Вячеслав Гавриков (г. Смоленск)

ШПАРГАЛКИ ДЛЯ РАЗРАБОТЧИКА: СНИППЕТЫ ДЛЯ STM32

Взаимодействие пользовательского кода с регистрами ядра и периферии микроконтроллеров STM32 *может быть осуществлено двумя способами: с помощью стандартных библиотек или с помощью наборов* ***сниппетов*** *(программных подсказок). Выбор между ними зависит от объема собственной памяти контроллера, требуемого быстродействия, срока выполнения разработки. В статье анализируются особенности структуры, достоинства и недостатки наборов сниппетов для микроконтроллеров семейств* ***STM32F1*** *и* ***STM32L0*** *производства компании* ***STMicroelectronics.***

Одно из преимуществ использования микроконтроллеров STMicroelectronics – широкий спектр средств разработки: документации, отладочных плат, программного обеспечения.

Программное обеспечение для STM32 включает в себя собственное ПО производства компании STMicroelectronics, источники Open Source, коммерческое ПО.

ПО от STMicroelectronics обладает важными достоинствами. Во-первых, оно доступно для бесплатного скачивания. Во-вторых, программные библиотеки представлены в виде исходных кодов – пользователь сам может модифицировать код, учитывая незначительные ограничения, описанные в лицензионном соглашении.

Библиотеки STMicroelectronics соответствуют ANSI-C и могут быть разделены по уровню абстракции (рисунок 1):

- CMSIS (*Core Peripheral Access Layer*) – уровень регистров ядра и периферии, ARM библиотека;
- Hardware Abstraction Layer – низкоуровневые библиотеки: стандартные библиотеки периферии (standard peripheral library), наборы сниппетов (snippets);
- Middleware – библиотеки среднего уровня: операционные системы реального времени (RTOS), файловые системы, USB, TCP/IP, Bluetooth, Display, ZigBee, Touch Sensing и другие;
- Application Field – библиотеки прикладного уровня: аудио, управление двигателями, автомобильные и промышленные решения.

На рисунке 1 видно, что для взаимодействия с уровнем CMSIS компания STMicroelectronics предлагает использовать два основных инструмента – стандартные библиотеки и сниппеты.

Стандартная библиотека – это набор драйверов. Каждый драйвер предоставляет пользователю функции и определения для работы с конкретным периферийным блоком (SPI, USART, ADC и так далее). Напрямую пользователь с регистрами уровня CMSIS не взаимодействует.

Наборы сниппетов – это высокоэффективные программные примеры, использующие прямой доступ к регистрам CMSIS. Разработчики ПО могут использовать реализации функций из этих примеров в собственном коде.

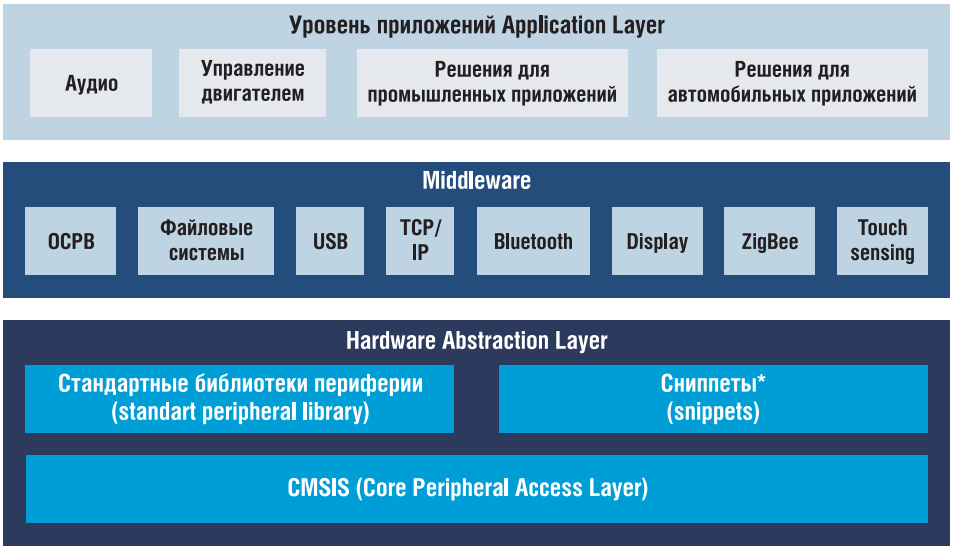
Каждый из способов имеет достоинства и недостатки. Выбор между ними делается с учетом доступного объема FLASH и ОЗУ, требуемого быстродействия, срока выполнения разработки, опытности программистов и других обстоятельств.

Уровень CMSIS

Микроконтроллер – это сложная цифро-аналоговая микросхема, состоящая из процессорного ядра, памяти, периферийных блоков, цифровых шин и так далее. Взаимодействие с каждым блоком происходит с помощью регистров.

С точки зрения программистов, микроконтроллер представляет собой пространство памяти. В нем размещены не только ОЗУ, FLASH и EEPROM, но и программные регистры. Каждому аппаратному регистру соответствует ячейка памяти. Таким образом, чтобы записать данные в регистр или вычитать его значение, программисту необходимо обратиться к соответствующей ячейке адресного пространства.

Человек имеет некоторые особенности восприятия. Например, символьные названия воспринимаются им гораздо лучше, чем адреса ячеек памяти. Это особенно заметно, когда используется большое число ячеек. В микроконтроллерах ARM число регистров, а значит, и используемых ячеек, превышает тысячу. Чтобы упростить работу, необходимо произвести определение символьных



* – В настоящий момент наборы сниппетов доступны для STM32F10 и STM32L0

Рис. 1. Структура ПО для микроконтроллеров STM32 производства STMicroelectronics

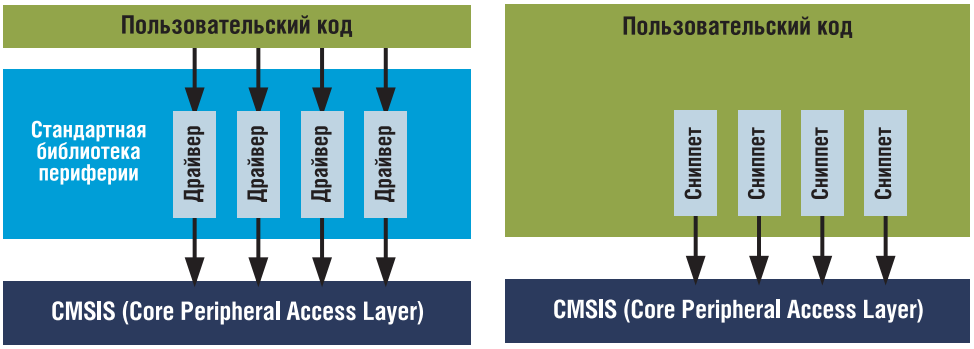


Рис. 2. **Взаимодействие с CMSIS с помощью стандартной библиотеки (а) и сниппетов (б)**

указателей. Это определение выполнено на уровне CMSIS.

Например, чтобы установить состояние выводов порта A, нужно записать данные в регистр GPIOA_ODR. Это можно сделать двумя способами – воспользоваться указателем с адресом ячейки 0xEBFF FCFF со смещением 0x14 или применить указатель с символьным названием GPIOA и готовую структуру, определяющую смещение. Очевидно, что второй вариант гораздо проще для восприятия.

CMSIS выполняет и другие функции. Он реализован в виде следующей группы файлов:

- **`startup_stm32l0xx.s`** содержит ассемблерный стартовый код Cortex-M0+ и таблицу векторов прерываний. После выполнения стартовой инициализации происходит передача управления сначала функции `SystemInit()` (ниже будут приведены пояснения), а затем – основной функции `int main(void);`
- **`stm32l0xx.h`** содержит определения, необходимые для выполнения основных операций с битами и определение типа используемого микропроцессора;
- **`system_stm32l0xx.c/.h.`** После начальной инициализации выполняется функция `SystemInit()`. Она производит первичную настройку системной периферии, таймингов блока RCC;
- **`stm32l0yyxx.h`** – файлы реализации конкретных микроконтроллеров (например, **`stm32l051xx.h`**). Именно в них определяются символьные указатели, структуры данных, битовые константы и смещения.

Взаимодействие со CMSIS. Стандартные библиотеки и сниппеты

Число регистров для микроконтроллеров STM32 в большинстве моделей превышает тысячу. Если использовать прямое обращение к регистрам, пользовательский код станет нечитаемым и абсолютно непригодным для поддержки и модернизации. Эта проблема может быть решена при использовании стандартной библиотеки периферии (standard peripheral library).

Стандартная библиотека периферии – это набор низкоуровневых драйверов. Каждый драйвер предоставляет пользователю набор функций для работы с периферийным блоком. Таким образом пользователь использует функции, а не обращается напрямую к регистрам. При этом уровень CMSIS оказывается скрытым от программиста (рисунок 2а).

Например, взаимодействие с портами ввода/вывода в STM32L0 реализовано с помощью драйвера, выполненного в виде двух файлов: **`stm32l0xx_hal_gpio.h`** и **`stm32l0xx_hal_gpio.c`**. В `stm32l0xx_hal_gpio.h` даны основные определения типов и функций, а в `stm32l0xx_hal_gpio.c` представлена их реализация.

Такой подход имеет вполне очевидные достоинства (таблица 1):

- Быстрота создания кода. Программисту не требуется изучать перечень регистров. Он сразу начинает работать на более высоком уровне. Например, для прямого взаимодействия с портом ввода/вывода в STM32L0 необходимо знать и уметь работать с одиннадцатью регистрами управления/состояния, большинство из которых имеют до 32 настраиваемых битов. При использовании библиотечного драйвера достаточно освоить восемь функций.
- Простота и наглядность кода. Пользовательский код не забит названиями регистров, может быть прозрачным и легко читаемым, что важно при работе команды разработчиков.
- Высокий уровень абстракции. При использовании стандартной библиотеки код оказывается достаточно платформонезависимым. Например, если сменить микроконтроллер STM32L0 на микроконтроллер STM32F0, часть кода, работающего с портами ввода/вывода, вообще не придется менять.

Наличие дополнительной оболочки в виде драйверов имеет и очевидные недостатки (таблица 1):

Таблица 1. **Сравнение способов реализации пользовательского кода**

Параметр сравнения	При использовании стандартной библиотеки периферии	При использовании наборов сниппетов
Размер кода	средний	минимальный
Затраты ОЗУ	средние	минимальные
Быстродействие	среднее	максимальное
Читаемость кода	отличная	низкая
Уровень независимости от платформы	средний	низкий
Скорость создания программ	высокая	низкая

Таблица 2. **Низкоуровневые библиотеки для STM32F10 и STM32L0**

Тип ПО	Фирменное название	Текущая версия
Комплекс программных средств STM32CubeF0	STM32CubeF0	1.0.0
STM32F0xx_HAL_Driver (входит в состав STM32CubeF0)		
Комплекс программных средств STM32CubeL0	STM32CubeL0	1.1.0
STM32L0xx_HAL_Driver (входит в состав STM32CubeL0)		
Сборник сниппетов для STM32F0	STM32SnippetsF0	1.0.0
Сборник сниппетов для STM32L0	STM32SnippetsL0	1.0.0

• Увеличение объема кода программы. Реализованные в библиотечном коде функции требуют дополнительного места в памяти.

• Повышенные затраты ОЗУ за счет увеличения числа локальных переменных и использования громоздких структур данных.

• Снижение быстродействия за счет увеличения накладных расходов при вызове библиотечных функций.

Именно наличие этих недостатков приводило к тому, что пользователь зачастую был вынужден оптимизировать код – самостоятельно реализовывать функции взаимодействия с CMSIS, оптимизировать библиотечные функции, убирая все лишнее, копировать реализации библиотечных функций непосредственно в свой код, использовать __INLINE-директивы для увеличения скорости выполнения. В результате, тратилось дополнительное время на доработку кода.

Компания STMicroelectronics, идя навстречу разработчикам, выпустила сборники сниппетов **STM32SnippetsF0** и **STM32SnippetsL0.**

Сниппеты входят в пользовательский код (рисунок 26).

Использование сниппетов предоставляет очевидные преимущества:

• повышение эффективности и быстродействия кода;

• уменьшение объема программы;

• снижение объемов используемой ОЗУ и нагрузки на стек.

Впрочем, стоит отметить и недостатки:

• уменьшение простоты и наглядности кода за счет «загрязнения» его названиями регистров и самостоятельной реализацией низкоуровневых функций;

• исчезновение платформонезависимости.

Таким образом, выбор между стандартной библиотекой и сниппетами не является очевидным. В большинстве случаев стоит говорить не о конкуренции, а о взаимном их использовании. На начальных этапах для быстрого построения «красивого» кода, логично использовать стандартные драйвера. При необходимости оптимизации можно обратиться к готовым сниппетам, чтобы не тратить время на разработку собственных оптимальных функций.

Стандартные библиотеки драйверов и сниппетов STM32F0 и STM32L0 (таблица 2) доступны для свободного скачивания на сайте *www.st.com.*

Более тесное знакомство со сниппетами, как и с любым ПО, следует начинать с рассмотрения особенностей лицензионного соглашения.

Лицензионное соглашение

Любой ответственный программист перед использованием сторонних программных продуктов внимательно изучает лицензионное соглашение. Несмотря на то, что сборники сниппетов производства ST Microelectronics не требуют лицензирования и доступны для свободного скачивания, это не значит, что на их использование не накладываются ограничения.

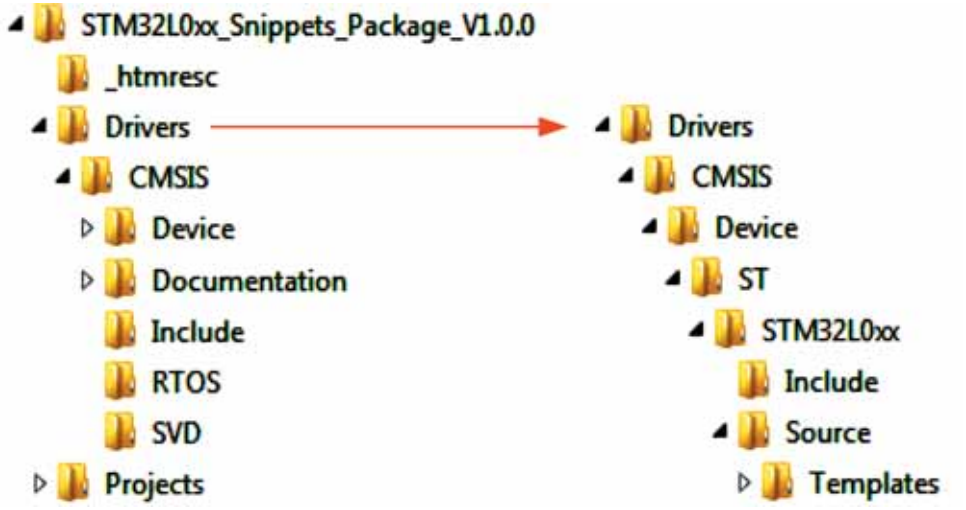


Рис. 3. **Структура сборника сниппетов STM32SnippetsL0**

Лицензионное соглашение входит в комплект всех свободно скачиваемых продуктов производства компании STMicroelectronics. После загрузки STM32SnippetsF0 и STM32SnippetsL0 в корневом каталоге легко обнаружить документ MCD-ST Liberty SW License Agreement V2.pdf, который знакомит пользователя с правилами использования данного ПО.

Пользователю STM32SnippetsF0 и STM32SnippetsL0 разрешается:

• копировать, модернизировать исходный код для создания программных продуктов, предназначенных для собственного использования;

• использовать/модернизировать исходный код для разработки и производства своих приборов;

• использовать предоставленную документацию для поддержки своих продуктов;

• производить, использовать, продавать приборы, созданные с использованием данного ПО или его модификаций.

Пользователю STM32SnippetsF0 и STM32SnippetsL0 запрещается:

• продавать, подвергать дополнительному лицензированию, сдавать в аренду предлагаемое ПО или его части;

• использовать, модернизировать предлагаемое ПО для процессоров других производителей;

• использовать предлагаемое ПО или его части в своих программных продуктах с целью дальнейшей продажи или лицензирования.

Стоит обратить внимание на пункт «NO WARRANTY», в котором четко говорится о том, что компания STMicroelectronics не гарантирует отсутствие ошибок в ПО, полную совместимость со всеми возможными конфигурациями «железа» и компиляторами. Компания не берет на себя ответственность при возникновении аварийных ситуаций, даже возникших по вине ПО.

Это значит, что пользователь обязан сам оценивать риски при использовании сниппетов и сам должен их модернизировать, если вдруг возникнут ситуации несовместимости с оборудованием или компиляторами.

Несмотря на столь угрожающие заявления, стоит отметить, что количество ошибок даже в новом ПО производства STMicroelectronics невелико. Как правило, ошибки оперативно устраняются при выходе свежих версий библиотек.

Структура наборов сниппетов STM32SnippetsF0 и STM32SnippetsL0

Рассмотрим структуру сборника сниппетов на примере STM32SnippetsL0. Набор STM32SnippetsF0 имеет схожую структуру.

После выполнения загрузки и распаковывания архива в распоряжении пользователя оказывается папка STM32L0xx_Snippets_Package_V1.0.0 (рисунок 3).

Основными подкаталогами сборника являются: Drivers\CMSIS и Project. Для STM32SnippetsF0, вместо папки Drivers\CMSIS реализована папка Library\CMSIS.

В папке Drivers\CMSIS можно найти файлы уровня CMSIS, в том числе – файлы stm32l0yyxx.h с определением символьных указателей для различных микроконтроллеров. Поддерживаются все микроконтроллеры **STM32L0: STM32L063, STM32L062, STM32L061, STM32L053, STM32L052, STM32L051.**

В папке Project содержатся подкаталоги с примерами для конкретных пери-

ферийных блоков, готовые проекты для ARM Keil и EWARM, а также файлы main.c.

Запуск и особенности использования наборов сниппетов STM32SnippetsF0 и STM32SnippetsL0

Особенностью данных наборов снипетов является их платформозависимость. Они предназначены для работы с конкретными платами. STM32SnippetsL0 использует платформу STM32L053 Discovery board, а STM32SnippetsF0 – плату STM32F072 Discovery board.

При использовании плат собственной разработки код и проекты должны быть изменены, об этом будет более подробно рассказано в последнем разделе.

Для запуска примера необходимо выполнить ряд шагов:

- запустить готовый проект из директории с требуемым примером. Для простоты можно воспользоваться готовыми проектами для сред ARM Keil или EWARM, расположенными в папке MDK-ARM\ и EWARM\ соответственно;
- включить питание отладочной платы STM32L053 Discovery/STM32F072 Discovery;
- подключить питание отладочной платы к ПК с помощью USB-кабеля. Благодаря встроенному отладчику ST-Link/V2 дополнительного программатора не потребуется;
- открыть, настроить и запустить проект;

Для ARM Keil:

– открыть проект;

– скомпилировать проект – Project → Rebuild all target files;

– загрузить его в контроллер – Debug → Start/Stop Debug Session;

– запустить программу в окне Debug → Run (F5).

Для EWARM:

– открыть проект;

– скомпилировать проект – Project → Rebuild all;

– загрузить его в контроллер – Project → Debug;

– запустить программу в окне Debug → Go(F5).

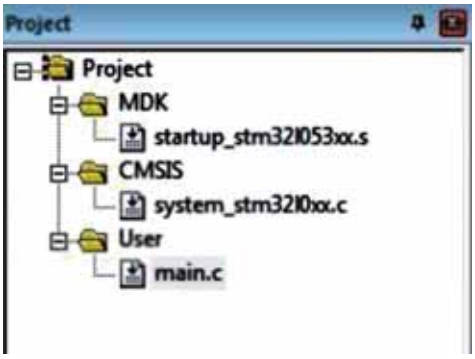


Рис. 4. **Состав стандартного проекта**

- провести тестирование в соответствии с алгоритмом, описанном в main.c.

Для анализа программного кода рассмотрим конкретный пример из STM32SnippetsL0: Projects\LPUART\01_WakeUpFromLPM\.

Запуск примера для LPUART

Отличительной особенностью новых микроконтроллеров семейства STM32L0 на ядре Cortex-M0+ является возможность динамического изменения потребления за счет большого числа нововведений. Одним из таких новшеств стало появление Low Power-периферии: 16-битного таймера **LPTIM** и приемопередатчика **LPUART.** Эти блоки обладают способностью тактирования, не зависящего от тактирования основной периферийной шины APB. При необходимости снижения потребляемой мощности рабочая частота шины APB (PCLK) может быть уменьшена, а сам контроллер переведен в режим пониженного потребления. При этом Low Power-периферия продолжает работу с максимальной производительностью.

Рассмотрим пример из директории Projects\LPUART\01_WakeUpFromLPM\, в котором рассматривается возможность независимой работы LPUART в режиме пониженного потребления.

При открытии проекта в среде ARM Keil отображаются всего три файла: `startup_stm32l053xx.s`, `system_stm32l0xx.c` и `main.c` (рисунок 4). В случае применения стандартной библиотеки в проект было бы необходимо добавить файлы драйверов.

Функционирование и анализ структуры файла Main.c

Программа из выбранного примера выполняется в несколько этапов.

После старта запускается функция `SystemInit()`, реализованная в `system_stm32l0xx.c`. Она проводит настройку параметров блока тактирования RCC (тайминги и рабочие частоты). Далее осуществляется передача управления в основную функцию `int main(void)`. В ней инициализируется пользовательская периферия – порты вводы/вывода, LPUART – после чего контроллер переводится в режим пониженного потребления STOP. В нем обычная периферия и ядро остановлены, работает только LPUART. Он ждет начала передачи данных от внешнего устройства. При приходе стартового бита LPUART пробуждает систему и принимает сообщение. Прием сопровождается мерцанием светодиода отладочной платы. После этого контроллер вновь переводится в состояние STOP и ждет следующей передачи данных, если не было обнаружено ошибок.

Передача данных происходит при помощи виртуального COM-порта и дополнительного ПО.

Рассмотрим main.c из нашего проекта. Этот файл представляет собой стандартный С-файл. Главной его особенностью является самодокументация – наличие подробных комментариев, пояснений и рекомендаций. Пояснительная часть содержит несколько разделов:

- заголовок с указанием названия файла, версии, даты, автора, краткого пояснения назначения;
- описание последовательности настройки системной периферии (RCC specific features): FLASH, ОЗУ, системы питания и тактирования, периферийных шин и так далее;
- перечень используемых ресурсов микроконтроллера (MCU Resources);
- краткое пояснение по использованию данного примера (How to use this example);
- краткое пояснение по тестированию примера и алгоритм его проведения (How to test this example).

Функция int main(void) имеет компактную форму и снабжена комментариями, которые в листинге 1, для большей наглядности, переведены на русский.

Листинг 1. Пример реализация функции main

```
int main(void)
{
 /* К началу выполнения этой части когда уже произведена конфигурация системных блоков в функции SystemInit(), реализованной в system_stm32l0xx.c. */

/* конфигурация периферийных блоков*/
 Configure_GPIO_LED();
 Configure_GPIO_LPUART();
 Configure_LPUART();
 Configure_LPM_Stop();

 /* проверка наличия ошибок при приеме */
 while (!error) /* бесконечный цикл */
```

```
 {
  /* ожидание готовности LPUART и переход в режим STOP */
  if((LPUART1->ISR & USART_ISR_REACK) == USART_ISR_REACK)
  {
   __WFI();
  }
 }
 /* при возникновении ошибки */
 SysTick_Config(2000); /* установка периода прерываний системного
таймера 1 мс */
 while(1);
}
```

В файле main.c объявлены и определены функции конфигурации периферии и две функции обработки прерываний. Рассмотрим их особенности.

В приведенном примере используются четыре функции конфигурации (листинг 2). Все они не имеют аргументов и не возвращают значений. Их главное предназначение – быстро и с наименьшими затратами занимаемого кода произвести инициализацию периферии. Это реализуется за счет двух особенностей: применения прямого обращения к регистрам и использования директивы __INLINE (листинг 3).

Листинг 2. Объявление функций конфигурации периферии

```
void Configure_GPIO_LED(void);
void Configure_GPIO_LPUART(void);
void Configure_LPUART(void);
void Configure_LPM_Stop(void);
```

Листинг 3. Пример реализации __INLINE-функции с прямым доступом к регистрам LPUART

```
__INLINE void Configure_LPUART(void)
{
 /* (1) Enable power interface clock */
 /* (2) Disable back up protection register to allow the access to
the RTC clock domain */
 /* (3) LSE on */
 /* (4) Wait LSE ready */
 /* (5) Enable back up protection register to allow the access to
the RTC clock domain */
 /* (6) LSE mapped on LPUART */
 /* (7) Enable the peripheral clock LPUART */
 /* Configure LPUART */
 /* (8) oversampling by 16, 9600 baud */
 /* (9) 8 data bit, 1 start bit, 1 stop bit, no parity, reception
mode, stop mode */
 /* (10) Set priority for LPUART1_IRQn */
 /* (11) Enable LPUART1_IRQn */
 RCC->APB1ENR |= (RCC_APB1ENR_PWREN); /* (1) */
 PWR->CR |= PWR_CR_DBP; /* (2) */
 RCC->CSR |= RCC_CSR_LSEON; /* (3) */
 while ((RCC->CSR & (RCC_CSR_LSERDY)) != (RCC_CSR_LSERDY)) /*(4)*/
 {
 /* add time out here for a robust application */
 }
 PWR->CR &=~ PWR_CR_DBP; /* (5) */
 RCC->CCIPR |= RCC_CCIPR_LPUART1SEL; /* (6) */
 RCC->APB1ENR |= RCC_APB1ENR_LPUART1EN; /*(7) */
 LPUART1->BRR = 0x369; /* (8) */
 LPUART1->CR1 = USART_CR1_UESM | USART_CR1_RXNEIE | USART_CR1_RE |
USART_CR1_UE; /* (9) */
 NVIC_SetPriority(LPUART1_IRQn, 0); /* (10) */
 NVIC_EnableIRQ(LPUART1_IRQn); /* (11) */
}
```

Обработчики прерываний от системного таймера и от LPUART также используют прямое обращение к регистрам.

Таим образом, общение с CMSIS производится без стандартной библиотеки. Код оказывается компактным и высокоэффективным. Однако его читаемость значительно ухудшатся из-за обилия обращений к регистрам.

Использование сниппетов в собственных разработках

Предложенные наборы сниппетов имеют ограничения: необходимо использовать отладочную плату STM32L053 Discovery board для STM32SnippetsL0, а плату STM32F072 Discovery board – для STM32SnippetsF0.

Для применения сниппетов в своих разработках потребуется произвести ряд изменений. Во-первых, необходимо переконфигурировать проект под нужный процессор. Для этого в нем нужно сменить стартовый файл `startup_stm32l053xx.s` на файл другого контроллера и определить нужную константу: **STM32L051xx, STM32L052xx, STM32L053xx, STM32L062xx, STM32L063xx, STM32L061xx, STM32F030, STM32F031, STM32F051** и другие. После этого при компиляции `stm32l0xx.h`, будет автоматически подключен нужный файл с определением периферии контроллера `stm32l0yyxx.h` (`stm32l051xx.h/stm32l052xx.h/stm32l053xx.h/stm32l061xx.h/stm32l062xx.h/stm32l063`). Во-вторых, нужно выбрать соответствующий программатор в настройках свойств проекта. Во-третьих – изменить код функций из примеров, если они не отвечают требованиям пользовательского приложения.

Заключение

Наборы сниппетов и стандартные библиотеки периферии производства компании ST Microelectronics не являются взаимоисключающими. Они дополняют друг друга, добавляя гибкость при создании приложений.

Стандартная библиотека дает возможность быстрого создания ясного кода с высоким уровнем абстракции.

Сниппеты позволяют повысить эффективность кода – увеличить производительность и сократить объем занимаемой памяти FLASH и ОЗУ.

Литература

1. Data brief. STM32SnippetsF0. STM32F0xx Snippets firmware package. Rev. 1. – ST Microelectronics, 2014.

2. Data brief. STM32SnippetsL0. STM32F0xx Snippets firmware package. Rev. 1. – ST Microelectronics, 2014.

3. MCD-ST Liberty SW License Agreement V2.pdfElectromechanical Relays. Technical Information. – ST Microelectronics, 2011.

4. Data brief. 32L0538DISCOVERY Discovery kit for STM32L053 microcontrollers. Rev. 1. – ST Microelectronics, 2014.

5. *http://www.st.com/*.