МЕТОДИКА СОЗДАНИЯ БИБЛИОТЕК ПОДПРОГРАММ ДЛЯ МИКРОКОНТРОЛЛЕРОВ ФИРМЫ SILABS

Олег Николайчук
onic@ch.moldpac.md

Схемотехника

Целью настоящей статьи является ознакомление читателей с методикой создания универсальных библиотек подпрограмм для микроконтроллеров фирмы Silicon Laboratories в интегрированной среде SiLabs IDE с компилятором языка C фирмы Keil.

Библиотека функций, или подпрограмм, - это набор часто применяемых, заранее составленных и отлаженных фрагментов программы, предназначенных для последующего использования в качестве целых частей при составлении новых программ. Библиотеки функций — одно из самых эффективных средств языка «С», позволяющих существенно экономить время при создании новых программ. Как правило, библиотеки подпрограмм разрабатываются за довольно длительный период времени путем постепенного добавления в библиотеку новых отлаженных модулей с соответствующим наращиванием номера версии библиотеки. Количество модулей средней библиотеки может достигать нескольких сотен, средств автоматического создания библиотек, как правило, нет даже во многих профессиональных комплектах разработки программного обеспечения и, как следствие, – создание новой версии библиотеки требует довольно значительных затрат времени. Поэтому разработчики программного обеспечения, как правило, создают для каждого компилятора свои собственные методики и соответствующие программные средства для облегчения процесса создания библиотек.

Существует насколько подходов к созданию исходных модулей текстов подпрограмм.

Согласно одному из них в библиотеку включают один общий модуль, содержащий все подпрограммы, реализующие те, или иные функции, необходимые для работы с выбранным объектом. Этот подход является самым простым с точки зрения процесса подготовки исходных модулей. Однако недостатком такого подхода является то, что при использовании в основной программе только части подпрограмм (функций) из этого модуля, к основной программе подключается весь модуль кодов, в том числе и кодов неиспользованных функций, а также, при компиляции выдаются предупреждения о наличии таких неиспользованных функций.

Второй подход заключается в том, что количество исходных модулей создается по числу имеющихся подпрограмм. Это самый громоздкий подход, поскольку при реализации, например, некоторых интерфейсов, возможно очень большое количество входящих подпрограмм. Соответственно будет достаточно большое количество и модулей исходных текстов. С другой стороны, такой подход позволяет создавать наиболее компактные по размеру коды основной программы, т.к. к ней из библиотеки подключаются только модули используемых подпрограмм.

Третий и наиболее часто используемый подход является компромиссным. Он заключается в том, что среди подпрограмм, которые должны быть включены в библиотеку, выделяются так называемые подпрограммы «нижнего уровня», которые будут использоваться в любом случае при использовании хотя бы одной из остальных подпрограмм - так называемого «верхнего уровня». Подпрограммы «нижнего уровня» размещаются в общем «базовом модуле» исходных текстов. Подпрограммы «верхнего уровня» размещаются каждая в индивидуальном модуле. При правильном определении подпрограмм «нижнего уровня» удается достичь таких же результатов по объему выходного кода, как и при втором подходе.

Обычно объявления для всех подпрограмм, которые могут быть использованы для работы с выбранным объектом, включаются в отдельный файл объявлений (заголовков). В свою очередь все файлы заголовков включаются в общий файл заголовков библиотеки.

Процесс создания библиотек для микроконтроллеров фирмы Silicon Laboratories (SiLabs) с помощью компилятора Keil C отличается от аналогичных процессов для других микроконтроллеров и с помощью других компиляторов. Это обусловлено следующими причинами:

1. Микроконтроллеры фирмы Silicon Laboratories (SiLabs) подразделяются на ряд семейств, имеющих различные наборы периферии. Кроме того, внутри каждого семейства периферийные узлы не имеют фиксированных выводов корпуса для линий ввода/вывода. В этих микроконтроллерах используется «плавающий» приоритетный механизм ассоциации линий ввода/вывода встроенной периферии на ряд свободных выводов корпуса – “crossbar”. Причем эти механизмы отличаются для различных семейств микроконтроллеров. Это обстоятельство приводит к тому, что при создании универсальных библиотек в состав библиотечных функций не могут быть включены подпрограммы, явно использующие линии ввода/вывода.
2. Второе ограничение накладывает компилятор языка C фирмы Keil. Это ограничение заключается в том, что объявления линий ввода/вывода типа sbit могут находиться только в том файле, в котором они используются. Это означает, что если в библиотечные файлы включены объявления конкретных линий ввода/вывода типа sbit, то они будут справедливы лишь для одной частной конфигурации микроконтроллера, но не будут универсальными ни для данного микроконтроллера (ввиду наличия механизмов crossbar), ни для всех семейств (ввиду различий в архитектуре и составе периферии).
3. Третье ограничение также накладывается компилятором языка C фирмы Keil. Это ограничение заключается в том, что компилятор не имеет расширенной командной строки, позволяющей производить компиляцию ряда исходных файлов библиотеки с одинаковыми параметрами. Например, нельзя откомпилировать все файлы *.c находящиеся в одной директории. Это приводит к тому, что для каждого исходного файла необходимо указывать индивидуальную строку для вызова компилятора, что значительно усложняет процесс создания библиотеки.

Однако существует механизм, позволяющий создавать и использовать универсальные библиотеки функций для всех семейств микроконтроллеров. Суть этого механизма заключается в том, что при написании программы с использованием библиотек в проекте создается специальный файл, например, «config_io.c», в котором размещаются и все объявления типа sbit, используемые в данном проекте, и все функции нижнего уровня, непосредственно работающие с линиями ввода/вывода, объявленными как sbit. Библиотечные же функции не содержат объявлений типа sbit, иными словами – не связаны с конкретной архитектурой микроконтроллера, а используют уже функции нижнего уровня, имеющиеся в исходном файле «config_io.c».

Кроме этого, в вышеуказанном файле с условным названием «config_io.c» следует располагать также подпрограммы конфигурации механизма crossbar, а также и другие подпрограммы общего назначения, например, функции программной задержки и функции перезапуска охранного таймера WDT.

В качестве примера рассмотрим использование библиотечного интерфейса для алфавитно-цифрового LCD (ЖКИ) индикатора. Допустим, что в качестве шины данных индикатора мы собираемся использовать порт P2 микроконтроллера C8051F021, а в качестве управляющих линий выборки индикатора EN, выбора режима чтения или записи RW и выбора типа операции (команда или данные) DC – мы будем использовать три свободные

линии порта P3. В этом случае файле проекта с условным названием «config_io.c» следует сделать следующие объявления:

```
sbit    EN      = P3^0;
sbit    RW      = P3^1;
sbit    DC      = P3^2;

#define DATABUS     P2
#define DATABUS_CF  P2MDOUT
```

Функции нижнего уровня для управляющих сигналов и функции ввода/вывода байта данных будут выглядеть следующим образом:

```
void LCD_EN (bit B )    {EN=B;}
void LCD_RW (bit B )    {RW=B;}
void LCD_DC (bit B )    {DC=B;}
bit INT0 (void)         {return INT0_Bit;}
//*************************************************************
byte Get_Data (void)
{
        DATABUS_CF=0;       // Режим с открытым истоком
        DATABUS=0xFF;       // Данные на ввод
        return DATABUS;     // Взять данные
}
//*************************************************************
void Set_Data (char CH)
{
        DATABUS_CF=0;       //  Режим с открытым истоком
        DATABUS=CH;         //  Поместить байт данных CH в порт
}
//*************************************************************
```

Кроме того, как было указано выше, как правило все библиотечные функции в той или иной мере используют функции программных задержек и перезапуска охранного таймера. Эти функции также следует располагать в файле с условным названием «config_io.c». Очевидно, что программные задержки сильно зависят от тактовой частоты микроконтроллера, поэтому их придется корректировать. Примерный вид вышеуказанных функций приведен ниже:

```
extern long     SYSCLK;

void WDT (void)
{
        WDTCN=0xA5;
}
/*********************************************************************/
void Time (unsigned mkS)
{
        if (SYSCLK==1105900) if (mkS>8)  mkS -= 7;
        if (SYSCLK==22118400)            mkS *= 2;
        while (mkS--) WDT();
}
/*********************************************************************/
void Delay (unsigned mS)
{
        while(mS--) {Time(1000);}
}
```

Следует отметить, что функция Time не генерирует точную задержку mkS. Внешняя переменная SYSCLK назначается равной частоте тактового генератора контроллера в подпрограммах инициализации тактовых генераторов, и в данном случае, используется для коррекции приблизительного времени задержки Time(). Если в проекте не используется охранный таймер, то его функцию также следует откорректировать, например, закрыв в комментарии выражение /* WDTCN=0xA5 */. Приведенная процедура перезапуска охранного таймера справедлива для полноформатных семейств микроконтроллеров, имеющих аппаратный охранный таймер WDT, например для C8051F02x.

Как уже указывалось выше, компилятор языка «C» фирмы Keil, к сожалению, имеет весьма ограниченные возможности по обработке исходных файлов из командной строки. Иными словами, этот компилятор не может обработать насколько указанных с помощью маски имен исходных «C» файлов с одинаковыми прочими параметрами. Это не позволяет произвести компиляцию всех исходных «C» файлов, находящихся в отдельной директории, например, для одной избранной модели памяти, и получить соответствующее число объектных файлов, необходимых для создания библиотеки. В свою очередь это означает, что пользователям этого компилятора не удастся создать единый *.bat файл для компиляции всех исходных кодов и генерации библиотеки, как это делалось, например, в популярном компиляторе языка Borland C.

Для решения описанной выше проблемы пользователям компилятора языка «C» фирмы Keil, работающим в среде «Silicon Laboratories (SiLabs) IDE», предлагается воспользоваться следующей методикой.

Компиляция всех модулей всех исходных текстов. Для компиляции всех исходных текстов разработчик «SiLabs IDE» обычно рекомендует создать в окне «Project Windows» новый проект «New_Project», а затем добавлять к нему все необходимые в проекте исходные файлы последовательным нажатием правой кнопки мыши с выбором функции «Add Files to project New_Project». Затем для включения компиляции каждого файла необходимо установить на его имя в окне «Project Windows» указатель мыши и опять нажать правую кнопку мыши с выбором функции «Add FileName.c to build». Очевидно, что такая последовательность действий применима для проектов, в которых число исходных файлов редко превышает один десяток. Но при большом количестве файлов такая последовательность действий явно не является оптимальной.

Оптимален другой путь, подразумевающий, что исходные файлы библиотеки, как правило, располагаются в одной директории. В этом случае в проект описанным выше способом включается хотя бы один исходный файл. Для включения всех остальных исходных текстов в проект необходимо войти в меню «Project -> Target Build Configuration…», затем установить опцию «Define Build Process» и нажать кнопку «Customize». Далее в появившемся новом окне выбрать закладку «Files to Compile» и нажать кнопку «Add all C files» (см. рис. 1). Такая последовательность действий подключит все исходные «C» файлы библиотеки, находящиеся в одной директории для компиляции. Аналогично можно ассемблировать и все имеющиеся ассемблерные исходные файлы.

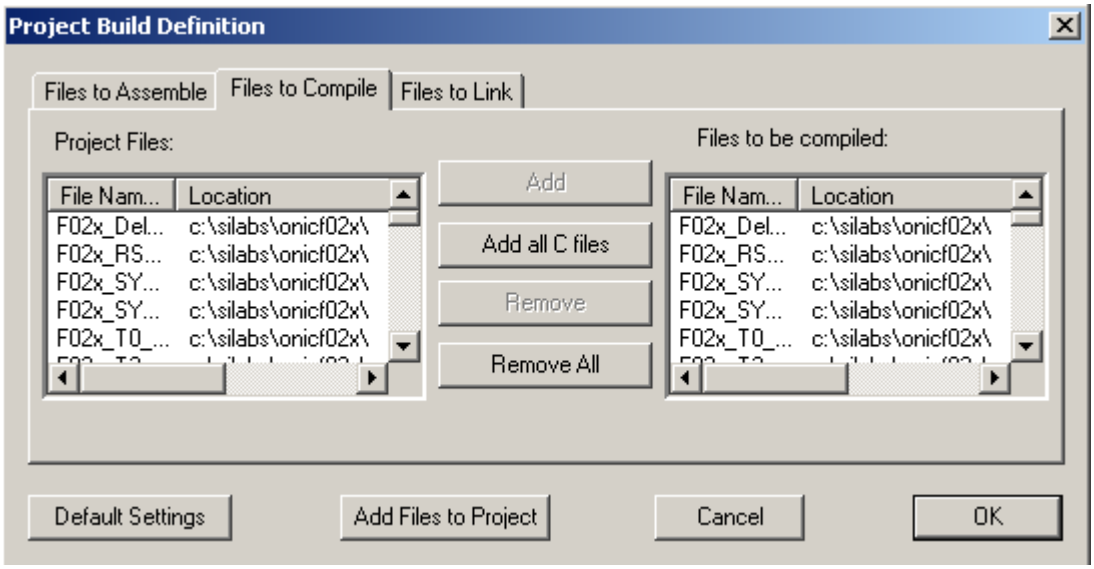

Рис.1. Подключение к проекту всех исходных «С» файлов библиотеки

Еще один вопрос, который нам необходимо рассмотреть, это вопрос об оптимальной настройке компилятора языка «С» фирмы Keil.

Во-первых, отметим, что компилятор языка «С» может обеспечить компиляцию исходных текстов в одной из трех моделей памяти: Small, Compact и Large.

Малая модель памяти – Small – генерирует самый компактный код, т.к. использует самые короткие вызовы. Это обусловлено тем, что все переменные по умолчанию располагаются во внутренней памяти – internal RAM. Стек также располагается во внутренней памяти, и поскольку объем этой памяти очень ограничен, написание программ в этой модели памяти накладывает определенные ограничения на количество используемых переменных и глубину стека.

Средняя модель памяти – Compact – характеризуется тем, что все переменные по умолчанию располагаются в одной странице внешней памяти данных. Объем всех переменных в данной модели не должен превышать 256 байтов. Поскольку обращение к внешней странице памяти данных в этом случае производится с помощью команд косвенной адресации (через регистры R0 и R1 - @R0, @R1), размер выходного кода несколько больше, чем в предыдущей модели.

Большая модель памяти – Large – характеризуется тем, что все переменные по умолчанию располагаются во внешней дополнительной памяти данных, объем которой может достигать 64 Кбайта. Обращение к внешней дополнительной памяти данных в данном случае производится с помощью указателя данных DPTR. Это самый громоздкий механизм адресации и он естественно генерирует самый большой код.

Совершенно очевидно, что при написании программы все ее фрагменты должны компилироваться в одной из вышеперечисленных моделей памяти. Это утверждение касается, в том числе, и библиотечных модулей. Поэтому, если разработчик планирует написание программы в одной из моделей памяти, библиотеки, которые он собирается использовать, также должны быть ориентированы на эту же модель памяти.

Во-вторых, необходимо напомнить, что модули, которые будут включаться в библиотеку, необходимо генерировать с выключенной опцией включения отладочной информации. Это позволит сократить размер выходного кода функций.

Все перечисленные выше настройки должны быть указаны в меню «Project ->Tool Chain Integration…». В появившемся окне необходимо выбрать производителя в выпадающем окне «Select Tool Vendor -> Keil», затем необходимо выбрать закладку «Compiler», нажать кнопку «Customize». После этого появится окно настройки компилятора, показанное на рис.2.

Именно в этом окне и нужно выбрать необходимую модель памяти для вашей библиотеки и отменит включение отладочной информации.

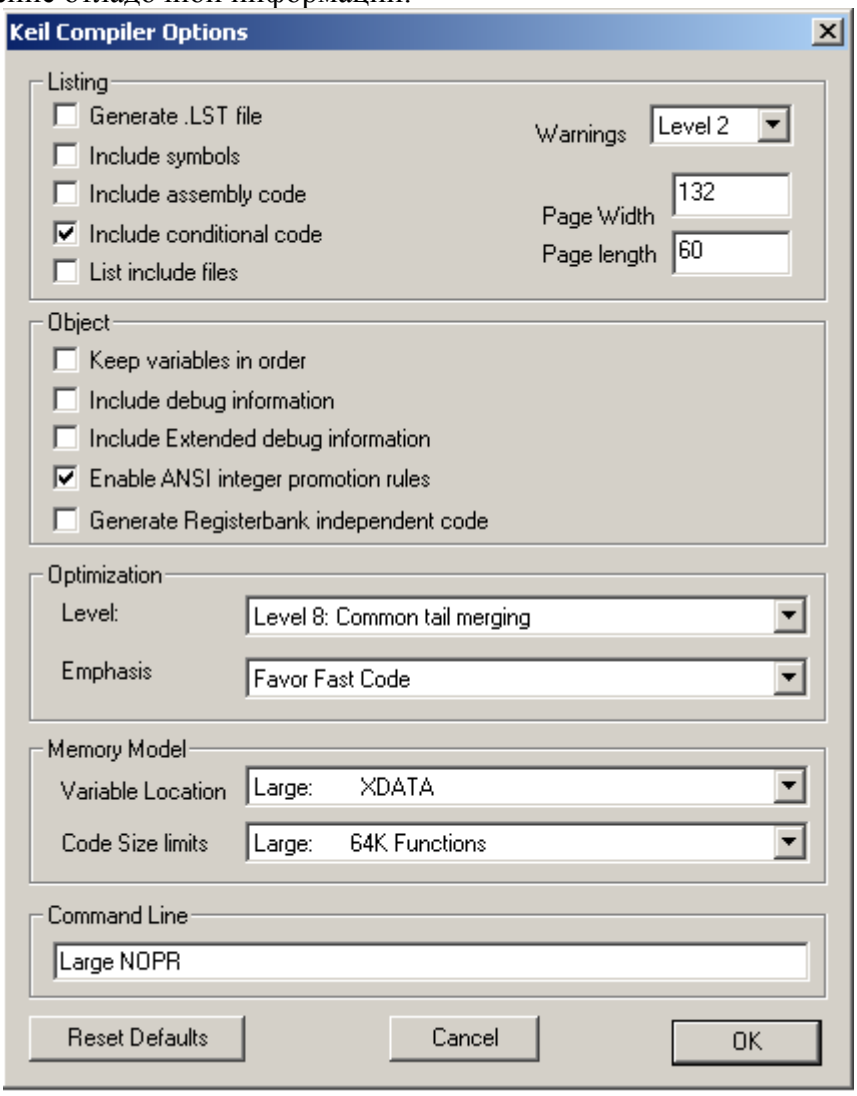

Рис.2. Окно настройки компилятора Keil

После того, как все исходные файлы включены в проект создания библиотеки, и все необходимые настройки компилятора произведены, необходимо произвести компиляцию всех исходных «С» текстов. Это делается обычным способом – путем нажатия кнопки «Build/Make Project» на панели «Project Window». После компиляции всех исходных файлов с помощью компилятора, оболочка SiLabs IDE попытается вызвать линкер для связки проекта, и естественно выдаст сообщение об ошибках, так как в проекте отсутствует функция main, а многие из библиотечных функций окажутся неиспользуемыми. На это не стоит обращать внимание. Для нас в данном случае главное, что все исходные тексты компилированы и соответствующие объектные коды сгенерированы. Таким образом, мы выполнили первую половину задачи.

Вторая половина задачи состоит в том, чтобы включить все полученные выше объектные файлы в единый файл библиотеки. Делается это обычно с помощью пакетного файла. Мы разберем этот файл построчно. Допустим, что мы будем генерировать библиотечный файл MyLib.lib.

В первой строке необходимо удалить старую версию библиотеки, если она имеется в рабочей директории, а также файл листинга, если он имеется:

```
DEL MyLib.l*
```

Далее мы создаем с помощью библиотекаря Lib51 новую пустую библиотеку:

```
LIB51 CREATE MyLib.lib
```

Далее мы должны описать включение с помощью команды ADD (добавить) каждого из имеющихся объектных модулей (с расширением .obj) в состав библиотеки:

```
LIB51 ADD F02x_CTime.OBJ      to MyLib.lib
LIB51 ADD F02x_Delay.OBJ      to MyLib.lib
….
```

После этого библиотека является сформированной. Теперь следует создать листинг библиотеки MyLib.lst содержащий названия всех функций, включенных в библиотеку:

```
LIB51 LIST MyLib.lib to MyLib.lst PUBLICS
```

Обычно исходные тексты библиотеки размещаются в директории, вложенной в директорию оболочки SiLabs IDE. В нашем случае это будет директория C:\SiLabs\MyLib\. В этой директории будут располагаться все исходные «C» и «Asm” модули, и соответствующие файлы заголовков. Чтобы не засорять эту директорию и облегчить работу в ней, после построения библиотеки следует убрать все сгенерированные объектные файлы и файлы листингов модулей во вложенные директории, например, \LIST и \OBJ, а затем удалить из библиотечной директории объектные файлы. Это достигается следующими тремя командами:

```
COPY *.LST LIST\*.LST
COPY *.OBJ OBJ\*.OBJ
DEL *.OBJ
```

Полученные библиотеку, файлы заголовков и листинг библиотеки обычно перемещают в отдельную директорию, например, C:\SiLabs\My_LIBS\, из которой эту библиотеку потом и используют. После этого удаляются все листинги. Достигается это следующими командами:

```
COPY *.H ..\My_LIBS\*.H
COPY *.LIB ..\My_LIBS\*.LIB
COPY MyLib.LST ..\My_LIBS\MyLib.LST
DEL *.LST
```

Еще раз напомним, что все эти команды записываются последовательно в текстовый пакетный файл, например, MyLib.bat.

Если вы предвидите возможность одновременной работы в разных моделях памяти, вы можно сгенерировать библиотечные файлы сразу для всех моделей. В этом случае рекомендуется дополнять имя библиотеки символом соответствующей модели, например, MyLibS (для Small), MyLibC (для Compact) или MyLibL (для Large).

В заключение еще раз напомним последовательность действий по созданию библиотеки:

1. Разрабатываются и отлаживаются все модули исходных текстов для включения в библиотеку.
2. Разрабатываются файлы объявлений для каждой группы модулей исходных текстов, объединенной по признаку общности программного или

аппаратного объекта, с которым они взаимодействуют. Файлы объявлений включают объявления всех функций, содержащихся в выделенной группе модулей, включают объявления всех переменных, используемых данной группой функций, а также объявления всех внешних функций и переменных. Кроме этого, разрабатывается общей библиотечный файл объявлений, включающий файлы объявления всех групп модулей.

3. Производится создание проекта библиотеки, т.е. настройка компилятора языка «С» (или компиляторов, например, если используется еще и ассемблер) на компиляцию всех исходных файлов библиотеки.
4. Производится настройка компилятора языка «С» (или компиляторов) на компиляцию исходных текстов в выбранной модели памяти без включения в объектные модули отладочной информации.
5. Производится компиляция всех модулей исходных текстов, необходимых для создания библиотеки.
6. Создается пакетный файл для включения всех объектных модулей в библиотеку. Пакетный файл сперва удаляет старую версию библиотеки, если она уже существует, удаляет старые листинги объектных файлов и библиотеки, создает новую библиотеку, включает в нее все необходимые объектные файлы, создает листинг библиотеки и помещает все использованные в предварительно определенные директории.

В заключение следует отметить, что при создании любой библиотеки вне зависимости от целей, для каких она разработана (для коммерческих целей или для личного использования), необходимо создать файл с текстовыми описаниями каждой из включенных в библиотеку функций, а также ее входных и выходных параметров.

Литература:

1. http://www.silabs.com/
2. http://www.keil.com/