

USB Shutdown Controller

Contributed by Evan
Thursday, 19 October 2006
Last Updated Wednesday, 28 March 2007

Note: This project is currently very much a work-in-progress. Some of what is described here is already functional, some is still just plans, and all of it is subject to change as I go along. Please see the next page for more information on my current status with the project!

This project is the next revision of my original shutdown controller. For quite some time, I have been trying to piece together a computer system for my carPC setup that can handle S3 standby (suspend-to-RAM), giving boot times as fast as 10-15 seconds for occasions where I am turning my car on and off a lot. However, S3 standby has the unfortunate advantage of using a non-negligible amount of power, meaning I can't safely leave the machine in S3 standby for more than a couple of days, I estimate. A traditional SDC controls the power of the computer solely by shorting the power button pins on the motherboard front panel header, thus allowing you to configure in windows whether that action is associated with shutdown, standby, or hibernate. So if set to standby, you either have to plan ahead and manually shutdown or hibernate if you know you will be away from the car for too long, or rely on the SDC to cut power to the PC when the battery voltage gets too low, as a safety feature. But this is far from an elegant solution, resulting in either added user attention and effort, or undue stress on the battery.

Advanced Power Control

Therefore, one of the primary goals I had in mind for this SDC was to allow it more advanced control of the PC than merely the power button. This could be achieved via the serial or parallel ports, but I chose to do it via the USB port. This is orders of magnitude more complicated, as learning to interface with the USB port in the first place took me many weeks of research and experimentation, and even with experience it still involves a lot of coding. However, I feel it is by far the most elegant solution, since USB ports are convenient, nearly every carPC will have them, and it also opens up many other possibilities for control and configuration of the SDC from the PC. This can include allowing a simple GUI interface for modifying timing parameters of the SDC and making changes to behavior, as well as easy software integration of different monitored values such as voltage and temperature, etc.

Thanks to this added flexibility, the SDC will be able to order the PC to stand by, shut down, hibernate, or reboot, all via the USB interface, with some simple PC-side interface software running in the background. It will also maintain the traditional power button interface, which is necessary for powering the computer up (since the USB interface obviously won't work when the computer is off) and as a backup method of powering off the PC in the event that the USB interface is disconnected, the software crashes, etc. The end result is that the SDC will have the ability to, for example, put the PC into standby, wait for a certain amount of time (perhaps 6 hours, or 12, or a day) and if the car isn't started during that time, it can boot the PC back up, wait for it to be ready (testing for that via the USB interface) and then ordering it to hibernate or shut down. This is a level of control that I have not seen implemented on any other SDC to date, but I think that it is essential to making standby a safe, reliable, and convenient solution, that won't require extra effort on the user's part, or put significant stress on the car battery.

Another advantage of using the USB interface is that it is more of a closed-loop system. Since the SDC communicates with the PC software, it has the ability to detect whether the computer is actually booted into windows and running, as opposed to only knowing whether it is ON or OFF. That can eliminate one annoying issue with many traditional SDC's, which is that if the car is started for only a few seconds (such as just to move it to the other side of the driveway) and is shut off again before the PC has finished booting into windows, then when the SDC sends the power button signal to shut it down, it will not respond. That generally results in the PC booting as usual, sitting there until the SDC timeout expires, and then the SDC cutting power hard, which is exactly the kind of thing you are trying to avoid by having an SDC in the first place. Since this SDC can tell if its software is running, it can simply wait for the computer to finish booting, and shut it down then, while still implementing a backup timeout in case the computer crashes or hangs during boot.

Modular Architecture

Beside the USB interface, perhaps the most important idea for this project is designing it in a modular fashion. It only makes sense to place the SDC in or near the PC itself; it needs to connect to the PC in various ways (power, signals, USB, voltage monitors, temp sensors, etc). However, many people would also find it convenient to have a way to interact with the system; perhaps some LED status indicators, and maybe some buttons for manual control; and some people might want it to be even fancier. But, many people mount their carPC's in inaccessible places, such as the trunk, or under a seat. This limits the direct usability in the event that you want to be able to interact with it, and if you wanted to mount a number of indicators, buttons, etc in the dash, it could require running a lot of wires.

Thus, it would make more sense to have a more intelligent board in the dash that handles user interaction, and have it

communicate with the main board of the SDC via an I2C serial link, requiring just 2 signal wires plus ground, allowing you to run a single thin cable with a few conductors (phone cable would be perfect) from the dash to the main SDC board. The dash-mounted UI board could then be as complex as desired; it could be as simple as a few LEDs and a button or two for manual control, and only require a couple dollars' worth of parts, or could be more advanced, including a character LCD display and a menu system for adjusting parameters and displaying real-time voltage and temperature readings, power status, etc. And, of course, those who only want a straightforward SDC and aren't control freaks like some of us, could be all set with the main SDC board alone.

An I2C serial link also opens up potential future possibilities because it is a bus - other devices can be attached to it without requiring any additional circuitry. This could mean additional I/O, relays, temperature sensors, etc.

Software Interface

As mentioned previously, the SDC is intended for use with a simple PC-side program that at its simplest, it would communicate with in order to gain the necessary power control of the PC. This could be as unobtrusive as a small app running in the taskbar, or perhaps even a service running in the background. But the USB interface also opens up the possibility of a much richer software interface. Such an application could also provide an interface for other applications to monitor voltages, temperatures, keep track of SDC status, signal alarms with on-screen alerts if a fault is detected, and allow an easy interface for modifying settings and timing values of the SDC. It is expected that such features could be easily implemented in frontend software, for an integrated and user-friendly solution. The software interface would be an excellent companion or alternative to a hardware user interface, as described previously. Another advantage of the USB interface is that a bootloader can be used, allowing the firmware of the SDC to be upgraded through software with no additional hardware required.

Features

Aside from the USB interface and the features afforded by it, I also intend to develop this as a feature-rich SDC in all the traditional ways. A list of the general features I am planning on:

- Configurable modes, from basic timed power-down, to multi-step sequenced shutdown involving multiple power modes
- Configurable delay times, allowing flexible control of power use
- Voltage monitoring, including configurable warning/alarm thresholds where the system can be automatically powered down for safety
- Temperature monitoring, on-board and optionally remote sensors, also including similar protection modes as for voltage
- Auxiliary output(s), configurable with certain delays to control additional devices during startup/shutdown, such as amplifiers for anti-thump
- Status LEDs on-board, providing basic status information, particularly for debugging purposes during setup
- Fused Input, reverse-voltage, and over-voltage protection
- Connector to allow manual use of existing PC power button as well

Additional features which I plan to implement (most likely on an auxiliary remote-mounted board) - I expect to have multiple versions, including varying levels of features.

- Stay-On button, for those instances where you just need to shut off your car for a couple of minutes and would rather your PC stayed on, such as while pumping gas or visiting a convenience store. Configurable timeout as a backup.
- Stay-Off (Valet) switch, for when you want your PC to remain off even with the car running, such as when it's with a valet, or in the shop, or being borrowed by a friend. This switch could be in a hidden location, or use a switch requiring a key, for security.
- Emergency-Off button, just in case. This would be to immediately kill the power, no shutdown procedure involved.
- Reset button, just in case.
- Status LEDs, for simple status verification from the drivers' seat.
- LCD display, utilizing a small character LCD to display important information, and provide a simple menu system for configuration changes.

{mos_sb_discuss:8}

{mospagebreak title=Page 2 - Current Status}

Project History:

It's hard to say when I started work on this project, because it was one of the main motivations for me to get started with PIC USB interfacing sometime in the summer of 2006. Since then I've written a number of articles on the things I've learned, and all of these subjects were ones that I needed in order to do this SDC project the way I wanted to. It wasn't until probably November or December of 2006 that I got most of the background work out of the way and actually started focused work on the SDC project itself.

Current Progress:

The new main SDC board

Auxiliary board The core SDC functionality is working. This is implemented using a state-machine type model, which allows for more expandability and flexibility (from a programming standpoint) for the number of different power states that need to be traversed through in order to achieve the desired power sequencing.

The USB interface, one of the most important parts of the whole thing, is working. From my C# .NET 2.0 application, I can send commands to the SDC, receive its responses, and the SDC can asynchronously send messages to the computer application as well. I have been successful in getting the C# program to order the PC to shut down, stand by, hibernate, etc.

The SDC currently sends information about its status any time something changes - this can include any level of status information: as brief as a few flags indicating its current mode and any important errors, or as complex as monitored parameters like voltage, temperature, digital I/O, etc. The software can successfully both read and write all the various delay values on the SDC, allowing them to be adjusted easily from the PC. All delay values and settings on the SDC are stored in non-volatile EEPROM memory, and are thus maintained even when power is removed.

Currently the SDC is configured to monitor several analog input channels, out of the possible 12 that are on the 18F4550. These are hooked to various things, particularly the input (battery) voltage, 5v and 12v rails. There are still 5 channels available, broken out to solder pads for later use.

The I2C interface between the 18F4550 (SDC board) and a 16F88 (auxiliary board) is working, enough to send packets in either direction. Due to the Master-Slave nature of the I2C bus, extra work was required to achieve a semi-asynchronous communication between the boards in a moderately efficient manner, but has resulted in a flexible protocol that keeps the Auxiliary board easily expandable for future features.

The current auxiliary board consists of a few buttons and a character LCD - enough to use a menu system to change settings, delay times, display monitored parameters, etc. There are still at least 4 pins available, that can be analog input or digital I/O, the usage of these has not yet been decided - however, given the intended modular nature of the auxiliary board it's somewhat irrelevant as a microcontroller with more I/O pins could just as easily be used.

The SDC board is still partway between a prototype/development version and a usable version - I didn't add a lot of connectors yet because connectors are expensive, and not necessary for development purposes. There will be a molex power connector on a short pigtail, to bring in 5v and 12v for monitoring, and also for providing a switched regulated 12v output through the relay. There are also pin headers for power button, reset, power LED, etc.

Current focus is on increasing the level of control between the PC, SDC, and auxiliary board to achieve the aforementioned functionality goals, as well as adding more of the smaller features. {mos_sb_discuss:8}