МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
ВОРОНЕЖСКАЯ ГОСУДАРСТВЕННАЯ ТЕХНОЛОГИЧЕСКАЯ АКАДЕМИЯ

КАФЕДРА ИНФОРМАЦИОННЫХ И УПРАВЛЯЮЩИХ СИСТЕМ

СОЗДАНИЕ ПОЛЬЗОВАТЕЛЬСКИХ ФУНКЦИОНАЛЬНЫХ БЛОКОВ ПРОГРАММИРОВАНИЕМ НА СИ++

Методические указания по выполнению лабораторной работы дисциплины
«Интегрированные системы проектирования и управления»

Для бакалавров, обучающихся по направлениям 220700 - Автоматизация технологических процессов и производств и 220400 – Управление в технических системах

Дневной и заочной формы обучения

ВОРОНЕЖ 2011

УДК

Создание пользовательских функциональных блоков программированием на СИ++: Методические указания для выполнения лабораторной работы по дисциплине «Интегрированные системы проектирования и управления» / Воронеж. гос. технол. акад.; Сост. И.А. Хаустов, А.А. Хвостов. Воронеж, 2011. 14 с.

Указания разработаны в соответствии с требованиями ООП подготовки бакалавров по направлениям 220700 – «Автоматизация технологических процессов и производств» и 220400 – «Управление в технических системах». Методические указания посвящены обучению навыкам создания и подключения пользовательских функциональных блоков программированием на языке СИ++ инструментальными пакетами BILDER и Visual C.

Ил. . Библиогр.: назв.

Составитель доценты И.А. ХАУСТОВ и А.А. ХВОСТОВ,

Научный редактор профессор, д.т.н. В.Ф. ЛЕБЕДЕВ

Рецензент профессор, д.т.н. Ю.А. ЧЕВЫЧЕЛОВ

Печатается по решению
редакционно-издательского совета
Воронежской государственной технологической академии

Цель работы

Получение навыков создания пользовательских функциональных блоков программированием на СИ++ в т. ч.:

- программирование и компиляция DLL библиотек в среде BILDER C++ или Visual C++;
- подключение и использование созданных DLL модулей в качестве функциональных блоков в алгоритмах обработки данных и управления.

Постановка задачи

1. Создать DLL библиотеку, реализующую одну или несколько функций обработки данных или управления в соответствии с выбранным вариантом задания.
   1.1. Составить математическую формулировку, спецификацию входов и выходов функционального блока
   1.2. Разработать алгоритм программы, реализующий поставленную задачу.
   1.3. Написать программу в среде BILDER C++ или Visual C.
   1.4. Скомпилировать программу в качестве DLL модуля.
2. Произвести подключение созданного модуля и тестирование с целью выявления и устранения ошибок функционирования.
3. Оформить отчет.

Краткие теоретические сведения

Динамически присоединяемые библиотеки DLL

Динамически присоединяемая библиотека DLL — это одна из возможностей повторного использования разработанных кодов. DLL — это специального вида исполняемый файл с расширением .dll, используемый для хранения функций и ресурсов отдельно от исполняемого файла. Обычно, когда пишется программа и создаются функции,

ресурсы и т.п., все они компонуются в исполняемый файл. Это называется *статическим связыванием*. Когда же подключается DLL, то вызываемые из нее функции используются вашим модулем в процессе его выполнения. DLL делает полезные, часто используемые функции доступными сразу для многих приложений одновременно, хотя работа ведется только с одной ее копией на диске и в памяти. Обычно DLL не загружается в память, пока это не станет необходимым, но, будучи однажды загружена, она делает свои функции и ресурсы доступными для любой программы.

Предположим, что различные полезные процедуры и функции необходимо использовать в различных приложениях. Конечно, можно просто скопировать эти процедуры и функции в каждый проект. Но тогда, если несколько приложений работает на компьютере одновременно, то копии процедур загружены в память в выполняемом модуле каждого приложения и память, таким образом, расходуется неэффективно. Да и затраты пространства на дисках также избыточны, поскольку эти процедуры в составе выполняемых модулей также тиражируются. Если же процедуры размещены в DLL, то все эти проблемы снимаются.

Создание DLL повышает также гибкость программы. Например, можно создать несколько библиотек, содержащих всё используемые приложением тексты — надписи, подсказки и т.п. Каждая из этих библиотек может содержать тексты на том или ином языке: русском, английском, немецком. Тогда в зависимости от того, какую из этих библиотек будет применять пользователь, программы будут общаться с ним на том или ином языке.

Еще одним преимуществом DLL является то, что они могут использоваться приложениями, написанными на других алгоритмических языках. Например, можно использовать библиотеки, написанные на Object Pascal,

Visual Basic, Access Basic и др. А библиотеки, созданные в C++Builder, смогут использоваться в любых системах, которые умеют вызывать DLL, независимо от того, на каких алгоритмических языках они написаны.

Статическое и динамическое связывание DLL с приложением

Библиотеки DLL могут связываться с приложением двумя путями: *статическим связыванием* или *динамическим связыванием.*

Статическое связывание означает, что DLL загружается сразу, как только начинает выполняться приложение, которое будет ее использовать. Это наиболее простой способ использования DLL. Вызов функций в приложении, использующем подобную DLL, почти не отличается от вызова любых других функций. Некоторыми недостатками такого подхода можно считать увеличение времени загрузки приложения (ведь кроме выполняемого модуля приложения в этот момент грузятся и модули соответствующих DLL) и невозможность выполнения приложения пользователем, у которого нет соответствующего файла DLL. Последнее трудно назвать существенным недостатком, поскольку, конечно, необходимо распространять приложение вместе с DLL. Но некоторые действия приложение могло бы делать и без DLL, т.е. оно могло бы в урезанном виде функционировать и при отсутствии DLL. Однако при статическом присоединении это невозможно. Еще одним недостатком статического присоединения является то, что DLL занимает память все время, в течение которого выполняется приложение, независимо от того, вызываются ли в данном сеансе работы с приложением какие-то функции библиотеки, или нет.

Статическое связывание подразумевает, что для DLL создан специальный файл описаний импортируемых функций (import library file). Этот файл имеет то же имя,

что и соответствующая DLL, и расширение **.lib.** Этот файл **.lib** должен быть связан с вашим приложением на этапе компиляции.

Динамическое связывание отличается от статического тем, что библиотека DLL загружается только в тот момент, когда необходимо выполнить какую-то хранящуюся в ней функцию. Затем эту библиотеку можно выгрузить из памяти. Это обеспечивает, конечно, более эффективное использование памяти. Но зато вызов соответствующих функций библиотеки существенно усложняется, да и время вызова тоже увеличивается из-за необходимости загружать и выгружать библиотеку.

Целью РГР не является получение навыков вызова библиотечных функций, однако для представления рассмотрим пример.

Для вызова библиотечной функции из библиотеки, связываемой динамическим образом, надо прежде всего загрузить библиотеку с помощью функции **LoadLibrary**. Затем с помощью функции API Windows **GetProcAddress** надо получить указатель на интересующую функцию библиотеки. Только после этого можно выполнять функцию. А затем с помощью функции **FreeLibrary** надо выгрузить библиотеку из памяти.

Пусть, например, создали библиотеку **mydll.dll,** содержащую некоторую функцию **char * MyFunction(char** *). Тогда для загрузки DLL вам надо выполнить оператор вида:

```
// загрузка DLL
HINSTANCE dllinstance = LoadLi-
brary("mydll.dll");
```

Получить указатель на импортируемую функцию можно, например, следующим кодом:

```
// получение указателя на функцию
typedef char (__import * FType(char *));
FType * MyFunc;
```

```
MyFunc = (FType *)
GetProcAddress(dllinstance,"_MyFunction");
```

Объявление **typedef** вводит некоторый тип **FType** (это произвольное имя). В объявлении сначала указывается тип возвращаемого значения, а в круглых скобках после имени типа перечисляются типы аргументов. Введенный тип используется для задания типа указателя на функцию **MyFunc.** Для получения значения этого указателя используется функция API Windows **GetProcAddress.** Она принимает в качестве параметров указатель на загруженный модуль DLL и имя функции, а возвращает указатель на функцию. Этот указатель надо привести к типу используемой функции. Для приведения типа используется объявленный тип **FType.**

Вызов функции осуществляется с помощью указателя на нее:

```
    // вызов функции
char * S = MyFunc("Привет!");
```

Когда работа с DLL завершена, ее можно выгрузить из памяти оператором вида:

```
    // выгрузка DLL
FreeLibrary(dllinstance);
```

Все это достаточно громоздко. Так что если только нет острой необходимости использовать именно динамическое связывание, лучше всегда использовать статическое связывание.

Создание DLL в среде BILDER C++

Создание DLL начинается с выполнения команды File | New | Other и выбора в окне New Items на странице New пиктограммы DLL Wizard — Мастера DLL (рис. 1). В нем выбирается язык DLL — С или С++. Индикатор Use VCL позволяет создавать DLL, которая может содержать компоненты библиотеки VCL (в случае использования классов, форм, функ-

ђqddqѣеЎ bЁ dәdЁbЎзә d ɛɔɑbɔѣ Ɉqddʎqѣè

ций, связанных с библиотекой визуальных компонентов). При этом в модуль включится файл VCL.h и установятся опции компоновки, обеспечивающие совместимость с объектами VCL. Индикатор Multi Threaded позволит работать с несколькими потоками (нитями) выполнения. А индикатор VC++ Style DLL обеспечит создание DLL в стиле Microsoft Visual C++

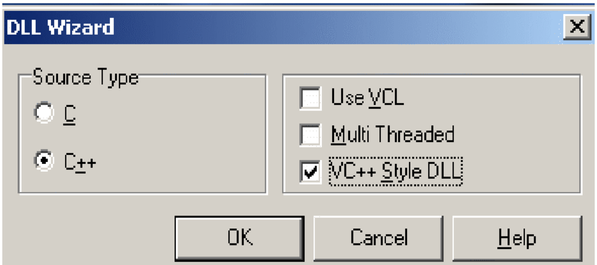

Рис. 1

После установки опций опции C++ и VC++ Style DLL и щелчка на OK запускается Редактор Кода, в котором появится следующий текст и коментарии:

```
int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason, void* lpReserved)
{return 1;}
```

Функция **DllEntryPoint,** необходимая для загрузки и выгрузки библиотеки. Она создает дескриптор **hinst** DLL. Он требуется при выполнении некоторых функций, например, **LoadIcon, LoadCursor** и др. В этих случаях вы можете используется параметр **hinst** для создания соответствующей глобальной переменной. Поскольку такие функции не предусмотрены, то в использовании функции **DllEntryPoint** нет необходимости.

Порядок выполнения

Итак, рассмотрим пример создания FBD-модуля с тремя входами и одним выходом. На его выходе форми-

руется сумма первых двух входов умноженная на третий вход.

Шаг 1. На первом этапе следует составить математическую формулировку задачи и спецификацию функционального блока.

$$RES=(A0+A1)*A2$$

Если задача состоит в создании алгоритма управления , то математическая формулировка задачи не требуется.

Спецификация входных и выходных переменных функционального блока

Идентификатор переменной	Тип	Краткое название	Назначение
in[0]	Вход	A0	Контролируемая величина на первом входе
in[1]	Вход	A1	Контролируемая величина на втором
in[2]	Вход	A2	Контролируемая величина на третьем
out[0]	Выход	RES	Выходная величина, равная *(A0+A1)*A2*

Шаг 2. Составим алгоритм работы программы.

Шаг 3. Создадим DLL библиотеку, функции которой реализуют поставленную задачу

1. Выполним команду File | New | Other и выберем в окне New Items на странице New пиктограмму DLL Wizard. Сделаем в диалоговом окне установки, показанные на рис. 1.
2. Удалим из текста комментарий и функцию **DllEntryPoint.**
3. Составим программу, реализующую поставленную задачу. Текст программы рассмотрим подробнее.

```
#include <string.h>

// Объявление функций
// FBD_DLL – предназначена для реализации кода FBD модуля
```

```
// zFBD_DLL – формирует указатель на описание FBD-модуля
// zFBD_DLL_varname – предназначена для формирования
// указателя на имена входов и выходов.

__declspec(dllexport) int FBD_DLL( float*, float*, int*, int* );

__declspec(dllexport) void zFBD_DLL_info( void* );

__declspec(dllexport) void zFBD_DLL_varname( char* );
```

Конструкция __declspec(dllexport), используемая при объявлении функций означает, что функция может экспортироваться из библиотеки, то есть может вызываться внешними приложениями. Подобным образом должны быть перечислены все функции DLL, предназначенные для прямого использования в приложениях. Помимо таких функций в DLL могут быть вспомогательные функции-утилиты, предназначенные только для использования другими функциями. Подобные утилиты не должны определяться как экспортируемые.

```
// Структура описывающая FBD-модуль

typedef struct
{
  int q_in;         // Количество входов
  int q_out;        // Количество выходов
  int q_int;        // Зарезервировано
  int type;         // Номер в списке функциональных разделов.
  char name[8]; // Короткое имя для вывода на экран
  char fname[8];// Полное имя
}DO_DEFAULT;

 // Тело FBD-модуля
int FBD_DLL(float *in,          // Указатель на массив входов
            float *out,         // Указатель на массив выходов
            int *I,             // Зарезервировано
            int *rl )           // Зарезервировано
{
  out[0] = (in[0] + in[1])* in[2];
```

```
  return 0;
}

// Формирование указателя на описание FBD-модуля
void zFBD_DLL_info( void *buf )
{
  DO_DEFAULT *dd;
  dd=(DO_DEFAULT *)buf;
  strcpy( dd->name, "fbd0" );   // краткое название модуля
  strcpy( dd->fname, "fbd0" );  // полное название модуля
  dd->q_in   = 4;  //Установим количество входов. В блоке
                   //необходимо предусмотреть наличие входа
                   //блокировки. Для этого количество входов надо
                   //указать на один больше, чем требуется.
  dd->q_out = 1;  // Установим количество выходов
  dd->type   = 17;//Номер в списке функциональных разделов.
                   // Должно быть равно 17
  dd->q_int = 0;   // Количество зарезервированных переменных
}

// Формирование указателя на имена входов и выходов
// на каждое имя отводится 8 байт
void zFBD_DLL_varname( char *varname )
{
  strcpy( varname+0, "RUN" );//вход запуска, в области памяти,
                             //на имя этого входа отводится 8
                             //байт, однако оно не отображается
                             //при редактировании
                             //FBD программы
  strcpy( varname+8, "A0" );    // первый вход
  strcpy( varname+16, "A1" );  // второй вход
  strcpy( varname+24, "A1" );  // третий вход
  strcpy( varname+32, "RES" );// выход
}
```

4. Сохраните проект — под именем **FBDn.** (**n** – номер в списке от 0 до 9)
5. Выполните команду Project | Build. В результате будут созданы файлы **FBDn.dll** и **FBDn.lib.** Ес-

ли выполнить команду Run | Run (F9), то получим сообщение: «Cannot debug project unless a host application is defined. Use Run | Parameters... dialog box». Это означает, что вы сначала с помощью команды Run | Parameters нужно определить хост тестирующего приложения. Однако это делать необязательно.

6. Созданная DLL библиотека по умолчанию разместится в директории BIN C++Builder. Ее следует поместить в директорию запуска MPB.

Шаг 4. Подключение программы.

Запустим редактор базы каналов и загрузим проект. Перейдем в окно редактирования FBD-программ и в диалоге **Меню FBD** выбрать раздел DLL, то в нем будет присутствовать только что созданный блок.

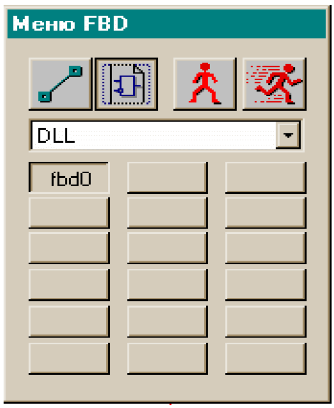

Рис. 2

При размещении функционального блока в поле создания и редактирования FBD программ блок будет выглядеть, как показано на рисунке.

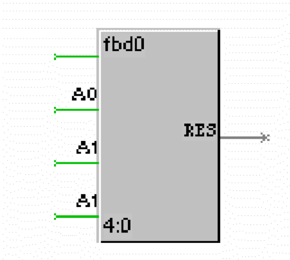

Рис. 3

Подключение FBD программы выполняется обычным способом.

Требования к отчету.

Отчет о работе должен содержать

1. Название работы;
2. Цель;
3. Постановку задачи;
4. Математическая формулировка задачи;
5. Спецификации входных и выходных переменных создаваемых функциональных блоков.
6. Алгоритмы функций, выполняемых функциональными блоками.
7. Тексты программ.

Библиографический список

Учебное издание

СОЗДАНИЕ ПОЛЬЗОВАТЕЛЬСКИХ ФУНКЦИОНАЛЬНЫХ БЛОКОВ ПРОГРАММИРОВАНИЕМ НА СИ++

Методические указания для выполнения лабораторной работы по дисциплине «Интегрированные системы проектирования и управления»

Для бакалавров направлений 220700, 220400

Составители: ХАУСТОВ Игорь Анатольевич,
ХВОСТОВ Анатолий Анатольевич

Компьютерный набор и верстка И.А. Хаустов

Подписано в печать
Формат 60x84 1/16. Бумага офсетная. Гарнитура Таймс. Ризография.
Усл. печ. л. . Уч.-изд. л. . Тираж 100 экз. Заказ
Воронежская государственная технологическая академия (ВГТА)
Участок оперативной полиграфии ВГТА
Адрес академии и участка оперативной полиграфии:
394000 Воронеж, пр. Революции, 19

ŋqddʮeY bɆ dəbɆbY3ə d ɞɔɑbɔʮ Ɉqdd Yqʮè