Слой аппаратной абстракции HAL

Последняя библиотека — слой аппаратной абстракции (англ. Hardware Abstraction Layer, HAL). Ее основные задачи — сократить время разработки и позволить писать портируемый на любое семейство STM32 (F0, F1 и т.д.) код. С одной стороны, она похожа на стандартную библиотеку: для инициализации периферийного блока используется структура. Перепишем инициализацию порта ввода-вывода с использованием библиотеки HAL:

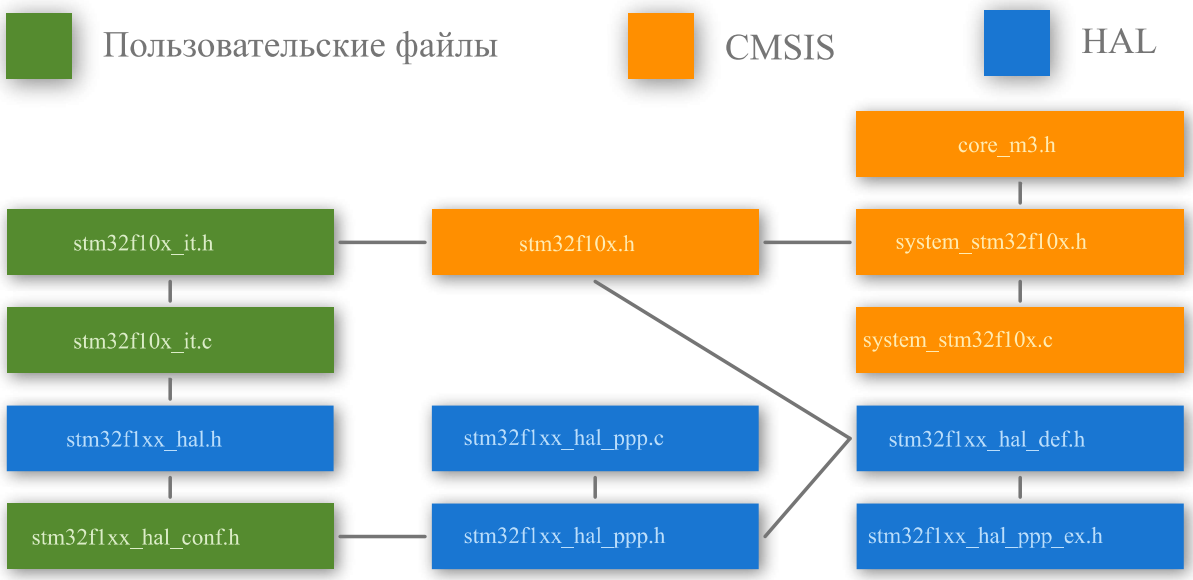
```
__HAL_RCC_GPIOA_CLK_ENABLE();
GPIO_InitTypeDef gpio;
gpio.Pin = GPIO_PIN_0;
gpio.Mode = GPIO_MODE_OUTPUT_PP;
gpio.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

Код не сильно отличается от StdPeriph или LL, а именование файлов осуществляется схожим образом: `stm32f1xx_hal_ppp.c` / `stm32f1xx_hal_ppp.h`, где `ppp` — название периферии. Учитывая опыт создания стандартной библиотеки, в HAL введено разделение на общие (т.е. применимые для всех семейств МК) и специфические функции. Вторые, если они есть, помещаются в отдельный файл `stm32f1xx_hal_ppp_ex.c` / `stm32f1xx_hal_ppp_ex.h` (суффикс `ex` происходит от слова extend, расширенный).

Все модули подключаются через конфигурационный файл `stm32f1xx_hal_conf.h`:

```
#define HAL_MODULE_ENABLED
#define HAL_ADC_MODULE_ENABLED
/*#define HAL_CRYP_MODULE_ENABLED   */
/*#define HAL_CAN_MODULE_ENABLED   */
/*#define HAL_CEC_MODULE_ENABLED   */
/*#define HAL_CORTEX_MODULE_ENABLED   */
// ...
```

Функции для работы с системными ресурсами (SysTick и NVIC) переопределены в файлах `stm32f1xx_hal_cortex.c` / `stm32f1xx_hal_cortex.h`. Инициализация периферийных устройств осуществляется через файл `stm32f1xx_hal_msp.c`.



Первое, что должно быть сделано в функции `main()` — вызов `HAL_Init()`, которая настраивает доступ к флеш-памяти и системный таймер. По умолчанию он используется для реализации функции задержки, по этой причине при использовании операционной системы реального времени в качестве источника системного тика должен быть выбран другой таймер.

Не надо думать, что на этом все особенности библиотеки заканчиваются. Если речь не идет об общих или системных ресурсах (GPIO, SysTick, NVIC, PWR, RCC, FLASH), вводится еще одна сущность — дескриптор (англ. handle). Он используется для полного описания объекта [1] в системе. Если мы говорим о модуле коммуникации (USART, SPI, I^2C и т.д.), мы должны помнить о его основной задаче — обмене данными, которые обычно хранятся в буфере. Проблема в том, что нужно завести как минимум два массива (на прием и на отправку) плюс еще две переменных (или макроса) с размером буферов. У блока может быть несколько режимов работы (через USART реализуется IrDA и SMARTCARD, например), кроме того, сам блок имеет внутреннее состояние (он сейчас что-то отправляет или, наоборот, ожидает команд). В устройстве при этом может находится несколько таких блоков — два, три, кто знает? В итоге получается, что для обслуживания одного «экземпляра» (англ. instance) USART требуется создать кучу переменных, в именовании которых можно легко запутаться. Логичным решением является обертка всех этих переменных в структуру.

```
typedef struct {
    USART_TypeDef                 *Instance;
    UART_InitTypeDef              Init;
    uint8_t                       *pTxBuffPtr;
    uint16_t                      TxXferSize;
    __IO uint16_t                 TxXferCount;
    uint8_t                       *pRxBuffPtr;
    uint16_t                      RxXferSize;
    __IO uint16_t                 RxXferCount;
    DMA_HandleTypeDef             *hdmatx;
    DMA_HandleTypeDef             *hdmarx;
    HAL_LockTypeDef               Lock;
    __IO HAL_UART_StateTypeDef    gState;
    __IO HAL_UART_StateTypeDef    RxState;
    __IO uint32_t                 ErrorCode;
} UART_HandleTypeDef;
```

```
// ...
UART_HandleTypeDef hGSM;
UART_HandleTypeDef hCOM;
```

Библиотека HAL реализована таком образом, что все функции в ней являются реентрантными (англ. reentrant), т.е. их можно без опаски выполнять «одновременно», что актуально для операционной системы реального времени. Реализуется это посредством механизма блокировки (файл `stm32f1xx_hal_def.h`).

```
typedef enum {
    HAL_UNLOCKED = 0x00U,
    HAL_LOCKED   = 0x01U
} HAL_LockTypeDef;
// ...
#define __HAL_LOCK(__HANDLE__) \
                                   do{                                   \
                                   if((__HANDLE__)->Lock == HAL_LOCKED)  \
                                   {                                     \
                                       return HAL_BUSY;                  \
                                   }                                     \
                                   else                                  \
                                   {                                     \
                                       (__HANDLE__)->Lock = HAL_LOCKED;  \
                                   }                                     \
                                }while (0U)

#define __HAL_UNLOCK(__HANDLE__)                                         \
                                   do {                                  \
                                       (__HANDLE__)->Lock = HAL_UNLOCKED;\
                                   }while (0U)
```

Взгляните на фрагмент из модуля SPI:

```
HAL_StatusTypeDef HAL_SPI_Transmit(SPI_HandleTypeDef *hspi, uint8_t *pData, uint16_t Size,
uint32_t Timeout) {
    // ...
    /* Process Locked */
    __HAL_LOCK(hspi);
    /* Init tickstart for timeout management*/
    tickstart = HAL_GetTick();
    if(hspi->State != HAL_SPI_STATE_READY) {
        errorcode = HAL_BUSY;
        goto error;
    }
    // ...
    if((Timeout == 0U) || ((Timeout != HAL_MAX_DELAY) && ((HAL_GetTick()-tickstart) >=
 Timeout))) {
        errorcode = HAL_TIMEOUT;
        goto error;
    }
    // ...
    /* Process Unlocked */
    __HAL_UNLOCK(hspi);
```

```
    return errorcode;
}
```

Блокировка предотвращает возможность одновременного доступа к ресурсу (в данном случае к периферийному блоку SPI). Сама функция, как можно заметить, возвращает код ошибки: если линия занята, то вернется `HAL_BUSY`, а если операция завершена успешно — `OK`.

```
typedef enum {
    HAL_OK       = 0x00U,
    HAL_ERROR    = 0x01U,
    HAL_BUSY     = 0x02U,
    HAL_TIMEOUT  = 0x03U
} HAL_StatusTypeDef;
```

Кроме всего прочего, библиотека предусматривает максимальное время работы (англ. time-out) с периферийным блоком. Если блок используется дольше определенного времени, функция завершает свою работу и возвращает код ошибки `HAL_TIMEOUT`.

Также HAL реализует механизм пользовательских обратных функций (англ. user-callback).

```
void HAL_UART_IRQHandler(UART_HandleTypeDef *huart);
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart);
void HAL_UART_TxHalfCpltCallback(UART_HandleTypeDef *huart);
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart);
void HAL_UART_RxHalfCpltCallback(UART_HandleTypeDef *huart);
void HAL_UART_ErrorCallback(UART_HandleTypeDef *huart);
void HAL_UART_AbortCpltCallback (UART_HandleTypeDef *huart);
void HAL_UART_AbortTransmitCpltCallback (UART_HandleTypeDef *huart);
void HAL_UART_AbortReceiveCpltCallback (UART_HandleTypeDef *huart);
```

Все они в файле библиотеки объявляются с «модификатором» `__weak` [2] (слабый) и могут быть переопределены разработчиком (менять их внутри библиотеки не нужно!)

```
__weak void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
{
    /* Prevent unused argument(s) compilation warning */
    UNUSED(huart);
    /* NOTE: This function Should not be modified, when the callback is needed,
    the HAL_UART_TxCpltCallback could be implemented in the user file
    */
}
```

Если необходимо произвести действие по событию завершения отправки сообщения, то функцию `HAL_UART_TxCpltCallback()` нужно поместить в файл обработчиков прерываний `stm32fxx_it.c`. Библиотека достаточно сильно абстрагирует от железа. Обязательно загляните в файлы исходного кода и ужаснитесь, какой ценой. Детальное описание библиотеки можно найти в документе **UM1850**.

Отношение к библиотеке HAL у многих разработчиков отрицательное: она очень громоздкая и запутанная. Еще и использует `goto`, что многими считается плохой практикой. Вот комментарий одного из пользователей на stackoverflow:

> My advice: forget the hal. Use bare registers instead

> Мой совет: забудь про HAL. Работай с голыми регистрами.

Если, однако, вы работаете с каким-нибудь stm32f7, у вас высокая тактовая частота и мегабайты flash-памяти, HAL можно использовать без раздумий — нужно только проникнуться ее «философией». Подключить библиотеку к проекту можно, определив макрос `USE_HAL_DRIVER` в настройках среды разработки.

Назад | **Оглавление** | **Дальше**

---

1. Си не является объектно-ориентированным языком программирования. ↩

2. Атрибут `__attribute__((weak))` позволяет сообщить компоновщику, что данная функция «слабая», т.е. имеет «меньший приоритет». Если находится такая же функция без данного атрибута, то она считается «сильной» и перезаписывает «слабую». ↩