Low-Power Systems Evaluation

Diogo Curto

Original date: September 09, 2014
Last Modified: December 05, 2014

Contents

Chapter 1

Low-power systems design techniques

Embedded systems usually have in its core a microcontroller. The "brain" of the application, in order to be considered low power, needs to meet certain requirements. Besides being specifically designed to consume as less power as possible, must provide tools to manage adverse situations with energy constraints, such as voltage drops in battery applications, low peak current, minimum acceptable battery duration and so on.

When defining a system as being low-power it really comes down to the application. Every one has a definition for what it needs to be low-power. In most cases, a low-power application is defined by its efficiency. The system needs to get the most work done in the less time as possible, with the minimum use of CPU while working as slow as possible and consume only the strictly necessary power [1].

With this thoughts in mind, is clear that low-power systems require optimization at all levels, from the microcontroller architecture up through the application layer.

Throughout this chapter will be introduced the main power categories and design techniques for microcontroller based systems and applications to be as efficient as possible.

1.1 Power Categories

There are two real factors involved in the application design: dynamic power consumption when it is running and static power for when it is asleep [2].

$$P_{dyn} = \alpha \times f \times C \times V^2 \qquad (1.1)$$

Dynamic power, as expressed in equation 1.1, is affected mainly by the supply voltage and the charging and discharging of capacitances at the clock frequency. The parameter α is a scaling factor that varies when considering

an entire MCU. This is the power consumed when the CPU is running and processing data. From a systems designer point of view, the changeable parameters, not arbitrarily because they are dependent on each other, during system design are the supply voltage and operating frequency.

In a more global view, dynamic power is not only associated to the CPU itself, but also to the peripherals, both internal and external. This is important since most microcontrollers have the ability to shutdown internal peripherals completely in order to save power.

External peripherals can also be shutdown by using, for instance, I/O pins to power those devices, or even to control a transistor working as switch.

Static power encompasses the power required to maintain proper system operation while code is not actively running and is highly dependable on voltage, temperature, leakage and bias currents for analog circuits. This is considered when the CPU is in standby (idle) and normally waiting for an event to occur. In battery applications this is the state in which the CPU stays longer and therefore consumes the most significant battery power. Most applications need memory to work properly and therefore data retention power must be taken into account when discussing static power.

In terms of embedded systems, the most common practice is to discuss current consumption of each part instead of power consumption due to the fact that normally, the voltage supply is fixed to a predetermined range and the most changing variable is current. Therefore, over this text, device characterization will be expressed in current consumption.

1.2   Design Techniques

When designing a low-power system, the first step is normally to do a power budget. Is necessary to evaluate the needed or allowable power modes and estimate for how long the system will be running to accomplish the determined objectives. It is also important to consider the use of the peripherals: the time they will run in each different power mode and so on. [3]

$$I_{AVG} = \frac{I_{dyn} \times t_{dyn} + I_{st} \times t_{st}}{t_{dyn} + t_{st}} \qquad (1.2)$$

After this, it is necessary to calculate the current consumption of each mode and the time spent on each mode to get an overall estimation of average current consumption for the application as shown in figure 1.1 and calculated in equation 1.2.

This allows us to get an overall view in what is needed from the power-supply and the expected battery life. It also allows to get a good view of what parts of the system need more focus in order to minimize current consumption.
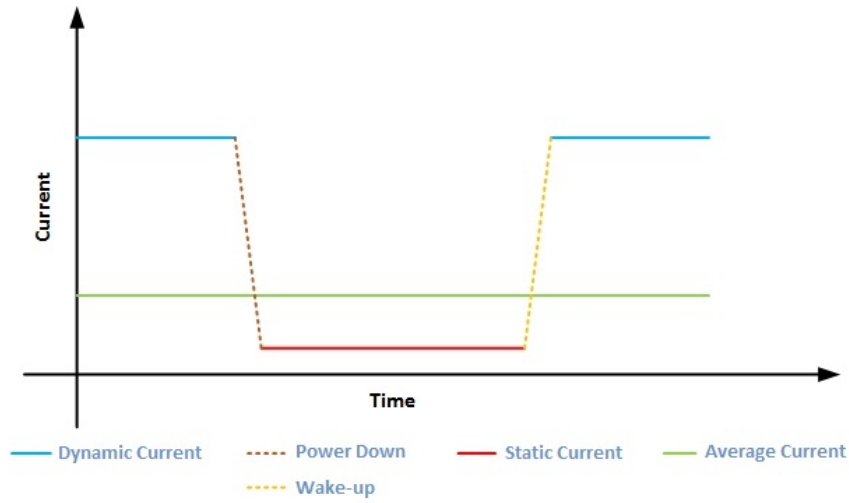
Figure 1.1: Generic system current consumption

With these thoughts in mind, over the next sections, will be presented microcontroller features capable of managing the current consumption of specific modules and tasks.

1.2.1 Supply Voltage Range

When considering a reduction in the supply voltage in order to reduce the current consumption, there is a limitation to it that cannot be overlooked. The minimum supply voltage is limited by the operating frequency, that is, to reduce the supply voltage is also necessary to change the frequency. As an example in figure 1.2 is shown the frequency limitations depending on the supply voltage.

1.2.2 Clock Speed

As discussed in the section 1.1, the clock speed is an important factor to reduce current consumption. When considering an application that runs for a specific time and then goes back to sleep, is important to consider the trade-off between speed and current consumption, that is, evaluate what consumes more current: execute at really low speeds or run fast and then go to sleep for a longer period.

In order to explain the statement above, an example is provided in figure 1.3 for active current consumption. This figure was obtained by taking a generic application with a certain amount of work that needs to run every ten seconds for five different clock speeds:
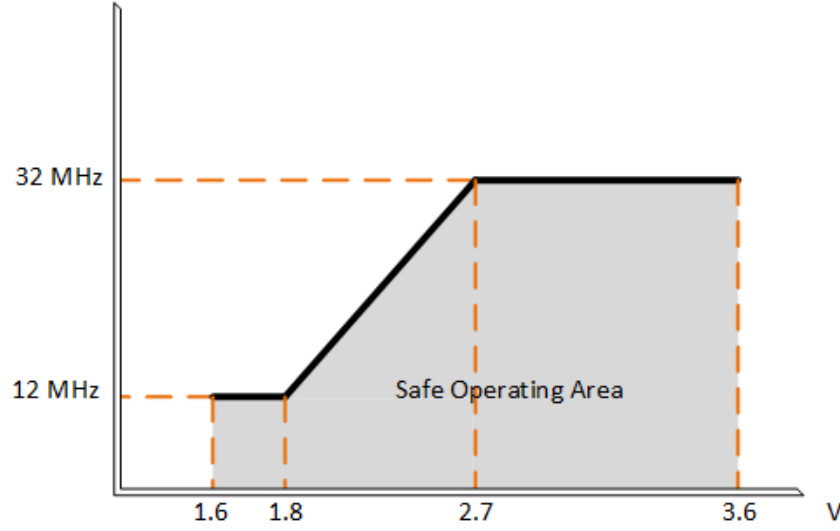
Figure 1.2: Frequency vs Supply voltage. src: Figure 32-22 [4]

$$
\begin{cases}
32 \text{ MHz} & - & \text{IAVG32} \\
8 \text{ MHz} & - & \text{IAVG8} \\
2 \text{ MHz} & - & \text{IAVG2} \\
1 \text{ MHz} & - & \text{IAVG1} \\
31 \text{ kHz} & - & \text{IAVG31k}
\end{cases}
$$

As can be seen, executing at 31 KHz is always worst than higher speeds. This occurs due to the very low execution speed the microcontroller is almost always working instead of in a Deep-Sleep state.

A better overview of this case is shown in figure 1.4. Here, the 31 KHz average current was removed. As the work load starts to increase, the most efficient frequencies are the highest. This indicates that executing at low speeds is only better if the work load is small.

These figures were obtained based on equation 1.2. More explicitly:

$$
\begin{cases}
i_{dyn}(f) & = & i_{DD}(f) + i_{\text{WU}}(f) + i_{\text{PD}}(f) \\
t_{dyn}(f) & = & t_{\text{n inst}}(f) + t_{\text{WU}} + t_{\text{PD}}(f) \\
I_{st} & = & I_{\text{DS}} \\
t_{st} & = & t_{\text{DS}} \\
t_{dyn}(f) + t_{st} & = & 10 \text{ s}
\end{cases}
$$

Another important aspect of frequency reduction to consider is the fact that in order to maintain the logic level, a certain amount of current must be used. This is necessary to maintain the charge level of the intrinsic capacitances of the microcontroller (recall equation 1.1). For lower frequencies, this
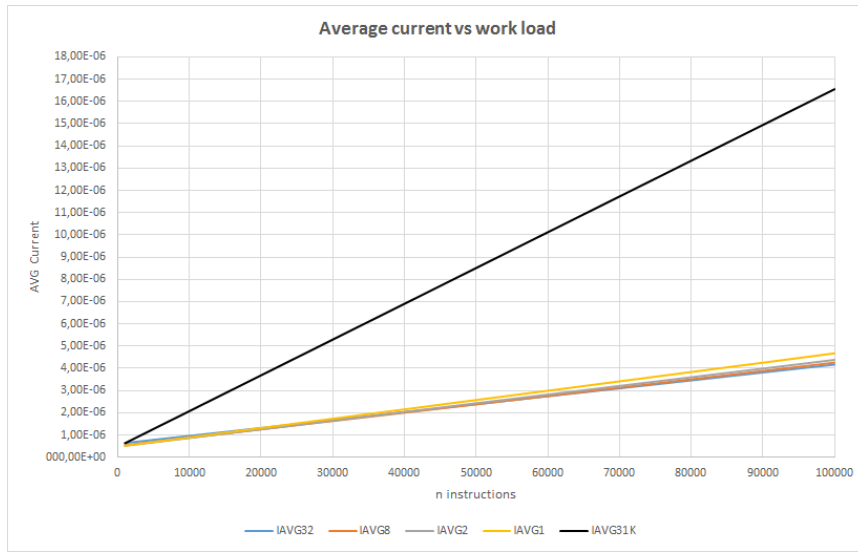
Figure 1.3: Clock speed average active current vs work load. src: CH 32 [5] [6]
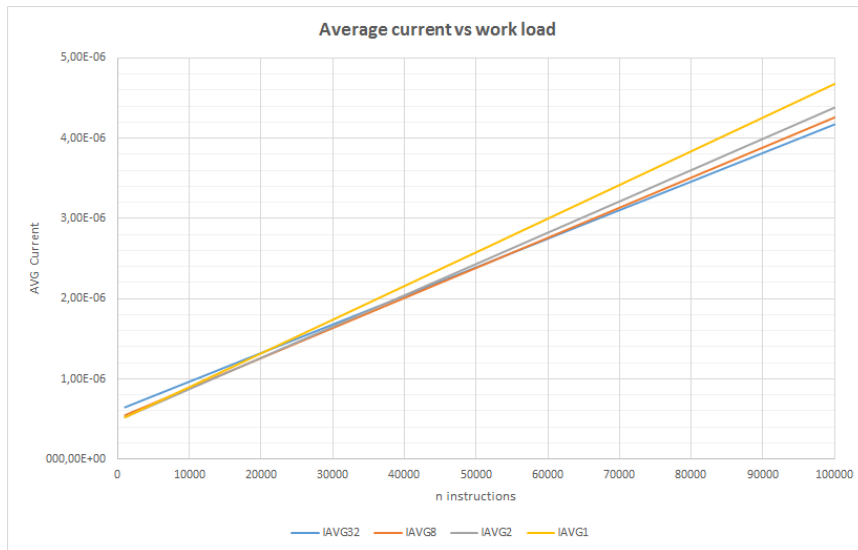

Figure 1.4: Clock speed average active current vs work load. src: CH 32 [5] [6]

logic level must be held for more time, increasing the current consumption. When using higher frequencies, the voltage (logic) level decay is negligible making these more efficient than lower frequencies. An example is shown in figure 1.5.

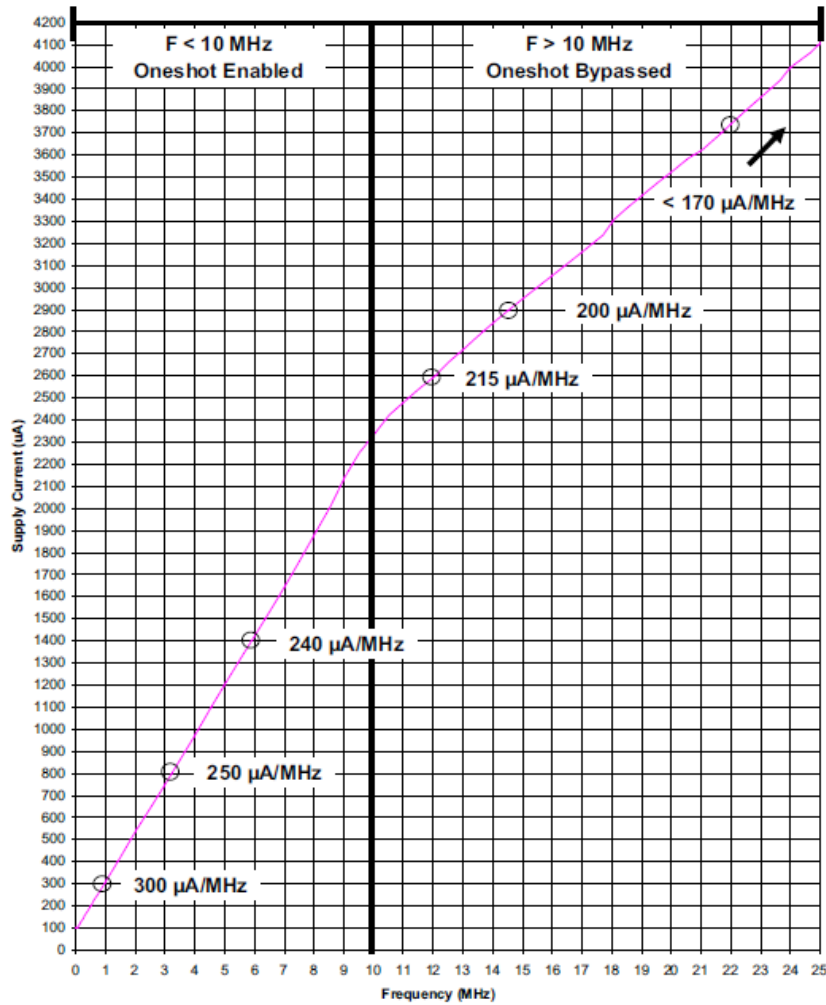When considering running the application at higher speeds there is an-

Figure 1.5: Active Mode Current Efficiency. src: Figure 4.1 [8]

other factor that rises as shown in section 1.2.1: Most microcontrollers need higher supply voltages in order to run faster. Therefore, to avoid errors in code execution, another feature must be used: A low-voltage detector, that enables, for instance, to reduce clock speed in runtime and therefore extend battery life. There is another feature useful both at higher and lower speeds called the Brown-out Reset. This allows to protect the application from errors as batteries die, or when high peaks of dynamic current bring the supply voltage to an unsupported level, resetting the microcontroller in both cases.

In some applications, the relevant mode is not when the application is running (active mode). This happens when the system spends a large portion of his time in a Sleep/Low-Power mode making this the most relevant

mode for current consumption. Meaning that the most important mode to consider really depends on the duty cycle between the various sleep and active modes. [7]

1.2.3 Wake-up time

Wake-up time refers to the transition from low-power to active modes. For systems with short active windows the wake-up process consumes a similar amount of current as normal operation. The more functions that are turned off, the less current the chip will consume, but the longer it will take it to wake up.

Another important factor is the oscillator start-up delay, which depends on the clock source as well as other frequency dividers/multipliers in the path. Wake-up time will usually be the limiting factor that determines which low-current mode can be used at a given time in the application. In order to minimize this issue, some additional peripherals can be used. For instance, an internal voltage regulator that allows to maintain the internal capacitances charged reduces wake-up time in 200 μs.

NOTE: The average wake-up current does **not** include the energy required in active mode; for example, for an interrupt service routine or to reconfigure the device.
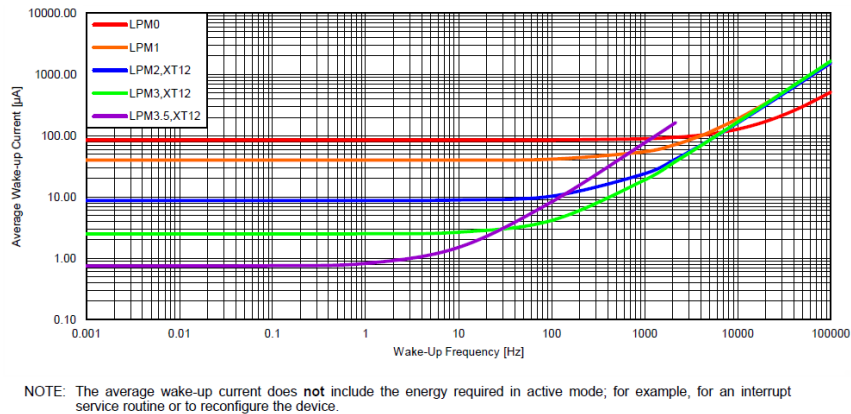
Figure 1.6: Average LPM Currents vs Wake-up Frequency at 25 °C. src: Figure 5-3 [9]

This figure comes in the sequence of figures 1.3 and 1.4. [WAITING FOR TEXAS INST. RESPONSE...]

1.2.4 Instruction Set Architecture

When evaluating current consumption of a microcontroller, one of the key aspects to take into account is the instruction set. That is, the amount of actual work done per unit of energy consumed. The clocking scheme - ratio between the input system clock and the instruction clock frequencies, cycles

per instruction and available instructions have major impact on current consumption.

The first thought that comes to mind is CISC vs RISC[1]. The CISC architecture goal is to complete a task in as few lines of assembly as possible. This is achieved by building processing hardware that is capable of understanding and executing a series of operations.

On the other hand, RISC architecture is focused in having only instructions that can be executed within one clock cycle. This is achieved by separating the complex instructions with a sequence of simpler instructions.

For instance the ALU (Arithmetic Logic Unit), is one of the most important components of a CPU. The level of complexity has a major impact on current consumption. When an ALU is highly complex with very built-in capabilities, for instance, multiplier/divider units, floating point, registry calculation and so on, this not always is good in therms of current consumption.

If the application requires very little of these features, a good portion of current is wasted. But if the application requires all these features, and they are unsupported by the ALU, they must be decomposed into simpler instructions, which may require more time to process.

This means that the microcontroller must be adjusted for the application in order to maximize current efficiency.

In order to materialize the above statements, for instance, in some devices, Texas Instruments [10], opted to reduce CPU (ALU) complexity and incorporated an Hardware Multiplier (32x32 bits), taking 1 CPU Cycle[2] to process data, as a peripheral, with no direct support for division. On the contrary, Microchip Technology [5], added hardware support for both multiplication (16x16 bits, 1 CPU cycle) and division (32/16 bits, 18 CPU cycles).

With no direct comparison intended beyond the point of giving two examples, these architectural choices clearly show that these two different devices are useful in different applications.

1.2.5 Data Retention

Most microcontrollers provide two main types of sleep modes. The first is a light-sleep mode, in which the MCU core is stopped, peripherals are disabled, and clock sources are turned off. However, the MCU stays powered up, preserving the contents of registers and SRAM making the wake-up process faster.

The second, is a deep-sleep mode, in which the entire MCU is powered down and SRAM contents are lost. In order to avoid the loss of data, is necessary to save the needed data to Flash or EEPROM memories. When

---

[1]Complex Instruction Set Computers and Reduced Instruction Set Computers
[2]Using Direct Memory Access

the CPU wakes-up the data needs to be restored in order to resume operation. Unfortunately, due to high write/erase time and current spent on this process, the energy used is substantial.

In order to mitigate some of these problems, another type of memory was introduced in microcontrollers: The Ferroelectric RAM (FRAM) with similar behavior as SRAM except for one important characteristic: it is non-volatile, which means an extra feature to reduce current consumption. FRAM can be used as Flash memory to store the application code and as backup to save data before power down. Although consumes more current when active, the zero consumption while shutdown allows major power savings.[11]

Table 1.1 shows a short comparison between the previously referred memories.

Table 1.1: Comparison between different types of memories.[12] [13]

	SRAM	FRAM	FLASH	EEPROM
Type	Volatile	Non-Volatile	Non-Volatile	Non-Volatile
Write Cycle [time]	$<< 125$ ns [3]	125 ns	10 μs	5 ms
Endurance [cycles]	unlimited	10^{13}	10^5	10^6
Average active current [/MHz]	< 60 μA	110 μA	260 μA	50 mA
Flexible Code and data partitioning	no	yes	no	no

1.2.5.1 Code Execution: Flash vs SRAM

Powering and reading the Flash memory of an MCU is one of the most significant contributions to current consumption. Some devices allow code execution from SRAM. This comes, as expected, with some trade-offs: The first step to run from SRAM is to copy the Flash contents to SRAM, which consumes significant amounts of time and current. Another disadvantage is the limited size of RAM, which is significantly smaller than Flash. This means that, in order to be worthwhile, the functions running from RAM must be limited in size or should be computationally-intensive functions.

Another important drawback is the use of a single memory. Since most microcontrollers use two separate buses to read from Flash and SRAM, the limitation of using only one could imply the stalling of the processor in instructions that write back to SRAM.

Can't be forgotten that in case of applications that switch from active mode to deep-sleep periodically need to restore the RAM contents after each power-on increasing the waste of power.

---

[3]Write cycle up to 200 MHz

1.2.5.2 Code Execution: FRAM vs SRAM

As shown in table 1.1, FRAM can work as unified memory with flexible code and data partitioning, meaning that runtime variables can be stored in the same 'place' as program data with the drawback explained earlier. However with an important advantage: the data saved is kept when the device is shut down. There's still the possibility to use SRAM to complement the FRAM adding the advantage of independent buses.

1.2.6 Proper use of Peripherals

The use of peripheral features in an embedded system is very usual, therefore, maximizing its efficiency is also a good practice to reduce current consumption.

1.2.6.1 Serial Interfaces: UART, SPI and I^{2}C

The serial interface is one of the most used peripherals in an embedded system. Since the protocols used may vary, is necessary to evaluate the best in terms of energy efficiency. Figure 1.7 shows the comparison between three serial interfaces.

| Parameters | UART | | SPI | | | | I^{2}C | | | |
| | | | RX selected | | RX unselected | | RX selected | | RX unselected | |
	1 byte	9 bytes	1 byte	9 bytes	1 byte	9 bytes	1 byte	9 bytes	1 byte	9 bytes
Interface implementation in hardware										
Power consumption with inactive interface, mW	6.06	6.06	4.66	4.66	4.66	4.66	6.13	6.13	6.13	6.13
Communication:										
Required time, ms	0.68	5.78	0.52	4.6	0.52	4.6	1.26	5.91	1.26	5.91
Required energy, μJ	7.31	53.31	2.5	23.53	2.42	22.52	8.11	31.71	8.01	30.23
Normalized energy, μJ	7.27	53.01	2.48	23.33	2.4	22.32	8.19	31.99	8.08	30.5
Interface implementation in software										
Power consumption with inactive interface, mW	11.04	11.04	11.17	11.17	11.17	11.17	11.23	11.23	11.23	11.23
Communication:										
Required time, ms	0.64	5.72	0.51	4.62	0.51	4.61	1.23	5.85	1.23	5.86
Required energy, μJ	9.00	80.06	7.26	65.25	7.26	65.25	17.45	82.84	17.44	83.6
Normalized energy, μJ	9.01	80.16	7.28	65.42	7.28	65.42	17.45	82.84	17.44	83.6

Figure 1.7: Energy consumption for evaluated serial interfaces. src: Table III [14]

The values present in figure 1.7 are normalized according to equation 1.3.

$$E_{norm} = (\frac{15.8 - DataRate_{exp}}{15.8} + 1)E_{exp} \qquad (1.3)$$

Where $DataRate_{exp}$ stands for the experimentally defined data rate in kbits/s and E_{exp} for the experimental measured energy consumption for transferring one or nine data bytes.

From figure 1.7 can be seen that the most energy-efficient interface is SPI. This is mainly due to the fact that SPI protocol does not have any overhead in data transmission.

1.2.6.2 Analog to Digital Converter

An analog to digital converter consists essentially of a measured voltage across a capacitor with reference voltage circuits, signal amplifiers and buffers, etc. When sampling and converting all this analog circuits are working and therefore consuming current. When the main objective is to save power, the best option is to run the ADC module at a higher speed than necessary for the application, disabling it between samples and reducing the sampling time to the minimum necessary to maintain measurement accuracy. As an addition, DMA can be used to save data into RAM, in order to maintain the CPU in a sleep mode, maximizing the module's efficiency.

In this case some considerations must be taken into account: If the module is being turned off and on, there is a minimum delay before taking a sample. When the source impedance of the analog signal is high the current drawn from the source by leakage can affect accuracy. If the input signal does not change too quickly an extra capacitor must be added to charge the internal holding capacitor, increasing accuracy. At last, putting the microcontroller in a sleep mode also improves accuracy, since the digital noise from the CPU and other peripherals is minimized.

1.2.6.3 DMA and FIFO Buffers

The Direct Memory Access controller is mainly used to reduce the CPU overhead, since it allows to perform data transfer tasks without CPU intervention. Since it can work even with the CPU in a sleep mode, it is a great tool to reduce power consumption.

Another important feature is the used of FIFO buffers. This buffers store received data until they are full. Once this happens, an interrupt is fed to the CPU in order to process the data and empty the FIFO. This allows a great power saving because the CPU does not need to handle individually every single transfer.

1.2.6.4 Brown-out Reset

As explained in section 1.2.2 an important peripheral feature is the Brown-out Reset (BOR). Since it protects the application from miss-execution due to voltage drop, it also consumes a significant amount of power.

The best way to deal with this issue, without letting the application unprotected from voltage drops, is to have a dynamic BOR. This is achieved by: first, allowing the microcontroller to disable this peripheral in sleep modes, since it's main objective is protect from miss-execution, in sleep modes it is no longer needed when the CPU is not running; second, by using a low-power BOR that ensures that the microcontroller will always have a power-on reset to place it in a known state.

The first approach is not the best, although the CPU is not working does not mean that other peripherals can't be affected by voltage drops. It is always best to have some sort of protection to avoid errors.

1.2.7 Hardware Design [3]

When designing low-power hardware to interface with the microcontroller there are some basic considerations to take into account that allow major power savings:

- Minimize operating duty-cycles.
  This applies to both internal and external hardware. By powering the hardware only when necessary and turn it off when is not needed.

- Minimize leakage currents.
  As explained in section 1.2.2, executing at higher speeds can be more efficient.

- Maximize impedance in current paths.
  Maximizing the impedance in current paths by using only the strictly necessary current.

- Minimize impedance in high-speed switching paths.
  Since there are always capacitances that need to be charged and discharged in switching circuits, reducing the path impedance allows for faster charge and discharge and therefore a greater working speed.

1.2.7.1 Unused Port Pins

An unused I/O pin should not be left unconnected. One of the best practices is to connect it to ground or the supply voltage through a resistor of high value. This prevents the internal transistors to be biased in the linear region due to floating signals, wasting power.

Bibliography

[1] W. Wong, "Low Power Microcontroller-based Design Techniques," 2009. [Online]. Available: http://electronicdesign.com/embedded/low-power-microcontroller-based-design-techniques

[2] A. Sedra and K. C. Smith, *Microelectronic Circuits*, 6th ed. Oxford University Press Inc, 2010.

[3] B. Ivey and Microchip Technology Inc, "AN1416, Low-Power Design Guide," 2011.

[4] Atmel Corporation, "8/16-bit Atmel XMEGA D4 Microcontroller," 2014.

[5] Microchip Technology Inc, "PIC24FJ128GA204 FAMILY Datasheet," 2014.

[6] ——, "Section 39. Power-Saving Features with Deep Sleep," 2009.

[7] A. M. Holberg and A. Saetre, "Innovative Techniques for Extremely Low Power Consumption with 8-bit Microcontrollers," 2006.

[8] Silicon Labs, "C8051F93x-C8051F92x," 2013.

[9] Texas Instruments Inc, "MSP430FR698x, MSP430FR598x Mixed-Signal Microcontrollers," 2014.

[10] ——, "MSP430x09x Family User ' s Guide," 2010.

[11] P. Thanigai and Texas Instruments Inc, "FRAMs as alternatives to flash memory in embedded designs." [Online]. Available: http://www.embedded.com/design/mcus-processors-and-socs/4390688/2/FRAMs-as-alternatives-to-flash-memory-in-embedded-designs

[12] FUJITSU SEMICONDUCTOR LIMITED, "FUJITSU Semiconductor FRAM," 2014.

[13] G. Allen, C. Stanfield, and G. Zheng, "Ferroelectric Random Access Memory," University of Michigan, Tech. Rep., 2012.

[14] K. Mikhaylov and J. Tervonen, "Evaluation of Power Efficiency for Digital Serial Interfaces of Microcontrollers," *2012 5th International Conference on New Technologies, Mobility and Security (NTMS)*, pp. 1–5, May 2012. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6208716