Introduction to the Software Platform

Introduction to the Software Platform

Organization of the Software Platform
Using the Software Platform Builder
Glossary

The Software Platform is a software framework to facilitate writing software to access peripherals on your hardware design. It also facilitates the implementation of software protocols and it provides extra functionality that you can use in your application, such as multithreading. In essence it is a collection of software modules, delivered as source code. You can add these modules to your embedded project to take care of various low level routines that are required to control or access peripherals. Each module also provides an interface to the application, offering specific functionality (for example, `set_baudrate()`, a function to dynamically change the baud rate).

The Software Platform Builder is the graphical user interface that you use to configure and add modules to your embedded project. To create a Software Platform, specific for your project, you use the Software Platform Builder. The Software Platform Builder becomes available when you add a special document to your embedded project: a Software Platform file with extension `.SwPlatform`. This document both represents the Software Platform for your project and provides you with a graphical interface to select and configure the functionality you need. What you need, of course, depends on the peripherals on your hardware design that you wish to access in your application. The Software Platform Builder can read your FPGA design and import the appropriate low-level modules for the peripherals on your FPGA design. You can use this import as a starting point and add more functionality at a higher abstraction level to the Software Platform file.

For a tutorial that explores the use of the Software Platform Builder, see Tutorial - Getting Started with the Software Platform Builder.

Why You Need the Software Platform

To write software that access or controls a peripheral, you need a thorough knowledge of how the peripheral works: which registers you need, which device specific commands to use, which communication protocols to use, and which interrupts to handle.

A simple design situation

For a simple FPGA design it is normally not too complex to write such software yourself. For example, to control a row of LEDs, you need to write some 8-bit binary number to an address. That address would be the location of the peripheral on the FPGA to which the physical LEDs on the NanoBoard are connected. It could well be the first address of the processor I/O space (for example 0xFF000000). If you write in your C source something to that address, the LEDs will behave accordingly.

No device specific commands, no interrupt handling, no special registers, no communication protocols
and no special data handling are required. It is easy to manually write software for this design. Even in this simple example, the C code is not fully portable since the base address of the LEDs is defined on the FPGA design and may become invalid if your hardware design changes.

**A more complex design situation**

However, things can get soon out of hand when you want to control more complex peripherals. Imagine a design that contains a PS/2 keyboard. An end-user may press for example the key combination "Shift + A" and the purpose of your application is to store the capital letter 'A' in a variable for later use in your application. This seems simple but a lot needs to be taken care of beneath the surface.

- At the lowest level you need to connect to the PS/2 port by defining the base address of the PS/2 port and the addresses to access the clock line and the data line of the port.
- At the next level, your application needs to be ready to receive an interrupt from the keyboard on the PS/2 peripheral. You need to correctly handle the interrupt and read incoming data from the data line on the PS/2 port. The keyboard sends serial data in 11-bit frames. Multiple frames make up a scancode which represent each possible event on the keyboard (i.e. a key press or a key release).
- At a higher level you need to collect the data and store the actual scan codes. A scan code can be 1 to 8 bytes long. (For example, 0xC1 corresponds to pressing the 'A'-key, 0xF0 0xC1 corresponds to releasing the 'A'-key.).
- At the highest level, you need to interpret the scan codes by mapping them to the correct keys. You need also need to detect key combinations. In our example you need to remember the fact that the scancode for a Shift-key press (make code) was received before you are received the 'A'-key make code which you should now interpret as a capital 'A'. The interpreted input should be buffered correctly.

Though this description is far from complete (it abstracts from sending data, keyboard initialization, etc), it indicates the complexity you will face when you need to write software to handle a peripheral device like a PS/2 keyboard.

At this point we introduce the Software Platform. The modules of the Software Platform will take care of all these laborious lower level routines and instead provide you with an easy to use Application Programming Interface for each peripheral that your application needs to control.

**What is the Software Platform?**

The Software Platform provides numerous software modules that take care of lower level software routines as well as modules that offer extra functionality by providing you with a convenient API which you can use in your application.

The lowest level modules provide configuration data and drivers for a particular peripheral device. Higher level modules are more abstract and more hardware independent. For example, at the higher, abstract level, you could choose to use a module to access a file system in your application. At the lower levels you still can select modules to decide which specific storage device you want to access (a hard drive, SD card, ram drive, ...) Thus, the lower level modules are more specific for a particular peripheral while the higher level modules are less hardware specific and can even be used in combination with multiple peripheral devices.

**Using the Software Platform in an Embedded Project**
Once you have an embedded project in Altium Designer, you can add a special document to the project with the extension .SwPlatform. This document is no ordinary source document, but merely a graphical representation of the software modules that you are going to select for your project. This Software Platform "document" also provides a graphical user interface, the Software Platform Builder, which is used to select the software modules. With these modules you can form device stacks: higher level modules are placed onto lower level modules. By placing more or less modules in a stack, you can choose the abstraction level you want to use in your application.

For the PS/2 keyboard example, you would select four modules, for each abstraction level one. The figure shows an example of the stack. Each colored block represents a software module. In general (unless specified otherwise), only the highest level module provides the interface to your application. In this example stack for the PS/2 keyboard, the top level module is the Keyboard I/O Context which provides you with a standard C interface to access the keyboard. This is realized because the standard C functions which you can call in your application, on their turn call lower level functions. The module provides a specific implementation for these lower level C I/O functions. In this example the following low level functions have a specific implementation:

- `_open()` Used by the functions `fopen()`
- `_read()` Reads a sequence of characters from a device (now from the keyboard driver)
- `_write()` Writes a sequence of characters to a device (now to the keyboard driver)
- `_close()` Used by the functions `close()` and `fclose()`

The next section discusses in more detail how the modules are organized and how you should choose them to assemble a stack.

**Technical Background**

As described before, modules from the Software Platform are configurable software packages, delivered as source code, that become part of your embedded project when you add them to the Software Platform document. When you compile your embedded project, these selected modules are compiled into a software library. This library will be linked with the embedded project after which you can call the functions in this library from your application.

The library is recompiled every time that the Software Platform encounters changes in the embedded configuration.
The Generated Software Platform Library

Every time the Software Platform saves a configuration, based on the settings you specified, it generates a *makefile* that contains all information to build the library. This makefile includes global compile options, macro definitions and source specific compile options among some other things such as whether the interface is visible or not.

In general, the files to be compiled can be divided into two sets: permanent and generated files. The *permanent* sources determine the Software Platform’s repository and their code will be shared between all the embedded projects that use them. The *generated* files are outputted by the Software Platform for each new configuration and are specific to your project. These generated files are C source files and include files and may contain embedded data, types and macros. When you build the embedded project, the permanent and generated files are both compiled to form the library that will be linked with your project.

**Instantiation of Modules**

The precise process of how the library is build, is beyond the scope of this manual. However, it is important to understand that the modules that you select to be part of your project, become in fact *instances*. A module can be configured globally, but once instantiated (with the global configuration settings) multiple instances of the same module can be configured individually. In other words, you specify which modules should become part of the Embedded Project when it is build and with which specific configuration. This combined information is used to build the Software Platform library which then will contain configured instances of the selected modules. This way, the generated library is always specific for a particular project.

**Conclusion**

The Software Platform provides a large number of ready made software modules that make it much easier to develop your software. It saves you from all the (hardware) details and communication protocols you would need to know about. In addition, because you can rapidly change the configuration of modules (by modifying the special Software Platform document), the Software Platform enables you to write portable, hardware independent applications. This allows you to change the hardware design without having to completely rewrite your application.

---

**Source URL:** https://techdocs.altium.com/display/AEE/Introduction+to+the+Software+Platform