10 Tips for designing a HAL

Jacob Beningo - June 02, 2015

Designing a HAL is a great first step to developing firmware that is reusable and hardware independent. No, a HAL is not the infamous artificial intelligence from 2001: A Space Odyssey. The HAL, or Hardware Abstraction Layer, provides the application developer with a set of standard functions that can be used to access hardware functions without a detailed understanding of how the hardware works.

HALs are essentially API’s designed to interact with hardware. A properly designed HAL provides developers with many benefits, such as code that is portable, reusable, lower cost, abstracted, and with fewer bugs. A poorly designed HAL, though, can result in increased costs and buggy software, and leave the developer wishing that they were dealing with the previously mentioned infamous HAL. To make sure it's proper, here are 10 tips developers can follow when designing their own HAL.

Tip #1 - Identify core features

A HAL needs to be a consistent and standard set of functions that can be used across multiple hardware platforms. Microcontrollers come with a standard set of peripherals all of which serve a particular purpose in an embedded system. When developing a HAL, examine each of the standard microcontroller peripherals and identify their core features. A few core features that would be needed for a communication device, for example, would be an initialization, transmit, and receive function. These are basic must-have functions that would be needed in nearly any application. An example of a core HAL for a UART can be found in Figure 1.

2 void Uart_Init(const Uart_ConfigType *Config);
3 uint8_t Uart_RxChar(Uart_ChannelType Channel);
4 void Uart_TxChar(int ToPrint, Uart_ChannelType Channel);

Figure1 - UART HAL
Tip #2 – Avoid an all-encompassing HAL

Engineers sometimes fall into the “one ring to rule them all” trap. The trap is that engineers start with something simple and elegant and then grow the solution to cover the universe. HAL designers should avoid trying to create an all-encompassing or singular HAL to rule every microcontroller device and peripheral. The reason to avoid an all-encompassing HAL is that complexity, cost, and the potential for bugs will drastically increase if you try to create one. Every microcontroller has niche features, so it would just be impossible to create a standard and elegant HAL for them all.

What can a developer do to handle niche peripheral features that aren’t handled by the HAL? The answer is to build register access functions into the HAL. A HAL can expose the fact that it doesn’t cover every possible use and state of the peripheral and instead provide write and read access to select registers within the driver. The register access functions would be considered “expert” mode HAL functions that should be used only by developers who are familiar with the inner workings of the microcontroller. An example of how the register access functions might look can be seen in Figure 2.

```c
7  void Uart_RegisterWrite(Uart_RegisterType Register, uint16_t RegisterData);
8  uint16_t Uart_RegisterRead(Uart_RegisterType Register);
```

Figure 2 – Register Access HAL

Tip #4 – Use Doxygen to outline the HAL

A great way to plan out and develop documentation for a HAL is to outline it using Doxygen. There are a number of advantages to using Doxygen to plan the HAL. First, Doxygen uses code comments to generate HTML, RTF, and PDF documents, which means the developer already has source comments on what the different function are supposed to do. Second, since the comments for the HAL are automatically developed, the HAL source files become a blank template from which developers can fill in the HAL functions per the software architecture and requirements. Finally any updates that are made to the HAL over time can be made in one place, the Doxygen files, and then the updates easily propagate to the documentation.

Tip #5 – Get a second set of eyes

Getting a second set of eyes on the HAL is a wonderful way to get a fresh perspective. In fact, one of the best things to do during any development cycle is to get multiple eyes on the design. Every engineer has his or her own views and experiences that can be contributed to the HAL. Feedback from multiple parties, especially those that may have to use the HAL, is a great way to minimize how many changes will need to be made to the HAL and helps to ensure that the HAL will survive long term to maximize code reuse and minimize cost.

Tip #6 – Don’t be afraid to iterate

During the first release of a HAL there are going to minor problems and discoveries that were overlooked during the design and review of the HAL. Don’t sweat it! Designing a perfect HAL is unrealistic and the goal should be to develop one that is good enough to start using. Gather feedback from the users of the HAL and then make minor, iterative updates. Make sure that the changes are well documented so that legacy HAL users can easily update to the latest revision. After a few iterations a developer will find that their HAL has become a very well-oiled machine that saves precious development time.
Tip #7 - Keep the view at 30,000 feet

Remember that one of the HAL's purposes is to provide a standard and consistent interface that abstracts the hardware functionality. Keep the interface simple and the level of detail of how the hardware works at the 30,000 feet view. A great test is to have a manager or a software newbie review the HAL and ensure that they can understand how it works.

Keeping the HAL at a high abstraction level will not only help to maximize its use it will also eliminate misunderstandings that can result in long debugging sessions, increased costs, or missed deadlines. Also keep in mind that the HAL should allow enough leeway that a developer can implement the HAL functions in a way that fits their requirements and application needs. The API or HAL should allow for different low-level implementation strategies to be implemented and supported.

Tip #8 - Use appropriate naming conventions

A safe bet when developing a HAL is to use an interface that is ANSI-C compliant. An ANSI-C compliant HAL will ensure portability across multiple compilers and tool chains. An example of an ANSI-C compliant requirement would be to limit the function name length to 31 significant characters. Additional considerations would be to use standard portable types and avoid compiler intrinsics. Another quick tip is to define a short coding standard with naming and coding standard best practices on how the HAL interface should be written.

Tip #9 - Include a parameter for initialization

One of the most common mistakes encountered when designing a HAL is to have a peripheral initialization function take no parameters. In essence, the initialization is hard coded for every application. A parameter-less initialization greatly limits the HAL’s portability. An initialization function would be better served passing a pointer to a configuration table. The simplest implementation would just have an empty void table. A more complex implementation would use the pointer to loop through the table and configure the peripheral. Either way, passing a pointer provides greater portability and reuse to the HAL.

Tip #10 - Deploy on multiple development kits

A simple and effective way to test out a HAL is to deploy it on multiple microcontrollers from different silicon vendors. Developing simple test code will help to shake out the HAL and elucidate any portability issues up front. Development kits are a great way to cheaply get hardware to test HAL’s on. Most microcontroller development kits cost less than $20.

Conclusions

There are many benefits to utilizing a HAL in the embedded software space. The chief benefits are having a set of functions that are well defined that allow application code to be easily ported from microcontroller to microcontroller. Additional benefits include reuse, decreased costs and the abstraction of the hardware. A few might argue that a HAL will decrease execution efficiency and increase code space. The modern microcontroller and compiler have nearly put these concerns to rest with their architectures and optimization capabilities. For additional examples on how to create a HAL or utilize an existing HAL visit www.beningo.com.

Jacob Beningo is a Certified Software Development Professional (CSDP) whose expertise is in embedded software. He works with companies to decrease costs and time to market while maintaining a quality and robust product. Feel free to contact him at jacob@beningo.com, at his
website www.beningo.com, and sign-up for his monthly Embedded Bytes Newsletter here.

Join over 2,000 technical professionals and embedded systems hardware, software, and firmware developers at ESC Silicon Valley July 20-22, 2015 and learn about the latest techniques and tips for reducing time, cost, and complexity in the embedded development process.

Passes for the ESC Silicon Valley 2015 Technical Conference are available at the conference’s official site with discounted advance pricing until July 17, 2015. The Embedded Systems Conference and EDN are owned by UBM Canon.