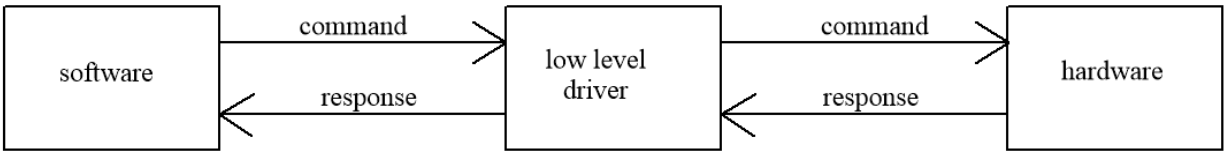
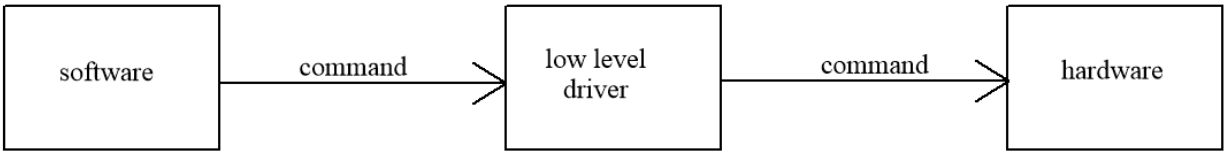
Creating a Shadow Model to Control Hardware That Cannot Respond back to the Software, Version 1.2

By R. G. Sparber

Warning: this technique should never be used where there Is a risk to life.

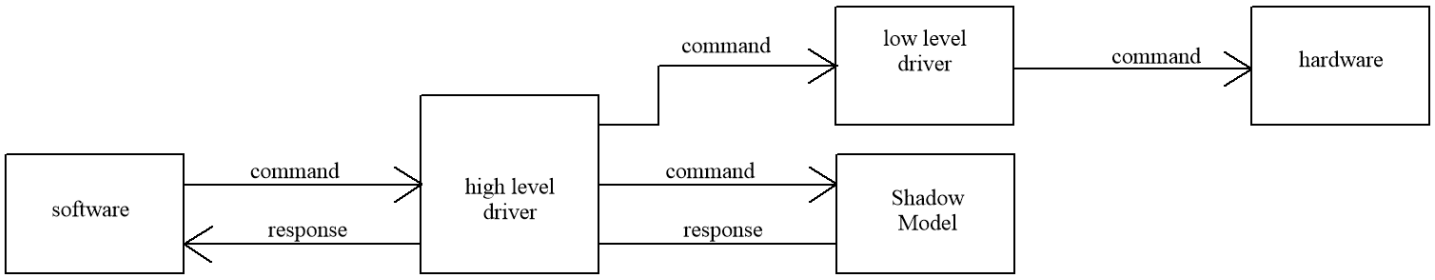


Ideally, I should be able to send a command from software, through a low-level driver, and on to a piece of hardware and have the hardware respond with status. Without this response, the software has no confirmation that the hardware is in the proper state.



Recently I had to control hardware that did not provide any response. Two commands existed that would toggle between states. Without knowing what state I was in, toggling could move me to the wrong state.

So how do I put lipstick on this pig?



To make the best of this bad situation, I created what I will call a "Shadow Model". This model behaves the way I want the hardware to work. I also added a high-level driver between the software and my hardware with low-level driver and my Shadow Model.

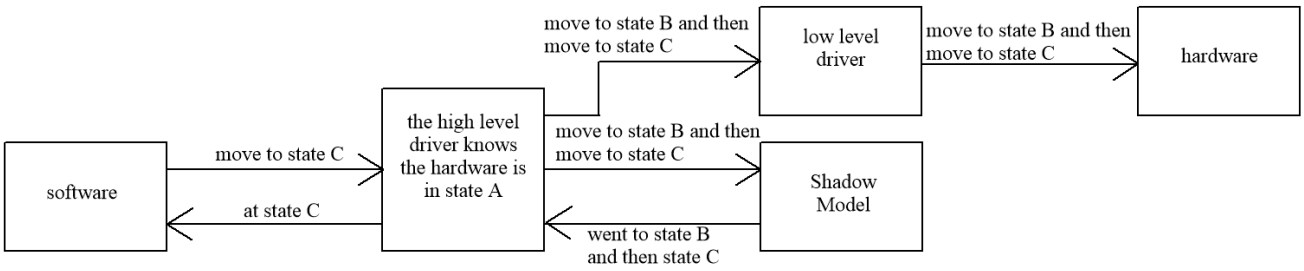
Each time the software sends a command to the real hardware, the high-level driver sends the command to the low-level driver and on to the hardware plus to my Shadow Model. The Shadow Model returns the needed response for the *assumed* hardware state.

What saves me here is that I can detect when the hardware powers up and be confident that at this point, it is in a known state. I then set the Shadow Model to this state. As the hardware moves between states, the Shadow Model moves between those same states ***and*** responds back. Then the high-level driver passes the Shadow Model's response back to the software as if it came from the hardware.

The high-level driver also restricts access to the hardware so all commands pass through it. This prevents the hardware and Shadow Model from being in different states. Without it, a user could directly command the low-level driver and forget to update the Shadow Model's state.

This is not as bulletproof as interfacing to properly designed hardware but it is better than just blinding firing off state change commands.

An added benefit to having the high-level driver is its ability to shield hardware complexity from the software. Some state transitions are illegal in the hardware but can be made legal as far as the software is concerned. The high-level driver just adds the extra commands to keep the hardware happy and the software interface simple.



In this example, the software wants the hardware to move to state C. Requiring the software to keep track of the current state of the hardware and legal state transitions is unnecessary complexity for the interface.

Instead, the high-level driver knows the hardware is in state A and that it must first command a change to state B before it can move from B to C. The two-state changes are passed to the low-level driver and on to the hardware while also being sent to the Shadow Model. The Shadow Model responds with a confirmation that it is in state B and then in state C. These responses go back to the high-level driver and is returned to the software as confirmation that the hardware is now in the expected state.

Acknowledgment

Thanks to Marv Klotz for pointing out that this control technique should never be used where there is any risk to life.

I welcome your comments and questions.

If you wish to be contacted each time I publish an article, email me with "Subscribe" in the subject line. In the body of the email please tell me if you are interested in metalworking, software, and/or electronics so I can put you on the best distribution list.

If you are on a list and have had enough, email me "Unsubscribe" in the subject line.

Rick Sparber
Rgsparber.ha@gmail.com
Rick.Sparber.org