



## SIMPLYING MULTI-CHEMISTRY BATTERY CHARGERS

*By Archana Yarlagadda, Applications Engineer Senior, Cypress Semiconductor Corp.*

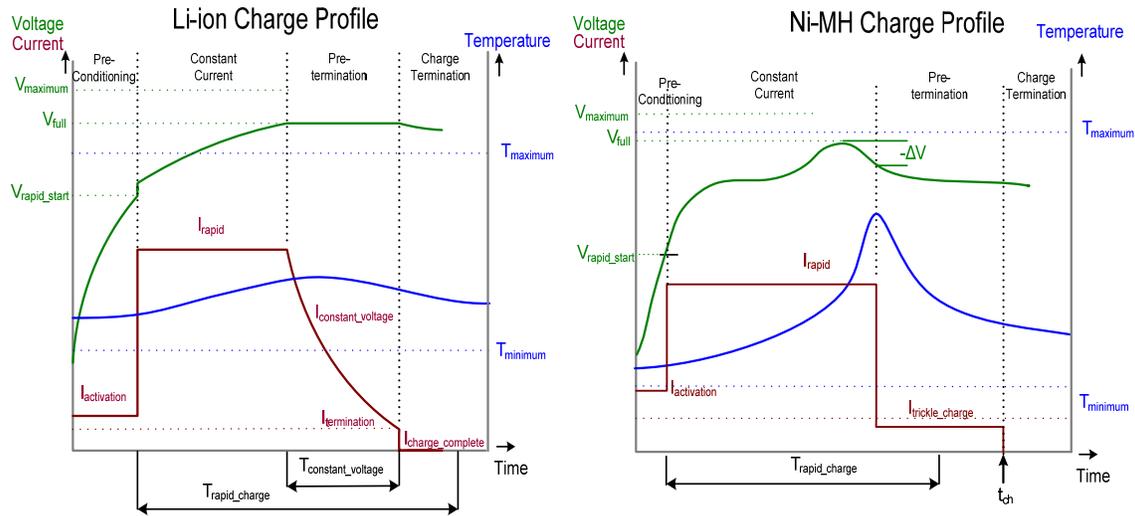
Portable electronic devices, whether personal electronics, remote scientific instrumentation, or simple garage flashlights, all have one thing in common: batteries. These can be NiCd, NiMH, Li-Ion, or any other rechargeable battery chemistries. This article discusses a flexible battery charging system that can be applied to a range of voltages, battery chemistries, and battery charge profiles.

When charging multi-chemistry batteries with different cell capacities, the battery voltage can be higher or lower than the supply voltage at various stages of the charging. Thus the supply voltage needs to be either boosted or attenuated to match the battery voltage. For example, a supply voltage of 3.3 V needs to be attenuated down when a single cell NiMH battery (typically 1.25 V) is being charged. When a single cell Li-ion battery (4.1 V) is used, the input voltage needs to be amplified. To address such cases, the primary charge path is chosen to be Single Ended Primary Inductor Converter (SEPIC) [1]. This topology of switch-mode DC-DC conversion has the ability to both buck and boost a wide range of voltages to provide supply voltage flexibility.

Two different rechargeable battery chemistries – Nickel-Metal Hydrate (NiMH) and Lithium-ion (Li-Ion) batteries – will be used as examples in this article. These two chemistries require different charge profiles but can both be readily serviced using the same flexible charging topology. The flexibility and simplicity in switching from one type of battery chemistry to another is implementation in software on a microcontroller. By designing the charging subsystem in a modular manner and encapsulating functions into various components, the same application can be implemented using different microcontrollers, depending upon system requirements. The use of components simplifies design, where the input and/or outputs are hardware and/or software [2]. This approach allows developers to add battery charging as an additional feature to another main application, like motor control, accurate medical measurements, etc.

A battery charger has to determine the state of the battery (i.e. voltage, current and temperature) and control the charging current. The hardware to determine the state of the battery is common to the batteries. The battery voltage can be higher or lower than the input range of the microcontroller; thus, the voltage is usually measured by using a resistor divider circuit to attenuate the voltage. The current can be measured on the high-side (the current going into the battery), low-side (the current coming out of the battery) or, in case of the SEPIC convertor, by using a resistor in the secondary side of the inductor. The batteries usually have an embedded thermistor that provides an accurate state of the battery temperature. This is sometimes omitted in some commercial batteries to reduce cost. In such cases, an external thermistor placed in contact with the battery can be used.

Based on these measured parameters, the charge current into the battery is determined and is controlled by the microcontroller. The main difference between different battery chemistries, from the battery charger perspective, is the charge profile. The charge profile of Li-ion and NiMH are given in Figure 1 [3].

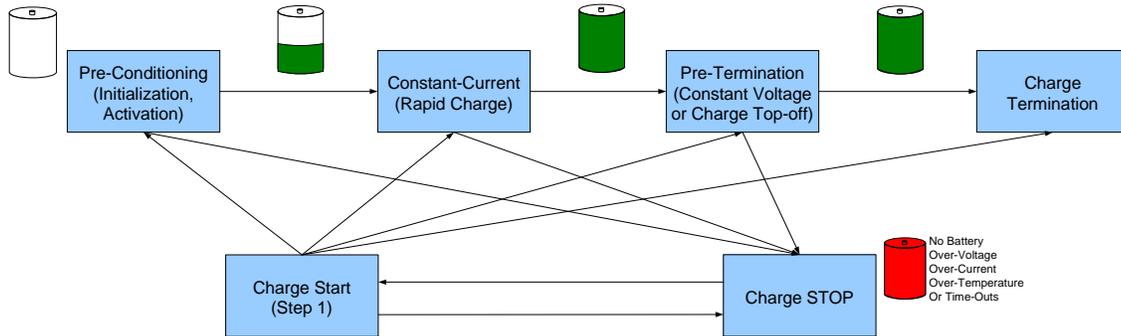


**Figure 1: Battery charging profile for Li-ion and NiMH batteries**

In the profile shown in Figure 1, the current is controlled by the microcontroller and the voltage and temperature are changes happening in the battery. The Li-ion battery uses a constant-current constant-voltage charge profile. Consider the nominal capacity of a battery to be denoted by “CA”. At startup, if the battery voltage is lower than the constant current threshold ( $V_{\text{rapid\_start}}$ ), the battery charger supplies a small amount of current (around 0.1 CA). This is the Pre-conditioning stage where the voltage in the battery gradually increases with this small charge current. When the voltage reaches the rapid charge threshold, the charge current is increased by the microcontroller to around 1 CA. This is the constant current stage that is maintained until the battery voltage reaches the specified voltage ( $V_{\text{full}}$ ). The battery charger then enters the constant-voltage stage, where the charge current is decreased while the battery voltage is maintained at  $V_{\text{full}}$ . When the current is decreased until the termination current, while maintaining the battery voltage, the battery charging is terminated. The current in the battery changes by a few °C during the whole charging process. If any of the battery conditions – voltage, current, or temperature – are outside the specified range for the corresponding battery charger stage, the battery charger shuts down the charging for protection.

The first two charger stages of an NiMH battery are similar to Li-Ion: Activation (with 0.2 CA) and Constant-Current (1 CA). The end of the constant-current stage in NiMH batteries is detected by a drop in the battery voltage (and drop in temperature) while the current is constant. After this drop in voltage, the NiMH charger profile enters a charge-top off stage where the current is reduced to a trickle charge level (around 0.05 CA). In this stage, a small amount of charge current is provided for a constant amount of time before charge termination.

Based on the charging requirements mentioned above, battery charging can be simplified to different levels using a state machine with predefined voltage, current, temperature, and time-out values. The state of the battery and the amount of current that needs to be provided for battery charging are controlled in the microcontroller state machine. A simplified state machine for charging both types of batteries is shown in Figure 2. This block diagram shows the different stages of charging.

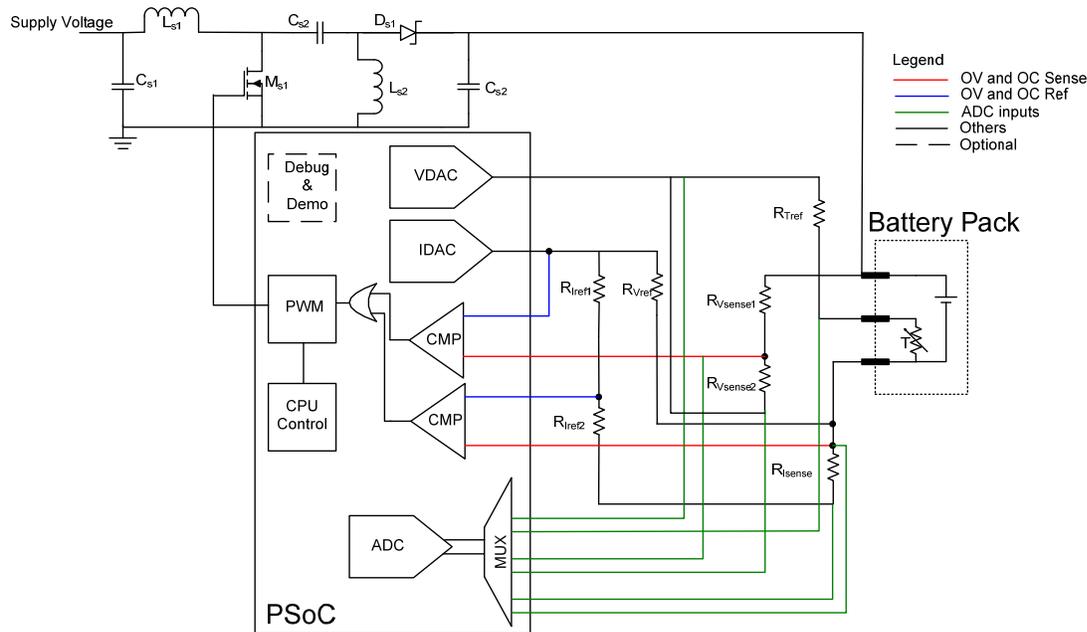


**Figure 2: Block diagram of different stages during charging**

Based on the battery chemistry chosen, the microcontroller goes through the state machine of that particular battery and controls the charge current. The profile used for charging a battery can be pre-programming, pre-startup, or automatic decision. For the first two methods, the type of the battery is taken as an input from the user. In pre-programming, the type of battery charging required is chosen in the component software and the microcontroller is programmed with the required profile. This type of decision is used in applications where battery charging is added as an additional feature to an existing product. In such applications, the type of the battery is known, or different types of batteries are used to make different versions of the same product.

In pre-startup, the microcontroller has both the battery profiles and the decision of the profile to be implemented is done through a conditional check. This check can be something as simple as a switch position that is checked during startup. The automatic decision is done by the microcontroller after startup and it chooses the battery charger profile by detecting the type of battery. For example, a typical voltage range for a single cell NiMH is between 0.9 V to 1.25 V, whereas a single cell Li-ion voltage ranges from 2.7 V to 4.2 V. Similarly, the temperature ranges also differ between the two, and these values can be saved and compared upon startup. The automatic check approach is limited to very specific conditions. In general, the pre-programming and pre-run time decisions are used for the majority of applications. In this article, the pre-programming is highlighted to focus on applications in which battery charging is an add-on feature.

As mentioned earlier, the hardware for sensing and controlling the battery charger is the same for the two types of chemistries. The schematic for the external hardware required for battery charging is shown in Figure 3.



**Figure 3: Simplified Schematic of Battery Charger**

To determine the battery state voltage, current and temperature measurements are performed by multiplexing these inputs to an ADC in the microcontroller. Based on these values, the state is decided in the firmware and the charge current is controlled by changing the duty cycle of the PWM. The PWM output is connected to the gate of the MOSFET in the SEPIC convertor that controls the current flowing into the battery. These steps involve the CPU and thus have some latency. Different batteries, specifically Li-ion, are very sensitive to overcharging and can become extremely unstable at higher voltages. To add additional protection for overvoltage and overcurrent conditions, hardware protection circuits are added through comparators. These comparators will shut-off charging until reset by the user or a safe condition is reached.

Based on the parameter values measured and the type of battery chemistry, the CPU decides the state of the battery and changes the duty cycle of the PWM accordingly. Traditionally, the conditions required by the CPU to detect the profile were defined and as constants in the code and were changed manually. Consider the following Pseudo-code.

```
#define LIION 0
#define NIMH 1
#define BATTERY_PROFILE 0

void main()
{
  Measure_Battery_Parameters(); //Use ADC to measure V, I and T
  Get_Battery_State();
  Control_PWM_DutyCycle();
}

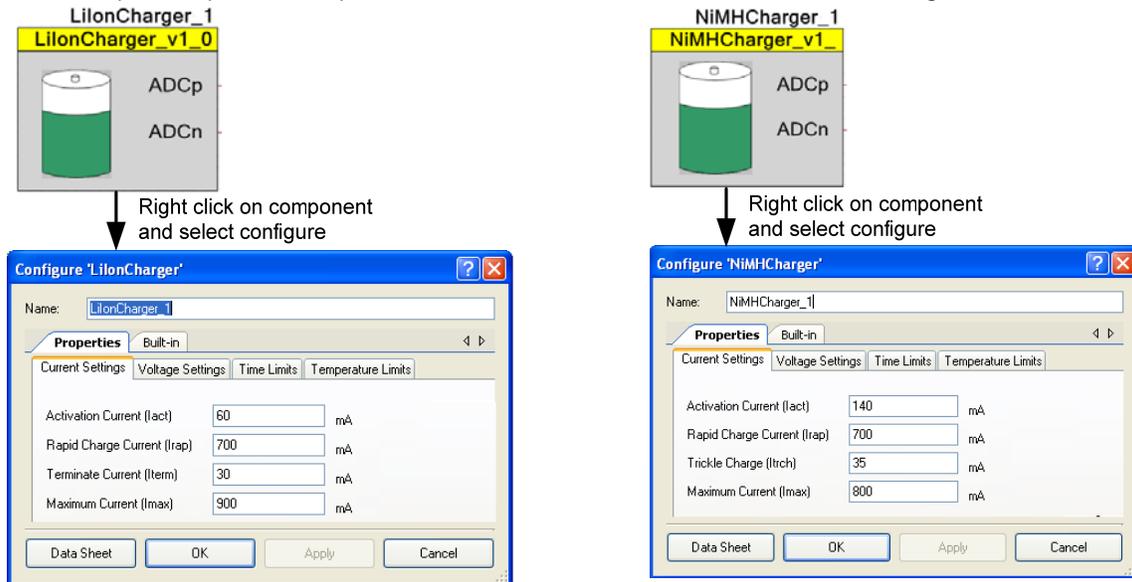
Get_Battery_State()
{
  if(BATTERY_PROFILE == LIION)
  {
    //set battery state for Li-Ion
  }
  else if(BATTERY_PROFILE == NIMH)
  {
    //set battery state for NiMH
  }
}
```

```

}
else
{
    //battery type error
}
}

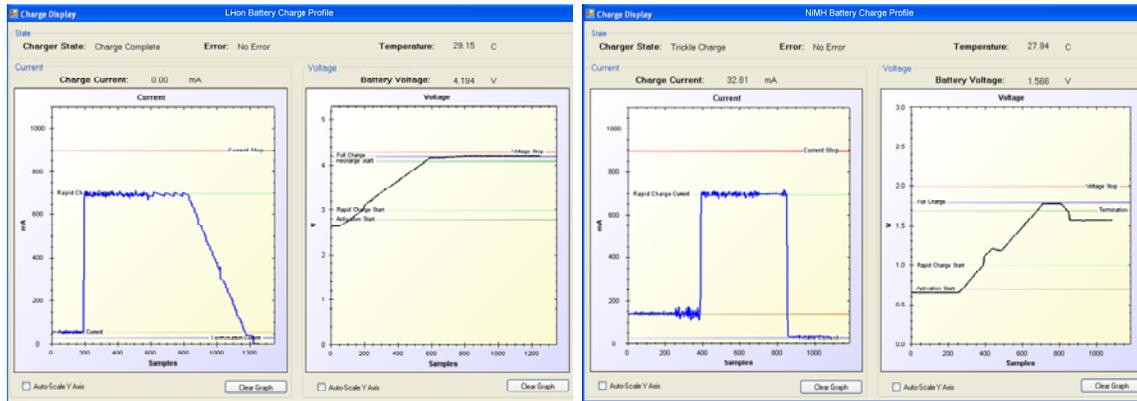
```

When the profile needs to be changed, the BATTERY\_PROFILE is set to 0 or 1 to switch between the two profiles. The voltage, current, and temperature limits for all the states are also saved as constants and changed accordingly. If a different voltage level for the same battery type is required, the code needs to be changed to enter the new parameters. This means the user of the application needs to be aware of the code to change the profiles and limits for battery charging. By taking a component approach, parameters can be entered for changing battery charger profile when developer select the appropriate IP blocks. For example, component encapsulations for Li-ion and NiMH batteries are shown in Figure 4.



**Figure 4: GUI to enter parameter limits of different battery chemistry**

By using these components, an application designer can add the charger component into an existing application and set up the appropriate profile. All the other hardware (comparators, PWM, etc) and software (state machines) are also generated by the component. Using a reprogrammable architecture such as the PSoC from Cypress, hardware components can be programmed and implemented with the software application. This method was used to program the hardware shown in Figure 3 with Li-ion and NiMH battery charge profiles. The battery parameters are sent to the computer by adding a USB component to the project. The data was plotted using software tool built using C#. Any other form of communication and similar tool can be used to plot the data. A battery emulator was used to emulate the Li-Ion and NiMH batteries to obtain the graphs in real time. The plots obtained as shown in Figure 5.



**Figure 5: Lilon and NiMH Charging Profile Plot**

The noise observed in the current is the switching noise due to the change of voltage using a battery emulator. Since the voltage was changed faster using a battery emulator, the response and settling time of the PWM output in response to change in voltage is seen as switching noise in the results above. The change of voltage in a battery is very gradual and thus switching noise is not significant with a real battery.

The stages of battery charging for the Li-ion and NiMH batteries in Figure 1 and Figure 2 can be observed in Figure 5. Thus it can be observed that with simple change in firmware of a SoC, a multi-chemistry battery charger can be obtained with the same hardware. The simplification obtained by making components of the profiles in components facilitates battery charging as an additional feature to a main application.

Cypress Semiconductor  
 198 Champion Court  
 San Jose, CA 95134-1709  
 Phone: 408-943-2600  
 Fax: 408-943-4730  
<http://www.cypress.com>

© Cypress Semiconductor Corporation, 2007. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC Designer™, Programmable System-on-Chip™, and PSoC Express™ are trademarks and PSoC® is a registered trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

This Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.