Core 64: Software Block Diagram

User Application

Speaker

Expansion Port

FreeRTOS

Process Monitor

Serial Debug Port Update

Memory Monitor

Button Detect

Battery Voltage

Heartbeat Blink

LED Array

HAL

Core Memory

HAL

LED Array

Driver

Core Memory

Driver

OLED

3rd Party Libraries (FastLED …)

Arduino Libraries

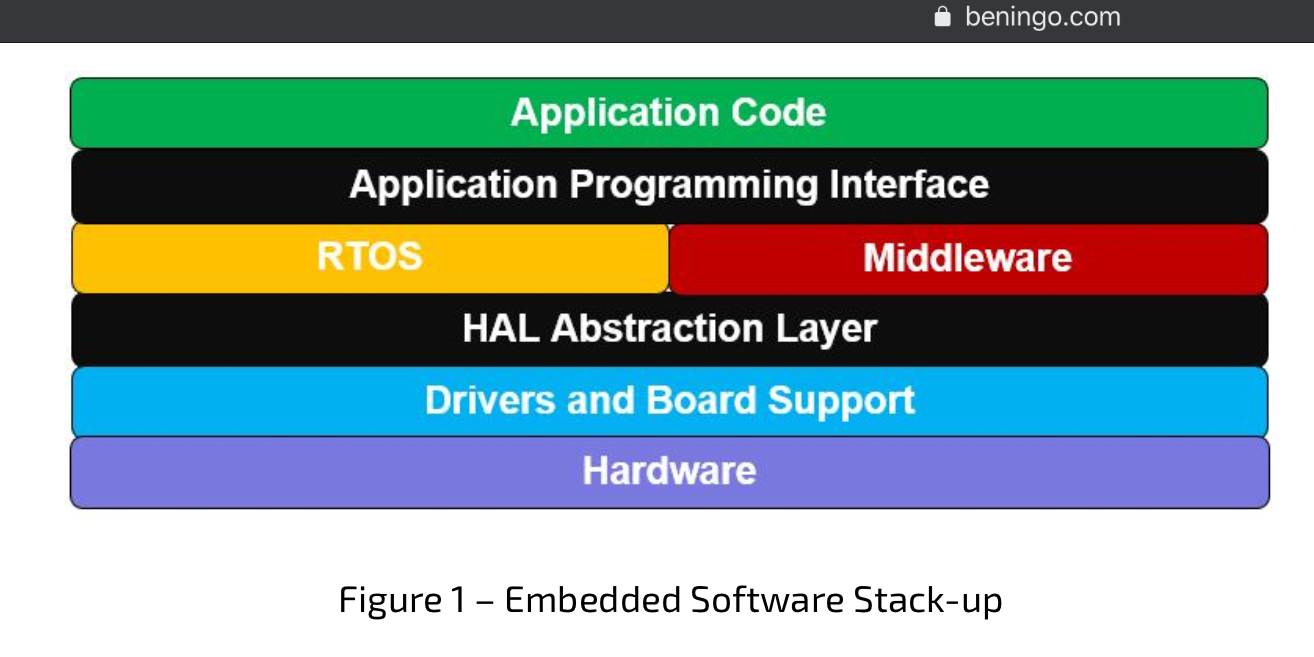
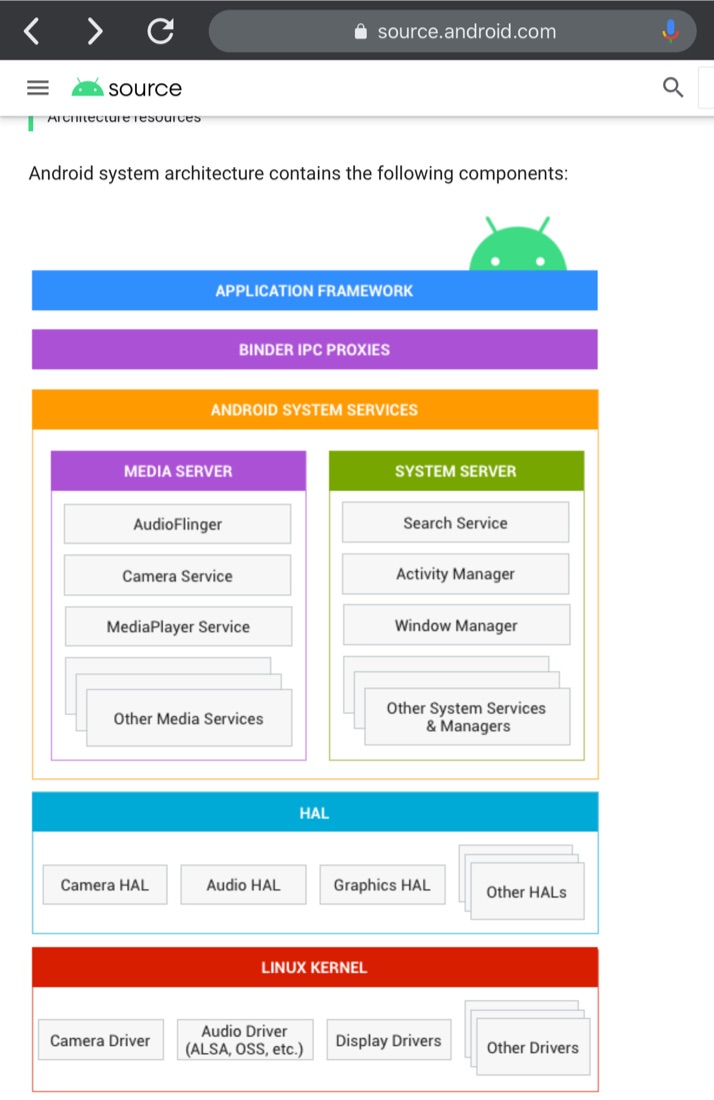
Hardware

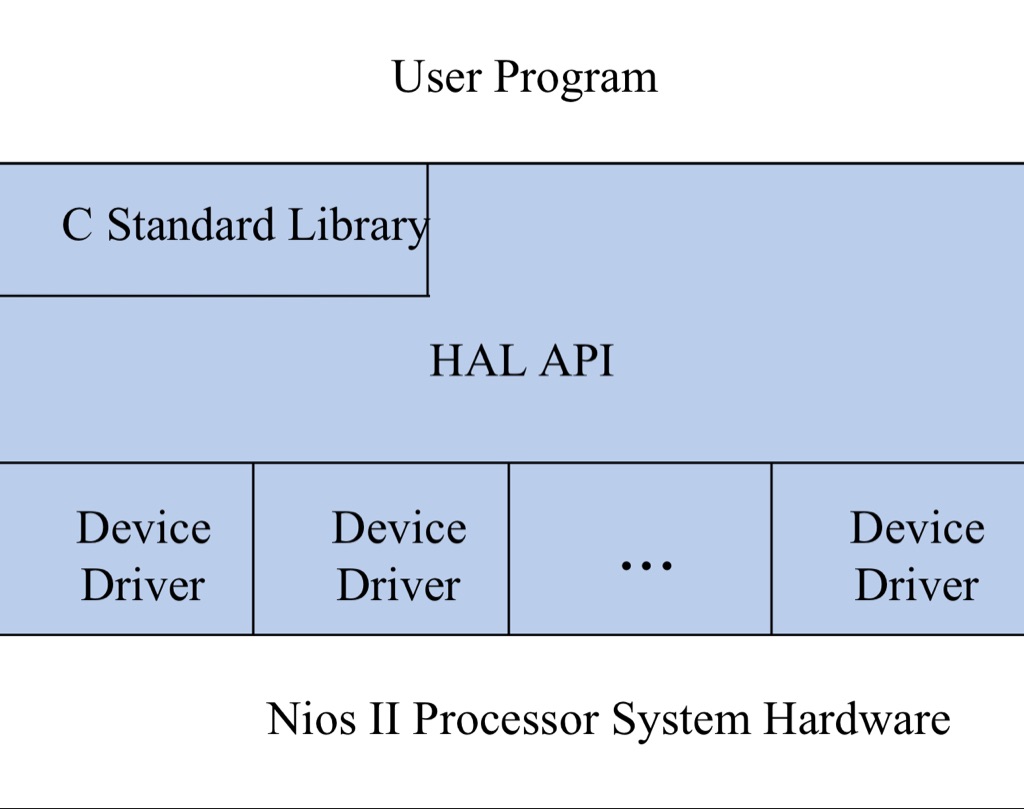
|  |  |  |  |
| --- | --- | --- | --- |
| LAYERS | SYSTEM | LED ARRAY EXAMPLE | CORE MEMORY EXAMPLE |
| Application | Task\_Top\_Level\_State\_Machine: The user application. Initially this a few different demo and test modes of operation, using different combinations of Middleware Functionality. |  |  |
| RTOS | Arduino\_FreeRTOS.h: manages everything else which are all running as tasks in the RTOS. |  |  |
| Middleware |  |  |  |
| HAL |  |  | Core\_Mem\_Driver (rename this to Core\_Mem\_HAL): |
| Drivers and Board Support | Arduino hidden includes  HardwareIOMap.h |  |  |
| Hardware | Teensy  MCU  GPIO | Teensy  MCU  GPIO |  |

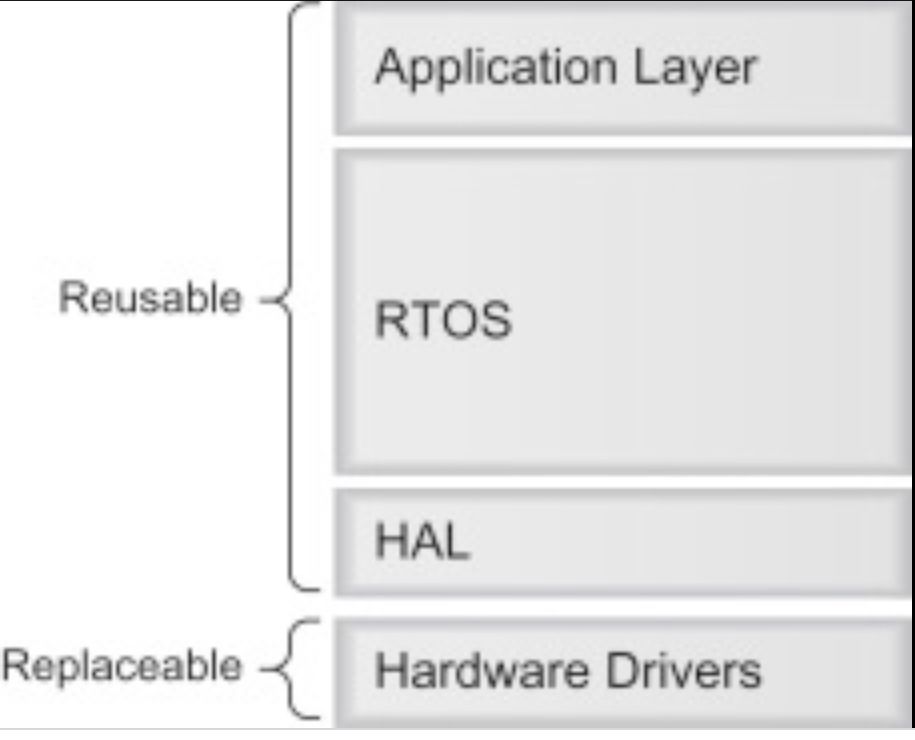
Considering changing the paradigm to HAL over DRIVER instead of the other way I was thinking about it.

Reading the book: Reusable Firmware Development by Jacob Beningo

HAL over DRIVER

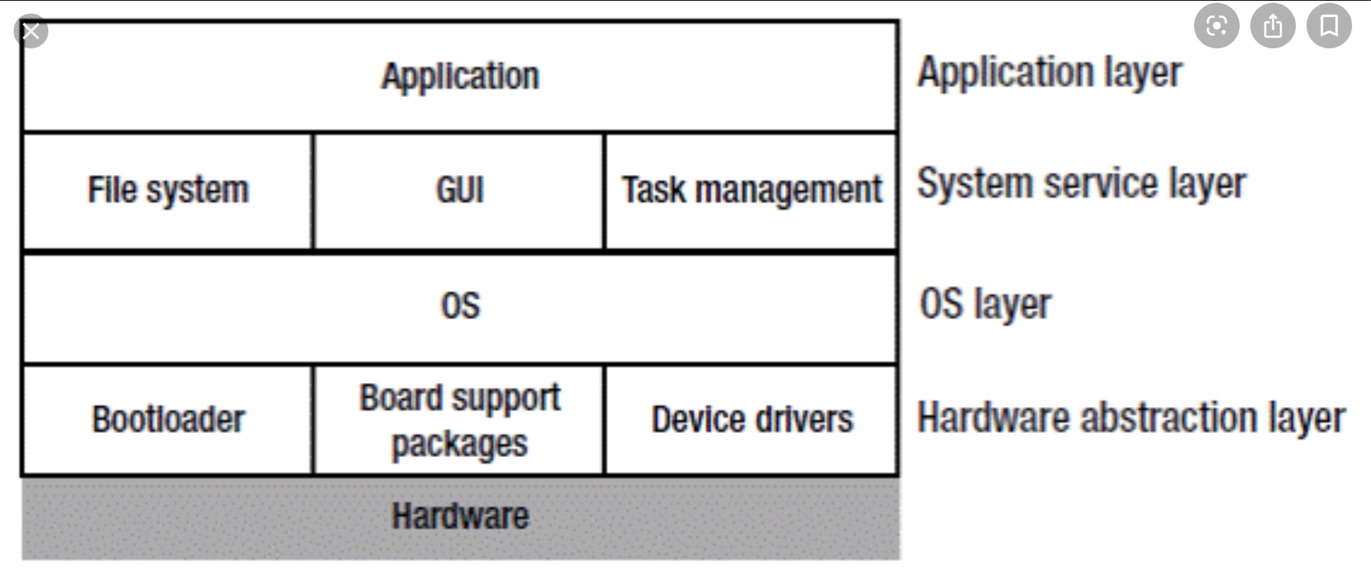


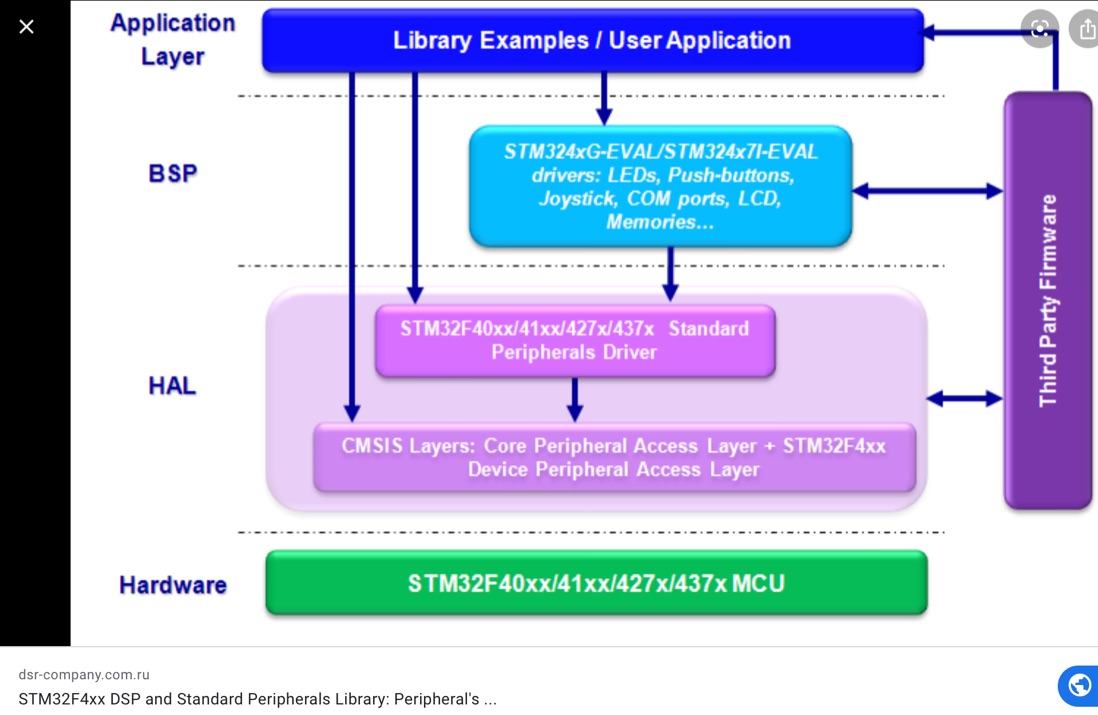




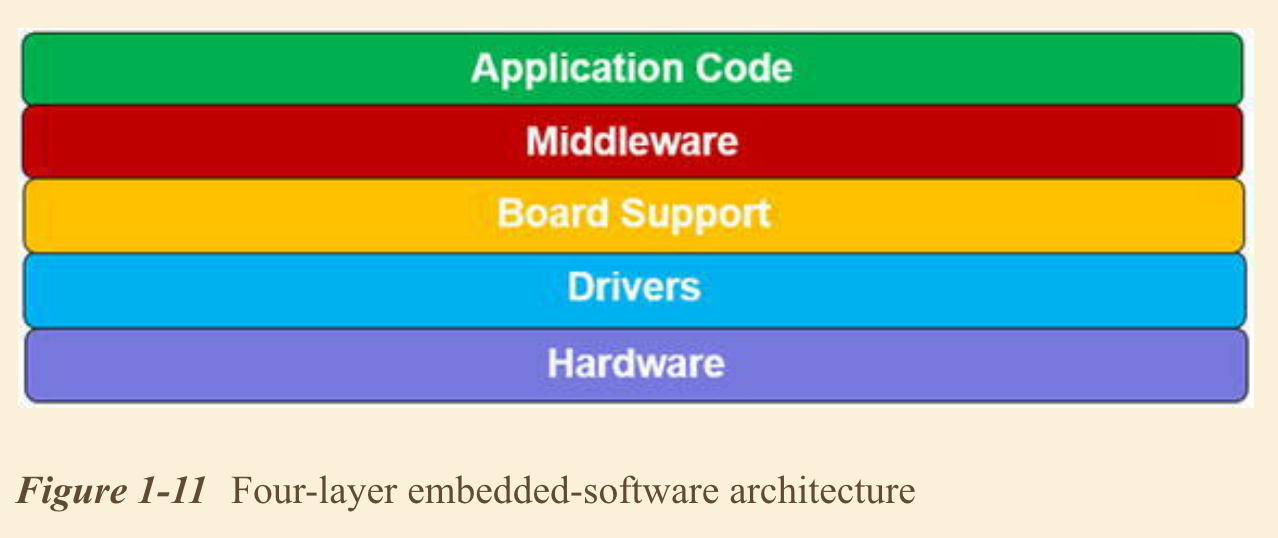
DRIVER over HAL

DRIVER IN the HAL



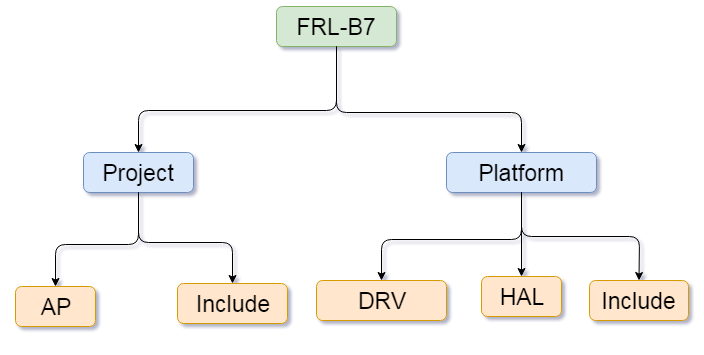
NO DRIVER

NO HAL



**Directory Structure Ideas**

The following figure shows the top-level source tree:



**Project**: It holds main application layer only.

1. **AP :** Application related all .c files are placed under the AP directory
2. **Include:** It holds .h files of application layer

**Platform**: It is divided into two sub-directory.

1. **DRV:** Drivers related APIs
2. **HAL:** Hardware application related APIs
3. **Include:** It holds .h files of DRV and HAL layer