УДК: 004.81: 159.953.52

Разработка собственных блоков пакета расширения Spektr_SM+VisSim+dll

В.В. Рыбин

В настоящее время для визуального математического моделирования используются интегрированные системы Simulink+Matlab[5] и VisSim+Mathcad [6]. Для изучения теории систем автоматического регулирования и управления на базе системы VisSim создан учебно-методический комплекс[8]. Для изучения спектральной формы математического описания систем управления разработаны и применяются в процессе обучения пакеты расширения Spektr_SM [2-4] системы Simulink+Matlab и системы VisSim+Mathcad.

В данной статье рассмотрены технологические особенности разработки блока пакета расширения Spektr_SM системы VisSim+dll.

Современные интегрированные средства разработки приложений Windows позволяют автоматизировать процесс создания приложения. Для этого используются генераторы приложений. Пользователь отвечает на вопросы генератора приложений, который создает приложение, отвечающее заданным требованиям. Пользуясь им как шаблоном, пользователь сможет быстро разрабатывать свои приложения. Подобные средства автоматизированного создания приложений включены в компилятор Microsoft Visual C++ и называются MFC AppWizard. В среде Visual C++ можно строить различные типы проектов. Такие проекты после их создания можно компилировать и запускать на исполнение. Так мастер dllWizard позволяет создавать блоки пользователя для версии VisSim 5.0 и выше [9].

1. Технология формирования блока и сопутствующих программ

Механизм разработки блоков элементарных операций спектрального метода в системе визуального моделирования VisSim демонстрирует схема, расположенная на рис. 1. В среде Visual C++6 алгоритм формирования dll-файла любого блока пакета расширения Spektr_SM один и тот же. Продемонстрируем его на примере разработки проекта блока SAPNN1 вычисления ДНПФ апериодического звена.

1.1. Создание основы программного кода блока

Как уже отмечалось в работе [9] описан процесс разработки блока пользователя СВМ VisSim при помощи мастера dllWizard, который разработан фирмой Visual Solution. Он позволяет создать основу программного кода блока, т.е. существенно упростить процесс создания блока

Разработка блоков элементарных операций спектрального метода для пакета Spektr_SM + VisSim

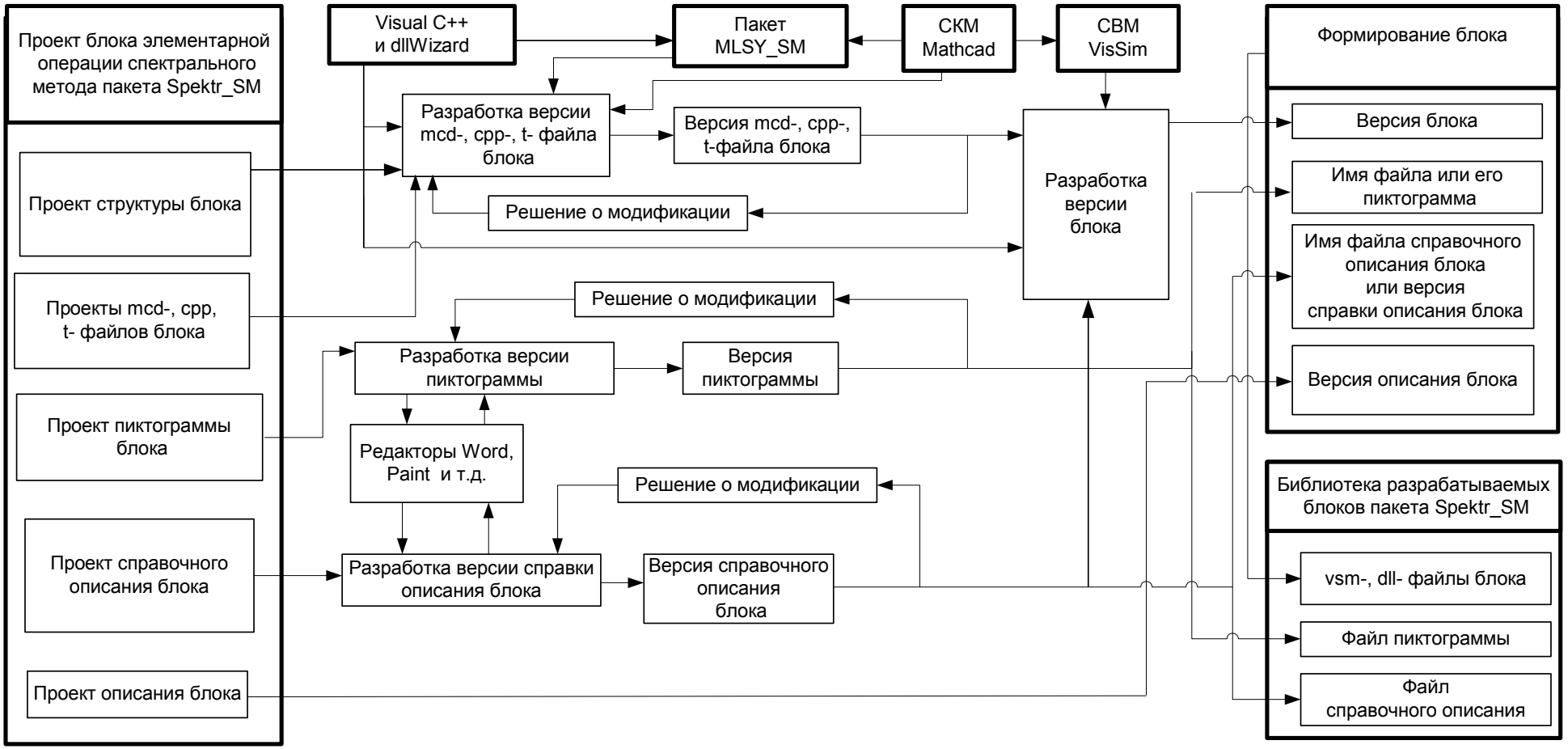


Рис. 1

пользователя, не углубляясь в тонкости программирования.

Пример 1. Разработать основу программного кода блока SAPNN1, который вычисляет усеченную матрицу ДНПФ апериодического звена и осуществляет преобразование НСХ входного сигнала в НСХ выходного сигнала.

Решение задачи. На компьютере, с установленными программами MS Visual C++ 6 и dllWizard, для создания проекта блока необходимо: 1) в Visual C++ выбрать File->New->VisSim DLL Wizard, 2) в поле **Project name** набрать название проекта в виде: **SAPNN1,** 3) в поле **Location** ввести путь к папке, в которой будут сохранены файлы проекта, и нажать кнопку **OK**.

После нажатия кнопки **OK**, последовательно начнут появляться окна **DLL-мастера**, в которых следует заполнить требуемые позиции. Вот эта последовательность:

1. Первое окно мастера **dllWizard (VisSim DLL Wizard - Step 1 of 7)** приглашает к созданию блока и предупреждает, что в построенную мастером основу программного кода потребуется внести дополнения, с тем, чтобы блок работал в соответствии с предъявляемыми к нему пользователем требованиями. Нажимаем кнопку **Next.**

2. В появившемся окне **(VisSim DLL Wizard - Step 2 of 7)** в поле **Base Function Name** вводим имя базовой функции в виде: **SAPNN1**, а в поле **Block Name** записываем название нашего блока в виде: **ДНПФ АЗ**. Первое поле отвечает за имя базовой функции. Второе – за текст, выводимый поверх блока в рабочей области. Это название всегда можно будет изменить непосредственно в программе. Нажимаем **Next.**

3. В появившемся окне (**VisSim DLL Wizard - Step 3 of 7**) задаем один вход (in0) и один выход (out0). Нажимаем **Next**.

4. В появившемся окне (**VisSim DLL Wizard - Step 4 of 7**) определяем тип входных \ выходных переменных. Важно задать этим переменным тип **MATRIX.** Размерности не важны, так как они будут переопределяться в дальнейшем. Нажимаем **Next**.

5. Появившееся окно (**VisSim DLL Wizard - Step 5 of 7**) оставляем без изменений и просто нажимаем **Next.**

6. В появившееся окно (**VisSim DLL Wizard - Step 6 of 7**) в поле **Top Level Menu Name** записываем имя в виде: **SPEKTR_SM**, в поле **Block Category Name (optional**) записываем имя в виде: **ДНПФ непрерывных систем**, а в поле **Block Name** записываем имя в виде: **ДНПФ апериодического звена.** Благодаря этому, создаваемый блок появится в меню **VisSim** по адресу “ **SPEKTR_SM -> ДНПФ непрерывных систем -> ДНПФ апериодического звена**”. Добавим галочку в **Include Custom MFC Dialog**, что позволит создать свой собственный диалог для изменения параметров, возникающий при щелчке правой кнопкой мыши на блоке. Если этого не сделать, то **VisSim** будет использовать стандартный диалог. Свой диалог требуется, чтобы задать

нужные параметры и подсказки о параметрах и их возможных значениях, что нельзя сделать в стандартном случае. Нажимаем **Next**.

7. В появившееся окно (**VisSim DLL Wizard - Step 7 of 7**) надо просто нажать кнопку **Finish.**

Все каркасные файлы для нашей программы написал **Visual C++,** с помощью мастера **VisSim DLL Wizard**.

После этого появится окно **New Project Information** в котором надо нажать **OK**.

Построенный **DLL Wizard** проект (остов приложения) создан. Естественно, никаких специфических для данного приложения свойств остов не содержит. Они появятся на следующем этапе, когда программист начнет работать с остовом, создавая из заготовки свое собственное приложение. Тем не менее, стартовое приложение можно транслировать и запускать на исполнение. Информация о нем отображается в окне **Workspace**. Окно состоит из трех вкладок – **ClassView**, **ResourceView**, **FileView,** отображающих информацию о проекте. Чтобы закрыть проект, надо выбрать пункт **меню File->Close Workspace.** Чтобы опять открыть его, надо в меню **File->Open** в списке **Тип Файлов** выбрать **WorkSpaces (.dsw,.mdp).**

Чтобы проект удачно компилировался, необходимо добавить к нему файл **vsuser.h**. Он находится в папке **…\VisSim50\Vsdk\Include**. Чтобы подключить этот файл к проекту, требуется переписать его в папку с созданным проектом, открыть диалог **Project->Add To Project->Files** и выбрать **vsuser.h**. После этого, в файл **vsi.cpp** добавляем строчку:

#include “vsuser.h

Если требуется изменить имя блока **SAPNN1**, заданное на втором шаге формирования проекта блока, надо открыть файл **vsi.cpp** и найти там функцию

SAPNN1Event(HWND h, int msg, WPARAM wParam, LPARAM lParam)

Ближе к её концу находится текст:

case WM_VSM_GET_BLOCK_NAME: return "ДНПФ АЗ";

Вместо имени блока **ДНПФ АЗ** можно вписать другое имя блока.

Если требуется изменить пункт меню, по которому располагается данный блок в меню **VisSim**, надо в файле **vsi.cpp** (ближе к началу) найти следующее объявление:

USER_MENU_ITEM um1[] = {{"&SPEKTR_SM", "SAPNN1", -2} ,{"&ДНПФ непрерывных систем", "SAPNN1", -1} ,{"ДНПФ апериодического звена", "SAPNN1", 1, 1, sizeof(SAPNN1_INFO),"Output data to SAPNN1 "},{0}};

и заменить в нем заданные имена на требуемые имена.

Как уже отмечалось, никакие средства автоматизированной разработки не смогут создать программу полностью без участия программиста. Прикладную часть приложения придется

разрабатывать ему самому. Однако мастер dllWizard создает необходимые функции для формирования основной функциональности блока. Они располагаются в файле vci.cpp и при формировании проекта блока подвергаются модификации. Рассмотрим эти функции.

Функции уровня моделирования (симуляции):

1. vsmInit – вызывается VisSim при запуске и используется для вставки пунктов в меню.

2. vsmEvent - вызывается при событиях уровня моделирования, таких как начало и конец моделирования, конец шага моделирования и т.д. Реакция на эти события может быть переопределена.

Функции уровня блока (в имени этих функций используется имя, которое задается во втором окне мастера dllWizard):

1. SAPNN1(SAPNN1_INFO *pi, SIGNAL *inSig[], SIGNAL outSig[]) – вызывается на каждом шаге моделирования и содержит основную функциональность блока; SAPNN1_INFO *pi – ссылка на структуры с текущими параметрами блока; SIGNAL *inSig[], - ссылка на входные параметры блока; SIGNAL outSig[] - ссылка на выходные параметры блока.

2. SAPNN1Event - функция вызывается при возникновении событий уровня блока.

3. SAPNN1PA - функция вызывается при загрузке блока и используется для распределения памяти под параметры.

4. SAPNN1PC - функция вызывается при щелчке правой кнопкой мыши на блоке. Если в шестом окне мастера dllWizard была установлена галочка (для изменения параметров используется пользовательский диалог). Если используется стандартный диалог, то функция должна вернуть строку, где через точку с запятой перечислены названия параметров, которые будут высвечиваться рядом с соответствующими полями в стандартном диалоге.

5. SAPNN1PI - функция вызывается при создании блока и позволяет устанавливать значения параметров по умолчанию.

6. SAPNN1SE - функция вызывается по окончании моделирования, чтобы совершить действия необходимые для выхода.

7. SAPNN1SS - функция вызывается непосредственно перед началом моделирования, чтобы провести необходимую инициализацию.

1.2. Формирование файлов, обеспечивающих выполнение элементарных операций спектрального метода в разных базисах.

Программные модули, обеспечивающие выполнение элементарных операций спектрального метода в разных базисах находятся в пакете расширения MLSY_SM C++ [1]. Заметим, что разработать и отладить этот пакет можно в Microsoft Visual C++, используя консольное приложение **Win32 Concole Application.** Из программных модулей этого пакета и

формируются файлы разрабатываемого проекта блока, обеспечивающие выполнение нужной элементарной операции спектрального метода в разных базисах. Имена этих файлов формируются из имени проекта и префикса **ПЭО_**.

Пример 2. Сформировать файлы **ПЭО_SAPNN1.cpp** и **ПЭО_SAPNN1.h** , обеспечивающие вычисление усеченной матрицы ДНПФ апериодического звена в разных базисах.

Решение задачи.

)1 В пакете расширения **MLSY_SM C++** находим программные модули: **si1pp1, si1cc1, si1ff1, si1tt1, si1uu1, si1xx1, si1yy1** и вставляем их через буфер обмена в файл **ПЭО_SAPNN1.cpp**. Каждый программный модуль вычисляет усеченную матрицу ДНПФ интегрирующего звена относительно заданной базисной системы. Например, программный модуль **si1pp1** имеет вид:

```
Matrix <double> si1pp1(double t, double L)
        {Matrix <double> m(L,L,0); int i = 0; m[i][i] = t/2;
            for(int k = 0; k <= L-2; k++)
            {m[k][k+1] = -t/(2*sqrt((2*double(k)+1)*(2*double(k)+3)));
            m[k+1][k] = t/(2*sqrt((2*double(k)+1)*(2*double(k)+3)));}
        return m;}
```

)2 В файле **ПЭО_SAPNN1.cpp** формируем программный модуль, обеспечивающий выбор базиса. Этот программный модуль имеет вид:

```
Matrix <double> si1nn1(double t, double L, int BAZIS)
                {switch(BAZIS)
                { case 1: return si1pp1(t,L); break;
                case 2: return si1tt1(t,L); break;
                case 3: return si1uu1(t,L); break;
                case 4: return si1cc1(t,L); break;
                case 6: return si1xx1(t,L); break;
                case 7: return si1yy1(t,L); break;}}
```

Параметр BAZIS (это целое число от 1 до 7), передаваемый в программу, обеспечивает выбор нужного базиса по следующей таблице соответствия: 1 - полиномы Лежандра; 2 - полиномы Чебышева первого рода; 3 - полиномы Чебышева второго рода; 4 - косинусоиды; 5 - комплексные экспоненциальные функции; 6 - функции Хаара; 7 - функции Уолша.

)3 В файле **ПЭО_SAPNN1.cpp** формируем программный модуль, обеспечивающий вычисление усеченной матицы ДНПФ апериодического звена в выбранном базисе. Этот программный модуль имеет вид:

```
Matrix <double> sapnn1(double L, double T, double k, double te, int BAZIS)
```

{ if(BAZIS == 6 || BAZIS == 7) { double x,y,n; x = log10(L)/log10(2); y = modf(x, &n); if(y != 0) { MessageBox(NULL,"Порядок усечения должен быть степенью двух", "Error", MB_ICONERROR | MB_OK); assert(0); } }Matrix <double> I1(L,L,0); Matrix <double> I2(L,L,0); Matrix <double> E(L,L,0); Matrix <double> AP(L,L,0); double T1; E = E.identity(); I1 = si1nn1(te, L, BAZIS); T1 = 1.0/T; I2 = E + T1*I1; I2 = I2.inverse(); AP = k*T1*I2*I1; return AP; }

)4 Формируем файл **ПЭО_SAPNN1.h**:

```
#include "MATRIXFULL.h"
#include <complex>
using namespace std;
typedef complex <double> cmplx;
//Функции для si1nn1
Matrix <double> si1pp1(double t, double L); Matrix <double> si1cc1(double t, double L);
Matrix <cmplx> si1ff1(double t, double L); Matrix <double> si1tt1(double t, double L);
Matrix <double> si1uu1(double t, double L); Matrix <double> si1xx1(double t, double L);
Matrix <double> si1yy1(double t, double L); Matrix <double> si1nn1(double t, double L, int
BAZIS); Matrix <double> sapnn1(double L, double T, double k, double te, int BAZIS);
```

)5 Для корректной работы формируемой программы в файле **ПЭО_SAPNN1.cpp** выполняем следующие подключения:

```
#include "stdafx.h"
#include "ПЭО_SAPNN1.h"
#include <iostream>
#include <assert.h>
#define PI 3.14159265358979
```

)6 Сформированные файлы **ПЭО_SAPNN1.cpp**, **ПЭО_SAPNN1.h** и файл **MATRIXFULL.h** (специальный шаблон матриц), обеспечивающий эффективную работу с матрицами перепишем в директорию проекта и добавим их к проекту, воспользовавшись диалогом **Project->Add To Project->Files**.

)7 Для корректной работы формируемой программы в файле в **vsi.cpp** выполняем следующие подключения:

```
#include “ MATRIXFULL.h ”
#include "ПЭО_SAPNN1.h"
```

Аналогично формируются файлы, обеспечивающих выполнение элементарных операций спектрального метода в разных базисах и в других проектах.

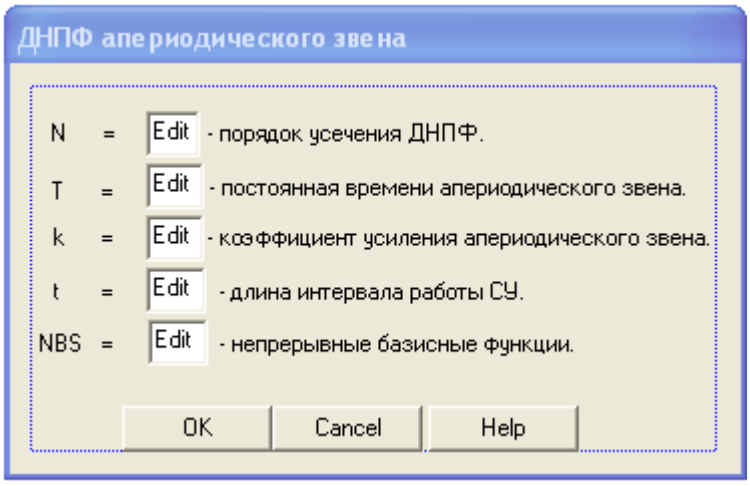
1.3. Создание диалога блока

Мастер dllWizard позволяет пользователю создавать в среде MS Visual C++ 6 не только типовые, но и более содержательные диалоговые окна [9]. Продемонстрируем технологию создания диалога блока разрабатываемого пакета Spektr_SM на примере блока SAPNN1.

Пример 3. Разработать форму окна диалога блока SAPNN1, который вычисляет усеченную матрицу ДНПФ апериодического звена и осуществляет преобразование НСХ входного сигнала в НСХ выходного сигнала.

Решение задачи. В окне **Workspace** проекта блока SAPNN1 перейдем на вкладку **ResourceView** и раскроем все списки:

Выбрав **IDD_FORMVIEW**, перейдем в редактор внешнего вида диалога. Добавим на него новые поля, отредактируем текст и получим следующий вид:



В середине окна диалога описаны параметры блока, которые можно поменять. Внизу окна диалога кнопки выхода из диалога и получения справки по настройке параметров блока. Щелкнув правой кнопкой мыши на поле окна ввода параметров, выбираем в выпадающем меню **Properties** (Свойства) и редактируем параметр **ID** (делаем **IDC_N1** для N**, IDC_T** для T**, IDC_K** для k**, IDC_TSMALL** для t**, IDC_NBS** для NBS). Названия кнопок можно менять. Для этого нужно щелкнуть правой кнопкой мыши на выбранной кнопке, в выпадающем меню выбрать **Properties** и изменить название кнопки. Далее необходимо создать обработчик события нажатия на кнопку **Help**. Для этого следует выделить кнопку **Help**, нажать на нее правой кнопкой мыши и в выпадающем меню выбрать инструментальное средство - **ClassWizard** (мастер классов). Мастер классов предоставляет широкий спектр услуг. В частности он позволяет добавлять к

существующему классу новые методы и данные. Например, включить в класс **CVsmDialog** элементы данных, связанные с полями диалоговой панели. Если это делается в данном проекте впервые, то MS Visual C++ 6 откроет диалоговую панель с именем «**Select Source File –SAPNN1**». В окошко **File name** вводим имя **SAPNN1.clw** и нажимаем кнопки **ADD** и **OK**. Открывается диалоговая панель, на которой нужно для проекта **SAPNN1** и объекта **IDC_BUTTON1** (параметр кнопки **Help)** выбрать класс **CVsmDialog** и обработчик **BN CLIKED.** Нажать на кнопку **Add function**, а затем, на появившейся панели, на кнопку **OK** и на диалоговой панель на кнопку **Edit code**.

Для завершения процесса формирования диалога блока необходимо отредактировать файлы **CVsmDialog.cpp, vsmDialog.h.** и **vsi.cpp.**

Пример 4. Отредактировать файлы **vsmDialog.cpp** и **vsmDialog.h** под разрабатываемый диалог блока SAPNN1.

Решение задачи. После того, как форма окна диалога блока SAPNN1 сформирована в файле **vcmdialod.cpp** появится вставка:

```
void CVsmDialog::OnButton1() {       }
```

Вводим в нее следующий текст:

```
void CVsmDialog::OnButton1 ()
{MessageBox("Вычисляется усеченная матрица ДНПФ\n"
"апериодического звена на интервале работы СУ [0, t].\n"
"N = 2, 3, 4, … для полиномов Лежандра, \n"
"Чебышева 1-го и 2-го рода, косинусоид.\n"
"N= 3,5,7,… для комплексных экспоненциальных функций.\n"
"N= 2, 4, 8,… для функций Уолша и Хаара.\n"
" \n"
"Задание базисной системы:\n"
"1 - полиномы Лежандра;\n"
"2 – полиномы Чебышева 1-го рода;\n"
"3 - полиномы Чебышева 2-го рода;\n"
"4 – косинусоиды; \n"
"5 - комплексные экспоненциальные функции;\n"
"6 - функций Хаара;\n"
"7 - функций Уолша.\n",
"          ДНПФ апериодического звена");}
```

Переходим в левом окне проекта обратно на вкладку **FileView** и выбираем файл **vsmdialog.h.** Находим там перечисление:

```
//{{AFX_DATA(CVsmDialog) enum{ IDD = IDD_FORMVIEW }; double
    m_myDouble; int m_myInt; CString m_myString;
 //}}AFX_DATA
```

и заменяем его на перечисление:

```
//{{AFX_DATA(CVsmDialog) enum { IDD = IDD_FORMVIEW };
double d_L;       double d_T;   double d_k;   double d_te;  int d_BAZIS;
//}}AFX_DATA
```

Таким образом, мы определяем переменные, для хранения значений параметров в диалоге до передачи их в основную программу.

Теперь переходим в файл **vsmdialog.cpp**. Заменяем перечисления:

```
//{{AFX_DATA_INIT(CVsmDialog)
m_myDouble = 0.0; m_myInt = 0; m_myString = _T("");
//}}AFX_DATA_INIT

//{{AFX_DATA_MAP(CVsmDialog)
DDX_Text(pDX, IDC_MY_DOUBLE, m_myDouble);
DDX_Text(pDX, IDC_MY_INT, m_myInt);
DDX_Text(pDX, IDC_MY_STRING, m_myString);
//}}AFX_DATA_MAP
```

на перечисления:

```
//{{AFX_DATA_INIT(CVsmDialog)
double d_L = 3; double d_T = 0.2; double d_k = 1; double d_te = 2; int d_BAZIS = 1;
//}}AFX_DATA_INIT
//{{AFX_DATA_MAP(CVsmDialog)
DDX_Text(pDX, IDC_N1, d_L); DDX_Text(pDX, IDC_T, d_T);
DDX_Text(pDX, IDC_K, d_k); DDX_Text(pDX, IDC_TSMALL, d_te);
DDX_Text(pDX, IDC_NBS, d_BAZIS);
//}}AFX_DATA_MAP
```

Таким образом, мы устанавливаем начальные значения переменных в диалоге и определяем, какое поле соответствует какой переменной (например, поле с **ID = IDC_N1** соответствует переменной **d_L** и при изменении параметров, значение, введенное в это поле, будет сохраняться в указанной переменной).

Отметим так же, что для работы с параметрами в блоке в файле **vsi.cpp** необходимо:

1) Определить переменные, соответствующие параметрам в структуре SAPNN1_INFO.

2) Константе SAPNN1_INFO_STR задать в качестве значения строку, в которой перечислены типы переменных из структуры SAPNN1_INFO в соответствии с шаблоном: I – целое (int); F – с плавающей точкой 64 бита (64 bit float); f - с плавающей точкой 32 бита (32 bit float); s – строка (string); c – символ (char); b – байт(byte).

3) Написать обработку параметров в функции SAPNN1PC.

4) Задать начальные значения параметров в функции SAPNN1PC, используя передаваемую этой функцией переменную pi.

Пример 5. Отредактировать файл **vsi.cpp** под разрабатываемый диалог блока SAPNN1.

Решение задачи. Добавим параметры блока и создадим диалог для их изменения. Для этого найдем в файле **vsi.cpp** структуру:

```
typedef struct { int myInt; // Add your structure contents here double myDouble;
  char *myStr;} SAPNN1_INFO;
```

и заменим её следующей структурой:

typedef struct { double L; // Add your structure contents here
double T; double k; double te; int BAZIS;} SAPNN1_INFO;

В этой структуре определяются переменные, в которых будут храниться значения параметров (**L** – порядок усечения ДНПФ; **T** – постоянная времени апериодического звена; **k** – коэффициент усиления апериодического звена; **te** – длинна интервала работы СУ; **BAZIS** – номер непрерывной базисной функции).

Сразу после неё идёт определение:

#define VS_INFO_STR "IFs"

Заменим это определение новым:

#define VS_INFO_STR "FFFFI"

Затем надо найти функцию **SAPNN1PI(SAPNN1_INFO *pi)** и добавить в неё следующую строчку:

(*pi).BAZIS = 1; (*pi).k = 1; (*pi).T = 0.2; (*pi).L = 3; (*pi).te = 2;

Таким образом, задаются начальные значения параметров. Именно их будет использовать блок, если после его добавления не вызывать диалог изменения параметров.

Далее находим функцию **SAPNN1PC(SAPNN1_INFO *pi, long *runCount).** Эта функция вызывается при щелчке правой кнопкой мыши на блоке в окне модели **VisSim**. Заменим её тело новым телом функции:

AFX_MANAGE_STATE(AfxGetStaticModuleState());
CVsmDialog *dlg; dlg = new CVsmDialog(); dlg->d_L = pi->L;
dlg->d_T = pi->T; dlg->d_k = pi->k; dlg->d_te = pi->te; dlg->d_BAZIS = pi->BAZIS;
if (dlg->DoModal() == IDCANCEL) return 0; pi->L = dlg->d_L; pi->T = dlg->d_T;
pi->k = dlg->d_k; pi->te = dlg->d_te; pi->BAZIS = dlg->d_BAZIS; delete dlg; return 0;

Здесь мы сначала объявляем переменную типа диалога и создаем сам диалог. Затем записываем в его переменные текущие значения параметров. Обрабатываем ситуацию, когда пользователь выбирает кнопку **Cancel** (ничего не изменяется). Присваиваем параметрам значения, введенные в диалоге, и уничтожаем диалог. Диалог создан.

1.4. Формирование матричных входов и выходов блока

При формировании проекта входу и выходу формируемого блока был присвоен тип **MATRIX. О**днако входу и выходу будет присвоен матричный тип только после некоторых изменений в файле **vsi.cpp**. Продемонстрируем внесение этих изменений на примере блока SAPNN1.

Пример 5. Отредактировать файл **vsi.cpp** так, чтобы вход и выход блока SAPNN1 стали матричного типа.

Решение задачи. В файле **vsi.cpp** находим функцию **SAPNN1Event(HWND h, int msg, WPARAM wParam, LPARAM lParam).** В ней находим следующий текст:

```
if (port >=0) return (LPSTR) inDesc[port-1].type; // return scaled int type for
port = -(port+1);  // Here if output, zero base out port
// Set scaled int radix
if ( outDesc[port].type == T_SCALED_INT)
{ outSig = (SIGNAL *)vissimRequest(VR_GET_BLOCK_OUTPUT, blockHandle, 0);
outSig += port; outSig->type = T_SCALED_INT;
outSig->u.scaledInt.radixPoint = outDesc[port].dim1;
outSig->u.scaledInt.wordLength = outDesc[port].dim2;}
return (LPSTR)outDesc[port].type;
```

и заменить его одной строкой:

```
return (LPSTR)((port == 1 || port == -1)?T_MAT_DOUBLE:T_DOUBLE);
```

Затем, сразу после этой строки, вставить следующий текст:

```
case WM_VSM_IS_VEC_CONNECT: // Scalar arg 1, matrix arg2
 port = wParam; return (LPSTR)(port == 1 || port == -1); // Negative port is output

case WM_VSM_ALLOC_VEC_OUTPUT: // Allocate matrix output
// Use dimensions of matrix on input 2
inSig = (SIGNAL FAR**)vissimRequest( VR_GET_BLOCK_INPUT, lParam, 0);
if (inSig[0] && inSig[0]->type == T_MAT_DOUBLE)
{ dim1 = inSig[0]->u.m->dim1; dim2 = inSig[0]->u.m->dim2; }
outSig = (SIGNAL FAR*)vissimRequest( VR_GET_BLOCK_OUTPUT, lParam, 0);
MatrixOpParam = (VS_INFO FAR*)vissimRequest( VR_GET_BLOCK_PARAMS, lParam, 0);
outSig[0].type = T_MAT_DOUBLE;
outSig[0].u.m = matReDeclare(outSig[0].u.m, dim1,dim2); // Allocate output matrix
break;
```

После этого, и вход, и выход станут типа MATRIX. Теперь, чтобы работать с данным типом в нашей программе, необходимо добавить в неё функцию:

```
double *mxCheck( MATRIX *m, unsigned j1, unsigned j2)
{  ASSERT(j1 < m->dim1);  ASSERT(!j2 || j2 < m->dim2);  return &m->d[(j1) + (j2) * m->dim1];}
```

и заменить функции:

```
static LPSTR inCxName[1]={"in0"};
static LPSTR outCxName[1]={"out0"};
typedef struct {  int type;  int dim1;  int dim2;} ARG_DESC;
static ARG_DESC inDesc[] = {  {T_MAT_DOUBLE,1,16}};
static ARG_DESC outDesc[] = { {T_MAT_DOUBLE,1,16}};
static int getBlockId(LPARAM blockHandle)
{ return vissimRequest(VR_GET_BLOCK_ID,blockHandle, 0);}
static int includedFile;
static void preSimInit()
{  includedFile = 0;}
```

на функции:

```
sstatic MATRIX *PASCAL matReDeclare( MATRIX *m, unsigned dim1, unsigned dim2)
{  unsigned len = dim1*sizeof(m->d[0]);
  if (dim2)    len *= dim2;  m = (MATRIX*)grealloc( m, sizeof(MATRIX)+len);
  m->dim1 = dim1;  m->dim2 = dim2;  return m;}

static MATRIX *getMatrix( MATRIX *m)
{  int a;  if (m) return m; // Already allocated  m = matReDeclare( m, 3, 3); // Allocate a 3x3
  for (a = 0; a < 3;a++) // Initialize ones on diagonal  MX( m, a,a) = 1;  return m;}
typedef struct {  int type;  int dim1;  int dim2;} ARG_DESC;
static ARG_DESC inDesc[] = {  {T_MAT_DOUBLE,1,16}};
static ARG_DESC outDesc[] = {  {T_MAT_DOUBLE,1,16}};
static int getBlockId(LPARAM blockHandle)
{ return vissimRequest(VR_GET_BLOCK_ID,blockHandle, 0);}
static int includedFile; static void preSimInit() { includedFile = 0;}
```

1.5. Формирование матричной связи вход-выход по усеченной матрице ДНПФ элементарного или типового звена

Формируемый блок должен обеспечить выполнение операции связи вход-выход. Это будет достигнуто только после редактирования некоторых функций из файла **vsi.cpp**. Продемонстрируем внесение этих изменений на примере блока SAPNN1.

Пример 6. Отредактировать файл **vsi.cpp** проекта блока SAPNN1 так, чтобы входной сигнал правильно преобразовывался в выходной сигнал.

Решение задачи. Обработка входного сигнала, и получение выходного происходит в функции **SAPNN1(SAPNN1_INFO *pi,SIGNAL *inSig[], SIGNAL outSig[]).**

Разберём, что надо добавить в неё в нашем случае.

Создаём переменные-ссылки на матрицы входного и выходного сигналов, далее работаем с ними:

```
MATRIX *mIn1 = inSig[0]->u.m;  MATRIX *mOut1 = outSig[0].u.m;
```

Выводим предупреждение, в случае несоответствия размерностей входной матрицы порядку усечения матрицы ДНПФ апериодического звена:

```
if((*pi).L != mIn1->dim1)
{MessageBox(NULL,"Число строк должно равняться порядку усечения ДНПФ",
"Error", MB_ICONERROR | MB_OK); assert(0);}
```

Вводим вспомогательные переменные для работы с матрицами:

```
Matrix <double> in0(mIn1->dim1,mIn1->dim2,0);
Matrix <cmplx> in0for5(mIn1->dim1, ceil((double)(mIn1->dim2)/2.0),0);
Matrix <double> AOut((*pi).L,(*pi).L,0);
Matrix <double> AOutfor5((*pi).L,mIn1->dim2,0);
Matrix <cmplx> I1((*pi).L,(*pi).L,0);
Matrix <cmplx> I2((*pi).L,(*pi).L,0);
Matrix <cmplx> E((*pi).L,(*pi).L,0);
Matrix <cmplx> AP((*pi).L,(*pi).L,0);
cmplx T1;
```

```
cmplx ka;
```

Присваиваем матрицу входного сигнала вспомогательной матрице **in0**. Доступ к элементам переменной типа MATRIX осуществляется при помощи функции **MX**:

```
for(int i = 0; i < mIn1->dim1; i++)
for(int j = 0; j < mIn1->dim2; j++)
in0[i][j] = MX(mIn1,i,j);
```

Если выбрана действительная система базисных функций, то усеченную матрицу ДНПФ апериодического звена (функция **sapnn1** из файла **ПЭО_SAPNN1.cpp**) умножаем на усеченную матрицу НСХ входного сигнала (входная матрица), а результат этой операции - усеченную матрицу НСХ выходного сигнала, присваиваем матрице выхода:

```
if((*pi).BAZIS != 5)
{AOut = sapnn1((*pi).L,(*pi).T,(*pi).k,(*pi).te,(*pi).BAZIS)*in0;
        for(int a = 0; a < mIn1->dim1; a++)
        for(int b = 0; b < mIn1->dim2; b++)
              MX(mOut1,a,b) = AOut[a][b];
return;}
```

Для комплексных систем базисных функций входная матрица образована приписыванием к матрице действительной части усеченной матрица НСХ входного сигнала ее мнимой части. Поэтому сначала входная матрица приводится к комплексному виду, а затем формируется усеченная матрица ДНПФ апериодического звена и умножается на усеченную матрицу НСХ входного сигнала. Результат этой операции - усеченная матрица НСХ выходного сигнала преобразуется в две матрицы и к матрице действительной части приписывается матрица мнимой части, а полученная матрица присваивается матрице выхода:

```
Else { if(mIn1->dim2 % 2 !=0)
{MessageBox(NULL,"Число столбцов должно быть четным", "Error", MB_ICONERROR
| MB_OK); assert(0);} for(int u = 0; u < (*pi).L; u++) { E[u][u] = 1;}
I1 = si1ff1((*pi).te, (*pi).L); T1 = complex<double>(1.0/(*pi).T,0); I2 = E + T1*I1; I2 =
I2.inverse(); ka = complex<double>((*pi).k,0); AP = ka*T1*I2*I1;
for(int i = 0; i < mIn1->dim1; i++)
for(int j = 0; j <= ceil((double)(mIn1->dim2)/2.0)-1;j++)
{(in0for5[i][j]) = complex<double>(in0[i][j], in0[i][j+floor((double)(mIn1->dim2)/2.0)]);}
in0for5 = AP*in0for5;
for(int a = 0; a < (*pi).L; a++)
for(int b = 0; b <= ceil((double)(mIn1->dim2)/2.0)-1; b++)
{AOutfor5[a][b] = (in0for5[a][b]).real();
AOutfor5[a][b + floor((double)(mIn1->dim2)/2.0)] = (in0for5[a][b]).imag();}
for(int q = 0; q < mIn1->dim1; q++)
for(int w = 0; w < mIn1->dim2; w++)
MX(mOut1,q,w) = AOutfor5[q][w];
return;}
```

На этом закончен процесс формирования версии проекта блока SAPNN1.

2. *Формирование dll-файла блока и его включение в пакет расширения Spektr_SM+dll*

Формирование проекта закончено. Теперь надо созданную программу скомпилировать, выбрав пункт меню **Build->Build SAPNN1.dll.** Если не было ошибок при вводе, то в нижнем окне Visual C++ 6 появится сообщение: **SAPNN1.dll - 0 error(s), 0 warning(s)**, а в папке **ПРОЕКТ SAPNN1** появится директория **Debug**, в которой находится файл **SAPNN1.dll**. Для получения окончательной версии файл **SAPNN1.dll,** которая не содержит отладочный код, в меню **Build** (построить) выбераем **Set Active Configuration** (установка активной конфигурации) и устанавливаем **Win32 Release** и проводим окончательную компиляцию.

Чтобы включить созданный блок в библиотеку СКМ **VisSim**, надо запустить систему **VisSim** и выбрать в меню пункт **Правка->Настройки…**, в появившемся окне перейти на вкладку **AddOns** – Библиотеки дополнительных инструментов, щелкнуть на многоточии и выбрать созданный файл **SAPNN1.dll**. После этого в меню СВМ VisSim появится новый раздел **SPECTR_SM**. Для загрузки блока вычисления ДНПФ апериодического звена (файл SAPNN1.dll) в окно формируемой модели нужно выбрать пункт меню **SPECTR_SM ->ДНПФ непрерывных систем -> ДНПФ Апериодического звена.**

Список литературы

1. Рыбин В.В. Разработка и применение пакетов расширения MLSY_SM СКМ Mathcad, Maple, Mathematica, Matlab.// Электронный журнал “Труды МАИ”, № 13. - http://www.mai.ru (21.10.2003)
2. Рыбин В.В. Разработка и применение пакета расширения Spektr_SM пакета Simulink СКМ Matlab.// Электронный журнал “Труды МАИ”, № 13. - http://www.mai.ru (21.10.2003)
3. Рыбин В.В. Разработка и применение пакета расширения Spektr_SM системы VisSim+Mathcad. .// Электронный журнал “Труды МАИ”, № . - http://www.mai.ru ( )
4. Рыбин В.В. Разработка и применение пакета расширения Spektr_SM пакета Simulink СКМ Matlab: Учебное пособие.- М.: МАИ, 2004. –66 с.
5. Дьяконов В.П. MATLAB 6/6.1/6.5 + Simulink 4/5. Основы применения. Полное руководство пользователя. - М.: СОЛОН-Пресс, 2002. – 768 с.
6. Дьяконов В.П. VisSim+Mathcad+MATLAB. Визуальное математическое моделирование. - М.: СОЛОН-Пресс, 2004. – 384 с.
7. Клиначёв Н.В. Технология создания внешних “dll-модулей” для моделирующей программы VisSim. –Website: http://www.vissim.nm.mydll.html, Челябинск, 2000.

8. Клиначёв Н.В. Теория систем автоматического регулирования и управления: Учебно-методический комплекс. - Website: http://www.vissim.nm.ru./tau_knv.zip, http://www.vissim.nm.ru./tau_lec.html, Челябинск, 2003.
9. Федосеев Б.Т. Создание блока пользователя для программы VisSim 5 в среде MS Visual C++6 с использованием мастера dllWizard. - Website: http://www.vissim.nm.ru, Челябинск, 2003.
10. Сайт “VisSim в России” http://www.vissim.nm.ru.
11. Сайт фирмы Visual Solutions Inc http://www.vissim.com.

Рыбин Владимир Васильевич, доцент кафедры математической кибернетики Московского Авиационного института (государственного технического университета), к.т.н.
контактный телефон: 158-48-11
E-mail:dep805@mai.ru