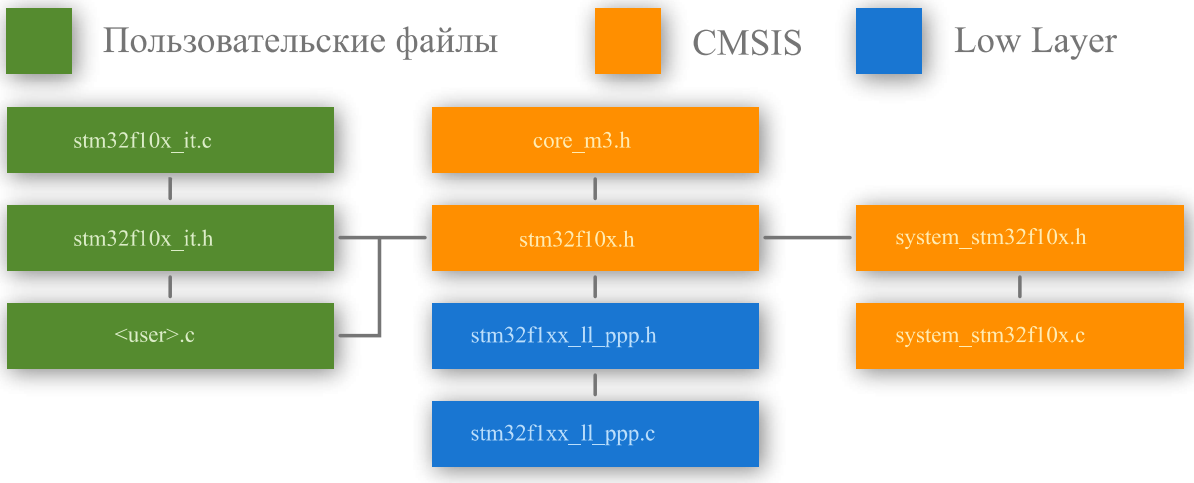
# Низкоуровневая библиотека

У стандартной библиотеки периферии есть два недостатка (личное мнение): в ходе разработки она слишком разрослась и не предоставляет унифицированного интерфейса (поэтому был придумал HAL); и не все операции являются атомарными (хотя в случае инициализации вряд ли это можно назвать проблемой).

По сути низкоуровневая библиотека (англ. low layer) — это реинкарнация стандартной (разработка которой прекращена). Однако она не такая гибкая, как ее предшественник: предусмотрены функции только для основных возможностей периферии [1], если вам нужно работать с USB, то сделать это через LL не получится. Кроме функций, дублирующих возможности StdPeriph (объявление, заполнение и передача в функцию инициализации структуры), низкоуровневая библиотека предоставляет `inline`-функции прямого доступа (атомарного) к регистрам.

Такой подход (атомарных операций) лучше: во-первых, их можно вызывать без опасения, что они будут прерваны исключительной ситуацией; во-вторых, не нужно тратить дополнительную память на хранение структур; и в-третьих, снижаются накладные расходы, ведь вызывать функцию (а значит, и сохранять стек) не приходится — `inline`-функция вставляется в место вызова, как макрос. Для того чтобы подключить библиотеку, нужно объявить макрос `USE_FULL_LL_DRIVER` (в настройках проекта).

Ниже приведена типичная структура проекта.




Низкоуровневая библиотека, как и стандартная, для своей работы использует CMSIS, имеет схожий принцип именования файлов ( `stm32yyxx_ll_ppp.h` , `stm32yyxx_ll_ppp.c` ) и разбита на три подуровня.

Низкоуровневая библиотека, как и стандартная, для своей работы использует CMSIS, имеет схожий принцип именования файлов ( `stm32yyxx_ll_ppp.h` , `stm32yyxx_ll_ppp.c` ) и разбита на три подуровня.

- **Уровень 1.** Обертки возможностей CMSIS: `LL_PPP_WriteReg()` / `LL_PPP_ReadReg()` .
- **Уровень 2.** Атомарные операции:
  - включение/выключение периферийных блоков (в том числе их частей), например `LL_PPP_Disable(PPPx)` ;

  - запуск периферии или установка ее в функциональное состояние, например `LL_PPP_Action()`;
  - вспомогательные функции, например `LL_PPP_State(PPPx)`;
  - работа с прерываниями (в том числе с флагами событий), например `LL_PPP_State(PPPx)`.
- **Уровень 3.** Функции инициализации периферии.

Функциональность включения/отключения периферийных блоков вынесена из `_rcc` в `stm32f1xx_ll_bus.h`. В файле `stm32f1xx_ll_system.h` расположены некоторые функции для работы с falsh-памятью и отладчиком. В файле `stm32f1xx_ll_utils.h` присутствуют функции для настройки PLL, задержки и считывания идентификатор МК (уникальный номер).

```
__STATIC_INLINE uint32_t LL_GetUID_Word0(void) // Word1, Word2
{
  return (uint32_t)(READ_REG(*((uint32_t *)UID_BASE_ADDRESS)));
}
```

Перепишем всё тот же пример инициализации ножки микроконтроллера на выход.

```
LL_APB2_GRP1_EnableClock(LL_APB2_GRP1_PERIPH_GPIOA);

LL_GPIO_InitTypeDef gpio;
gpio.Pin = LL_GPIO_PIN_0;
gpio.Mode = LL_GPIO_MODE_OUTPUT;
gpio.Speed = LL_GPIO_SPEED_FREQ_LOW;
gpio.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
LL_GPIO_Init(GPIOA, &gpio);
```

Для перевода старого проекта на низкоуровневую библиотеку ST разработала утилиту SPL2LL-Converter. Детальное описание библиотеки можно найти в документе **UM1850**.

---

**Назад** | **Оглавление** | **Дальше**

---

1. Модули FLASH, NVIC (есть в CMSIS), DFSDM, CRYP, HASH, SDMMC(SDIO) низкоуровневой библиотекой не поддерживаются. ↩