without CPU operation. This avoids the ‘wake-up’ time delay, as well as the power required for the normal transition from Stop Mode to Run Mode.

Data reception from the synchronous serial port, the UART a data conversion by the A/D converter can operate in the Snooze Mode by waking-up the associated port, but not the CPU. The A/D converter can ‘wake-up’ when the RTC or the Interval Timer generate interrupt signals to start a conversion. Similarly, the synchronous serial port can ‘wake-up’ when the Serial Clock input pin (SCKp) edge is detected, and the UART can ‘wake-up’ when an edge on the RxD input is detected.

After any data reception operation in the Snooze Mode is completed, a ‘match condition’ is checked. If the condition is a match, then the MCU exits the Snooze Mode and enters the Run Mode. If the condition isn’t a match, operation returns to the Stop Mode. Thus, the CPU can be activated only when the data received requires action from the CPU.

For example, an A/D conversion can operate in the Snooze Mode without the CPU running. The result of the A/D conversion can be checked to see if it is within a pre-defined range. If it’s in the ‘safe range’ the CPU need not be enabled, so the MCU transitions from the Snooze Mode to the Stop Mode. However, if – and only if – the A/D conversion returns a result outside of the pre-defined range, then the CPU is activated. Thus, the use of the Snooze Mode can avoid the costly power dissipation associated with turning on the CPU during most measurements. Because the A/D conversion uses only 0.5mA in Snooze Mode, rather than 5mA in Run Mode, dramatic power-consumption advantages are obtained.

The remainder of this white paper provides more details about the Renesas RL78/G13 MCU. Subsequently, low-power design techniques applicable to a wide range of power-constrained designs are described. Finally, the power savings made possible by using the Snooze Mode are illustrated by a typical system design.

An Introduction to the Renesas RL78/G12/G13 MCU

The Renesas RL78/G12 and RL78/G13 microcontrollers offer best-in-class high processing performance, as well as outstanding low-power operation. When high performance is required, the RL78 CPU achieves up to 41 DMIPS performance while requiring as low as 66μA/MHz operating current. When high performance is not required, current draw drops to a very low 0.57μA by using the internal 32.768 kHz oscillator with an active Real Time Clock (RTC) and Low Voltage Detector (LVD). Because these members of the RL78 family provide both high computing performance and low power consumption, they are excellent design choices for a broad span of power-constrained applications.

RL78 Architecture Overview

Renesas RL78/G12 and RL78/G13 microcontrollers offer outstanding low-power operation as well as best-in-class high processing performance. They are low-end general-purpose products suitable for home appliances, audio-visual devices, consumer electronics products such as personal digital assistants (PDAs), industrial equipment, office equipment, and computers and peripherals.

The RL78 family has a distinguished heritage, having evolved from two popular Renesas MCU families, the 78K and R8C product lines. The design of the RL78 leverages the years of experience behind these product families, including the CPU core and the large variety of proven peripheral functions. This heritage allows software resources that were used with previous 78K and R8C products to be reused effectively.
To simplify system development and deployment, the RL78/G12 and RL78/G13 groups feature an on-chip single-power-supply flash memory, self-programming (with a boot-swap function/flash-shield window function) and an on-chip debug feature. Devices in the G12 and G13 series operate from 1.6V to 5.5V power supplies and have a wide ambient operating-temperature range: −40°C to +85°C.

**RL78/G13 Key Features**

Figure 1 shows a simplified diagram of an RL78/G13 MCU (128-pin version), highlighting the key processing and peripheral functions in this product group. Although the RL78 CPU core is based on the 78K0R CPU, the new design provides both better performance (up to 41 DMIPs) and lower power consumption (0.57μA with RTC and LVD). It can operate from either a high-speed internal oscillator (at 32 MHz, 24 MHz, 16 MHz, 12 MHz, 8 MHz, 4 MHz, or 1MHz) or from an ultra-low-speed subsystem oscillator (at 32.768 kHz).

The RL78 CPU operates on 32 registers, organized as 4 banks. It includes a 16-bit barrel shifter and supports a variety of addressing modes to achieve better code efficiency and performance.

![Figure 1: Diagram of RL78/G13 (128-pin)](image)

The RL78/G13 MCU provides from 16KB to 512KB of on-chip Flash program memory, from 1 to 32KB of RAM data memory and up to 8KB of Flash-type data memory. The device’s system functions include a full-featured DMA Controller, Interrupt Controller, Clock Generator, Power-on Reset, and Low Voltage Detector, as well as hardware support for complex arithmetic functions: 16-bit by 16-bit multiply, 32-bit by 32-bit divide, 16-bit by 16-bit multiply with 32-bit accumulate). A single-wire Debug Interface aids system development.
Safety features built into the MCU include special self-diagnostics and error-checking functions that improve system reliability. The RAM has a hardware parity-check function that protects the system from data corruption, while the A/D converter has a self-diagnostic function that performs a test A/D conversion using the internal reference voltage. Moreover, there is a timer that can be used to check that the oscillator frequency is set properly, and the Flash memory is protected by a CRC function that detects errors in stored data.

**Power Management**

Power Management functions of the RL78/G13 MCU include special operating Modes to stop chip functions (like the CPU, for example) to reduce power when only a portion of the device is required to operate. For the example design that will be discussed later, the MCU’s power management features will be an important element in reducing operating power.

The three main power management modes are Halt, Stop and Snooze.

- In the Halt Mode the CPU clock is disabled to save power, but all peripheral functions are operable.
- In the STOP Mode, the high-speed system clock oscillator and internal high-speed oscillator are disabled, stopping the whole system, thereby considerably reducing power.
- In the Snooze Mode, some peripheral function can ‘wake-up’ and execute simple operations even though the rest of the device is stopped. This saves a lot of power compared to Run or Halt, since the CPU is off and only the peripherals that need to operate are enabled. When the Snooze Mode transitions to the Run Mode, the wake-up time (required for clock stabilization) is 20μsec.

Figure 2 shows an operational flow diagram for these three modes.

> **Figure 2:**
> Operational Flow of Low Power Modes

From the Run Mode, the program can execute either a Halt instruction to enter the Halt Mode or a Stop instruction to enter the Stop Mode. The Halt Mode can return back to the Run Mode via an unmasked-interrupt request or the generation of a reset signal. Since the high-speed clock source is already running during the Halt Mode, the return to the Run Mode only requires a normal interrupt-service delay.

The Stop Mode can return to the Run Mode via an unmasked-interrupt request or the generation of a reset signal. During Stop Mode, the high-speed clock oscillator is stopped. Therefore, if that
oscillator is required in the Run Mode, an additional clock stabilization wait is inserted prior to the return from interrupt. The Stop Mode can also exit to the Snooze Mode when specific peripheral events occur.

If the entry to Snooze Mode is enabled, when specific peripherals trigger a data-reception event, the applicable peripheral clock is enabled and data is received without CPU operation. After the completion of the data-reception operation in Snooze Mode, a ‘match condition’ is checked. If the condition is a match, the Snooze Mode is exited and the Run Mode is entered. If the condition is not a match, operation returns to the Stop Mode.

An interrupt from any peripheral or MCU input active during the Snooze Mode can also trigger the MCU to return to Run Mode. This includes interrupts such as a pin-change or RTC alarm (different from a periodic interval), or other externally clocked peripheral inputs. Also, the UART will wake the chip after a full data byte is received. (For a complete set of ‘match conditions’, refer to the detailed description of the interrupt function in the RL78/G13 data sheet.)

It’s important to be aware that the Snooze Mode uses less operating power than the Halt Mode. In Halt Mode all peripherals are operable. By contrast, in Snooze Mode the clocks to peripherals are disabled, except for the ADC, UART and CSI when so configured.

Halt Mode Detailed Description

In the Halt Mode, the CPU clock is disabled. If the high-speed system clock oscillator, internal high-speed oscillator, or subsystem clock oscillator is operating before the Halt Mode is set, oscillation of the operating clocks continues. In this mode, the operating current is decreased by the CPU current, but other functions can continue to operate and consume power.

Halt Mode is the best choice when it is important to restart CPU operation immediately upon interrupt request generation. This would be the case, for example, when a result must be computed and delivered in a shorter amount of time than the ‘wake-up’ time of the other low-power modes.

Operation During Halt Mode

As previously mentioned, in Halt Mode, the CPU is stopped to save power. If one of the high-speed clocks was selected prior to entering the Halt Mode, that clock continues to operate and the others are disabled. Additionally, subsystem clock settings are retained during the Halt Mode. All peripheral functions can continue operation.

Notice that the DMA function can operate during the Halt Mode. This allows data transfers from the A/D converter, UART or other DMA enabled peripherals to be performed during Halt without enabling the CPU, a capability that saves power.

If the CPU is operating off the subsystem clock when the Halt Mode is entered, the Halt Mode operates as described above with a few exceptions. All high-speed clocks are disabled. Also, the peripherals that require a high-speed clock for proper operation are disabled. This includes the high-speed CRC, IICA Serial Interface, A/D Converter and illegal-memory-access detection function. If the low-power consumption RTC Mode is enabled, additional functions are disabled – specifically, the Timer Array Unit, Clock Output, Serial Array Unit, Multiplier/Divider and DMA are not operating.

Several important safety and operation-related functions are all operable in Halt Mode. They are the RTC, Interval timer, Watchdog Timer, Power-on Reset, Voltage detection, external interrupt, Key Interrupt and general-purpose CRC. Typically, however, these functions can be selectively disabled if lower power operation is desired.
Stop Mode Detailed Description

The Stop Mode can be entered by executing the Stop instruction if the CPU is using the high-speed clock system (internal high-speed oscillator clock, X1 clock or external main system clock). In the Stop Mode the main system clocks are all stopped. Thus, the operating current is lower than the Halt Mode, since not only the CPU clock is disabled, but also the high-speed peripheral clocks.

Stop Mode offers very low power consumption. However, because the high-speed clocks must ‘wake-up’ and stabilize prior to operation, Stop Mode is best used when the response time required is longer than the ‘wake-up’ time, which could be as long as 20μsec.

Operation During Stop Mode

In the Stop Mode, the CPU is stopped, as are all the high-speed clocks. Subsystem clock settings are retained. Many peripheral functions are disabled, too, with some exceptions. Specifically, the RTC, Interval timer, watchdog timer, Power on Reset, Voltage detector, External interrupt and Key interrupt can all operate. However, they also can be selectively disabled to save more power if necessary.

The Serial Interface (IICA) can generate a ‘wake-up’ signal on an address match and return operation to the Run Mode. The Clock output is operable only if the subsystem clock is selected as the count clock. The Timer Array Unit, Multiplier/Divider, DMA Controller, high-speed CRC, general-purpose CRC and illegal-memory-access functions are disabled. The A/D converter, CSI00 and UART0 functions can be selectively enabled and Snooze Mode can be entered from the Stop Mode if Snooze is enabled.

Snooze Mode Detailed Description

The Snooze Mode allows some simple peripheral functions to be enabled (from the Stop Mode) without ‘waking’ the CPU. This avoids the ‘wake-up’ time associated with the normal transition from Stop to Run and can save a lot of power. Data reception from CSI00 or UART0 or a data conversion by the A/D converter can operate in the Snooze Mode. This is accomplished by exiting the Stop Mode and enabling the appropriate peripheral clock needed for the operation being performed. (Thus, this mode can only be specified if the CPU clock is the internal high-speed oscillation clock).

Operation During Snooze Mode

In the Snooze Mode, the CPU is stopped to save power and the subsystem clock settings are retained. Most peripheral functions are disabled, with some exceptions; i.e., the RTC, Interval timer, Watchdog Timer, Power-on Reset, Low Voltage Detector, External Interrupt and Key Interrupt can all operate. Yet they, too, can be selectively disabled to save more power.

The Clock output is operable only if the subsystem clock is selected as the count clock. The Timer Array Unit, Multiplier/Divider, DMA Controller, high-speed CRC, CRC operation and illegal memory access functions are disabled.

Upon entering the Snooze Mode, the high-speed internal clock is started. After the oscillator stabilization time passes, that clock is available for use by special peripherals. The A/D converter, CSI00 or UART0 functions (on RL78/G13 chips with less than 96KB of Flash memory), and CSI20 or UART2 (in addition to CSI00 or UART0 on devices with 96KB or more of Flash) can have the clock selectively enabled by the associated ‘wake-up’ signal. The A/D converter can ‘wake-up’ when the RTC or Interval timer interrupt requests a conversion. CSI00/20 can ‘wake-up’ when the Serial Clock input pin (SCKp) edge is detected, and UART0/2 can ‘wake-up’ when an edge on the RxD input is detected.
Other On-Chip Peripherals

The remaining on-chip peripherals of the RL78/G13 MCU are organized by function: timers, analog and communications. For our example design we will use the Real Time Clock, UART, and ADC. Therefore, these functions are described in more detail below. Please refer to the RL78/G13 User's Manual (listed in the reference section at the end of this document) for details on other key peripheral functions.

Timers

The timer block contains a timer unit array, Interval Timer, Watchdog Timer (WDT) and a Real-Time Clock (RTC). The timer array contains one or two timer units, each of which has 4 or 8 channels. (The number of units and channels depend on the pin count of the MCU). Each channel has a 16-bit counter and a variety of Mode and Control registers, as each channel is capable of implementing any of the functions shown in Table 1, below.

Table 1: Timer Unit Channel Functions

<table>
<thead>
<tr>
<th>Independent Channel Operation Function</th>
<th>Simultaneous Channel Operation Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interval Timer</td>
<td>One-Shot Pulse Output</td>
</tr>
<tr>
<td>Square Wave Output</td>
<td>PWM Output</td>
</tr>
<tr>
<td>External Event Counter</td>
<td>Multiple PWM Output</td>
</tr>
<tr>
<td>Divider Function</td>
<td></td>
</tr>
<tr>
<td>Measurement of High/Low-level of Input Signal</td>
<td></td>
</tr>
<tr>
<td>Delay Counter</td>
<td></td>
</tr>
<tr>
<td>Input Pulse Interval Measurement</td>
<td></td>
</tr>
</tbody>
</table>

Real-Time Clock

The Real-Time Clock (RTC) contains a series of counters for generating the year, month, week, day, hour, minute and second, for a period of up to 99 years. A constant-period interrupt can be generated on intervals of 0.5 seconds, 1 second, 1 minute, 1 hour, 1 day or 1 month. Additionally, alarm registers can be used to set an alarm for the week, hour and second. The clock source can be selected from the subsystem clock (FSUB, 32.768 kHz) or the internal low-speed oscillator clock (FIL, 15 kHz).

Control registers are used to select various RTC operating Modes. Also, each of the time-of-day counters can be read by the CPU over the peripheral bus.

RTC Operation Using the Snooze Feature

The RTC interrupt (INTRTC) can be used to start an A/D conversion—a useful capability when a periodic conversion must be made. In our example design, we will want to initiate an A/D conversion every 0.5 seconds, so we will set the RTC to generate an interrupt every 0.5 seconds and then use that interrupt signal to initiate A/D conversions. The RTC will also be used to supply a time stamp, so a precise log is kept of when the measurement was made.

Communications

The RL78/G13 MCU's communication functions are implemented by the Serial Array Unit (SAU). There are either one or two SAUs, depending on the device's pin count. Each Serial Array Unit can implement a variety of serial communications protocols, including I2C master/slave, I2C multi-master, CSI/SPI, UART and LIN. The number of channels available for each serial function varies by pin-count.
UART

The UART is a start-stop communications function that uses two lines: serial data transmission (TXD) and serial data reception (RXD). Each data frame, consisting of a start bit, data, parity bit, and stop bit, is transferred asynchronously between the MCU and another communication device at the internal baud rate. Full-duplex UART communication can be performed by using a channel dedicated to transmission (an even-numbered channel) and a channel dedicated to reception (an odd-numbered channel). Key features of the UART are described below:

**Data transmission/reception**
- Data length of 7, 8, or 9 bits
- Select the MSB/LSB first
- Level setting of transmit/receive data and reverse selection
- Parity-bit appending and parity check functions
- Stop-bit appending

**Interrupt function**
- Transfer end-interrupt/buffer-empty interrupt
- Error interrupt in case of framing error, parity error, or overrun error

**Error detection flag**
- Framing error, parity error, or overrun error

**UART Operation using the Snooze Feature**

The UART reception channel (channel 1) supports the Snooze feature. When the Stop Mode is active, the Snooze function is started when a change on the RxDn pin input is detected. The Snooze function allows the UART (UART 0 on devices with 64KB or less of Flash; UART0 and UART2 on devices with 96KB or more of Flash) to receive data without the requirement that the CPU be active. This can greatly reduce power consumption. (Note that the maximum transfer rate when using the Snooze function is 9600 bps.)

After the data is received, the UART generates an interrupt, either INTSR on normal completion, or INTSRE if an error has occurred. The interrupt can be used to exit from the Snooze Mode and enter the normal operation (Run Mode) for processing the data or responding to the error.

**Analog**

Analog functions of the RL78/G13 MCU include the A/D converter, internal voltage reference and internal temperature sensor. These functions support a wide variety of common ‘real-world’ sensing and measuring applications. The use of the A/D converter is of special interest here, since it is a key component in our design example.

**ADC**

The RL78 ADC converts analog inputs into 10-bit digital values. It can select from up to 26 inputs (channels) and operates in a variety of configurable modes: trigger mode (how the conversion is started), channel selection mode (how the ADC analog input is selected) and conversion operation mode (how many times the conversion is performed). Table 2 below shows the various options within each of these modes.
### Table 2: ADC Operation Mode Option Table

<table>
<thead>
<tr>
<th>Trigger Mode</th>
<th>Channel Selection Mode</th>
<th>Conversion Operation Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Software Trigger</strong></td>
<td><strong>Select Mode</strong></td>
<td><strong>One-shot Conversion Mode</strong></td>
</tr>
<tr>
<td>Conversion is started by specifying a software trigger</td>
<td>A/D conversion is performed on the analog input of one channel</td>
<td>A/D conversion is performed on the selected channel once</td>
</tr>
<tr>
<td><strong>Hardware trigger no-wait mode</strong></td>
<td><strong>Scan Mode</strong></td>
<td></td>
</tr>
<tr>
<td>Conversion is started by detecting a hardware trigger</td>
<td>A/D conversion is performed on the analog input of four channels in order</td>
<td></td>
</tr>
<tr>
<td><strong>Hardware trigger wait mode</strong></td>
<td><strong>Sequential Conversion Mode</strong></td>
<td>A/D conversion is sequentially performed on the selected channels until it is stopped by software</td>
</tr>
<tr>
<td>The power is turned on by detecting a hardware trigger while the system is off and in the conversion standby state, and conversion standby state, and conversion is then started automatically after the stabilazation with time passes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### ADC Operation Using the Snooze Feature

One of the most important modes for the example design we will discuss is the hardware trigger wait mode. In this mode, power to the converter is turned off. When a hardware trigger is received (an RTC interrupt for example) the clock to the A/D converter is turned on, the stabilization wait time elapses and then a conversion operation begins. This 3-step procedure allows the chip to dissipate the lowest amount of power while waiting for a periodic (usually infrequent) conversion operation to be requested and performed.

The A/D converter must use the hardware trigger wait mode and must specify a trigger interval sufficiently long that enough time can pass for clock stabilization, followed by the A/D conversion time. The one-shot conversion mode must also be used, so that only one conversion at a time is performed. The A/D converter also has a Scan mode (in which conversion occurs over 4 different channels sequentially) and a Select mode (in which conversion occurs on a specified channel).

In the Select mode, if an A/D conversion end interrupt request signal is generated, the A/D converter switches from the Snooze Mode to the normal operation (Run) mode with the high-speed oscillator clock as its source. This is true in the Scan mode, if even one of the scanned A/D converter channels generates an end-interrupt request signal.

In the Select mode, if the value of the result of the A/D conversion is outside the range of values specified by the A/D conversion result comparison function, the A/D conversion end-interrupt request signal is not generated and the high-speed oscillator is removed from the A/D converter. The process is the same if in the Scan mode no end-interrupt request signal is generated on any channel. In either ADC mode, a new hardware trigger can restart the A/D conversion process via Snooze.

The A/D converter must use the hardware-trigger wait mode and must specify a trigger interval long enough so that sufficient time can pass for clock stabilization followed by the A/D conversion time. The one-shot conversion mode must also be used to ensure that only one conversion at a time is performed.

The conversion times for the hardware trigger wait mode are given in Table 11.3 (5/6) in the RL78/G13 data sheet. For our example design, we will assume we are using Normal 2 and Fclk/6, so that our conversion time is 4.6875μsec. We need to add a 1μsec stabilization time, too; thus we end up with a 5.7μsec operation time. We will use the scan mode so we can test 4 channels sequentially. This gives us a total conversion time of 22.8μsec.
RL78/G13 Operating Power Characteristics

The RL78/G13 has four main operating modes, each of which has different power characteristics. Typical Supply current characteristics are given in Table 3, below. The clock source is from the Internal High-speed (FIH) oscillator and is set at 32MHz. The supply voltage is 3V and the temperature range is from -40°C to +85°C. The RL78/G13 device we will use has 96KB Flash. The numbers shown below are from Table 29.4.2 (2) and (4) in the RL78/G1 User's Manual, Rev 0.07, which is listed in the Reference section at the end of this document.

Note that the Run, Halt and Stop Mode currents don’t include ADC or LVD currents, so if those functions are used during a mode, their currents must be added in. For example, Run Mode with ADC active requires 4.7mA + 0.5mA = 5.2mA. The Snooze Mode current is the current in Stop Mode plus the active function. For example, Snooze with ADC, RTC and LVD active is 0.23μA + 0.5mA + 0.02μA + 0.08μA = 0.5mA).

Table 3: RL78/G13 Operating Current (FIH, 32MHz, 3V, 96KB, typ.)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Current</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power during Run</td>
<td>4.7mA at 32MHz, 3.0V, typ.)</td>
</tr>
<tr>
<td>Power during Halt</td>
<td>0.62mA (at 32MHz, 3.0V, typ.)</td>
</tr>
<tr>
<td>Power during Stop</td>
<td>0.23μA (at -40°C to +70°C, no WDT, typ.)</td>
</tr>
<tr>
<td>Power during Snooze</td>
<td>Stop current plus the active function</td>
</tr>
<tr>
<td>RTC Operating Current</td>
<td>0.02μA (at 3.0V, 32kHz, typ.)</td>
</tr>
<tr>
<td>RTC + 32kHz Oscillation Current</td>
<td>0.54μA (at 3.0V, 32kHz, typ.)</td>
</tr>
<tr>
<td>WDT Operating Current</td>
<td>0.22μA (at 3.0V, 15kHz, typ.)</td>
</tr>
<tr>
<td>ADC Operating Current</td>
<td>0.5mA (low-voltage mode, 3V, typ.)</td>
</tr>
<tr>
<td>Temperature Sensor</td>
<td>75μA (3V, typ.)</td>
</tr>
<tr>
<td>LVD Operating Current</td>
<td>0.08μA (3V, typ.)</td>
</tr>
</tbody>
</table>

During the Run Mode, all features of the RL78/G13 MCU operate and the typical current is highest. During the Halt Mode, the CPU is not running and typical current is less. During the Stop Mode, all CPU and Peripheral functions are disabled. Individual functions (RTC, ADC, WDT, etc) can be left enabled, in which case the appropriate additional current must be added in, per the data at the bottom of the table.

The Stop Mode offers the lowest operating current. When the Snooze function is enabled, the operating current depends on the functions (RTC, ADC, WDT, etc.,) that are enabled. Again, the operating current(s) of the functions that are running must be added to the Stop current to get the actual Snooze operating current.

Low Power Design Techniques

Having described the operation of the RL78 in sufficient detail, let’s proceed to the more detailed Low-Power oriented sections of this white paper. In order to set the stage for the calculation of the power dissipation savings that the Snooze Mode delivers in the example design, we will first explore some specific Low-Power design techniques. In particular, we will explain the concept of the Power-Use Profile and the associated Power-Profile Transition Diagram that are helpful for estimating the power dissipation of a periodic system.
Several techniques are useful for reducing power in MCU designs. The first step in a low-power design is to identify the main Power Use Profiles; that is, the modes of operation in which the application will dissipate different amounts of power. Four common Power-Use Profiles are given below:

- Run – Clocks stay at a steady rate
- Limited Run – Clock or peripherals are clocked at reduced rates when not needed
- Real-Time Clock – CPU core is halted much of the time, but wakes at regular intervals
- Stop – CPU core is halted most of the time, but is awakened by an external event.

At a high level, any application can fit roughly into one of these categories. It is important to understand that not every low-power application will require the most drastic conservation measures. Some applications that don’t have to worry about power might just leave all on-chip circuits running, causing the MCU to use 10's of milliamps. In line-powered systems or automotive applications, for example, this amount of current might not be a problem.

The first technique for reducing power consumption might only involve lowering the MCU’s clock frequencies or turning off peripherals when they aren’t needed. Such actions may reduce average current draw to a few milliamps or less. Another technique makes use of a timer such as a RTC to maintain system control, while allowing the CPU and peripherals to be stopped most of the time. With this technique, it is usually possible to get average currents down to 10's or 100's of microamps (μA). When a system is allowed to be completely inactive until needed, with an interrupt used to wake-up the chip, even lower currents – typically less than a microamp – can be achieved.

It is hard to predict average current and battery life without knowing how often the system will be interrupted. However, in many cases the standby current may be low enough that the system’s operating lifetime is limited more by the shelf life of the battery than by the current consumed by the load.

Here are some additional techniques for reducing power consumption:

1) Floating I/O pins can account for a huge percentage of the system’s power loss, so it is important to always account for each pin of the MCU. Each one should be tied to a clear logic level, or set to an output. Any floating input pins that are not connected thusly can each add several microamps to the total MCU current draw. These unnecessary losses add up very quickly, shortening the lifetimes of battery powered systems.

2) Analog peripherals often have resistor ladders or other static power drains that are independent of clock rate. For these loads, the best low-power design technique is to turn them on, use them quickly, and then turn them off. Due to the stabilization time of analog peripherals, it is prudent to carefully determine when they should be enabled and disabled; otherwise the function may not be immediately ready when it is needed. If possible, use the stabilization time to perform other CPU tasks. For example, do a calculation on the previous A/D conversion data while waiting for the completion of the latest A/D converter stabilization time.

3) Digital peripherals, such as timers, counters, or serial interfaces that are cycling from a clock, consume power in proportion to the clock frequency. Their currents may be small, but they can become large quickly if the peripherals interact with external loads. Use these peripherals as needed, of course, but minimize power by halting their clocks, provided that those clocks can be re-enabled quickly enough when needed.

To estimate the power consumption for the entire system to determine whether or not a particular battery or power supply will be sufficient, it’s necessary to add up the consumption of the each of the individual sections of the system. For the MCU and code portion, program execution must be analyzed in detail to see when the CPU is active, what peripherals are active and the amount of time used in each MCU power mode. To obtain a good power estimate, the system design should minimize both the number of program processes and the amount of time they are running.
Power Use Profile

Power dissipation will vary, depending on the low-power operating modes the system uses and when these modes are entered and exited. In a simple periodic system, there may only be one or two entries into a low-power state (perhaps to wait for an interrupt from a timer or an external event). In a more complex system, several low-power modes may be initiated by a variety of internal or external conditions.

Regardless of the complexity of the application, it helps to create a list, or profile, of the various power states the design uses. This list or ‘Power Use Profile’ will form the basis for estimating power dissipation. Each state will have a typical usage time period associated with it and an estimated power dissipation that takes into account the low-power state and the specific MCU blocks that are enabled or disabled.

A simplified example of a Power Use Profile is shown in Table 4, below. It shows three main functions:

1) Initialization – where the MCU and peripherals are configured
2) Wait for Event – where a periodic time interval is generated
3) Measure Analog Input – where the A/DC samples and converts an analog signal.

The Wait-for-Event function is implemented in the Low-Power Stop Mode, while the others use the Run Mode. Depending on the peripherals active during the function, the current usage varies, as shown in the Current column. (Review Table 3 to see the current contributions from the Mode and the function enabled.).

The estimated time the function takes is shown in the Time column. A similar Power Use Profile will form the basis of the power estimates in our example design.

Table 4: Example Power Use Profile

<table>
<thead>
<tr>
<th>Function</th>
<th>Mode</th>
<th>Peripherals</th>
<th>Current</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>Run</td>
<td>UART, CSI, RTC, LVD</td>
<td>4.7mA</td>
<td>400μs</td>
</tr>
<tr>
<td>Wait for Event</td>
<td>Stop</td>
<td>RTC, LVD</td>
<td>0.62μA</td>
<td>1sec</td>
</tr>
<tr>
<td>Measure Analog Input</td>
<td>Run</td>
<td>ADC, RTC, LVD</td>
<td>5.2mA</td>
<td>40μs</td>
</tr>
</tbody>
</table>

After the main operating modes of the design have been identified in the Power Use Profile, it is helpful to create a Power Profile Transition Diagram. This technique is explained next. Later, we will use it for our example design to estimate power consumption in two scenarios: one without using the Snooze Mode and one that applies Snooze Mode.

Power Profile Transition Diagram Description

Estimating the power dissipation for a design that uses multiple low-power modes is eased by the creation of a Power Profile Transition Diagram (PPTD). This diagram should show the amount of time and amount of power dissipated (or current used) during specific operations.

Figure 3, which illustrates the use of a PPTD, shows three different power states. Current is shown on the Y axis. It is displayed in logarithmic form, since the current used spans several orders of magnitude.

The example program performs an A/D conversion periodically. This MCU function begins in the Stop state, wakes to the Run state, does the D/A conversion and then returns to the Stop state.
Prior to Active Time 1, the MCU is in the Stop state and dissipates 0.001mA (1μA). Upon waking from Stop Mode, the current jumps to Active Current 1, where the MCU is processing the previous conversion results while the A/D converter is stabilizing. After Active Time 1, the MCU has finished processing and the A/D converter is stabilized.

A new data conversion operation then begins. This is shown on the PPTD by the transition to the lower Active Current 2 state. After Active Time 2, the conversion has been completed, so the MCU enters Stop Mode and transitions back to the low-power Inactive Current level. The MCU stays at this level until the next conversion is requested.

The PPTD can be used to calculate the average current drain, maximum current drain and current draw per function. We calculate the average current as a sum of all the currents in the system. Since most currents are not continuous, we have to weigh each one with the time that it is active. An example calculation is given below for the average current in a periodic system.

$$I_{ave} = \frac{(I_1 \times Time_1 + I_2 \times Time_2 + \ldots)}{Period}$$

(Note: The individual Times must add up to the total time Period)

Example: Wake current is 1mA for 150ms; Stop Mode is 3μA for the rest of the time (850ms).

Average current = \[\frac{(1000\mu A \times 0.15s) + (3\mu A \times 0.85s)}{1s} = 152\mu A\]

**Example Design Showing the Benefit of Snooze Operation**

To gain a better understanding of the benefits of Snooze Mode operation, we will examine the expected power dissipation of an example design. The design selected is a common type of application that must periodically measure an analog signal. (In this case, the analog signal represents a temperature, but it could be any of a variety of periodic sensor measurements.) After measuring the temperature, the system raises an alarm if the measurement exceeds the ‘safe’ range.

We will describe the design in more detail, and then calculate the expected power dissipation without using the Snooze Mode. Next, we will calculate what the power dissipation would be if the Snooze Mode were used. A comparison of the resulting power dissipation estimates will provide a numeric measure of the advantage the Snooze Mode can provide in a common type of design.

**Example Design Detailed Description**

Our example design is a remote, battery-operated temperature sensor that can be used to monitor and record the storage environment of transported goods. This Remote Temperature Sensor will use the Renesas RL78/G13 MCU as the main controller to implement the higher-level program functions. An on-chip A/D converter will be used to measure the temperature via an analog input pin. Temperatures will be measured using four separate sensors at four locations.
A UART port will be used to communicate with the Central Control System via an external wireless transceiver. An external serial Flash memory will store event logging data and the RL78/G13 chip will communicate with that memory via an SPI port. The MCU’s GPIO port will be used to show measurements and system status on a 7-segment LCD. These aspects of the system are shown in the block diagram in Figure 4, below.

![Block Diagram of the Remote Temperature Sensor](image)

**Figure 4: Block Diagram of the Remote Temperature Sensor**

**Program Flow**

The operation of the Remote Temperature Sensor includes two main functions. In the first function, the temperatures must be periodically sensed. The RTC in the RL78/G13 will be used to generate a periodic interrupt from which an A/D conversion will be initiated. The on-chip A/D converter will use its scan mode to check four channels sequentially.

Each A/D conversion will generate a temperature reading that will be tested to see if it is within the defined limits. If all four temperature readings are within the specified limits, no action will be taken. If the temperature moves outside the defined limits of the temperature reading, a log event will be triggered. The temperature measurement and the recorded time from the RTC are logged by storing the data into external serial Flash memory via the SPI port.

When warning alerts are generated, they are sent over the UART via the external wireless transmitter to the central control system. The transmitted data will show the temperature reading and the time that the measurement was taken.

In the second function, the Remote Temperature Sensor responds to instructions from the Central Control System. Those instructions come via the wireless receiver and are captured by the UART in the RL78/G13 chip. The central control instruction can include commands such as ‘read the current temperature’, ‘read the log information’, ‘erase the log data’, ‘test the system’, or ‘change the temperature range’. These commands are decoded and executed by the MCU. The Read Temperature command operates similarly to the exception-logging command described previously; i.e., it is basically a forced Fault Event and uses the same code.
Having described the functions required by the example design, it is helpful to create a Power Use Profile identifying the various Low-Power states being used. In the first implementation, we do not use the Snooze Mode, so the Temperature measurement and command reception via the UART will be done in the Run Mode, in which the CPU is active. The resulting Power Use Profile is given in Table 5, below, along with operating currents and operation-time estimates (Table 3 is the source for the current components):

**Table 5: Power Use Profile without Snooze Mode**

<table>
<thead>
<tr>
<th>Function</th>
<th>Mode</th>
<th>Peripherals</th>
<th>Current</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>Run</td>
<td>UART, CSI, RTC, LVD</td>
<td>4.7mA</td>
<td>200µs</td>
</tr>
<tr>
<td>Wait for Event</td>
<td>Stop</td>
<td>RTC, LVD</td>
<td>0.62µA</td>
<td>0.5sec</td>
</tr>
<tr>
<td>Wake from Stop</td>
<td>Run</td>
<td>ADC, RTC, LVD</td>
<td>5.2mA</td>
<td>20µs</td>
</tr>
<tr>
<td>Measure Temperature</td>
<td>Run</td>
<td>ADC, RTC, LVD</td>
<td>5.2mA</td>
<td>22.8µs</td>
</tr>
<tr>
<td>Store Event</td>
<td>Run</td>
<td>CSI, RTC, LVD</td>
<td>4.7mA</td>
<td>8µs</td>
</tr>
<tr>
<td>Send Notification</td>
<td>Run</td>
<td>UART, RTC, LVD</td>
<td>4.7mA</td>
<td>6.7ms</td>
</tr>
<tr>
<td>Receive Command</td>
<td>Run</td>
<td>UART, RTC, LVD</td>
<td>4.7mA</td>
<td>1.7ms</td>
</tr>
<tr>
<td>Measure Temperature</td>
<td>Run</td>
<td>ADC, RTC, LVD</td>
<td>5.2mA</td>
<td>22.8µs</td>
</tr>
<tr>
<td>Store Event</td>
<td>Run</td>
<td>CSI, RTC, LVD</td>
<td>4.7mA</td>
<td>8µs</td>
</tr>
<tr>
<td>Send Notification</td>
<td>Run</td>
<td>UART, RTC, LVD</td>
<td>4.7mA</td>
<td>6.7ms</td>
</tr>
</tbody>
</table>
The above Power Use Profile can be translated into the Periodic Power Profile Transition Diagram (PPTD) shown in Figure 5, below.

The Initialization step requires 4.7mA current for 200μs. Because this function is done only once, its current contribution will be negligible. Thus, we need not include it as part of our Periodic PPTD.

After initialization, the MCU enters the Wait for Event state using the Stop low-power mode. The RTC and Low Voltage Detector (LVD) are running in this mode as well, so the total current is 0.62μA (Refer to Table 3 for the individual estimates).

When triggered, the MCU must wait for clock stabilization, a 20μs delay. The current used during this phase is estimated to be the same as when running (5.2mA). Subsequently, the Temperature Measurement will require 22.8μs and the current used is 5.2mA. These two steps are added together to create a single 42.8μs step with 5.2mA current requirement.

Next, the temperature measurements are logged, a procedure requiring 8μs and 4.7mA. If the temperature is within range, operation transitions to the Stop state at 0.62μA.

If the temperature is out of range, however, an alert is sent to the central controller (as indicated by the dotted line). Notice that the alert is not expected to occur very often (less than one out of 60 times, so it need not be included in the periodic power estimate. After the alert is sent, operation of the RL78/G13 MCU returns to the Stop state. This state lasts until the RTC times out (0.5sec), so this is our periodic time base.

Because the Receive Command function is not periodic and would probably occur only once every several minutes, we will not include it in the PPTD or our average current calculations. If it occurred more frequently, we could have to create another Power Use Profile and PPTD to determine its average current contribution.

![Figure 5: Periodic Power Profile Transition Diagram – No Snooze Mode](image-url)
To estimate the total current needed for each operating period we can simply multiply each current component by the operation time and then add each of these together. The computation is shown in Table 6, below. The operational current during the cycle is 0.57μA over 0.5sec, so the average current is 1.14μA.

Table 6: Average Current Estimate – No Snooze

<table>
<thead>
<tr>
<th>Function</th>
<th>Operation- No Snooze</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Current</td>
<td>Time</td>
<td>Total</td>
</tr>
<tr>
<td>Measure Temperature</td>
<td>5.2mA</td>
<td>42.8μsec</td>
<td>2.2256E-07 Amp-Sec</td>
</tr>
<tr>
<td>Store Event</td>
<td>4.7mA</td>
<td>8μs</td>
<td>3.76E-08 Amp-Sec</td>
</tr>
<tr>
<td>Wait for Event</td>
<td>0.62μA</td>
<td>(0.5s-50.8μs)</td>
<td>3.1E-07 Amp-Sec</td>
</tr>
<tr>
<td>Average Current</td>
<td>0.57μA</td>
<td>0.5s</td>
<td>1.14 Amp-Sec</td>
</tr>
</tbody>
</table>

Power Use Estimate – Snooze Implementation

The Snooze Mode allows the MCU to measure the temperature without using the CPU. The revised Power Use Profile when using Snooze is shown in Table 7 below:

Table 7: Power Use Profile – with Snooze

<table>
<thead>
<tr>
<th>Function</th>
<th>Operation – Snooze</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mode</td>
<td>Peripherals</td>
<td>Current</td>
</tr>
<tr>
<td>Initialization</td>
<td>Run</td>
<td>UART, CSI, RTC, LVD</td>
<td>4.7mA</td>
</tr>
<tr>
<td>Wait for Event</td>
<td>Stop</td>
<td>RTC, LVD</td>
<td>0.62μA</td>
</tr>
<tr>
<td>Measure Temperature</td>
<td>Snooze</td>
<td>ADC, RTC, LVD</td>
<td>0.62mA+0.5mA</td>
</tr>
<tr>
<td>Store Event</td>
<td>Run</td>
<td>CSI, RTC, LVD</td>
<td>4.7mA</td>
</tr>
<tr>
<td>Send Notification</td>
<td>Run</td>
<td>UART, RTC, LVD</td>
<td>4.7mA</td>
</tr>
<tr>
<td>Receive Command</td>
<td>Snooze</td>
<td>UART, RTC, LVD</td>
<td>0.62μA</td>
</tr>
<tr>
<td>Measure Temperature</td>
<td>Snooze</td>
<td>ADC, RTC, LVD</td>
<td>0.62mA+0.5mA</td>
</tr>
<tr>
<td>Store Event</td>
<td>Run</td>
<td>CSI, RTC, LVD</td>
<td>4.7mA</td>
</tr>
<tr>
<td>Send Notification</td>
<td>Run</td>
<td>UART, RTC, LVD</td>
<td>4.7mA</td>
</tr>
</tbody>
</table>
The above Power Use Profile can now be translated into the Periodic Power Profile Transition Diagram shown in Figure 6, below. (As in Figure 5, the non-periodic Initialization and Command functions are not included). The change between the previous Periodic PPTD is due to the use of Snooze Mode for the Temperature Measurement function. It is shown in red to make it easy to see the difference.

![Periodic Power Profile Transition Diagram – With Snooze Mode](image.png)

**Figure 6: Periodic Power Profile Transition Diagram – With Snooze Mode**

The power estimate when using the Snooze Mode is given in Table 8, below.

The use of the Snooze Mode reduces the operational current of the Measure Temperature function from 5.2mA to 1.12mA, a reduction of over 78 percent in the power consumed for this operation. The other operational currents are unchanged. The operational current during the cycle is reduced to 0.39μA and an average of 0.791μA.

<table>
<thead>
<tr>
<th>Function</th>
<th>Operation- Snooze</th>
<th>Current</th>
<th>Time Period</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measure Temperature</td>
<td></td>
<td>1.12mA</td>
<td>42.8μs</td>
<td>4.79E-08 Amp-sec</td>
</tr>
<tr>
<td>Store Event</td>
<td></td>
<td>4.7mA</td>
<td>8μs</td>
<td>3.76E-08 Amp-sec</td>
</tr>
<tr>
<td>Wait for Event</td>
<td></td>
<td>0.62μA</td>
<td>(0.5s-50.8μs)</td>
<td>3.10E-07 Amp-sec</td>
</tr>
<tr>
<td>Average Current</td>
<td></td>
<td>0.791μA</td>
<td>0.5s</td>
<td>3.95E-07 Amp-sec</td>
</tr>
</tbody>
</table>

**Table 8: Average Current Estimate – with Snooze**

**Power Use Comparison – No Snooze vs Snooze Implementation**

Because the Snooze Mode reduces the operation current of the Measure Temperature function to 1.12mA, the average operational current of the system drops from 1.14μA to 0.791μA, for a power savings of about 31%.

**Conclusion**

This white paper has provided a detailed description of the Low Power Modes available in the Renesas RL78/G13 MCU and showed how to map key program functions into low-power modes by using a Power Use Profile technique. An example application was used to illustrate the power savings possible by applying the special Snooze function that this MCU provides. In the example application, an average current savings of 31% was achieved.

**References**

1) RL78/G13 User’s Manual, Hardware, Rev 0.07: http://am.renesas.com/products/mpumcu/rl78/doc.html