SILICON LABORATORIES

AN138

# POWER MANAGEMENT TECHNIQUES FOR THE 'F30X AND 'F31X

## Relevant Devices

This application note applies to the following devices:
C8051F300, C8051F301, C8051F302, C8051F303, C8051F304, C8051F305, C8051F310, and C8051F311.

## Introduction

The C8051F30x and C8051F31x are excellent choices for low power applications. They provide flexible clocking hardware and 3V operation which significantly reduces power consumption. In addition, the pipelined core executes instructions at an average rate of one system clock per opcode byte.

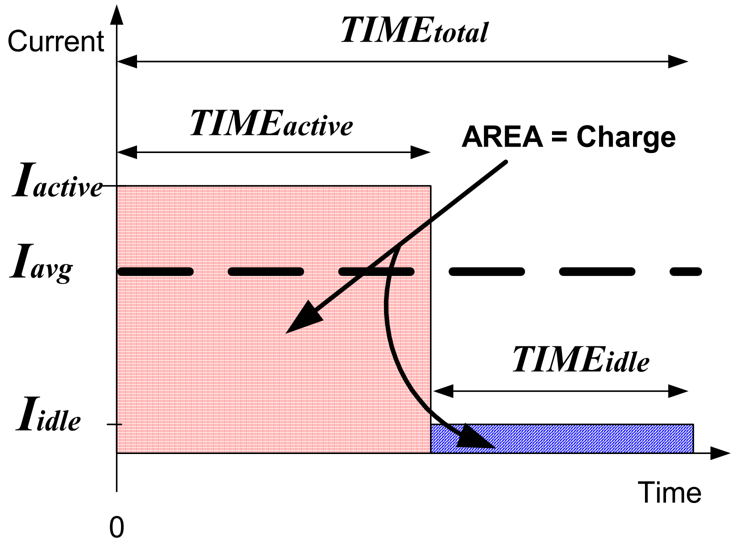
This application note discusses power calculation techniques and power saving strategies for for C8051F30x and C8051F31x devices. It discusses how the internal and external oscillators, CPU power management modes, system clock frequency, and supply voltage play a role in the power consumption of the device. This note also discusses and gives examples of implementing a "sleep" mode to reduce power consumption. Software examples are included to demonstrate how the techniques discussed in this note can be applied in actual systems.

## Key Points

- Flexible clocking hardware makes it easy to switch between a high-performance mode and a low-power mode.
- Managing power smartly can significantly reduce the total power consumption of the system.
- The missing clock detector will cause a device reset if the operating frequency drops below 10 kHz.

## Calculating Power Consumption

When designing a system with a power budget, being able to estimate the system's power consumption on paper can save time and resources by allowing the designer to experiment with different configurations before hardware is built. Since Silicon Labs devices often provide single-chip solutions to many applications, the device power



consumption is often equivalent to the system power consumption.

The device power consumption is calculated by taking the sum of individual contributions. The main contributors to power consumption include the oscillators, digital power, analog peripherals, and Port I/O.

The oscillator power consumption consists of contributions from the internal and external oscillators. The internal oscillator power consumption is discussed in the analog peripherals section.

Digital power consumption depends on CPU mode, supply voltage, and system clock frequency. Temperature and the digital peripherals themselves have a minimal effect on digital power consumption.

Analog peripheral power consumption is dominated by the ADC, VREF, temperature sensor, bias generator, and internal oscillator. Comparators also contribute a small amount to the total analog peripheral power consumption.

All current sourced by a port pin is taken from the device supply current. For example, if 10 mA are being used to power an LED connected to a port pin, the device power consumption will increase by 10 mA.

## *External Oscillator Power Consumption*

The external oscillator circuitry on these devices is very flexible. It may derive its timebase from a crystal or ceramic resonator, a capacitor, an RC network, or an external CMOS clock. Each of these clocking methods has its advantages. Since the oscillator can change clocking modes from application code, switching modes on the fly can significantly reduce power consumption. In C, RC, and CMOS clock modes, it is possible to achieve very low operating frequencies.

### External CMOS Clock

When the External Oscillator is in CMOS Clock mode, the external oscillator driver is turned off and the circuitry consumes a negligible amount of current. When a CMOS clock signal is present on XTAL2, it may be used as a clocking source for the CPU, Timers, PCA, or other peripherals. Note that power consumption increases slightly when a high frequency signal is applied to any port pin.

### External Crystal

An external crystal provides the most accurate timebase, but may consume more power at a given frequency when compared with other clocking methods discussed in this note. The external crystal current depends on the crystal frequency and the external oscillator drive current setting (XFCN). Table 1 and Table 2 show typical current values for the external oscillator circuitry when driving various crystals.

**Table 1. Typical External Oscillator Power Consumption in Crystal Mode (C8051F30x)**

| XFCN | Crystal Frequency | Current (3.0 Volts) |
|---|---|---|
| 1 | 32.768 kHz | 3.7 µA |
| 5 | 4.000 MHz | 240 µA |
| 7 | 11.0592 MHz | 3.8 mA |
| 7 | 25.000 MHz | 4.1 mA |

**Table 2. Typical External Oscillator Power Consumption in Crystal Mode (C8051F31x)**

| XFCN | Crystal Frequency | Current (3.0 Volts) |
|---|---|---|
| 1 | 32.768 kHz | 4.1 µA |
| 5 | 4.000 MHz | 280 µA |
| 7 | 11.0592 MHz | 4.4 mA |

**Table 2. Typical External Oscillator Power Consumption in Crystal Mode (C8051F31x)**

| XFCN | Crystal Frequency | Current (3.0 Volts) |
|---|---|---|
| 7 | 25.000 MHz | 4.7 mA |

## External C Mode

External C mode can provide low-power clocking to the device with a single capacitor connected to XTAL2. A wide range of frequencies can be achieved by varying the XFCN bits in the OSCXCN register. Table 3 through Table 6 show how frequency and current are affected by capacitance and XFCN settings.

Since the frequency of the external oscillator in C mode depends on capacitance, it will vary from system to system due to capacitor tolerance and stray capacitance. The tolerance of the internal current source also plays a role in determining the oscillation frequency. Once the capacitor starts oscillation, the frequency remains relatively stable.

The external oscillator frequency may be measured to 2% accuracy using the 24.5 MHz internal calibrated oscillator. For even greater accuracy, the internal oscillator frequency may first be measured using a 32.768 kHz watch crystal. Once the frequency of the internal oscillator is found, it may be used to more accurately measure the external oscillator frequency in C mode. Having up to three clock sources with one hardware configuration is possible because of the ability to switch oscillator modes on-the-fly.

**Table 3. C mode Frequency Range and Typical Oscillator Power Consumption with a 33 pF Capacitor on XTAL2 (C8051F30x)**

| XFCN | Approximate Frequency (3.0 Volts) | Current (3.0 Volts) |
|---|---|---|
| 0 | 5 kHz | 2 µA |
| 1 | 15 kHz | 4 µA |
| 2 | 44 kHz | 10 µA |
| 3 | 130 kHz | 26 µA |
| 4 | 380 kHz | 73 µA |
| 5 | 1.0 MHz | 220 µA |
| 6 | 3.8 MHz | 960 µA |
| 7 | 9.5 MHz | 4.0 mA |

**Table 4. C mode Frequency Range and Typical Oscillator Power Consumption with a 10 pF Capacitor on XTAL2 (C8051F30x)**

| XFCN | Approximate Frequency (3.0 Volts) | Current (3.0 Volts) |
|---|---|---|
| 0 | 9 kHz | 2 µA |
| 1 | 27 kHz | 4 µA |
| 2 | 80 kHz | 10 µA |
| 3 | 230 kHz | 27 µA |
| 4 | 650 kHz | 78 µA |
| 5 | 1.8 MHz | 230 µA |
| 6 | 5.8 MHz | 990 µA |
| 7 | 11.9 MHz | 4.0 mA |

**Table 5. C mode Frequency Range and Typical Oscillator Power Consumption with a 100 pF Capacitor on XTAL2 (C8051F30x)**

| XFCN | Approximate Frequency (3.0 Volts) | Current (3.0 Volts) |
|---|---|---|
| 0 | 2 kHz | 2 µA |
| 1 | 7 kHz | 4 µA |
| 2 | 20 kHz | 9 µA |
| 3 | 56 kHz | 24 µA |
| 4 | 170 kHz | 70 µA |
| 5 | 480 MHz | 210 µA |
| 6 | 2.0 MHz | 930 µA |
| 7 | 6.3 MHz | 4.0 mA |

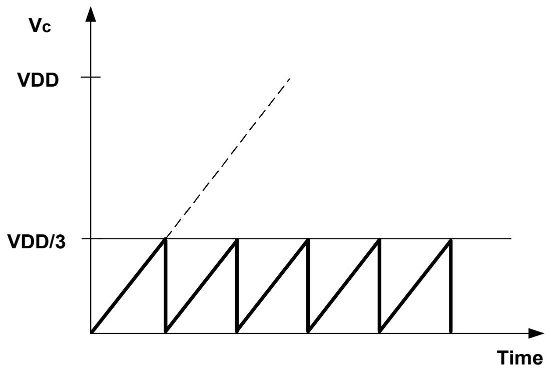
**Table 6. C mode Frequency Range and Typical Oscillator Power Consumption with a 33 pF Capacitor on XTAL2 (C8051F31x)**

| XFCN | Approximate Frequency (3.0 Volts) | Current (3.0 Volts) |
|---|---|---|
| 0 | 7 kHz | 2 µA |
| 1 | 21 kHz | 4 µA |
| 2 | 61 kHz | 11 µA |
| 3 | 170 kHz | 29 µA |
| 4 | 500 kHz | 85 µA |
| 5 | 1.4 MHz | 260 µA |
| 6 | 5.0 MHz | 1.1 mA |
| 7 | 11.2 MHz | 4.5 mA |

## Frequency Generation in C Mode

The external oscillator in C mode generates a clock signal by constantly charging and discharging the capacitor connected to XTAL2. As Figure 1 shows, the capacitor charges linearly from a constant current source. When the voltage on the capacitor reaches VDD/3, the comparator creates a path to ground, discharging the capacitor. Once the capacitor is discharged, the comparator opens the switch and the cycle repeats. The resulting waveform is shown in Figure 2. The output of the comparator, a digital signal, is fed to a "divide-by-two" circuit whose output can be selected as the system clock.

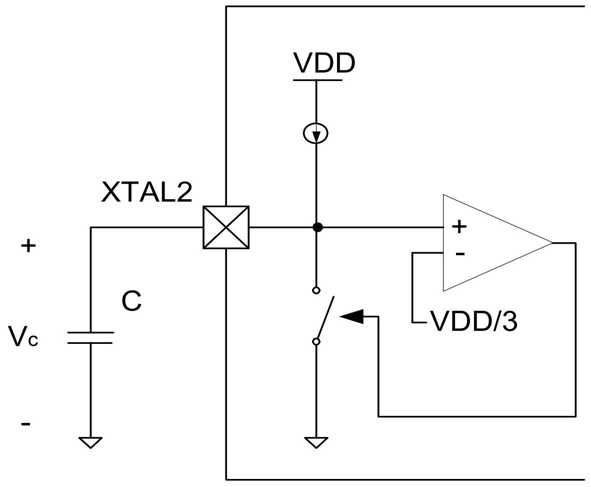
**Figure 2. C Mode Waveform at XTAL2**



## External RC Mode

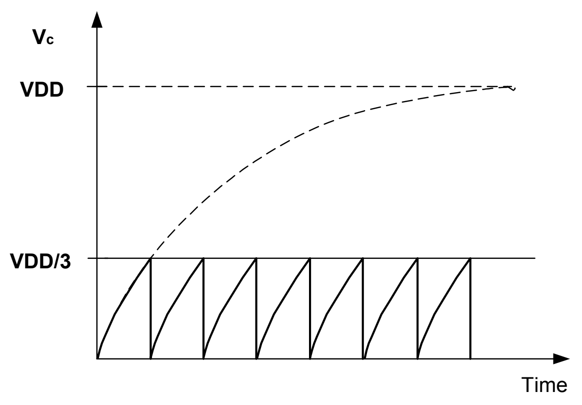
RC mode operates similarly to C mode with the exception that in RC mode the capacitor is charged

**Figure 1. C Mode Overview**



through an external resistor, as shown in Figure 3. Once the capacitor voltage reaches VDD/3, the comparator creates a path to ground, discharging the capacitor. Figure 4 shows the waveform at XTAL2 generated by this cycle.

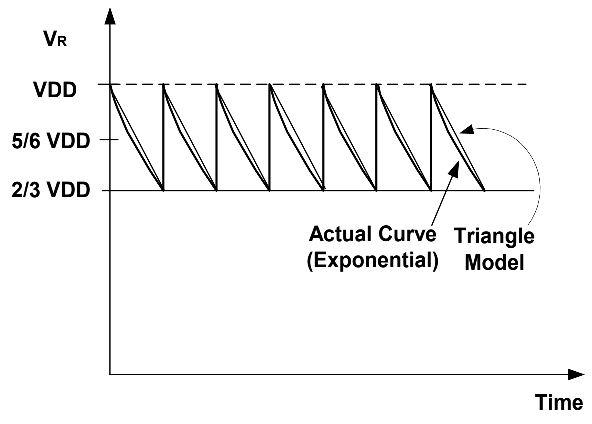
**Figure 4. RC Mode Waveform at XTAL2**



## Calculating Power in RC Mode

The average power consumption for the external oscillator in RC mode is determined by the the average current through the resistor. The voltage drop across the resistor is exponential, but can be modeled as a triangular waveform to simplify finding the average, as shown in Figure 5.

**Figure 5. RC Mode Resistor Voltage**



With this simplification, Equation 1 can be used to calculate the average voltage. The external oscillator average current and power are shown in Equation 2 and Equation 3, respectively.

**Equation 1. Average Resistor Voltage**

$$V_{avg} \cong \frac{5}{6} \times VDD$$

**Equation 2. Average Current**

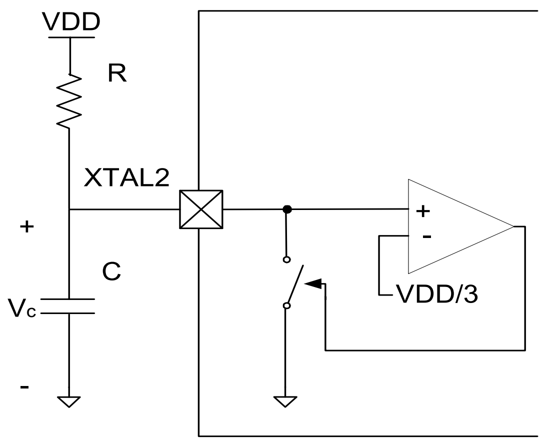
$$I_{avg} = \frac{V_{avg}}{R}$$

**Equation 3. Average Power**

$$P_{avg} = \frac{V^2_{avg}}{R}$$

Note that the power consumption of the external oscillator in RC mode depends on the resistor value and **not** on the capacitor value.

**Figure 3. RC Mode Overview**



## Digital Power Consumption

Digital power consumption is dominated by CPU current. The factors that play a major role in determining this current are CPU power management mode, supply voltage, and system clock frequency. Temperature and digital peripherals have a minimal effect on digital power consumption.

### CPU Power Management Modes

The CPU has three modes of operation: Normal, Idle, and Stop. Figure 6 and Figure 7 show typical supply current curves when the internal oscillator is in divide by eight mode (3.0625 MHz) and is selected as the system clock. The Idle mode current is dominated by the internal oscillator. The Normal mode current minus the Idle mode current is approximately the amount of current needed by the CPU to execute instructions at 3.0625 MHz.

The CPU is in Normal mode whenever it is executing instructions. On writing a ‘1’ to the IDLE bit (PCON.0), the CPU finishes executing the current instruction and enters a low-power mode until awakened by an interrupt or device reset. **In Idle mode, all analog and digital peripherals, memory, and internal registers remain operational.** When awakened, the CPU resumes execution at the instruction following the write to the IDLE bit.

**Figure 6. Effect of CIP-51 Power Management Mode on Supply Current (C8051F30x)**

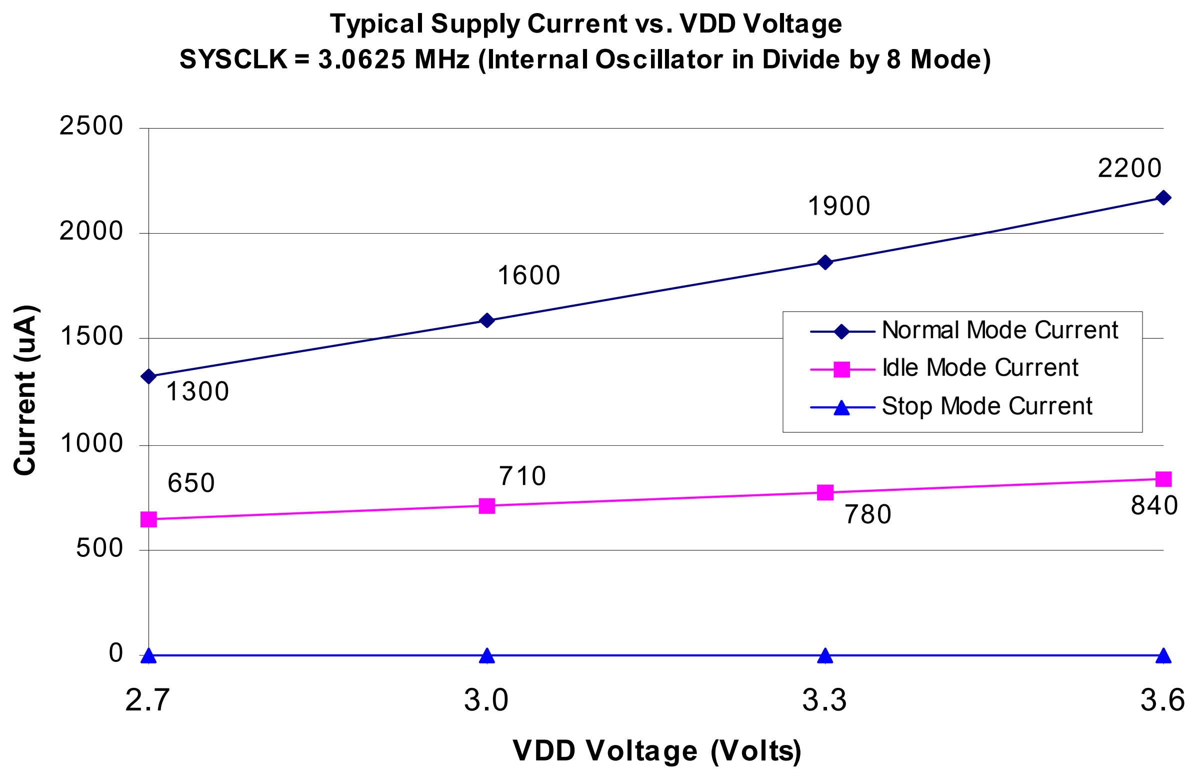


**Figure 7. Effect of CIP-51 Power Management Mode on Supply Current (C8051F31x)**

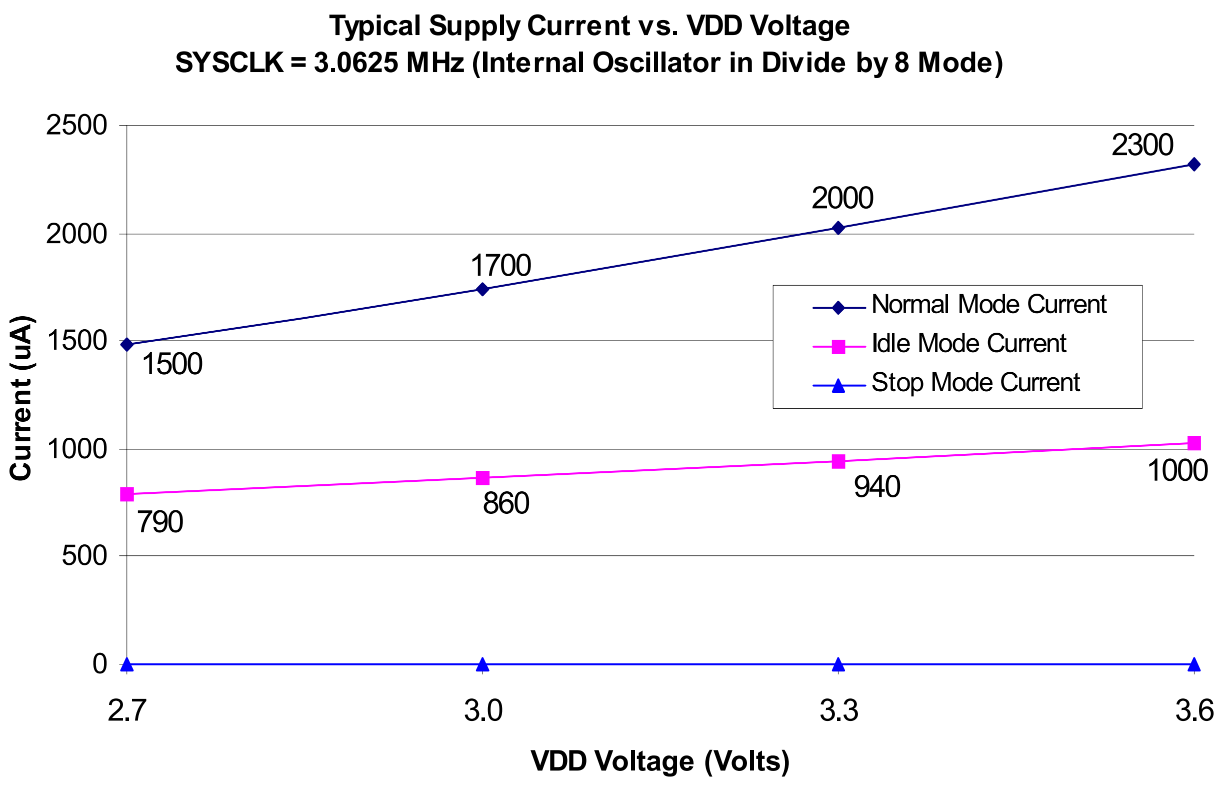
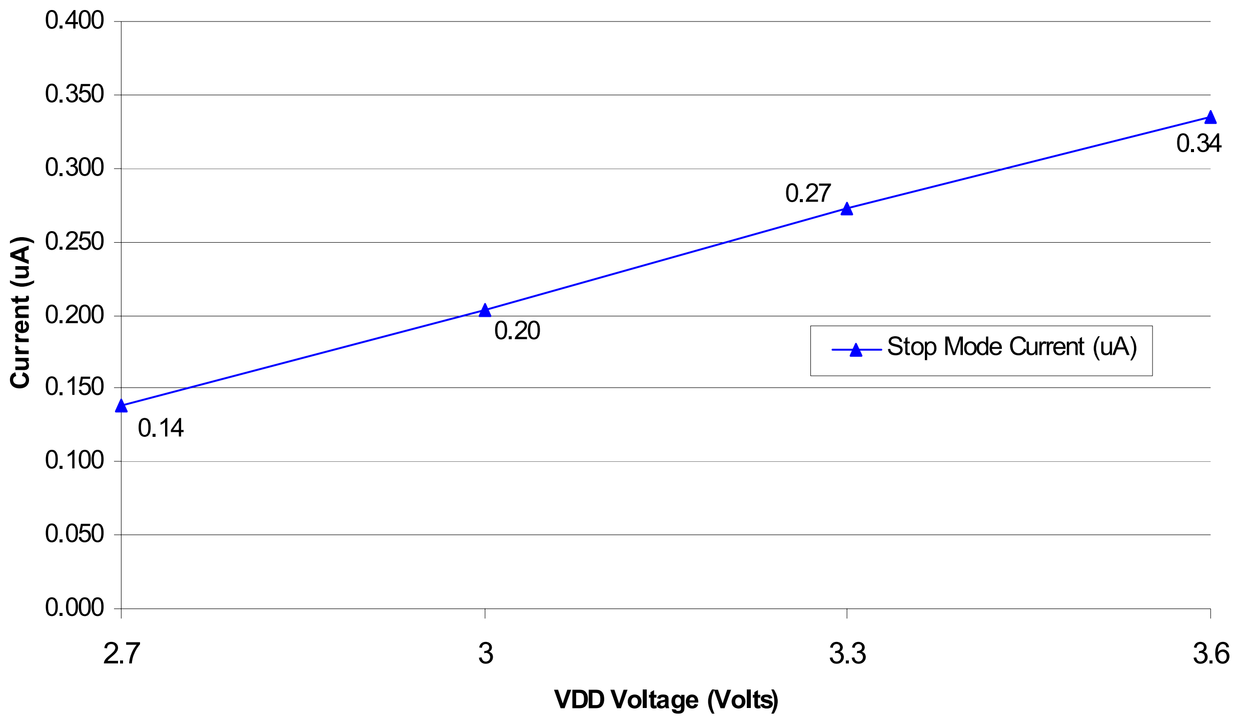


Figure 8.  Stop Mode Current vs. Supply Voltage (C8051F30x)

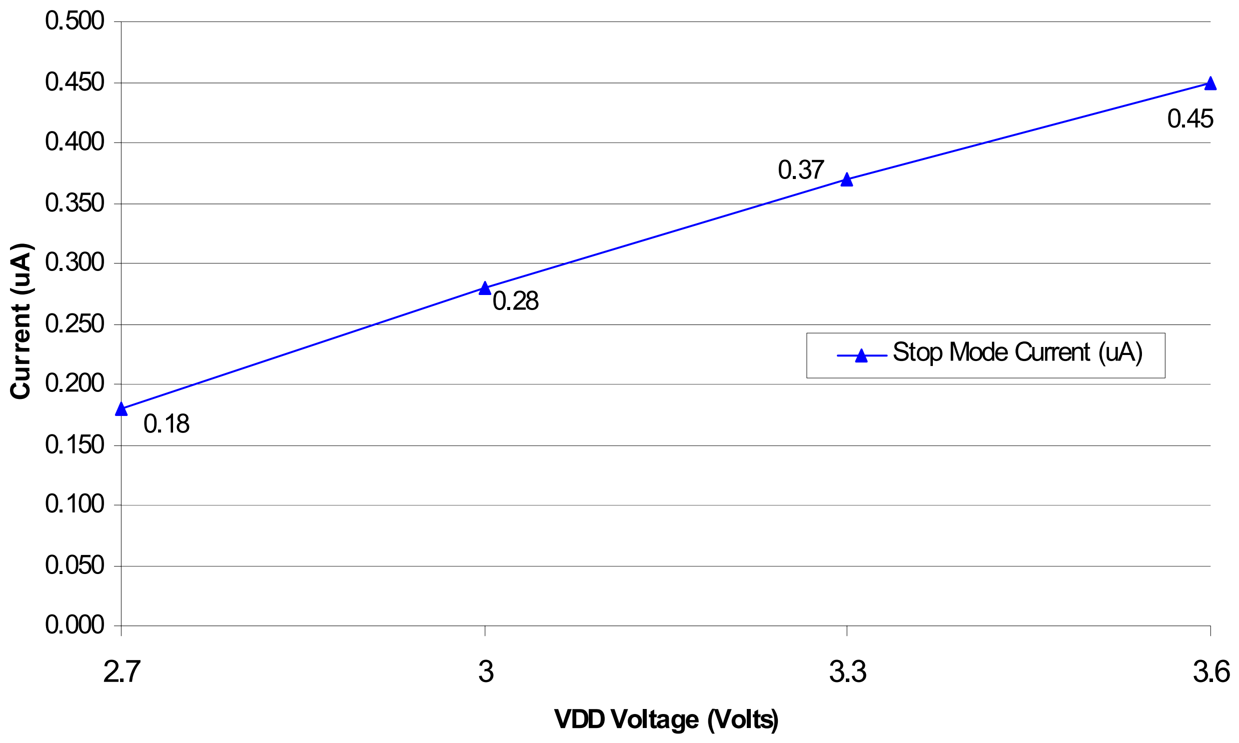


The CPU enters Stop mode on writing a ‘1’ to the STOP bit (PCON.1). **After the current instruction is executed, the internal oscillator, and all digital peripherals are disabled. Analog peripherals such as comparators and the external oscillator remain in their current state.** The MCU consumes the least amount of current when in Stop mode. Figure 8

and Figure 9 show that the Stop mode current is typically less than 500 nA when the internal oscillator is used for system clocking.

Any reset source can be used to recover from Stop mode. This includes Comparator, Missing Clock Detector, power on, or VDD monitor reset. Software Example 3 at the end of this note shows how Stop mode may be used on C8051F30x and C8051F31x systems to save power.

**Figure 9. Stop Mode Current vs. Supply Voltage (C8051F31x)**



## Supply Voltage

Supply current increases with supply voltage. This relationship can be observed at all operating frequencies but has the greatest impact at higher frequencies. Figure 10 and Figure 11 show typical supply current vs. supply voltage curves when operating from the internal 24.5 MHz system clock. The minimum supply voltage specified in the datasheet is 2.7 Volts. However, since many voltage regulators have a +/- 10% accuracy, systems are not typically designed for a supply voltage less than 3.0 Volts.

## Temperature

Device temperature does not have an appreciable effect on power consumption on these devices.

**Figure 10. Effect of Supply Voltage on Power Consumption (C8051F30x)**

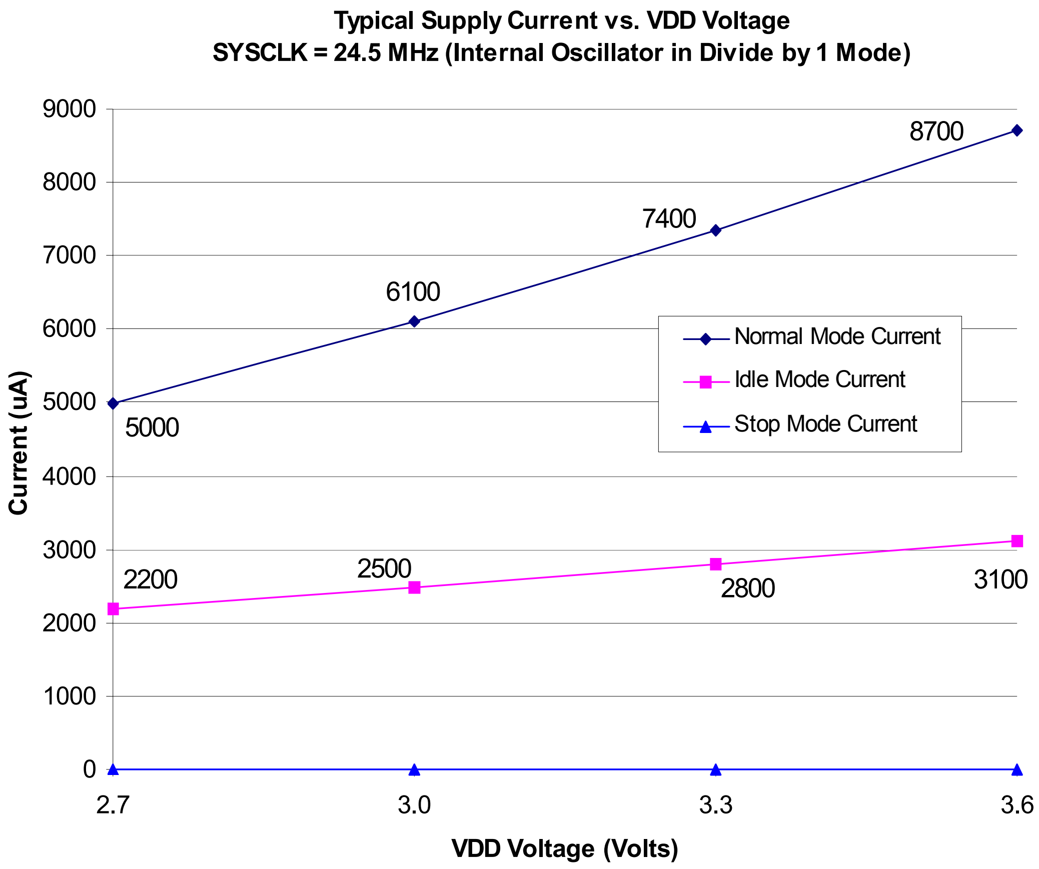
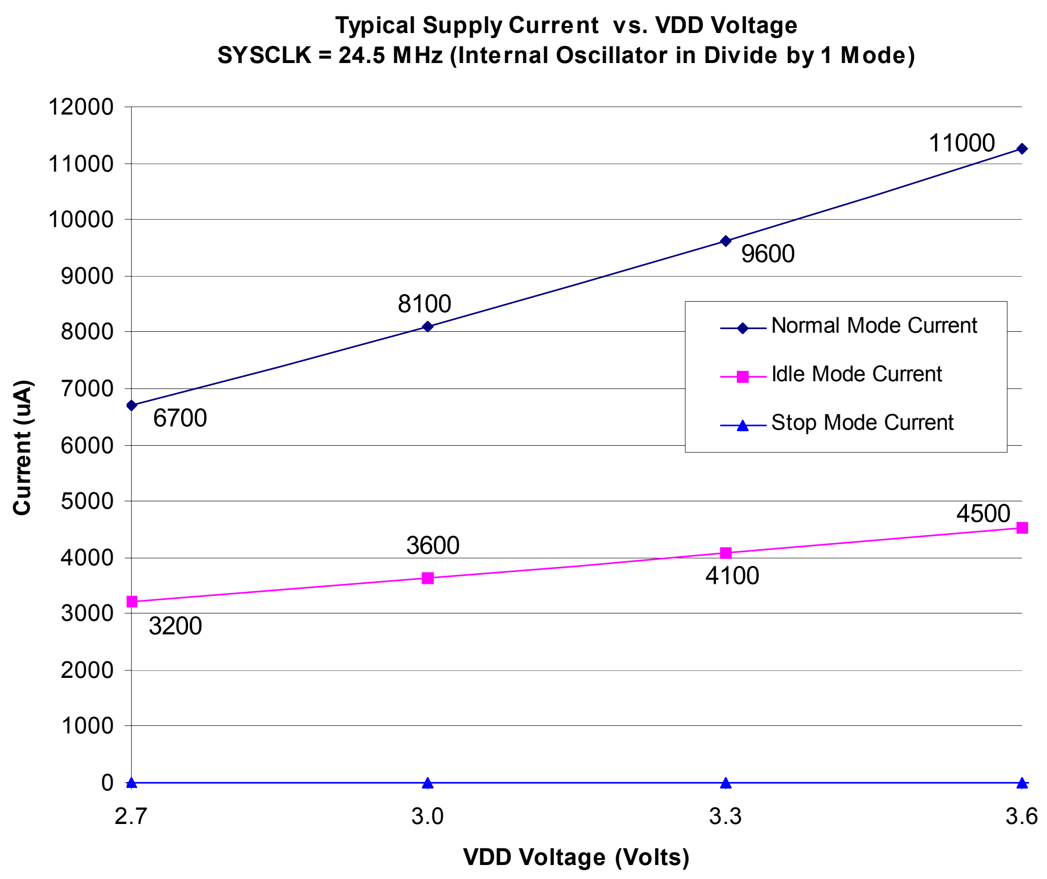


**Figure 11. Effect of Supply Voltage on Power Consumption (C8051F31x)**



## Operating Frequency (SYSCLK)

The CPU operating frequency has the greatest impact on power consumption.

Figure 12 and Figure 13 show the effect of operating frequency on power consumption when the CPU is in Normal mode. Near 13 MHz (C8051F30x devices) and 16 MHz (C8051F31x devices), the current drops slightly and changes slope because of a switchover in the FLASH read timing mechanism. Table 7 and Table 8 show the slope and offset for the graphs in Figure 12 and Figure 13, respectively. “Region A” in Figure 12 refers to frequencies less than 13 MHz and “Region B” refers to frequencies higher than 13 MHz. The same is true for Figure 13 except the switchover occurs near 16 MHz.

**Figure 12. Effect of Operating Frequency on Normal Mode Power Consumption (C8051F30x)**

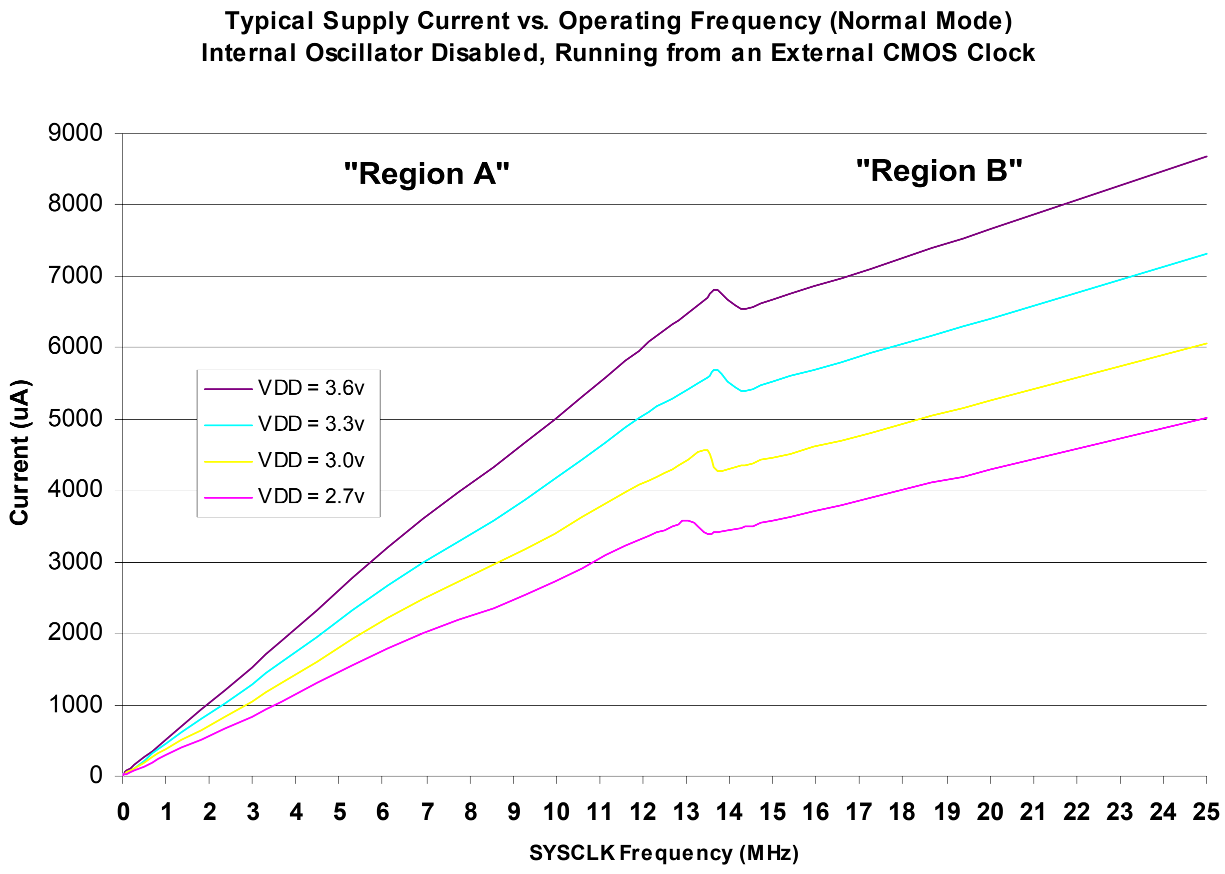


**Table 7. Slope and Offset Values for the Curves in Figure 12 (C8051F30x)**

| VDD | Slope A mA/MHz | Offset A mA | Slope B mA/MHz | Offset B mA |
|---|---|---|---|---|
| 2.7 | 0.28 | 0.01 | 0.15 | 1.4 |
| 3.0 | 0.34 | 0.01 | 0.16 | 2.0 |
| 3.3 | 0.42 | 0.02 | 0.18 | 2.8 |
| 3.6 | 0.50 | 0.02 | 0.20 | 3.7 |

**Figure 13. Effect of Operating Frequency on Normal Mode Power Consumption (C8051F31x)**

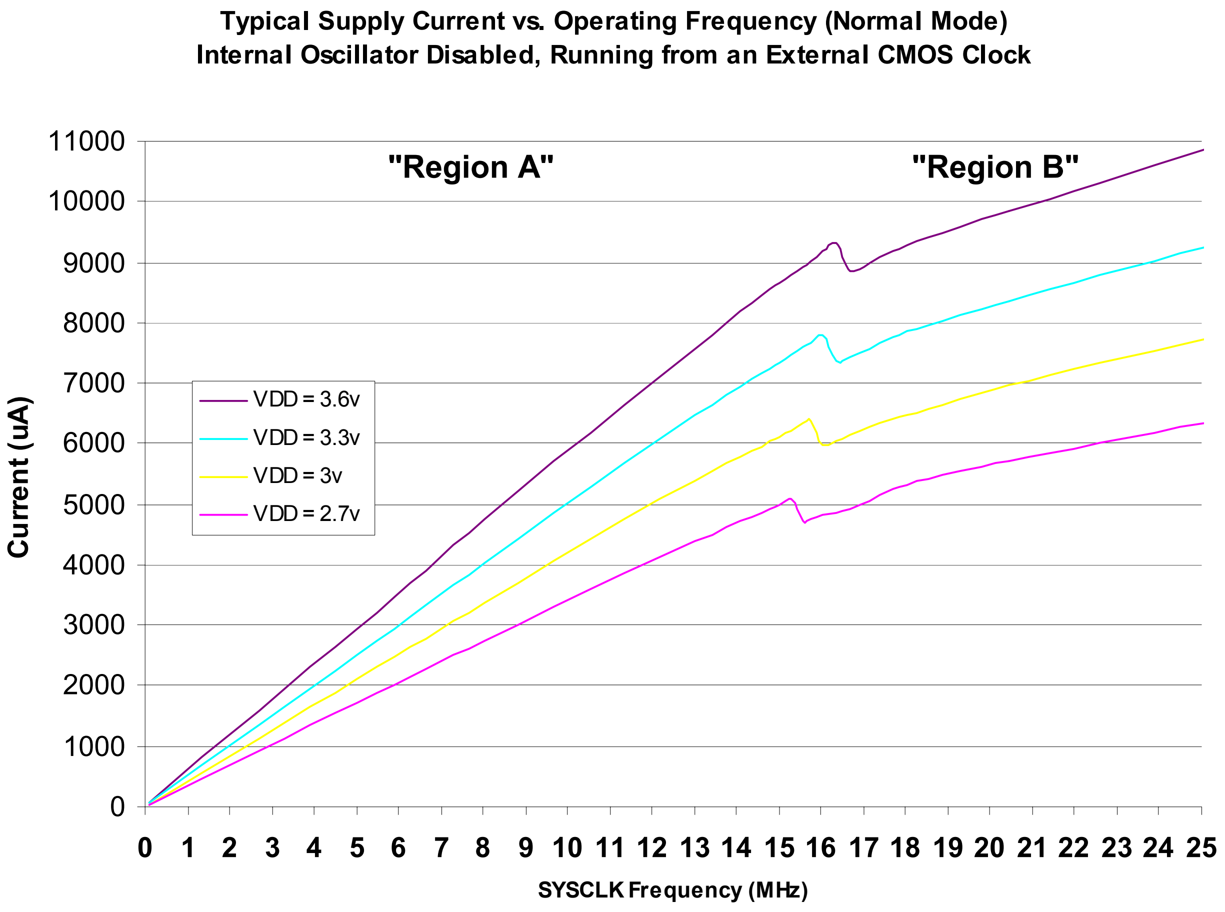


**Table 8. Slope and Offset Values for the Curves in Figure 13 (C8051F31x)**

| VDD | Slope A mA/MHz | Offset A mA | Slope B mA/MHz | Offset B mA |
|---|---|---|---|---|
| 2.7 | 0.34 | 0.03 | 0.16 | 2.5 |
| 3.0 | 0.41 | 0.04 | 0.18 | 3.2 |
| 3.3 | 0.49 | 0.05 | 0.21 | 4.1 |
| 3.6 | 0.58 | 0.05 | 0.24 | 5.0 |

When operating in “Region A”, turning off the FLASH one-shot by writing a ‘0’ to the FOSE bit in the FLSCL register will extend the “Region B” curve across the entire operating range of the device. This is only useful if operating near the switchover point, where “Region B” operation consumes less power than “Region A”.

When the CPU is in Idle mode, the Current vs. Operating Frequency curve is a single line over the operating range of the device. Figure 14 and Figure 15 show the effect of operating frequency on power consumption when the CPU is in Idle mode.

**Figure 14. Effect of Operating Frequency on Idle Mode Power Consumption (C8051F30x)**

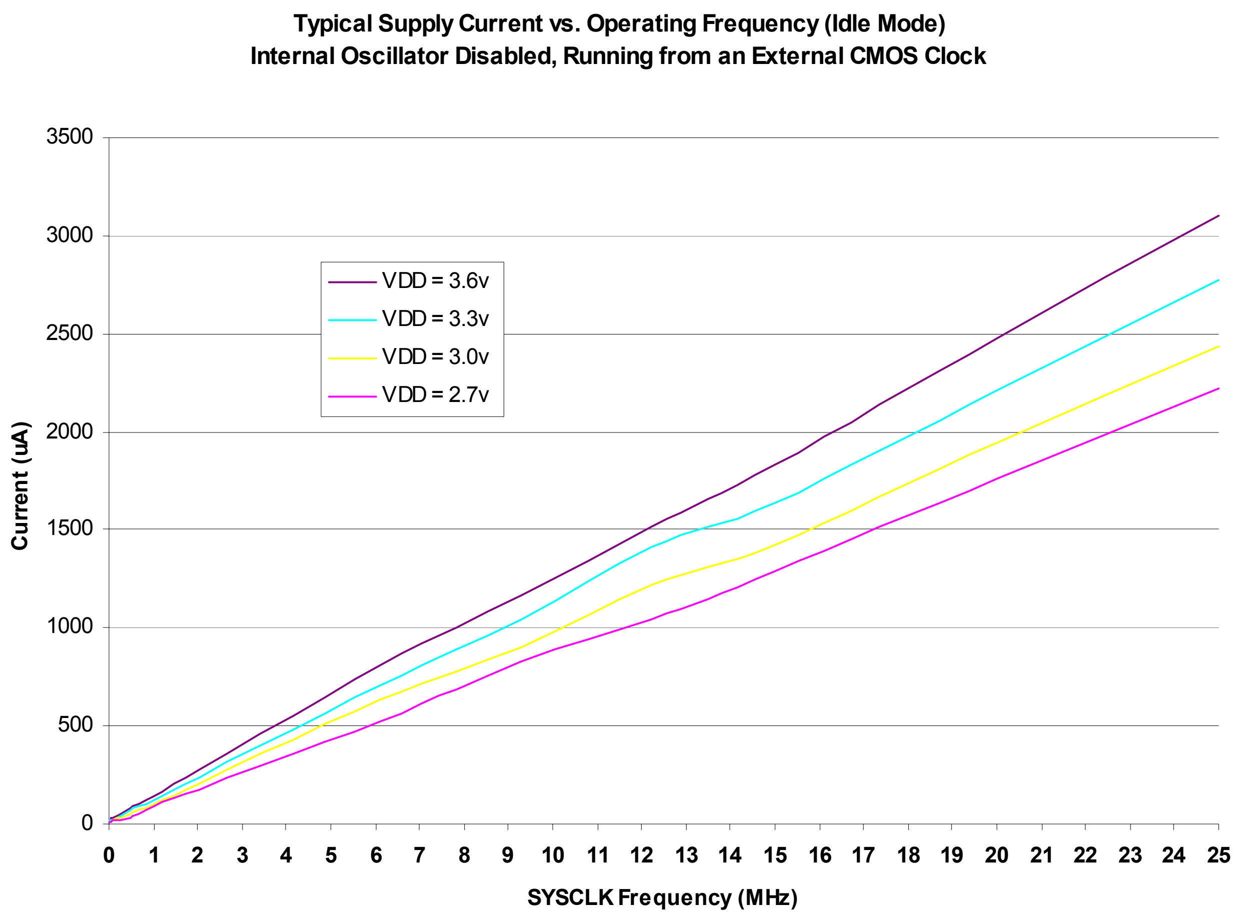


**Table 9. Slope and Offset Values for the Curves in Figure 14 (C8051F30x)**

| VDD | Slope mA/MHz | Offset mA |
|---|---|---|
| 2.7 | 0.09 | 0.00 |
| 3.0 | 0.10 | 0.01 |
| 3.3 | 0.11 | 0.02 |
| 3.6 | 0.12 | 0.02 |

## Figure 15. Effect of Operating Frequency on Idle Mode Power Consumption (C8051F31x)

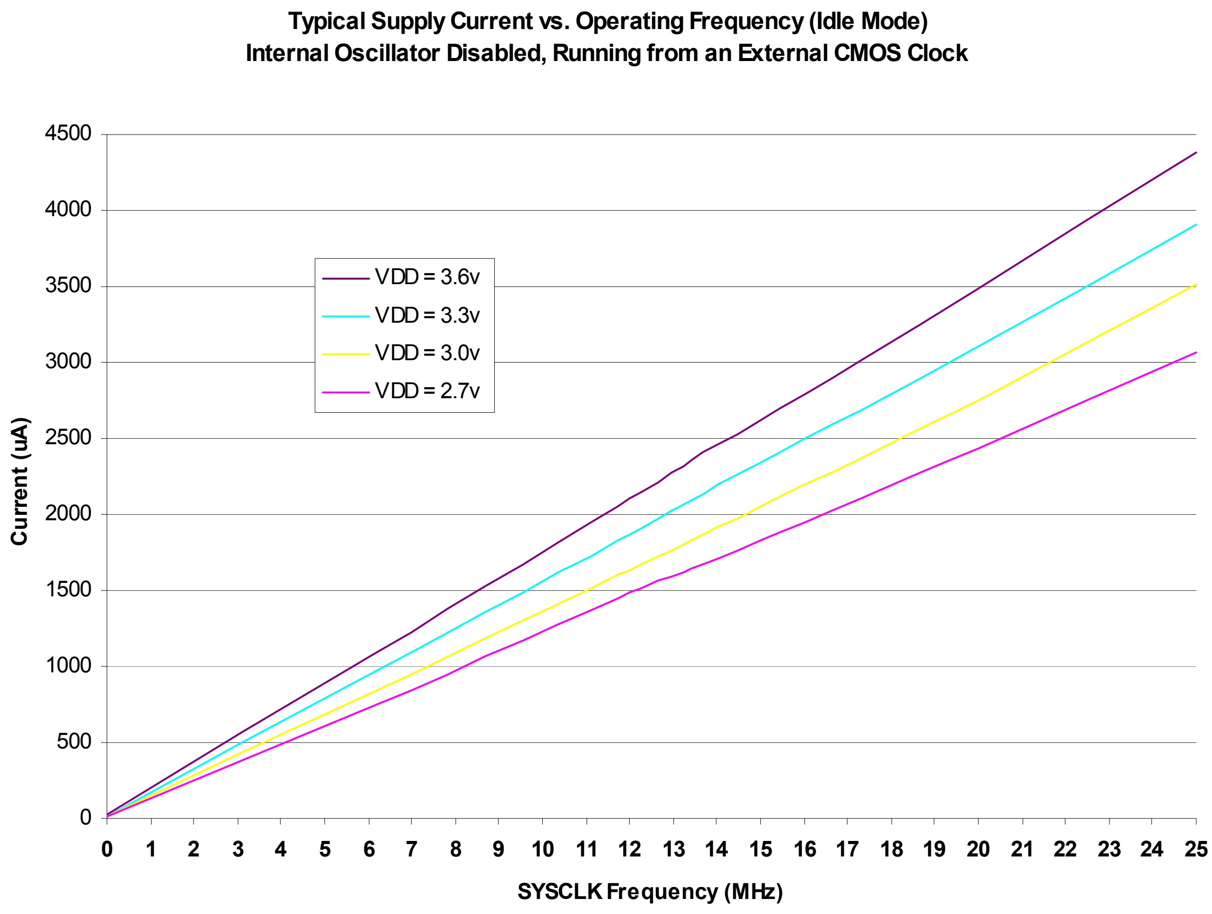


**Table 10. Slope and Offset Values for the Curves in Figure 15 (C8051F31x)**

| VDD | Slope<br>mA/MHz | Offset<br>mA |
|---|---|---|
| 2.7 | 0.12 | 0.00 |
| 3.0 | 0.14 | 0.01 |
| 3.3 | 0.16 | 0.01 |
| 3.6 | 0.17 | 0.01 |

## Digital Peripherals and Port I/O

Digital peripherals (timers, UART, PCA, SPI, etc.) account for a small percentage of the total power consumption. For example, operating a C8051F300 at 3.06 MHz (internal oscillator divided by 8) and 3.0 Volts, the average device supply current without any digital peripherals enabled is approximately 700 µA. This number typically increases by 18 µA ( 3 % ) when Timer 1 is started at its fastest clocking setting with UART0 constantly transmitting data. The power consumption for timers and UART depends on the frequency at which they are clocked and the supply voltage.

The Crossbar and configuration of GPIO pins to push-pull mode can also affect power consumption. In the example above, enabling the Crossbar, routing the UART0 TX signal to P0.4, and configuring the port pin to push-pull mode adds another 82 µA (10%) to the total device supply current. The power consumption for output port pins depends on the frequency that the pin is toggled and the external circuitry connected to the pin.

## *Analog Peripherals*

The ADC, temperature sensor, internal bias generator, and the internal oscillator consume power when enabled. Table 11  and Table 12  show typical current values for the analog peripherals on these devices. The internal bias generator is automatically enabled whenever the ADC, internal oscillator, or temperature sensor is enabled.

The peak ADC current during a conversion is typically 30% to 50% higher than when the ADC is not converting. The SAR Conversion Clock frequency and sampling rate also affect the power consumption. In general, increasing the SAR Conversion Clock frequency and decreasing the sampling rate will provide the greatest power savings because the ADC will spend less time in each conversion and more time idle between conversions.

**Table 11. Typical Current Contribution for Analog Peripherals at 3.0V (C8051F30x)**

| Peripheral | Typical Current Consumption |
|---|---|
| Internal Bias Generator | 110 µA |
| Temperature Sensor (ADC Enabled) | 85 µA |
| Temperature Sensor (ADC Disabled) | < 1 µA |
| ADC (enabled) | 430 µA |
| ADC (converting) | 630 µA |
| Internal Oscillator | 340 µA |
| Voltage Comparator | 0.4 - 7.6 µA (depending on speed mode) |

**Table 12. Typical Current Contribution for Analog Peripherals at 3.0V (C8051F31x)**

| Peripheral | Typical Current Consumption |
|---|---|
| Internal Bias Generator | 110 µA |
| Temperature Sensor (ADC Enabled) | 83 µA |
| Temperature Sensor (ADC Disabled) | < 1 µA |
| ADC (enabled) | 480 µA |
| ADC (converting) | 650 µA |
| Internal Oscillator | 360 µA |
| Voltage Comparator | 0.4 - 7.6 µA (depending on speed mode) |

## *Example Calculations*

The following examples show how to calculate the total device power consumption by adding the contributions from the oscillator, digital power, and analog power. The examples assume a supply voltage of 3.0V.

### Example 1: 32.768 kHz Watch Crystal in Idle Mode (C8051F31x)

In this example, we calculate the total device power consumption for operating a C8051F31x device from a 32.768 kHz watch crystal in Idle mode.

**Oscillator Current.** From Table 2 , the external oscillator requires **~4.1 µA** to drive the watch crystal.

**Digital Current.** Using the slope and offset information from Table 10 , the CPU current in Idle mode can be calculated using Equation 4.

**Equation 4. Calculating Digital Current from Slope and Offset**

$$\text{Current [mA]} = (\text{SYSCLK [MHz]} \times \text{Slope}) + \text{Offset}$$

In this example, the slope (0.14 mA/MHz) and offset (0.01 mA) give an estimated digital current of **~15 µA** for operation at 32.768 kHz and 3.0 V.

**Analog Current.** In this system, the internal oscillator is disabled when running from the external watch crystal. We add **0 µA** for the analog current contribution.

**Total Current.** The total supply current is **~19 µA** for a C8051F31x in this configuration. This number is an estimate and may vary by a few microamps in an actual system.

### Example 2: 24.5 MHz Internal Oscillator in Normal Mode with ADC On (C8051F30x)

In this example, we calculate the total device power consumption for operating a C8051F30x device from the internal 24.5 MHz calibrated oscillator in Normal mode with the ADC turned on.

**Oscillator Current.** From Table 11 , the internal oscillator requires **~340 µA** to generate the 24.5 MHz clock signal.

**Digital Current.** Using the slope and offset information from Table 7 , the CPU current in Normal mode can be calculated using Equation 4. In this example, the slope (0.16 mA/MHz) and offset (2.0 mA) in “Region B” of Figure 12 give an estimated digital current of **~6.0 mA** for operation at 24.5 MHz and 3.0 V.

**Analog Current.** Since the internal bias generator automatically switches on whenever the ADC or the internal oscillator are being used, it contributes 110 µA. When the ADC is enabled (not sampling), it contributes 430 µA. The total analog contribution comes to **540 µA**.

**Total Current.** There are two ways to estimate the total current from the data provided in this application note. First, we can sum the contributions from the oscillator, digital current, and analog current. An alternative is using Figure 10 to determine the combined digital, internal oscillator, and bias generator current then adding the ADC contribution to obtain the total supply current. Both methods should yield similar results; however, the second method is faster.

Using the first method, we add the 340 µA oscillator current, 6.0 mA digital current, and 540 µA analog current to obtain a total of **6.9 mA**.

Using the second method, we add 6.1 mA (supply current at 3.0V from Figure 10) and 430 µA ADC current for a total of **6.5 mA**.

The estimates from the two methods are within 5% of each other.

## Example 3: 25.000 MHz Crystal in Normal Mode with ADC On (C8051F30x)

In this example, we calculate the total device power consumption for operating a C8051F30x device from a 25.000 MHz crystal in Normal mode with the ADC on.

**Oscillator Current.** From Table 1 , the external oscillator requires **~4.1 mA** to drive the crystal.

**Digital Current.** Using the slope and offset information from Table 7 , the CPU current in Normal mode can be calculated using Equation 4.

In this example, the slope (0.16 mA/MHz) and offset (2.0 mA) in “Region B” of Figure 12 give an estimated digital current of **~6 mA** for operation at 25.0 MHz and 3.0 V.

**Analog Current.** In this system, the internal oscillator is disabled when running from the external crystal. Since we are using the ADC, we add the 110 µA internal bias current to the 430 µA ADC current for a total **540 µA** analog contribution.

**Total Current.** The total supply current is **~11 mA** for a C8051F30x in this configuration. Comparing this total to the total from Example 2, we find that using the internal oscillator saves approximately 4 mA while achieving a comparable operating frequency.

## Power Saving Strategies

In most low-power applications, the high performance processing capabilities of the device are not needed 100% of the time. The ability of these devices to switch between clock sources and power modes on-the-fly gives them the flexibility of performing high speed tasks and meet the requirements of a low power budget.

In most systems with low power requirements, the average power consumption is optimized. For example, in a battery powered application, the average current determines the battery life. Equation 5 shows how to calculate battery life in a system based on its average current and rating. Batteries contain a fixed amount of charge, specified in a battery datasheet in units of milliamp-hours (mA-h).

**Equation 5. Calculating Battery Life**

$$\text{Battery Life [hours]} = \frac{\text{Qtotal[mA-h]}}{\text{Iavg[mA]}}$$

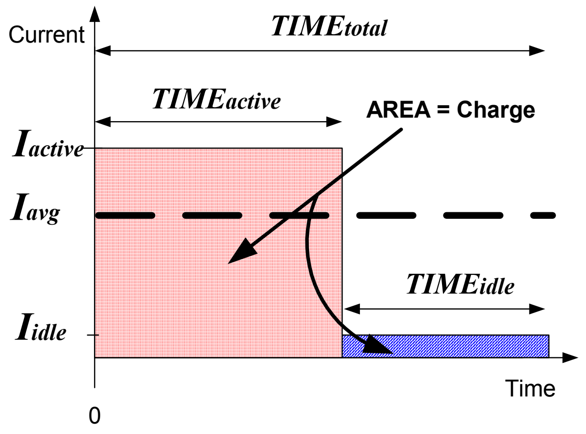
### *Minimizing Average Power Consumption*

There are two classes of optimizations that can be used to minimize average power consumption. The first kind involves adjusting system parameters that affect the system at all times. One of the main system level parameters is supply voltage. The supply voltage can be derived from a voltage regulator or from a battery. In low power systems, supply voltage should be minimized in order to save power.

The second kind involves structuring the firmware to save power. This involves having a high performance mode and a low power “sleep” mode. These two modes have different design criteria. The device should spend as much time as possible in “sleep” mode in order to save power.

Since the supply voltage is typically constant, minimizing average current is directly proportional to minimizing the average power consumption. Average current is the amount of charge consumed per unit time, or the area under a Current vs. Time chart divided by time, as shown in Figure 16.

**Figure 16. Average Current – Charge Consumed per Unit Time**



Equation 6 shows that the average current is the total charge (area) divided by the total time.

**Equation 6. Calculating Average Current**

$$\text{Iavg} = \frac{(I_{\text{active}} \times \text{Time}_{\text{active}}) + (I_{\text{idle}} \times \text{Time}_{\text{idle}})}{\text{Time}_{\text{total}}}$$

The charge required for a given task can be reduced by minimizing the “active” time or minimizing the peak active current. The designer should always consider minimizing “active” time and peak current to save power.

## Decreasing Supply Voltage

Supply voltage can have a large impact on power consumption. Low-power systems should always be designed to use the minimum supply voltage that allows the device to operate reliably within its specified voltage limits.

Many voltage regulators have +/-10% accuracy. If a regulator with this accuracy is used, the minimum design voltage should be 3.0 V, since the regulator output can vary between 2.7 V and 3.3 V.

An alternative to a voltage regulator is to use a battery. Lithium manganese dioxide batteries output 2.85 Volts for a majority of their useful life and can be directly connected to the power pins on the device. Batteries can provide a constant voltage that does not need regulation. In these systems, the on-chip VDD monitor should be enabled to ensure that the device is held in reset when the battery is drained.

### Designing a Low Power “Sleep” Mode

The design goal of a low-power “sleep” mode is to minimize current because the system can spend long intervals of time in this mode. A “sleep” mode can be implemented by putting the device in Idle or Stop mode. Stop mode provides a lower standby current than Idle mode, but Idle mode is easier to recover from. Examples of both implementations are provided at the end of this note.

**In “sleep” mode it is important to turn off any peripherals (ADC, Internal Oscillator, etc. ) that are not required.**

In “sleep” mode, it is usually best to operate on an external oscillator. This allows the system to disable the internal oscillator and operate from a very low frequency timebase. Two appealing external oscillator configurations to consider for “sleep” mode are a 32.768 kHz watch crystal and a single capacitor.

A capacitor oscillator can consume less power than a crystal, but is less accurate. The main advantage of using a capacitor oscillator is the ability to clock peripherals (such as timers) at rates less than 10 kHz. There is also a cost and PCB space savings associated with using a single capacitor, as opposed to a crystal, two loading capacitors and a resistor. If a high frequency crystal is used in the design, the loading capacitor connected to the XTAL2 pin can be used by the external oscillator in C mode to derive a low-frequency clock source for “sleep” mode.

### Designing a High Performance Mode

A high performance mode should be designed to accomplish tasks in a minimal amount of time so that the system can go back to “sleep” mode as quickly as possible. This involves adjusting the peak current and the SYSCLK frequency to reduce the area under the Current vs. Time curve. Example 1 shows a system in which the average power consumption is reduced by increasing the SYSCLK frequency in the high performance mode.

From a power standpoint, most systems will benefit by using the internal oscillator in high performance mode.

## *Measuring Average Current*

The average system current is best calculated by measuring the power consumption in various modes using lab bench equipment and estimating the amount of time the system spends in each mode. Example 1 shows the power calculations for a sampling system that uses the on-chip ADC.

## *Examples*

Four examples are provided that demonstrate the concepts discussed in this application note. The software for these examples is included at at the end of this note. The examples are:

- **ADC Sampling (C8051F30x).** This example compares the power savings of two different ADC sampling systems. One system uses a 32.768 kHz crystal while sampling and the other switches to the internal oscillator to minimize the time that the ADC remains on. Both systems are identical in Idle mode.

- **Waking From Idle mode on UART Activity (C8051F30x).** This example shows how a "sleep" mode can be implemented using the device's Idle mode. The device wakes up when it detects UART activity.

- **Waking From Stop Mode Using a Comparator (C8051F30x and C8051F31x).** This example shows how a "sleep" mode can be implemented using the device's Stop mode. The device wakes up when it detects a button press. A connection diagram of how to wake on SMBus activity is also included in this example.

- **32.768 kHz Watch Crystal Low Power Startup Procedure (C8051F30x).** This example shows how to start a 32.768 kHz watch crystal, minimizing the time in Normal mode waiting for the crystal to start.

## Example 1: ADC Sampling

The two systems in this example take an ADC sample from the on-chip temperature sensor at a rate of 10 Hz. A 32.768 kHz watch crystal and associated loading capacitors and shunt resistor are connected between XTAL1 and XTAL2. Timer 2 overflows every 100 ms generating an interrupt that wakes the device up from Idle mode. When the device wakes up, it captures one ADC sample then goes back into Idle mode until the next interrupt occurs.

Since this system is battery powered, one of its goals is to minimize the amount of charge consumed per ADC sample. Since charge is current integrated over time, there is a choice between minimizing time or peak current required to take a sample. For example, to capture the ADC sample, the device may switch to the 3 MHz internal oscillator and use a larger amount of current for a short period of time or remain at 32 kHz and use less current for a longer period of time. Figure 17 and Figure 18 show current vs. time for the two systems, one with minimized time in Normal mode and the other with minimized peak current.

Both systems use 4.8 µA in Idle mode to drive a 32 kHz watch crystal and Timer 2.

When the system in Figure 17 wakes from Idle mode, it turns on the internal oscillator and the ADC, switches SYSCLK to the internal oscillator in divide-by-8 mode, and starts the ADC conversion. After the conversion is complete, it reads the ADC value, disables the ADC and internal oscillator, and puts the CPU back in Idle mode. **To capture the ADC sample, the device spends less than 400 µs consuming a peak current of 2.2 mA.**

**Figure 17. Current vs. Time (Minimized Time in Normal Mode)**

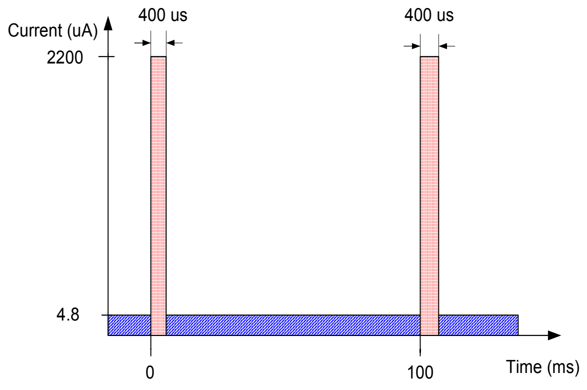
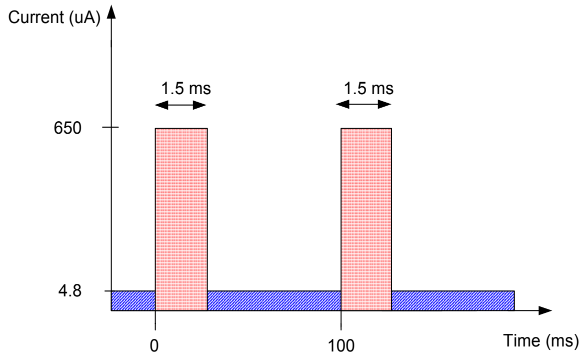


**Figure 18. Current vs. Time (Minimized Peak Current)**



When the system in Figure 18 wakes up from Idle mode, it immediately turns on the ADC and initiates a sample. It does not turn on the internal oscillator and SYSCLK remains at 32.768 kHz. After the conversion is complete, it reads the ADC value, turns off the ADC and goes into Idle mode. **To capture the ADC sample, the device spends less than 1.5 ms consuming a peak current of 0.65 mA.**

Using Equation 6, the system in Figure 17 has an average current of **14 µA**. If the system is powered from an ideal 3.0 Volt (actual voltage 2.85V) lithium manganese dioxide watch battery with a capacity of 575 mA-h, the battery life would be approximately 42,000 hours, or over four and a half years.

The average current for the system in Figure 18 is **15 µA**. If this system is powered with the same 3.0 Volt lithium manganese dioxide watch battery, the battery life would also be around 40,000 hours.

**In this example, increasing the system clock frequency in the high power mode decreased the average current.**

These examples do not take into account changes in temperature and battery performance variations over the life of the battery. One source for Application Manuals and Battery Datasheets is the *Technical Info* section of the Energizer web site (www.energizer.com).

## Example 2: Waking from Idle Mode on UART Activity

This example configures External Interrupt 0 to wake the device from Idle mode on detecting activity on the UART receive signal. Once the CPU comes out of Idle Mode, it disables External Interrupt 0 as shown in Figure 19, switches to the internal oscillator, enables UART0 reception, and discards the first UART frame. In this system, the first UART frame received after a period of inactivity is interpreted as a “wakeup” signal.

In order for the device to wake up and remain synchronized for UART communication, the “wakeup” signal has to have exactly one falling edge followed by one rising edge, as shown in Figure 20. Since the UART start bit is always ‘0’

**Figure 19. Waking from Idle Mode on UART Activity**

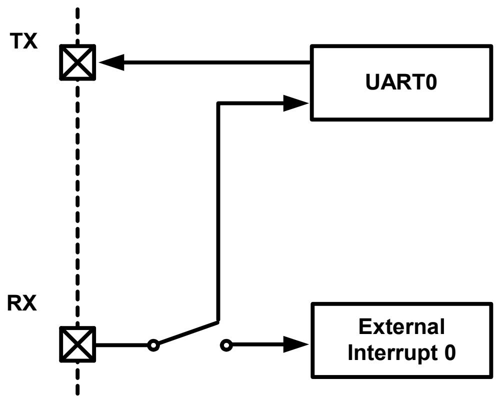
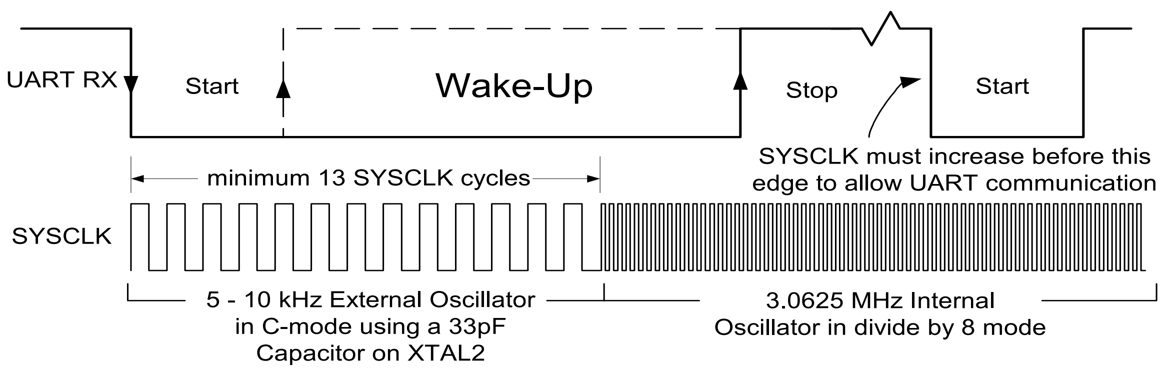


**Figure 20. UART “Wake-up” Signal**


UART RX
Start
Wake-Up
Stop
Start
SYSCLK must increase before this edge to allow UART communication
minimum 13 SYSCLK cycles
SYSCLK
5 - 10 kHz External Oscillator in C-mode using a 33pF Capacitor on XTAL2
3.0625 MHz Internal Oscillator in divide by 8 mode

and the stop bit is always ‘1’, the data bits can be any value as long as there is only one falling edge and one rising edge in the received signal. Keep in mind that characters are sent LSB first. Example “wakeup” characters are 0x00, 0xFF, and 0xF0 but not 0x0F.

The maximum baud rate supported by the system will be limited by the frequency of the external oscillator used while the device is in Idle mode. When programming in ‘C’, it takes a minimum of 13 SYSCLK cycles after the falling edge of the “wakeup” signal to enter an External Interrupt ISR and switch to the internal oscillator. If the UART character following the “wakeup” signal arrives before the internal oscillator is enabled, then the two UART systems may become unsynchronized.

Equation 7 can be used to calculate the maximum baud rate supported by a system with a given external clock frequency. Equation 7 is based upon 10 bits per UART frame and 13 external clock cycles required to turn on the internal oscillator. For example, if the external oscillator is a capacitor oscillating at 5 kHz, the highest standard baud rate supported by the system would be 2400 baud.

**Equation 7. Calculating Maximum Supported Baud Rate**

$$\text{MAX\_BAUD\_RATE} < \left(\frac{10}{13} \times \text{EXTCLK}\right)$$

The software for this example uses a 33 pF capacitor connected to the XTAL2 pin as an external oscillator. Considering stray capacitance and other effects, the system clock frequency is between 5 kHz and 10 kHz when the external oscillator is selected and the external oscillator drive current (XFCN) is set to its lowest value. The “wakeup” signal chosen in this example is ‘0xFF’. When the system wakes up, it waits for the next character and transmits a string containing that character. Then it disables UART reception, enables External Interrupt 0, and goes back into Idle mode.

The average power consumption for this system scales with the amount of UART activity. As Figure 21 shows, the system consumes 4.2 µA of current in Idle mode and approximately 1.5 mA in Active mode at 3.0 Volts.

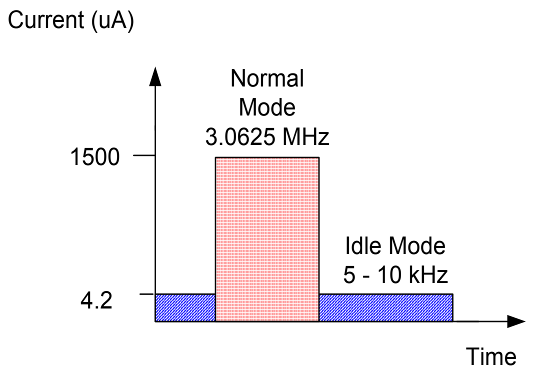
In some applications, it is possible to recover the first character but this places more restrictions on maximum UART baud rate and minimum external clock frequency.

## Example 3: Waking from Stop Mode Using a Comparator

This example shows how a “sleep” mode can be implemented in a system using the Stop mode of the CPU. This application implements a software counter that is incremented approximately every second when the device is in Normal mode. If the system is powered down when it is in Stop mode, the counter resumes counting the next time it enters Normal mode. If the system is powered down while it is in Normal mode, the counter will reset to zero. Every time the counter is updated, the current value of the counter is printed to the UART.

The S2 switch toggles the system back and forth between Stop mode and Normal mode. On power up, the system is in Stop mode. The system enters

**Figure 21. Example 2 Active Mode vs. Idle Mode Power Consumption**



Normal mode when the S2 switch on the target board is pressed, causing a Comparator 0 reset.

Upon entering Normal mode, External Interrupt 0 (/INT0) is activated to sense the S2 switch and Comparator 0 is disabled as a reset source. Pressing S2 in Normal mode will cause the INT0_ISR to put the system in Stop mode.

Three target board connections are needed to run this example on the C8051F30x and four connections are needed for the C8051F31x. They are circled in Figure 22 and Figure 23. The C8051F30x requires one less pin because the CP0+ signal and the External Interrupt 0 input can use the same pin.

In this example, the LED on the target board is used to conveniently provide a voltage that is between VDD and GND. This voltage can be generated using a resistor network or DC power supply. Also, the target board provides a convenient pull-up resistor (R4) for the S2 switch. The on-chip

**Figure 22. Example 3 Target Board Connection Diagram (C8051F30x)**

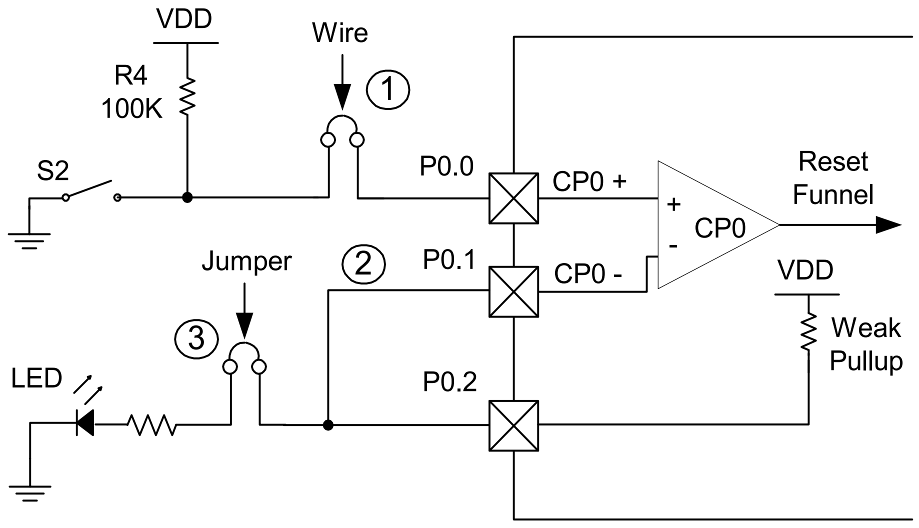
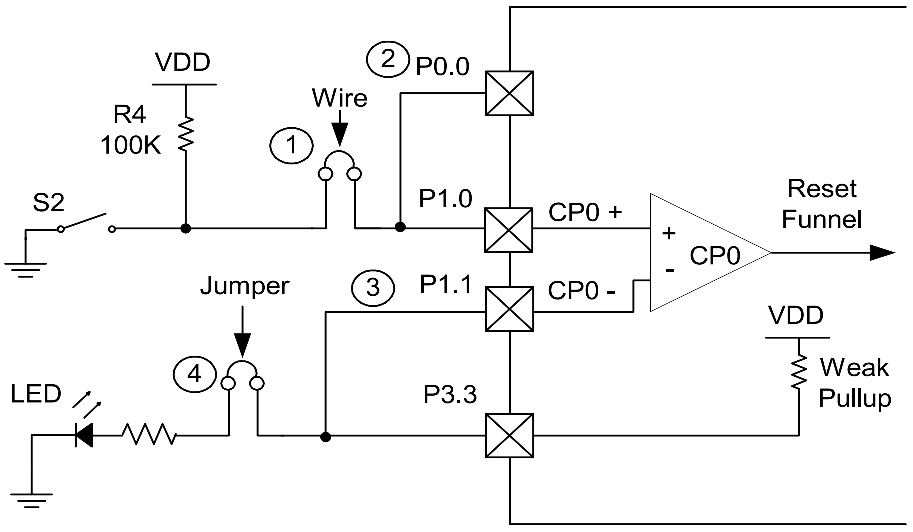


**Figure 23. Example 3 Target Board Connection Diagram (C8051F31x)**



weak pull-up for the port pin can replace R4 if the port pin is configured as a digital input and weak pull-ups are enabled.

The voltage at the CP0- input is used by Comparator 0 to detect if the CP0+ signal is high or low. When Comparator 0 is enabled as a reset source, it will generate a reset when the non-inverting (CP0+) input is lower than the inverting (CP0-) input.

The system remembers its state by storing a single-byte <SLEEP> flag and a copy of the counter in FLASH. The <SLEEP> flag is defined to be TRUE if it has a value of 0x55. All other values are defined as FALSE. On every reset, the device decodes the <SLEEP> variable stored in FLASH and the RSTSRC register to determine its state. **Note that if the PORSF (Power On Reset Flag) is set in the RSTSRC register, then all other flag bits in that register are undefined.** Table 13 shows how the device decodes the RSTSRC register to determine its state.

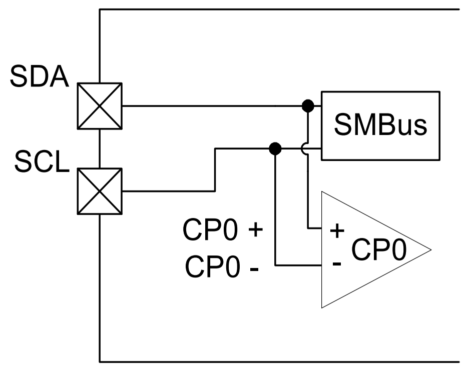
The techniques in this example can be used to wake a device from Stop mode on SMBus activity. An SMBus start signal consists of a

**Table 13. Reset Source Register Decoding for Example 3**

| Reset Type | RSTSRC | Action Taken |
|---|---|---|
| Hardware,<br>Power On,<br>Missing Clock Detector,<br>Watchdog Timer,<br>or FLASH error | 0x01,<br>0x02,<br>0x04,<br>0x08,<br>or 0x40 | Prepare the device for Stop mode.<br>1. Enable and configure Comparator 0.<br>2. Enable Comparator 0 as a reset source.<br>3. Go into Stop Mode waiting for the User to press the S2 switch to generate a comparator reset. |
| Comparator | 0x20 | If the <SLEEP> flag is set to TRUE, then prepare device to operate in Normal Mode and resume counting.<br>1. Restore the <COUNT> (READ FLASH).<br>2. Set <SLEEP> flag to FALSE (ERASE FLASH).<br>3. Enable External Interrupt 0. This interrupt will save the <COUNT> in FLASH, set the <SLEEP> flag, and put the CPU in Stop mode when S2 is pressed.<br><br>If the <SLEEP> flag is set to FALSE, start counting at zero. This condition only happens the first time after a firmware download or if power is lost while the device is in Normal mode.<br>1. Set <COUNT> to zero.<br>2. Enable External Interrupt 0.<br><br>The device should now be operating in Normal mode. |

falling edge on SDA while SCL is high. When a start condition is detected, the comparator resets the device. If the device is a slave, it will NACK the first transfer (considered a “wakeup” signal), but respond to all transfers that follow it. Figure 24 shows a possible connection diagram.

**Figure 24. Example Wake-On-SMBus Connection Diagram**



## Example 4: 32.768 kHz Watch Crystal Low-Power Startup Procedure

This example shows how to start an external 32.768 kHz watch crystal in low power applications. Since a watch crystal can take longer than one second to start, the device goes into Idle mode after turning on the external oscillator. At 100 ms intervals, Timer 2 generates an interrupt and wakes the device to check the XTLVLD flag. Once the watch crystal has started, the internal oscillator is disabled and the system uses the crystal as the system clock source.

## Software Examples

### Example 1A: ADC Sampling System (Minimized “Active” Time)

```
//-----------------------------------------------------------------------------
// ADC_A_F30x.c
//-----------------------------------------------------------------------------
// Copyright 2003 Cygnal Integrated Products, Inc.
//
// AUTH: FB
// DATH: 23 JAN 03
//
// This example captures ADC samples at a rate of 10 Hz from P0.0 and is clocked
// from a 32.768 kHz watch crystal. This program keeps the CPU in Idle mode
// until a Timer2 overflow. The Timer2 interrupt turns on the ADC and internal
// oscillator, takes a sample, then turns the ADC and the internal oscillator
// off to save power. While peak current increases  when the internal oscillator
// is turned on, the power saved by minimizing the time needed to take a sample
// is more than the power consumed by increasing the system clock frequency.
//
//
// Target: C8051F30x
//
// Tool chain: KEIL Eval ‘c’
//

//-----------------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------------
#include <c8051f300.h>                    // SFR declarations
#include <math.h>

//-----------------------------------------------------------------------------
// 16-bit SFR Definitions for ‘F30x
//-----------------------------------------------------------------------------

sfr16 DP       = 0x82;                    // data pointer
sfr16 TMR2RL   = 0xca;                    // Timer2 reload value
sfr16 TMR2     = 0xcc;                    // Timer2 counter
sfr16 PCA0CP1  = 0xe9;                    // PCA0 Module 1 Capture/Compare
sfr16 PCA0CP2  = 0xeb;                    // PCA0 Module 2 Capture/Compare
sfr16 PCA0     = 0xf9;                    // PCA0 counter
sfr16 PCA0CP0  = 0xfb;                    // PCA0 Module 0 Capture/Compare

//-----------------------------------------------------------------------------
// Global CONSTANTS
//-----------------------------------------------------------------------------

#define INTCLK       24500000 / 8         // Internal Oscillator frequency
                                          // in Hz (divide by 8 mode)
#define EXTCLK       32768                // Frequency for 32.768 kHz External
                                          // crystal oscillator
#define SAMPLERATE   10                   // ADC Sampling Rate in Hz


sbit LED = P0^2;                          // LED=’1’ means ON
sbit SW2 = P0^3;                          // SW2=’0’ means switch pressed
```

```
//-----------------------------------------------------------------------------
// Global VARIABLES
//-----------------------------------------------------------------------------
char ADC_READING = 0;

//-----------------------------------------------------------------------------
// Function PROTOTYPES
//-----------------------------------------------------------------------------
void SYSCLK_Init (void);
void PORT_Init (void);
void Crystal_Stabilize (void);
void Timer2_Init (int counts);
void Timer2_ISR (void);

//-----------------------------------------------------------------------------
// MAIN Routine
//-----------------------------------------------------------------------------
void main (void) {

   // disable watchdog timer
   PCA0MD &= ~0x40;                       // WDTE = 0 (clear watchdog timer
                                          // enable)

   PORT_Init();                           // initialize the Crossbar and GPIO
   SYSCLK_Init();                         // start external oscillator
   Timer2_Init(EXTCLK/8/SAMPLERATE);      // configure Timer2 to overflow at
                                          // <SAMPLERATE> times per second

   EA = 1;                                // enable global interrupts

   while(1){
      PCON |= 0x01;                       // put the device in Idle mode
   }
}



//-----------------------------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------------------------
//
// This routine initializes the system clock to use the external 32.768 kHz
// watch crystal as its clock source and disables the internal oscillator.
//
void SYSCLK_Init (void)
{

   int i;                                 // delay counter

   OSCXCN = 0x61;                         // start external oscillator

   for (i=0; i < 256; i++) ;              // wait for osc to start up

   while (!(OSCXCN & 0x80)) ;             // wait for crystal osc. to settle


   Crystal_Stabilize();                   // wait for crystal osc. to stablilize
   OSCXCN = 0x60;                      // decrease crystal drive current
   RSTSRC = 0x04;                      // enable missing clock detector
   OSCICN = 0x08;                      // switch to external oscillator


}

//-----------------------------------------------------------------------------
// Crystal_Stabilize
//-----------------------------------------------------------------------------
//
// Low-frequency crystal stabilization wait routine:
//
// This routine measures the period of the external oscillator with respect
// to the internal oscillator and loops until the external oscillator period is
// measured to be within 4 internal oscillator periods for 500 cycles in
// a row.  This is only necessary for tuning fork crystals, which have
// abnormally long stabilization times (on the order of seconds).
//
// Assumes that the internal oscillator operating in divide-by-8 mode is
// selected as the system clock source.  Also assumes that the external
// oscillator has been enabled, configured, and is oscillating.
//
// Here we measure the number of system clocks in 8 “EXTCLK/8” periods.
// We compare successive measurements.  When we obtain 500 measurements
// in a row that are all within 4 system clocks of each other the
// routine will exit.  This condition will only occur once the crystal
// oscillator has fully stabilized at its resonant frequency.
//
// Note that this can take several seconds.
//
void Crystal_Stabilize (void)
{
   int current, last;                  // used in osc. stabilization check
   int tolerance_count;

   // init PCA0
   PCA0CN = 0x00;                      // Stop counter; clear all flags
   PCA0MD = 0x0b;                      // PCA counts in IDLE mode;
                                       // EXTCLK / 8 is time base;
                                       // overflow interrupt is enabled

   // init Timer0
   TCON &= ~0x30;                      // Stop timer; clear TF0
   TMOD &= ~0x0f;                      // Timer0 in 16-bit counter mode
   TMOD |=  0x01;
   CKCON |= 0x08;                      // Timer0 counts SYSCLKs

   tolerance_count = 500;              // wait for 500 external cycles in
                                       // a row to lie within 4 internal
                                       // clocks of each other
   current = 0;

   do {
      PCA0CN = 0x00;
      PCA0L = 0xFF;                    // set PCA time base to ‘-1’
      PCA0H = 0xFF;
      TCON &= ~0x30;
      TH0 = 0x00;                          // init T0 time base
      TL0 = 0x00;

      // start PCA0
      CR = 1;
      while (CF == 0);                     // wait for edge
      TR0 = 1;                             // Start Timer0
      CF = 0;                              // clear PCA overflow
      PCA0L = -8;                          // set PCA to overflow in 8 cycles
      PCA0H = (-8) >> 8;
      while (CF == 0);
      TR0 = 0;
      last = current;
      current = (TH0 << 8) | TL0;
      if (abs (current - last) > 4) {
         tolerance_count = 500;            // falls outside bounds; reset
                                           // counter
      } else {
         tolerance_count--;                // in-bounds; update counter
      }

   } while (tolerance_count != 0);

}

//-----------------------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------------------
//
// Configure the Crossbar and GPIO ports.
// P0.0 - ADC Input
// P0.1 -
// P0.2 - XTAL1
// P0.3 - XTAL2
// P0.4 -
// P0.5 -
// P0.6 -
// P0.7 - C2D
//
void PORT_Init (void)
{
   XBR0     = 0x0d;                        // skip crystal pins and P0.0 in crossbar
   XBR2     = 0x40;                        // enable crossbar and weak pull-ups

   P0MDIN  &= ~0x0c;                       // configure XTAL1 and XTAL2 as analog
                                           // inputs
   P0MDIN  &= ~0x01;                       // configure P0.0 as an analog input
}

//-----------------------------------------------------------------------------
// Timer2_Init
//-----------------------------------------------------------------------------
//
// Configure Timer2 to auto-reload at interval specified by <counts>
// using EXTCLK / 8 as its time base.
//
void Timer2_Init (int counts)
{
   TMR2CN = 0x01;                          // Stop Timer2;
```

```
                                          // Timer2 timebase is EXTCLK/8

   TMR2RL = -counts;                      // Init reload value
   TMR2 = TMR2RL;                         // Init Timer2
   ET2 = 1;                               // enable Timer2 interrupts
   TR2 = 1;                               // start Timer2
}



//-----------------------------------------------------------------------------
// Timer2_ISR
//-----------------------------------------------------------------------------
//
// This ISR is called at <SAMPLERATE> Hz on Timer2 overflows
//
void Timer2_ISR (void) interrupt 5
{

   TF2H = 0;                              // clear Timer2 overflow flag

   OSCICN |= 0x04;                        // Start Internal Oscillator
   OSCICN &= ~0x08;                       // Switch to Internal Oscillator

   ADC0CN = 0x80;                         // enable ADC

   REF0CN |= 0x0A;                        // Select voltage reference and enable
                                          // bias generator

   AMX0SL = 0x80;                         // ADC in single-ended mode sampling P0.0

   ADC0CF = (INTCLK << 3);                // Set SAR clock frequency to ~ 3MHz

   ADC0CF |= 0x01;                        // Set PGA gain

   // Settling time starts at this point. Sampling should not start until
   // the appropriate settling time has passed. Each SYSCLK cycle is 326.5 ns.

   AD0INT = 0;                            // Clear conversion complete flag
   AD0BUSY = 1;                           // Start a conversion
   while(!AD0INT);                        // Wait until conversion complete

   AD0EN = 0;                             // Disable ADC
   REF0CN &= ~0x02;                       // Turn off bias generator

   ADC_READING = ADC0;                    // Capture ADC Reading

   OSCICN |= 0x08;                        // Switch to external oscillator
   OSCICN &= ~0x07;                       // disable internal oscillator

}
```

## Example 1B: ADC Sampling System (Minimized "Active" Peak Current)

```
//-----------------------------------------------------------------------------
// ADC_B_F30x.c
//-----------------------------------------------------------------------------
// Copyright 2003 Cygnal Integrated Products, Inc.
//
// AUTH: FB
// DATH: 23 JAN 03
//
// This example captures ADC samples at a rate of 10 Hz from P0.0 and is clocked
// from a 32.768 kHz watch crystal. This program keeps the CPU in Idle mode
// until a Timer2 overflow. The Timer2 interrupt turns on the ADC, takes a
// sample, then turns it off to save power.
//
// This program is meant to be used as a comparison to ADC_A_F30x to show that
// reducing the peak current required to take an ADC sample does not always
// save power. In this case, decreasing the SYSCLK frequency increased the
// average system current because the ADC was "on" for a longer period of time.
//
// Target: C8051F30x
//
// Tool chain: KEIL Eval 'c'
//

//-----------------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------------
#include <c8051f300.h>                    // SFR declarations
#include <math.h>

//-----------------------------------------------------------------------------
// 16-bit SFR Definitions for 'F30x
//-----------------------------------------------------------------------------

sfr16 DP       = 0x82;                    // data pointer
sfr16 TMR2RL   = 0xca;                    // Timer2 reload value
sfr16 TMR2     = 0xcc;                    // Timer2 counter
sfr16 PCA0CP1  = 0xe9;                    // PCA0 Module 1 Capture/Compare
sfr16 PCA0CP2  = 0xeb;                    // PCA0 Module 2 Capture/Compare
sfr16 PCA0     = 0xf9;                    // PCA0 counter
sfr16 PCA0CP0  = 0xfb;                    // PCA0 Module 0 Capture/Compare

//-----------------------------------------------------------------------------
// Global CONSTANTS
//-----------------------------------------------------------------------------

#define INTCLK       24500000 / 8         // Internal Oscillator frequency
                                          // in Hz (divide by 8 mode)
#define EXTCLK       32768                // Frequency for 32.768 kHz External
                                          // crystal oscillator
#define SAMPLERATE   10                   // ADC Sampling Rate in Hz

sbit LED = P0^2;                          // LED='1' means ON
sbit SW2 = P0^3;                          // SW2='0' means switch pressed

//-----------------------------------------------------------------------------
```

```
// Global VARIABLES
//-----------------------------------------------------------------------------
char ADC_READING = 0;

//-----------------------------------------------------------------------------
// Function PROTOTYPES
//-----------------------------------------------------------------------------
void SYSCLK_Init (void);
void PORT_Init (void);
void Crystal_Stabilize (void);
void Timer2_Init (int counts);
void Timer2_ISR (void);
//-----------------------------------------------------------------------------
// MAIN Routine
//-----------------------------------------------------------------------------
void main (void) {

   // disable watchdog timer
   PCA0MD &= ~0x40;                       // WDTE = 0 (clear watchdog timer
                                          // enable)

   PORT_Init();                           // initialize the Crossbar and GPIO
   SYSCLK_Init();                         // start external oscillator
   Timer2_Init(EXTCLK/8/SAMPLERATE);      // configure Timer2 to overflow at
                                          // <SAMPLERATE> times per second

   EA = 1;                                // enable global interrupts

   while(1){

      PCON |= 0x01;                       // put the device in idle mode
   }
}



//-----------------------------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------------------------
//
// This routine initializes the system clock to use the external 32.768 kHz
// watch crystal as its clock source and disables the internal oscillator.
//
void SYSCLK_Init (void)
{

   int i;                                 // delay counter

   OSCXCN = 0x61;                         // start external oscillator

   for (i=0; i < 256; i++) ;              // wait for osc to start up

   while (!(OSCXCN & 0x80)) ;             // wait for crystal osc. to settle


   Crystal_Stabilize();


   OSCXCN = 0x60;                         // decrease XFCN (crystal drive current)
   RSTSRC = 0x04;                        // enable missing clock detector
   OSCICN = 0x08;                        // switch to external oscillator


}

//-----------------------------------------------------------------------------
// Crystal_Stabilize
//-----------------------------------------------------------------------------
//
// Low-frequency crystal stabilization wait routine:
//
// This routine measures the period of the external oscillator with respect
// to the internal oscillator and loops until the external oscillator period is
// measured to be within 4 internal oscillator periods for 500 cycles in
// a row.  This is only necessary for tuning fork crystals, which have
// abnormally long stabilization times (on the order of seconds).
//
// Assumes that the internal oscillator operating in divide-by-8 mode is
// selected as the system clock source.  Also assumes that the external
// oscillator has been enabled, configured, and is oscillating.
//
// Here we measure the number of system clocks in 8 “EXTCLK/8” periods.
// We compare successive measurements.  When we obtain 500 measurements
// in a row that are all within 4 system clocks of each other the
// routine will exit.  This condition will only occur once the crystal
// oscillator has fully stabilized at its resonant frequency.
//
// Note that this can take several seconds.
//
void Crystal_Stabilize (void)
{
   int current, last;                    // used in osc. stabilization check
   int tolerance_count;

   // init PCA0
   PCA0CN = 0x00;                        // Stop counter; clear all flags
   PCA0MD = 0x0b;                        // PCA counts in IDLE mode;
                                         // EXTCLK / 8 is time base;
                                         // overflow interrupt is enabled


   // init Timer0
   TCON &= ~0x30;                        // Stop timer; clear TF0
   TMOD &= ~0x0f;                        // Timer0 in 16-bit counter mode
   TMOD |=  0x01;
   CKCON |= 0x08;                        // Timer0 counts SYSCLKs

   tolerance_count = 500;                // wait for 500 external cycles in a row
                                         // to lie within 4 internal clocks of each
                                         // other
   current = 0;

   do {
      PCA0CN = 0x00;
      PCA0L = 0xFF;                      // set PCA time base to ‘-1’
      PCA0H = 0xFF;
      TCON &= ~0x30;
      TH0 = 0x00;                        // init T0 time base
      TL0 = 0x00;
      // start PCA0
      CR = 1;
      while (CF == 0);                    // wait for edge
      TR0 = 1;                            // Start Timer0
      CF = 0;                             // clear PCA overflow
      PCA0L = -8;                         // set PCA to overflow in 8 cycles
      PCA0H = (-8) >> 8;
      while (CF == 0);
      TR0 = 0;
      last = current;
      current = (TH0 << 8) | TL0;
      if (abs (current - last) > 4) {
         tolerance_count = 500;           // falls outside bounds; reset
                                          // counter
      } else {
         tolerance_count--;               // in-bounds; update counter
      }

   } while (tolerance_count != 0);

}

//-----------------------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------------------
//
// Configure the Crossbar and GPIO ports.
// P0.0 - ADC Input
// P0.1 -
// P0.2 - XTAL1
// P0.3 - XTAL2
// P0.4 -
// P0.5 -
// P0.6 -
// P0.7 - C2D
//
void PORT_Init (void)
{
   XBR0     = 0x0d;                       // skip crystal pins and P0.0 in crossbar
   XBR2     = 0x40;                       // enable crossbar and weak pull-ups

   P0MDIN  &= ~0x0c;                      // configure XTAL1 and XTAL2 as analog
                                          // inputs
   P0MDIN  &= ~0x01;                      // configure P0.0 as an analog input
}

//-----------------------------------------------------------------------------
// Timer2_Init
//-----------------------------------------------------------------------------
//
// Configure Timer2 to auto-reload at interval specified by <counts>
// using EXTCLK / 8 as its time base.
//
void Timer2_Init (int counts)
{
   TMR2CN = 0x01;                         // Stop Timer2;
                                          // Timer2 timebase is EXTCLK/8
   TMR2RL = -counts;                         // Init reload value
   TMR2 = TMR2RL;                            // Init Timer2
   ET2 = 1;                                  // enable Timer2 interrupts
   TR2 = 1;                                  // start Timer2
}


//-----------------------------------------------------------------------------
// Timer2_ISR
//-----------------------------------------------------------------------------
//
// This ISR is called at <SAMPLERATE> Hz on Timer2 overflows
//
void Timer2_ISR (void) interrupt 5
{


   TF2H = 0;                                 // clear Timer2 overflow flag

   ADC0CN = 0x80;                            // enable ADC

   REF0CN |= 0x0A;                           // Select voltage reference and enable
                                             // bias generator

   AMX0SL = 0x80;                            // ADC in single-ended mode sampling P0.0

   ADC0CF = (EXTCLK << 3);                   // Set SAR clock frequency to ~32kHz

   ADC0CF |= 0x01;                           // Set PGA gain

   // settling time starts at this point, sampling should not start until
   // the appropriate settling time has passed. At this point, we using
   // a 32.768 kHz so each SYSCLK cycle is 30.5 us.

   AD0INT = 0;                               // Clear conversion complete flag
   AD0BUSY = 1;                              // Start a conversion
   while(!AD0INT);                           // Wait until conversion complete

   AD0EN = 0;                                // Disable ADC
   REF0CN &= ~0x02;                          // Turn off bias generator
   ADC_READING = ADC0;                       // Capture ADC Reading

}
```

## Example 2: Waking From Idle Mode on UART Activity (C8051F30x)

```
//-----------------------------------------------------------------------------
// UART_Idle_F30x.c
//-----------------------------------------------------------------------------
// Copyright 2002 Cygnal Integrated Products, Inc.
//
// AUTH: FB
// DATE: 6 NOV 02
//
//
// This example shows how a system can wake from Idle mode upon receiving
// a wakeup signal on the UART RX line.  The system operates on the internal
// oscillator divided by 8 in Normal mode. When in Idle mode, the system uses
// the external oscillator in C-mode as its clock source. This code assumes
// a 33pF capacitor is present between XTAL2 and GND. The capacitor causes
// oscillation between 5kHz and 10kHz when the external oscillator drive
// current (XFCN) is set to its lowest value.
//
// When in Normal mode, the program gets one character from the UART at 2400
// baud, transmits a string containing the character, and goes back into Idle
// mode. The system consumes approximately 4.2 uA in Idle mode and 1.5 mA in
// Normal mode.
//
// The wakeup character must have only one falling edge followed by only one
// rising edge. Since the start bit is a ‘0’ and the stop bit is a ‘1’,
// example wakeup characters are 0x00, 0xFF, and 0xF0 but not 0x0F. Keep in
// mind that characters are sent LSB first.
//
// The text file “FF_H.txt” contains an 0xFF character followed by ‘H’. It
// can be sent over UART to wake up the system from Idle mode.
//
// Target: C8051F30x
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51
//

//-----------------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------------

#include <c8051f300.h>                 // SFR declarations
#include <stdio.h>
#include <math.h>

//-----------------------------------------------------------------------------
// 16-bit SFR Definitions for ‘F30x
//-----------------------------------------------------------------------------

sfr16 DP       = 0x82;                 // data pointer
sfr16 TMR2RL   = 0xca;                 // Timer2 reload value
sfr16 TMR2     = 0xcc;                 // Timer2 counter
sfr16 PCA0CP1  = 0xe9;                 // PCA0 Module 1 Capture/Compare
sfr16 PCA0CP2  = 0xeb;                 // PCA0 Module 2 Capture/Compare
sfr16 PCA0     = 0xf9;                 // PCA0 counter
sfr16 PCA0CP0  = 0xfb;                 // PCA0 Module 0 Capture/Compare

//-----------------------------------------------------------------------------
// Global CONSTANTS
//-----------------------------------------------------------------------------
```

```
#define INTCLK       24500000 / 8      // Internal Oscillator frequency
                                       // in Hz (divide by 8 mode)
#define EXTCLK       5000              // Frequency of external capacitor
                                       // oscillator

#define BAUDRATE     2400              // Baudrate in bits per second

sbit LED = P0^2;                       // LED='1' means ON
sbit SW2 = P0^3;                       // SW2='0' means switch pressed
sbit TX0_PIN = P0^4;                   // UART TX0 pin
sbit RX0_PIN = P0^5;                   // UART RX0 pin

//-----------------------------------------------------------------------------
// Function PROTOTYPES
//-----------------------------------------------------------------------------

void PORT_Init (void);
void UART0_Init (void);
void INT0_ISR (void);


//-----------------------------------------------------------------------------
// Global VARIABLES
//-----------------------------------------------------------------------------

bit UART_ACTIVE = 0;                   // Flag indicating system is in Normal
                                       // mode operating at 3.0625 MHz

//-----------------------------------------------------------------------------
// MAIN Routine
//-----------------------------------------------------------------------------

void main (void) {

   char c;

   // Disable Watchdog timer
   PCA0MD &= ~0x40;                    // WDTE = 0 (clear watchdog timer
                                       // enable)
   PORT_Init ();                       // initialize crossbar and GPIO
   UART0_Init ();                      // initialize UART0

   IT01CF = 0x05;                      // Configure External Interrupt 0 to
                                       // generate an interrupt on the falling
                                       // edge of P0.5 (UART RX signal)

   EA = 1;                             // Enable global interrupts

   while (1) {

      if(UART_ACTIVE){
         c = getchar();                // Get the next character
         printf("\nThe character you entered was: %c", c);

      }

      printf("\n\nTransmit an (0xFF) to wake up the system.\n");
      UART_ACTIVE = 0;                  // Make device ready for Idle mode

      P0MDOUT &= ~0x10;                 // Make TX0_PIN open-drain
      TX0_PIN = 1;                      // Make TX0_PIN high impedance

      REN0 = 0;                         // Disable UART reception

      OSCXCN = 0x50;                    // Start external oscillator in C mode

      RSTSRC = 0x00;                    // Disable missing clock detector

      OSCICN = 0x08;                    // Switch to external oscillator
                                        // and disable internal oscillator

      TR1 = 0;                          // Disable Timer1

      EX0 = 1;                          // Enable External Interrupt 0

      PCON |= 0x01;                     // Go into Idle mode
   }
}

//-----------------------------------------------------------------------------
// Interrupt Service Routines
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// INT0_ISR
//-----------------------------------------------------------------------------
//
// This Interrupt Service Routine is called when a UART character is received
// when the system is in Idle mode.
//
// It enables the UART and sets the system state variable <UART_ACTIVE> to ‘1’.
//
void INT0_ISR (void) interrupt 0 {


   OSCICN = 0x04;                       // Enable Internal oscillator in divide
                                        // by 8 mode and switch to it

   EX0 = 0;                             // Disable External Interrupt0

   TR1 = 1;                             // Enable Timer1
   REN0 = 1;                            // Enable UART reception

   P0MDOUT |= 0x10;                     // enable TX0 as a push-pull output

   UART_ACTIVE = 1;                     // Indicate UART is ready for communication
}


//-----------------------------------------------------------------------------
// Initialization Subroutines
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------------------
```

```
//
// Configure the Crossbar and GPIO ports.
// P0.0 -
// P0.1 -
// P0.2 - LED (push-pull)
// P0.3 - SW2
// P0.4 - UART TX (push-pull)
// P0.5 - UART RX
// P0.6 -
// P0.7 - C2D
//
void PORT_Init (void)
{
   XBR0    = 0x08;                     // skip XTAL2 in the crossbar assignments
   XBR1    = 0x03;                     // UART0 TX and RX pins enabled
   XBR2    = 0x40;                     // Enable crossbar and weak pull-ups
   P0MDOUT |= 0x10;                    // enable TX0 as a push-pull output
   P0MDIN &= ~0x08;                    // Configure XTAL2 as an analog input


}

//-----------------------------------------------------------------------------
// UART0_Init
//-----------------------------------------------------------------------------
//
// Configure the UART0 using Timer1, for <BAUDRATE> and 8-N-1.
//
void UART0_Init (void)
{
   SCON0 = 0x10;                      // SCON0: 8-bit variable bit rate
                                      //        level of STOP bit is ignored
                                      //        RX enabled
                                      //        ninth bits are zeros
                                      //        clear RI0 and TI0 bits
   if (INTCLK/BAUDRATE/2/256 < 1) {
      TH1 = -(INTCLK/BAUDRATE/2);
      CKCON |= 0x10;                  // T1M = 1; SCA1:0 = xx
   } else if (INTCLK/BAUDRATE/2/256 < 4) {
      TH1 = -(INTCLK/BAUDRATE/2/4);
      CKCON &= ~0x13;
      CKCON |=  0x01;                 // T1M = 0; SCA1:0 = 01
   } else if (INTCLK/BAUDRATE/2/256 < 12) {
      TH1 = -(INTCLK/BAUDRATE/2/12);
      CKCON &= ~0x13;                 // T1M = 0; SCA1:0 = 00
   } else {
      TH1 = -(INTCLK/BAUDRATE/2/48);
      CKCON &= ~0x13;
      CKCON |=  0x02;                 // T1M = 0; SCA1:0 = 10
   }

   TL1 = TH1;                         // set Timer1 to reload value
   TMOD &= ~0xf0;                     // TMOD: timer 1 in 8-bit autoreload
   TMOD |=  0x20;
   TR1 = 1;                           // START Timer1
   TI0 = 1;                           // Indicate TX0 ready
}
```

## Example 3: Waking from Stop Mode Using a Comparator (C8051F30x)

```
//-----------------------------------------------------------------------------
// CP0_Stop_F30x.c
//-----------------------------------------------------------------------------
// Copyright 2003 Cygnal Integrated Products, Inc.
//
// AUTH: FB / GV
// DATE: 23 JAN 03
//
// This example shows how a sleep mode can be implemented in a system using
// the Stop mode of the CPU. This application implements a software counter
// that is incremented approximately every second when the device is in
// Normal mode. If the system is powered down when it is in Stop mode, the
// counter resumes counting the next time it enters Normal mode. If the
// system is powered down while it is in Normal mode, the counter will reset
// to zero. Every time the counter is updated, the current value of the
// counter is printed to the UART.
//
// The S2 switch toggles the system back and forth between Stop mode and
// Normal mode. On power up, the system is in Stop mode. The system enters
// Normal mode when the S2 switch on the target board is pressed, causing
// a Comparator 0 reset.
//
// Upon entering Normal mode, External Interrupt 0 (/INT0) is activated
// to sense the S2 switch and Comparator 0 is disabled as a reset source.
// Pressing S2 in Normal mode will cause the INT0_ISR to put the system
// in Stop mode.
//
// This program uses Comparator 0 as a reset source.  When S2 on the target
// board is pressed, the the CP0+ input drops below CP0- (VDD/2).  This causes
// Comparator 0 to issue a system reset.
//
// For this example, it is necessary to make the following connections:
// 1. P0.3_SW  -> P0.0
// 2. P0.2     -> P0.1
// 3. P0.2_LED -> P0.2
//
// P0.1 is used as a reference voltage for the comparator and should be
// approximately halfway between VDD and GND. When the LED is connected
// to P0.2 (high-impedance with weak pull-up), the voltage on P0.2 is
// around 1.7 Volts.
//
// Since this program writes to FLASH, the VDD monitor is enabled.
//
// Target: C8051F30x
// Tool Chain: KEIL C51 6.03 / KEIL EVAL C51
//

//-----------------------------------------------------------------------------
// Include Files
//-----------------------------------------------------------------------------

#include <c8051f300.h>
#include <stdio.h>

//-----------------------------------------------------------------------------
```

```
// 16-bit SFR Definitions for 'F30x
//-----------------------------------------------------------------------------

sfr16 DP       = 0x82;                 // data pointer
sfr16 TMR2RL   = 0xca;                 // Timer2 reload value
sfr16 TMR2     = 0xcc;                 // Timer2 counter
sfr16 PCA0CP1  = 0xe9;                 // PCA0 Module 1 Capture/Compare
sfr16 PCA0CP2  = 0xeb;                 // PCA0 Module 2 Capture/Compare
sfr16 PCA0     = 0xf9;                 // PCA0 counter
sfr16 PCA0CP0  = 0xfb;                 // PCA0 Module 0 Capture/Compare

//-----------------------------------------------------------------------------
// Global Constants
//-----------------------------------------------------------------------------

#define SYSCLK   3062500               // System Clock Frequency in Hz
#define BAUDRATE 9600                  // UART Baud Rate in bps

sbit S2 = P0^0;                        // Switch on Target Board

//-----------------------------------------------------------------------------
// Global Variables
//-----------------------------------------------------------------------------
long COUNT = 0;                        // Software Counter

char code SLEEP _at_ 0x1000;           // Flag that indicates device
                                       // is in a low-power state. The
                                       // flag is TRUE when it contains
                                       // an 0x55 pattern. Any other
                                       // pattern indicates FALSE.

long code COUNT_SAVE _at_ 0x1001;      // Non-volatile storage for
                                       // the current count



//-----------------------------------------------------------------------------
// Function Prototypes
//-----------------------------------------------------------------------------

void SYSCLK_Init (void);
void PORT_Init (void);
void CPT0_Init (void);
void ResetSRC_Init(void);
void EX0_Init(void);
void UART0_Init (void);
void Check_Reset_Source(void);
void wait_ms(int ms);

//-----------------------------------------------------------------------------
// MAIN Routine
//-----------------------------------------------------------------------------

void main (void)
{

   PCA0MD &= ~0x40;                    // disable the watchdog timer
   RSTSRC = 0x02;                      // enable VDD monitor
   EX0_Init();                           // initialize External Interrupt 0
   PORT_Init ();                         // initialize crossbar and GPIO
   SYSCLK_Init();                        // initialize the system clock
   UART0_Init();                         // initialize UART communication

   EA = 1;                               // Enable global interrupts



   Check_Reset_Source();                 // check whether the source of
                                         // of the last reset was due to
                                         // a power-on condition or due to
                                         // a comparator



   while(1){

      // print current count
      printf("Current Count: %ld\n", COUNT);
      COUNT++;

      // wait for one second
      wait_ms(1000);

   }
}

//-----------------------------------------------------------------------------
// Check_Reset_Source
//-----------------------------------------------------------------------------
//
// This routine is called on every device reset.
//
// On each comparator reset, it restores the value of <COUNT> from the
// <COUNT_SAVE> variable stored in FLASH if the device was in a low-power state
// prior to the reset(i.e. the SLEEP flag in FLASH was set to an 0x55 pattern).
// If the <SLEEP> flag was not set then the <COUNT> variable is set to zero and
// the device starts normal mode operation.
//
// On each power-on reset or HW pin reset, the device goes into a low power
// mode waiting for a comparator reset.
//
void Check_Reset_Source(void)
{
   char EA_SAVE;                         // interrupt state preservation
   char xdata * idata ptrSLEEP = &SLEEP;  // FLASH write pointer

   printf("\nDevice Reset -- RESET SOURCE = 0x%02bX\n\n", RSTSRC);

   // check for power-on, HW pin, watchdog timer or missing clock detector reset
   if(RSTSRC & 0x4F){
      CPT0_Init();                       // initialize comparator 0
      ResetSRC_Init();                   // set comparator 0 as a reset source

      printf("Entering Stop Mode\n\n");
      PCON |= 0x02;                      // put device in stop mode
   }
   // check for a comparator reset
   else if( RSTSRC & 0x20){

      while(!S2);                        // wait while switch down
      wait_ms(5);                        // wait until switch stabilizes


      // if the device was in a low-power state ( <SLEEP> flag is set to TRUE),
      // then resume counting, otherwise start counting from zero
      if(SLEEP == 0x55){

         // 1. restore <COUNT>
         COUNT = COUNT_SAVE;

         // 2. Set <SLEEP> flag to FALSE by erasing the FLASH page containing
         //    the variable
         EA_SAVE = EA;                   // preserve interrupt state
         EA = 0;                         // disable interrupts
         PSCTL = 0x01;                   // MOVX writes write FLASH byte
         FLKEY = 0xA5;                   // FLASH lock and key sequence 1
         FLKEY = 0xF1;                   // FLASH lock and key sequence 2
         *ptrSLEEP = 0x00;               // clear SLEEP flag to indicate device
                                         // is no longer in Stop mode
         PSCTL = 0x00;                   // disable FLASH writes/erases
         EA = EA_SAVE;                   // restore interrupt state

         // 3. Enable External Interrupt 0
         EX0 = 1;
      }

      // otherwise start counting at zero
      else{
         // 1. Set <COUNT> to zero
         COUNT = 0;

         // 2. Enable External Interrupt 0
         EX0 = 1;                        // Enable External Interrupt0
      }

   }

   // handle error condition for unrecognized reset source
   else {
      printf(“\n**UNRECOGNIZED RESET SOURCE = 0x%02bX\n”, RSTSRC);
      PCON |= 0x02;                      //  place device in Stop mode
   }


}

//-----------------------------------------------------------------------------
// wait_ms
//-----------------------------------------------------------------------------
//
// This routine inserts a delay of <ms> milliseconds.
//
void wait_ms(int ms)
{
   TMR2CN = 0x00;                        // Configure Timer 2 as a 16-bit
```

```
                                          // timer counting SYSCLKs/12
   TMR2RL = -(SYSCLK/1000/12);            // Timer 2 overflows at 1 kHz
   TMR2 = TMR2RL;

   TR2 = 1;                               // Start Timer 2

   while(ms){
      TF2H = 0;
      while(!TF2H);                       // wait until timer overflows
      ms--;                               // decrement ms
   }

   TR2 = 0;                               // Stop Timer 2
}


//-----------------------------------------------------------------------------
// Interrupt Service Routines
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// INT0_ISR
//-----------------------------------------------------------------------------

void INT0_ISR (void) interrupt 0
{
   // pointer to COUNT
   unsigned char* ptrCOUNT = &COUNT;

   // FLASH write pointer
   char xdata * idata ptrCOUNT_SAVE = &COUNT_SAVE;

   // FLASH write pointer
   char xdata * idata ptrSLEEP = &SLEEP;

   char EA_SAVE = EA;                     // save interrupt status

   printf("Entering Stop Mode\n\n");

   EA = 0;                                // disable interrupts

   PSCTL = 0x03;                          // MOVX writes erase FLASH page
   FLKEY = 0xA5;                          // FLASH lock and key sequence 1
   FLKEY = 0xF1;                          // FLASH lock and key sequence 2
   *ptrCOUNT_SAVE = 0;                    // initiate page erase

   PSCTL = 0x01;                          // MOVX writes write FLASH byte

   // copy <COUNT> to the <COUNT_SAVE> variable in FLASH

   FLKEY = 0xA5;                          // FLASH lock and key sequence 1
   FLKEY = 0xF1;                          // FLASH lock and key sequence 2
   ptrCOUNT_SAVE[0] = ptrCOUNT[0];        // copy first byte

   FLKEY = 0xA5;                          // FLASH lock and key sequence 1
   FLKEY = 0xF1;                          // FLASH lock and key sequence 2
   ptrCOUNT_SAVE[1] = ptrCOUNT[1];        // copy second byte

   FLKEY = 0xA5;                          // FLASH lock and key sequence 1
   FLKEY = 0xF1;                          // FLASH lock and key sequence 2
   ptrCOUNT_SAVE[2] = ptrCOUNT[2];        // copy third byte

   FLKEY = 0xA5;                          // FLASH lock and key sequence 1
   FLKEY = 0xF1;                          // FLASH lock and key sequence 2
   ptrCOUNT_SAVE[3] = ptrCOUNT[3];        // copy fourth byte

   FLKEY = 0xA5;                          // FLASH lock and key sequence 1
   FLKEY = 0xF1;                          // FLASH lock and key sequence 2
   *ptrSLEEP = 0x55;                      // set SLEEP flag to indicate device
                                          // is in Stop mode and <COUNT> has
                                          // been saved in FLASH

   PSCTL = 0x00;                          // disable FLASH writes and erases

   EX0 = 0;                               // disable External Interrupt 0

   EA = EA_SAVE;                          // restore interrupt status

   while(!S2);                            // wait while switch down

   CPT0_Init();                           // initialize comparator 0
   ResetSRC_Init();                       // set comparator 0 as a reset source

   PCON |= 0x02;                          // put the device in Stop mode


}

//-----------------------------------------------------------------------------
// Initialization Subroutines
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------------------------
// This routine initializes the system clock to use the precision internal
// oscillator divided by 8 as its clock source.
//
void SYSCLK_Init (void)
{

   OSCICN = 0x04;                         // SYSCLK is internal osc.
                                          // in divide by 8 mode running
                                          // at 3.0625 MHz
}

//-----------------------------------------------------------------------------
// PORT Initialization
//-----------------------------------------------------------------------------
//
// Configure the Crossbar and GPIO ports
//
// P0.0 - CP0+ input (connected to S2)
// P0.1 - CP0- input
// P0.2 - Comparator Voltage Reference  (connected to P0.1)
// P0.3 - Used as a weak pull-up for P0.0
//
void PORT_Init (void)
```

```
{
   XBR0     = 0x07;                      // skip P0.0 - P0.2 in crossbar


   XBR1     = 0x03;                      // Enable UART0
   XBR2     = 0x40;                      // Enable crossbar and weak pull-ups

   P0MDOUT |= 0x10;                      // TX0 is a push-pull output

}


//-----------------------------------------------------------------------------
// Comparator0 Initialization
//-----------------------------------------------------------------------------
//
// Initialize Comparator 0 to detect when the SW2 switch is pressed.
//
void CPT0_Init(void)
{

   P0MDIN  &= ~0x03;                     // Comparator 0 inputs (P0.0
                                         // and P0.1) are analog inputs.

   CPT0CN = 0x8F;                        // Comparator enabled with maximum
                                         // positive and negative hysteresis

   CPT0MX = 0x00;                        // P0.1 = Inverting Input for
                                         // the comparator
                                         // P0.0 = Non-Inverting Input for the
                                         // comparator

   wait_ms(500);                         // wait for comparator inputs to settle

   CPT0CN &= ~0x30;                      // clear interrupt flags


}

//-----------------------------------------------------------------------------
// Reset Source Initialization
//-----------------------------------------------------------------------------
//
// Configure Comparator 0 as a reset source.
//
void ResetSRC_Init(void)
{
   RSTSRC = 0x22;                        // Comparator 0 is a reset source
                                         // VDD Monitor enabled
}

//-----------------------------------------------------------------------------
// External Interrupt 0 Initialization
//-----------------------------------------------------------------------------
//
// Configure External Interrupt 0 to generate an interrupt on the falling
// edge of P0.0.
//
void EX0_Init(void)
```

```
{
   IT01CF = 0x00;                          // Configure External Interrupt 0 to
                                           // generate an interrupt on the falling
                                           // edge of P0.0 (S2 switch)
}
//-----------------------------------------------------------------------------
// UART0_Init
//-----------------------------------------------------------------------------
//
// Configure the UART0 using Timer1, for <BAUDRATE> and 8-N-1.
//
void UART0_Init (void)
{
   SCON0 = 0x10;                           // SCON0: 8-bit variable bit rate
                                           //        level of STOP bit is ignored
                                           //        RX enabled
                                           //        ninth bits are zeros
                                           //        clear RI0 and TI0 bits
   if (SYSCLK/BAUDRATE/2/256 < 1) {
      TH1 = -(SYSCLK/BAUDRATE/2);
      CKCON |= 0x10;                       // T1M = 1; SCA1:0 = xx
   } else if (SYSCLK/BAUDRATE/2/256 < 4) {
      TH1 = -(SYSCLK/BAUDRATE/2/4);
      CKCON &= ~0x13;
      CKCON |=  0x01;                      // T1M = 0; SCA1:0 = 01
   } else if (SYSCLK/BAUDRATE/2/256 < 12) {
      TH1 = -(SYSCLK/BAUDRATE/2/12);
      CKCON &= ~0x13;                      // T1M = 0; SCA1:0 = 00
   } else {
      TH1 = -(SYSCLK/BAUDRATE/2/48);
      CKCON &= ~0x13;
      CKCON |=  0x02;                      // T1M = 0; SCA1:0 = 10
   }

   TL1 = TH1;                              // set Timer1 to reload value
   TMOD &= ~0xf0;                          // TMOD: timer 1 in 8-bit autoreload
   TMOD |=  0x20;
   TR1 = 1;                                // START Timer1
   TI0 = 1;                                // Indicate TX0 ready

}
```

## Example 3: Waking from Stop Mode Using a Comparator (C8051F31x)

```
//-----------------------------------------------------------------------------
// CP0_Stop_F31x.c
//-----------------------------------------------------------------------------
// Copyright 2003 Cygnal Integrated Products, Inc.
//
// AUTH: FB
// DATE: 23 JAN 03
//
// This example shows how a sleep mode can be implemented in a system using
// the Stop mode of the CPU. This application implements a software counter
// that is incremented approximately every second when the device is in
// Normal mode. If the system is powered down when it is in Stop mode, the
// counter resumes counting the next time it enters Normal mode. If the
// system is powered down while it is in Normal mode, the counter will reset
// to zero. Every time the counter is updated, the current value of the
// counter is printed to the UART.
//
// The S2 switch toggles the system back and forth between Stop mode and
// Normal mode. On power up, the system is in Stop mode. The system enters
// Normal mode when the S2 switch on the target board is pressed, causing
// a Comparator 0 reset.
//
// Upon entering Normal mode, External Interrupt 0 (/INT0) is activated
// to sense the S2 switch and Comparator 0 is disabled as a reset source.
// Pressing S2 in Normal mode will cause the INT0_ISR to put the system
// in Stop mode.
//
// This program uses Comparator 0 as a reset source.  When S2 on the target
// board is pressed, the the CP0+ input drops below CP0- (VDD/2).  This causes
// Comparator 0 to issue a system reset.

// For this example, it is necessary to make the following connections:
// 1. P0.7_SW  -> P0.0
// 2. P0.7_SW  -> P1.0
// 3. P3.3     -> P1.1
// 4. P3.3_LED -> P3.3
//
// P0.1 is used as a reference voltage for the comparator and should be
// approximately halfway between VDD and GND. When the LED is connected
// to P0.2 (high-impedance with weak pull-up), the voltage on P0.2 is
// around 1.7 Volts.
//
// Since this program writes to FLASH, the VDD monitor is enabled.
//
// Target: C8051F31x
// Tool Chain: KEIL C51 6.03 / KEIL EVAL C51
//

//-----------------------------------------------------------------------------
// Include Files
//-----------------------------------------------------------------------------

#include <c8051f310.h>
#include <stdio.h>
```

```
//-----------------------------------------------------------------------------
// 16-bit SFR Definitions for 'F31x
//-----------------------------------------------------------------------------

sfr16 DP       = 0x82;                 // data pointer
sfr16 TMR2RL   = 0xca;                 // Timer2 reload value
sfr16 TMR2     = 0xcc;                 // Timer2 counter
sfr16 TMR3     = 0x94;                 // Timer3 counter
sfr16 TMR3RL   = 0x92;                 // Timer3 reload value
sfr16 PCA0CP0  = 0xfb;                 // PCA0 Module 0 Capture/Compare
sfr16 PCA0CP1  = 0xe9;                 // PCA0 Module 1 Capture/Compare
sfr16 PCA0CP2  = 0xeb;                 // PCA0 Module 2 Capture/Compare
sfr16 PCA0CP3  = 0xed;                 // PCA0 Module 3 Capture/Compare
sfr16 PCA0CP4  = 0xfd;                 // PCA0 Module 4 Capture/Compare
sfr16 PCA0     = 0xf9;                 // PCA0 counter
sfr16 ADC0     = 0xbd;                 // ADC Data Word Register
sfr16 ADC0GT   = 0xc3;                 // ADC0 Greater-Than
sfr16 ADC0LT   = 0xc5;                 // ADC0 Less-Than

//-----------------------------------------------------------------------------
// Global Constants
//-----------------------------------------------------------------------------

#define SYSCLK   3062500               // System Clock Frequency in Hz
#define BAUDRATE 9600                  // UART Baud Rate in bps

sbit S2 = P0^0;                        // Switch on Target Board

//-----------------------------------------------------------------------------
// Global Variables
//-----------------------------------------------------------------------------
long COUNT = 0;                        // Software Counter

char code SLEEP _at_ 0x1000;           // Flag that indicates device
                                       // is in a low-power state. The
                                       // flag is TRUE when it contains
                                       // an 0x55 pattern. Any other
                                       // pattern indicates FALSE.

long code COUNT_SAVE _at_ 0x1001;      // Non-volatile storage for
                                       // the current count

//-----------------------------------------------------------------------------
// Function Prototypes
//-----------------------------------------------------------------------------

void SYSCLK_Init (void);
void VDMON_Init (void);
void PORT_Init (void);
void CPT0_Init (void);
void ResetSRC_Init(void);
void EX0_Init(void);
void UART0_Init (void);
void Check_Reset_Source(void);
void wait_ms(int ms);

//-----------------------------------------------------------------------------
// MAIN Routine
//-----------------------------------------------------------------------------
```

```
void main (void)
{

   PCA0MD &= ~0x40;                       // disable the watchdog timer

   VDMON_Init();                          // initialize VDD monitor


   EX0_Init();                            // initialize External Interrupt 0
   PORT_Init ();                          // initialize crossbar and GPIO
   SYSCLK_Init();                         // initialize the system clock
   UART0_Init();                          // initialize UART communication


   EA = 1;                                // Enable global interrupts



   Check_Reset_Source();                  // check whether the source of
                                          // of the last reset was due to
                                          // a power-on condition or due to
                                          // a comparator



   while(1){

      // print current count
      printf("Current Count: %ld\n", COUNT);
      COUNT++;

      // wait for one second
      wait_ms(1000);

   }
}

//-----------------------------------------------------------------------------
// Check_Reset_Source
//-----------------------------------------------------------------------------
//
// This routine is called on every device reset.
//
// On each comparator reset, it restores the value of <COUNT> from the
// <COUNT_SAVE> variable stored in FLASH if the device was in a low-power state
// prior to the reset(i.e. the SLEEP flag in FLASH was set to an 0x55 pattern).
// If the <SLEEP> flag was not set then the <COUNT> variable is set to zero and
// the device starts normal mode operation.
//
// On each power-on reset or HW pin reset, the device goes into a low power
// mode waiting for a comparator reset.
//
void Check_Reset_Source(void)
{
   char EA_SAVE;                          // interrupt state preservation
   char xdata * idata ptrSLEEP = &SLEEP;  // FLASH write pointer

   printf("\nDevice Reset -- RESET SOURCE = 0x%02bX\n\n", RSTSRC);
   // check for power-on, HW pin, watchdog timer or missing clock detector reset
   if(RSTSRC & 0x4F){
      CPT0_Init();                        // initialize comparator 0
      ResetSRC_Init();                    // set comparator 0 as a reset source

      printf("Entering Stop Mode\n\n");
      PCON |= 0x02;                       // put device in stop mode
   }

   // check for a comparator reset
   else if( RSTSRC & 0x20){

      while(!S2);                         // wait while switch down
      wait_ms(5);                         // wait until switch stabilizes


      // if the device was in a low-power state ( <SLEEP> flag is set to TRUE),
      // then resume counting, otherwise start counting from zero
      if(SLEEP == 0x55){

         // 1. restore <COUNT>
         COUNT = COUNT_SAVE;

         // 2. Set <SLEEP> flag to FALSE by erasing the FLASH page containing
         //    the variable
         EA_SAVE = EA;                    // preserve interrupt state
         EA = 0;                          // disable interrupts
         PSCTL = 0x01;                    // MOVX writes write FLASH byte
         FLKEY = 0xA5;                    // FLASH lock and key sequence 1
         FLKEY = 0xF1;                    // FLASH lock and key sequence 2
         *ptrSLEEP = 0x00;                // clear SLEEP flag to indicate device
                                          // is no longer in Stop mode
         PSCTL = 0x00;                    // disable FLASH writes/erases
         EA = EA_SAVE;                    // restore interrupt state

         // 3. Enable External Interrupt 0
         EX0 = 1;
      }

      // otherwise start counting at zero
      else{
         // 1. Set <COUNT> to zero
         COUNT = 0;

         // 2. Enable External Interrupt 0
         EX0 = 1;                         // Enable External Interrupt0
      }

   }

   // handle error condition for unrecognized reset source
   else {
      printf("\n**UNRECOGNIZED RESET SOURCE = 0x%02bX\n", RSTSRC);
      PCON |= 0x02;                       //  place device in Stop mode
   }


}
```

```
//-----------------------------------------------------------------------------
// wait_ms
//-----------------------------------------------------------------------------
//
// This routine inserts a delay of <ms> milliseconds.
//
void wait_ms(int ms)
{
   TMR2CN = 0x00;                       // Configure Timer 2 as a 16-bit
                                        // timer counting SYSCLKs/12
   TMR2RL = -(SYSCLK/1000/12);          // Timer 2 overflows at 1 kHz
   TMR2 = TMR2RL;

   TR2 = 1;                             // Start Timer 2

   while(ms){
      TF2H = 0;
      while(!TF2H);                     // wait until timer overflows
      ms--;                             // decrement ms
   }

   TR2 = 0;                             // Stop Timer 2
}


//-----------------------------------------------------------------------------
// Interrupt Service Routines
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// INT0_ISR
//-----------------------------------------------------------------------------

void INT0_ISR (void) interrupt 0
{
   // pointer to COUNT
   unsigned char* ptrCOUNT = &COUNT;

   // FLASH write pointer
   char xdata * idata ptrCOUNT_SAVE = &COUNT_SAVE;

   // FLASH write pointer
   char xdata * idata ptrSLEEP = &SLEEP;

   char EA_SAVE = EA;                   // save interrupt status

   printf("Entering Stop Mode\n\n");

   EA = 0;                              // disable interrupts

   PSCTL = 0x03;                        // MOVX writes erase FLASH page
   FLKEY = 0xA5;                        // FLASH lock and key sequence 1
   FLKEY = 0xF1;                        // FLASH lock and key sequence 2
   *ptrCOUNT_SAVE = 0;                  // initiate page erase

   PSCTL = 0x01;                        // MOVX writes write FLASH byte

   // copy <COUNT> to the <COUNT_SAVE> variable in FLASH
   FLKEY = 0xA5;                          // FLASH lock and key sequence 1
   FLKEY = 0xF1;                          // FLASH lock and key sequence 2
   ptrCOUNT_SAVE[0] = ptrCOUNT[0];        // copy first byte

   FLKEY = 0xA5;                          // FLASH lock and key sequence 1
   FLKEY = 0xF1;                          // FLASH lock and key sequence 2
   ptrCOUNT_SAVE[1] = ptrCOUNT[1];        // copy second byte

   FLKEY = 0xA5;                          // FLASH lock and key sequence 1
   FLKEY = 0xF1;                          // FLASH lock and key sequence 2
   ptrCOUNT_SAVE[2] = ptrCOUNT[2];        // copy third byte

   FLKEY = 0xA5;                          // FLASH lock and key sequence 1
   FLKEY = 0xF1;                          // FLASH lock and key sequence 2
   ptrCOUNT_SAVE[3] = ptrCOUNT[3];        // copy fourth byte

   FLKEY = 0xA5;                          // FLASH lock and key sequence 1
   FLKEY = 0xF1;                          // FLASH lock and key sequence 2
   *ptrSLEEP = 0x55;                      // set SLEEP flag to indicate device
                                          // is in Stop mode and <COUNT> has
                                          // been saved in FLASH

   PSCTL = 0x00;                          // disable FLASH writes and erases

   EX0 = 0;                               // Disable External Interrupt 0

   EA = EA_SAVE;                          // restore interrupt status

   while(!S2);                            // wait while switch down

   CPT0_Init();                           // initialize comparator 0

   ResetSRC_Init();                       // set comparator 0 as a reset source


   PCON |= 0x02;                          // put the device in Stop mode


}


//-----------------------------------------------------------------------------
// Initialization Subroutines
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// VDMON_Init
//-----------------------------------------------------------------------------
// This routine initializes the VDD monitor.
//
void VDMON_Init (void)
{
   VDM0CN = 0x80;                         // enable VDD monitor
   while(!(VDM0CN & 0x40));               // wait until power supply is above
                                          // VDD threshold
}
```

```
//-----------------------------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------------------------
// This routine initializes the system clock to use the calibrated internal
// oscillator divided by 8 as its clock source.
//
void SYSCLK_Init (void)
{

   OSCICN = 0x80;                           // SYSCLK is internal osc.
                                            // in divide by 8 mode running
                                            // at 3.0625 MHz
}

//-----------------------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------------------
//
// Configure the Crossbar and GPIO ports
// P0.0 - (connected to S2)
// P1.0 - CP0+ input (connected to S2)
// P1.1 - CP0- input
// P3.3 - Comparator Voltage Reference  (connected to P1.0)
//
void PORT_Init (void)
{
   P0SKIP  = 0x01;                          // skip P0.0 in crossbar
   P1SKIP  = 0x03;                          // skip P1.0, P1.1 in crossbar

   XBR0     = 0x01;                         // Enable UART0
   XBR1     = 0x40;                         // Enable crossbar and weak pull-ups

   P0MDOUT |= 0x10;                         // TX0 is a push-pull output

}


//-----------------------------------------------------------------------------
// Comparator0 Initialization
//-----------------------------------------------------------------------------
//
// Initialize Comparator 0 to detect when the SW2 switch is pressed.
//
void CPT0_Init(void)
{

   P1MDIN  &= ~0x03;                        // Comparator 0 inputs (P1.0
                                            // and P1.1) are analog inputs.

   CPT0CN = 0x8F;                           // Comparator enabled with maximum
                                            // positive and negative hysteresis

   CPT0MX = 0x00;                           // P0.1 = Inverting Input for
                                            // the comparator
                                            // P0.0 = Non-Inverting Input for the
                                            // comparator

   wait_ms(500);                            // wait for comparator inputs to settle
   CPT0CN &= ~0x30;                           // clear interrupt flags


}

//-----------------------------------------------------------------------------
// Reset Source Initialization
//-----------------------------------------------------------------------------
//
// Configure Comparator 0 as a reset source.
//
void ResetSRC_Init(void)
{
   RSTSRC = 0x22;                             // Comparator 0 is a reset source
                                              // VDD Monitor enabled
}

//-----------------------------------------------------------------------------
// External Interrupt 0 Initialization
//-----------------------------------------------------------------------------
//
// Configure External Interrupt 0 to generate an interrupt on the falling
// edge of P0.0.
//
void EX0_Init(void)
{
   IT01CF = 0x00;                             // Configure External Interrupt 0 to
                                              // generate an interrupt on the falling
                                              // edge of P0.0 (S2 switch)
}

//-----------------------------------------------------------------------------
// UART0_Init
//-----------------------------------------------------------------------------
//
// Configure the UART0 using Timer1, for <BAUDRATE> and 8-N-1.
//
void UART0_Init (void)
{
   SCON0 = 0x10;                              // SCON0: 8-bit variable bit rate
                                              //        level of STOP bit is ignored
                                              //        RX enabled
                                              //        ninth bits are zeros
                                              //        clear RI0 and TI0 bits
   if (SYSCLK/BAUDRATE/2/256 < 1) {
      TH1 = -(SYSCLK/BAUDRATE/2);
      CKCON |= 0x08;                          // T1M = 1; SCA1:0 = xx
   } else if (SYSCLK/BAUDRATE/2/256 < 4) {
      TH1 = -(SYSCLK/BAUDRATE/2/4);
      CKCON &= ~0x0B;
      CKCON |=  0x01;                         // T1M = 0; SCA1:0 = 01
   } else if (SYSCLK/BAUDRATE/2/256 < 12) {
      TH1 = -(SYSCLK/BAUDRATE/2/12);
      CKCON &= ~0x0B;                         // T1M = 0; SCA1:0 = 00
   } else {
      TH1 = -(SYSCLK/BAUDRATE/2/48);
      CKCON &= ~0x0B;
      CKCON |=  0x02;                         // T1M = 0; SCA1:0 = 10
   }
   TL1 = TH1;                              // set Timer1 to reload value
   TMOD &= ~0xf0;                          // TMOD: timer 1 in 8-bit autoreload
   TMOD |=  0x20;
   TR1 = 1;                                // START Timer1
   TI0 = 1;                                // Indicate TX0 ready
}
```

## Example 4: 32.768 kHz Watch Crystal Low Power Startup Procedure (C8051F30x)

```
//-----------------------------------------------------------------------------
// Watch_XTAL_F30x.c
//-----------------------------------------------------------------------------
// Copyright 2003 Cygnal Integrated Products, Inc.
//
// AUTH: FB
// DATE: 23 JAN 03
//
// This example shows how to start an external 32.768 kHz watch crystal in low
// power applications. Since a watch crystal can take longer than one second
// to start, the device goes into Idle mode after turning on the external
// oscillator. Timer2, configured to generate an interrupt every 100 ms using
// a timebase derived from the internal oscillator, wakes the device to check
// the XTLVLD flag. Once the watch crystal has started, the internal oscillator
// is disabled and the system uses the crystal as the system clock source.
//
// Target: C8051F30x
//
// Tool chain: KEIL Eval 'c'
//

//-----------------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------------
#include <c8051f300.h>                    // SFR declarations

//-----------------------------------------------------------------------------
// 16-bit SFR Definitions for 'F30x
//-----------------------------------------------------------------------------

sfr16 DP       = 0x82;                    // data pointer
sfr16 TMR2RL   = 0xca;                    // Timer2 reload value
sfr16 TMR2     = 0xcc;                    // Timer2 counter
sfr16 PCA0CP1  = 0xe9;                    // PCA0 Module 1 Capture/Compare
sfr16 PCA0CP2  = 0xeb;                    // PCA0 Module 2 Capture/Compare
sfr16 PCA0     = 0xf9;                    // PCA0 counter
sfr16 PCA0CP0  = 0xfb;                    // PCA0 Module 0 Capture/Compare

//-----------------------------------------------------------------------------
// Global CONSTANTS
//-----------------------------------------------------------------------------

#define INTCLK       24500000 / 8         // Internal Oscillator frequency
                                          // in Hz (divide by 8 mode)
#define EXTCLK       32768                // Frequency for 32.768 kHz External
                                          // crystal oscillator


//-----------------------------------------------------------------------------
// Global VARIABLES
//-----------------------------------------------------------------------------
bit OSC_READY = 0;                        // flag to indicate when external
                                          // oscillator is ready

//-----------------------------------------------------------------------------
```

```
// Function PROTOTYPES
//-----------------------------------------------------------------------------
void SYSCLK_Init (void);
void PORT_Init (void);
void Timer2_Init (int counts);
void Timer2_ISR (void);

//-----------------------------------------------------------------------------
// MAIN Routine
//-----------------------------------------------------------------------------
void main (void) {

   // disable watchdog timer
   PCA0MD &= ~0x40;                       // WDTE = 0 (clear watchdog timer
                                          // enable)

   PORT_Init();                           // initialize the crossbar and GPIO
   SYSCLK_Init();                         // start external oscillator


   // The system should be running from the external oscillator at this point

   while(1){
      PCON |= 0x01;                       // put the device in Idle mode
   }
}



//-----------------------------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------------------------
//
// This routine starts the external 32.768 kHz watch crystal and puts the system
// in Idle mode. Timer 2 interrupts check the status of the XTLVLD bit and
// switches the system clock to the external oscillator when it is ready. The
// system remains in Idle mode until the oscillator starts.
//
void SYSCLK_Init (void)
{

   OSCXCN = 0x61;                         // start external oscillator

   Timer2_Init(INTCLK/12/10);             // configure Timer2 to overflow
                                          // at 10 Hz (every 100 ms)

   EA = 1;                                // enable global interrupts

   while(!OSC_READY){
      PCON |= 1;                          // put device in Idle mode
   }

}


//-----------------------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------------------
//
```

```
// Configure the Crossbar and GPIO ports.
// P0.0 -
// P0.1 -
// P0.2 - XTAL1
// P0.3 - XTAL2
// P0.4 -
// P0.5 -
// P0.6 -
// P0.7 - C2D
//
void PORT_Init (void)
{
   XBR0     = 0x0c;                        // skip crystal pins
   XBR2     = 0x40;                        // enable crossbar and weak pull-ups

   P0MDIN  &= ~0x0c;                       // configure XTAL1 and XTAL2 as analog
                                           // inputs
}

//-----------------------------------------------------------------------------
// Timer2_Init
//-----------------------------------------------------------------------------
//
// Configure Timer2 to auto-reload at interval specified by <counts>
// using the system clock / 12 as its time base.
//
void Timer2_Init (int counts)
{
   TMR2CN = 0x00;                          // Stop Timer0;
                                           // Timer2 timebase is SYSCLK/12

   TMR2RL = -counts;                       // Init reload value
   TMR2 = TMR2RL;                          // Init Timer2
   ET2 = 1;                                // enable Timer2 interrupts
   TR2 = 1;                                // start Timer2
}



//-----------------------------------------------------------------------------
// Timer2_ISR
//-----------------------------------------------------------------------------
//
// This interrupt service routine is called on Timer2 overflows
//
void Timer2_ISR (void) interrupt 5
{
   TF2H = 0;                               // clear Timer2 overflow flag

   if(OSCXCN & 0x80)                       // if crystal osc. has settled
   {
      OSCXCN = 0x60;                       // decrease crystal drive current
      RSTSRC = 0x04;                       // enable missing clock detector
      OSCICN = 0x08;                       // switch to external oscillator
                                           // and disable internal oscillator

      TR2 = 0;                             // stop Timer2

      OSC_READY = 1;                       // indicate that the external osc.
```

```
                                                  // is ready

   }
}
```

## Contact Information

Silicon Laboratories Inc.
4635 Boston Lane
Austin, TX 78735
Tel: 1+(512) 416-8500
Fax: 1+(512) 416-9669
Toll Free: 1+(877) 444-3032

Email: productinfo@silabs.com
Internet: www.silabs.com

SILICON LABORATORIES