Система высокоуровневого проектирования аппаратного обеспечения HLCCAD: открытый универсальный интерфейс моделируемых компонентов

**Михаил Долинский,
Вячеслав Литвинов,
Андрей Галатин,
Наталья Шалаханова**

Введение

Высокоуровневые модели (то есть модели, разработанные с помощью языков программирования высокого уровня) могут использоваться для выполнения следующих видов работ:

- эффективное исследование проектного пространства;
- имитация устройств ввода-вывода;
- интеграция в симулируемую систему стороннего программного обеспечения;
- интеграция в симулируемую систему стороннего аппаратного обеспечения;
- замена отсутствующих компонентов проекта;
- ускорение симуляции;
- контроль адекватности проекта (его частей) «золотым моделям»;
- моделирование процессоров;
- совместная отладка программного и аппаратного обеспечения;
- моделирование мультипроцессорных систем;
- развитие стандартной библиотеки компонентов; в этом случае нужно дополнительно обеспечивать создание адекватной VHDL-модели одним из следующих способов: генерация схемы для HLCCAD или генерация синтезируемого VHDL;
- синтез микропрограммных автоматов: симуляция модели виртуального процессора МПА (микропрограммных автоматов), отладка микропрограмм, генерация схемы микропрограммного автомата с жесткой логикой.

К основным достоинствам высокоуровневых моделей можно отнести высокую скорость симуляции и простоту создания (относительно получения тех же сущностей композицией стандартных элементов). К основным проблемам использования высокоуровневых моделей можно отнести потребность в квалифицированных специалистах одновременно в программном и аппаратном обеспечении, а также отсутствие в подавляющем большинстве САПР (чтобы не сказать во всех), эксплуатируемых сегодня на территории России, возможностей интегрировать в них высокоуровневые модели. Одним из важных достоинств HLCCAD [1–4] как раз является наличие такой возможности. Более того, HLCCAD обеспечивает два универсальных интерфейса, существенно упрощающих разработку высокоуровневых моделей:

- интерфейс (IModel) моделей с системой компиляции, моделирования и анализа, то есть набор процедур и функций, которые должен создать разработчик высокоуровневой модели, чтобы она была эффективно интегрирована в систему HLCCAD;
- интерфейс (IHLCCAD), предоставляемый системой HLCCAD для упрощения разработки моделей — готовый набор процедур и функций, выполняющих полезные действия по доступу к информации, хранящейся во внутренней структуре данных HLCCAD.

1. COM-интерфейс моделей с системой компиляции, моделирования и анализа

Высокоуровневая модель (устройства или внешней среды) представляется виде COM-объекта, расположенного в 32-битной динамически загружаемой библиотеке (DLL — Dynamic Link Library). Каждая такая DLL представляет набор функций, доступных извне. Система HLCCAD может загрузить динамическую библиотеку и получить необходимые «точки входа» (адреса начала функций библиотеки) по их номеру или имени. Интерфейс передачи параметров предварительно оговорен. COM-объекты, по определению, можно создавать на любом языке программирования, способном поддержать данный интерфейс (например, Object Pascal, C++).

Для инициализации COM-объекта в параметрах корпуса указывается имя файла динамической библиотеки и <префикс> имени функции. Функция <префикс>_Init из указанной DLL вызывается системой HLCCAD для создания модели. В качестве параметра этой модели передается интерфейс IHLCCAD. Функция должна вернуть указатель на интерфейс модели IModel.

Процедуры и функции модели (определенные интерфейсом IModel и реализуемые разработчиком модели) позволяют системе HLCCAD получить информацию о свойствах модели, провести анализ корпуса устройства, обеспечить своевременный вызов модели для симуляции и визуализации результатов.

В свою очередь, процедуры и функции, определенные интерфейсом IHLCCAD (и уже реализованные в системе HLCCAD ее разработчиками) обеспечивают модели доступ к информации, интерактивно определяемой пользователем в процессе работы в HLCCAD на этапах компиляции, моделирования и анализа.

2. Технология создания высокоуровневых моделей устройств

2.1. Общие принципы построения исходного текста модели

Создание высокоуровневой модели осуществляется путем реализации объекта, поддерживающего интерфейс модели IModel и процедуры инициализации. Для поддержки этого используется наследование объекта:

```
TDeviceModel = class(TInterfacedObject, IModel)
```

Затем описываются внутренние элементы объекта для хранения данных, такие, как:

- I,O: pointer; — указатели на контакты корпуса, для установки значений на контакты;
- valI: pChar; — указатель на значение входа, для чтения значения на линиях контакта;
- ISelf: IModel; — указатель на интерфейс модели
- HLCCAD: IHLCCAD; — указатель на интерфейс системы для вызова функций управления моделированием и параметрами моделирования данного устройства.

Далее создается конструктор объекта:

Constructor Create(aHLCCAD: IHLCCAD);

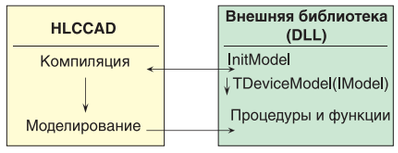

Рис. 1. Структурная схема взаимодействия с моделью

В этой функции («конструкторе») происходит проверка соответствия условного графического обозначения (УГО) корпуса и модели, а также инициализация внутренних элементов объекта. Соответствие определяется при помощи функций интерфейса IHLCCAD.

После того как данные проверены и получен указатель на интерфейс модели, вызывается функция Correct для определения отсутствия ошибок. Если ошибки были обнаружены моделью, то процесс компиляции прерывается. В противном случае продолжается инициализация модели устройства. Вызывается функция Options, которая информирует систему о режиме моделирования модели:

- flOnChange — активизация при изменении значений на входах;
- flOnTime — активизация после указанного промежутка времени;
- flCPU — дополнительный флаг, информирующий систему о том, что данная модель является моделью процессора. (Описание наработанных технологий разработки моделей процессоров — тема отдельной статьи.)

Затем следует вызов функции AfterCreate, которая предназначена для создания внутренних объектов модели — памяти, регистров, флагов, битов, окон визуализации или ввода и прочее.

И наконец, происходит вызов процедуры AutoStart, внутри которой модель инициализирует свои внутренние данные, устанавливает значения на выходы корпуса.

На этом процесс инициализации закончен.

Для реализации функционирования устройства предназначены две функции, одна из них вызывается после изменения значений на входах, а другая вызывается системой через указанный промежуток времени. Допустимо использование обоих способов активизации модели.

Для реализации первого подхода необходимо заполнить процедуру OnChanged, для второго — ExecAfter. Функция ExecAfter в качестве результата возвращает величину задержки модельного времени в пикосекундах до следующего вызова. Если величина будет меньше нуля, то вызов данной функции прекращается.

При реализации этих функций модель может обрабатывать значения входных контактов, изменять значения выходов, а также значения внутренней памяти, регистров, флагов и битов.

Окно «Инспектор проекта» (рис. 2) предоставляет разработчику гибкую систему документирования разрабатываемых устройств.

Рис. 2. Окно «Инспектор проектов»

Среда хранения документов подразумевает древовидную структуру. В каталоге файла проекта должен быть создан подкаталог с именем «Help». Далее должны быть созданы каталоги с названием языка, на котором составлен текст документа (например English, Russian, Belorussian, German, French). В каждом каталоге необходим главный файл документа «Index.htm». Этот файл ассоциируется с файлом помощи к проекту. Для каждого устройства можно создать закладку в формате HTML. При вызове помощи к устройству или папке будет открыт файл «Index.htm» и произведен переход к указанной закладке при наличии таковой.

Для одноязычных версий файлы документов можно располагать непосредственно в каталоге «Help».

2.2. Порядок разработки модели

2.2.1. Как интегрировать модель в HLCCAD

Последовательность шагов по интеграции модели в систему HLCCAD такова:

1. Создать (или открыть существующий) проект, в котором предполагается описать устройство (соответствующее разрабатываемой модели), которое в дальнейшем можно будет использовать в других проектах (рис. 3).
2. Создать новое устройство (рис. 4).
3. Отредактировать (рис. 5) его корпус (этот корпус будет появляться на схеме, где вы будете использовать это устройство): создать контакты необходимого типа и размерности, установить размеры корпуса и т. п.

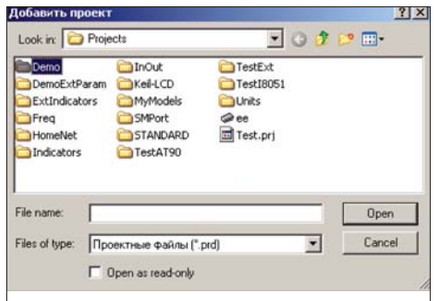

Рис. 3. Окно создания проектов

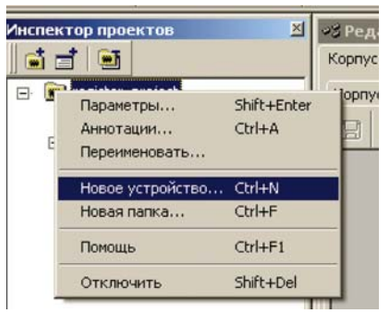

Рис. 4. Добавление нового устройства

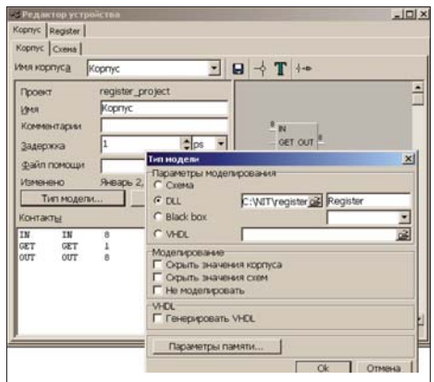

Рис. 5. Окна редактирования корпуса и параметров моделирования

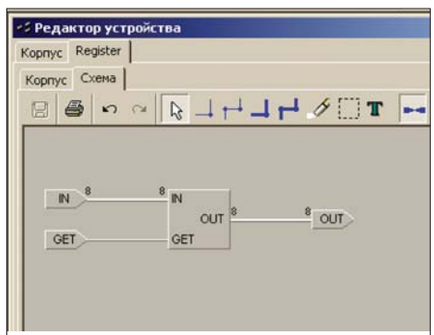

Рис. 6. Схема устройства

4. В параметрах корпуса (см. на рис. 5 — кнопка «Тип модели») указать имя DLL-файла (в котором находится или будет находиться модель) и имя модели (префикс функции инициализации) (рис. 5).

После этого устройство можно использовать как компонент в любых проектах.

2.2.2. Внешняя модель устройства

В общем случае модель должна подавать на выходы устройства значения в зависимости от значений на входах устройства и его внутреннего состояния. Модель может активизироваться в одной из двух следующих ситуаций (иногда в обеих):

- OnChange — произошло изменение на контактах устройства.
- ExecAfter — прошло указанное время.

Первого способа достаточно, чтобы описывать работу различных комбинационных схем (когда в каждый момент времени значения на выходах зависят только от значений на входах, то есть устройство не имеет элементов памяти).

Второй способ бывает удобно использовать при создании моделей более сложных устройств, таких, как тактовые генераторы, микропроцессоры, микроконтроллеры.

При желании разработчик высокоуровневой модели устройства может реализовать методы построения синтезируемого VHDL-описания работы этого устройства, чтобы обеспечить автоматическое построение VHDL-текста для схем, на которых используется это устройство.

2.2.3. Использование моделью внутренней памяти

Система предоставляет модели механизм работы с памятью. Разработчик может создать память (выделить блок указанного размера) или наложить новую память на участок уже созданной ранее памяти. В обоих случаях модели предоставляется интерфейс IMemory, через который она может работать с этой памятью. В дополнение, модель может получить указатель на саму память (pointer) и работать с ней прямо (это может оказаться существенно производительнее). Однако если вся работа с памятью осуществляется через интерфейс, то пользователь сможет в системе HLCCAD просматривать трассу состояния этой памяти в любой момент времени из смоделированного диапазона. Поэтому настоятельно рекомендуется при разработке устройств типа RAM, ROM и т. п. (где содержимое памяти существенно для того, кто использует модель) пользоваться внутренним механизмом памяти системы.

2.2.4. Функция создания интерфейса модели

Функция создания интерфейса модели должна импортироваться из динамической библиотеки и иметь имя ИМЯ_Init, где ИМЯ — это имя процедуры, указанное для корпуса устройства в параметрах моделирования в редакторе. Формат функции следующий:

```
TInit = function(HLCCAD: IHLCCAD; Action: integer):IModel; stdcall;
```

HLCCAD — интерфейс работы с системой.

Action — указывает, для чего создается модель:

- acModel = 1 — для моделирования;
- acVHDL = 2 — генерации VHDL;
- acCPU = 3 — дизассемблирования.

Пример

```
function ROM_Init(HLCCAD: IHLCCAD; Action: integer):IModel;
stdcall;
begin
        Result:=TROM.Create(HLCCAD, Action);
end;
```

2.2.5. Формат значения на контакте

Значение на контакте представляется символьной строкой, длина которой совпадает с размерностью контакта — то есть каждой реальной линии соответствует один собственный символ.

Входное значение (подаваемое системой моделирования) обозначается следующими символами:

0 — логическое значение 0 на линии;

1 — логическое значение 1 на линии;

U — невозможно определить состояние линии;

P — запрещенное состояние на линии;

Z — состояние высокого сопротивления на линии.

Выходное значение, устанавливаемое моделью, должно содержать только символы «0», «1» или «Z».

2.2.6. Интерфейс модели IModel

Интерфейс модели IModel — это фактически набор процедур и функций, которые вызываются системой HLCCAD в различных ситуациях в процессе использования модели: при включении устройства в схему, компиляции, симуляции и отладке схемы, визуализации результатов симуляции, генерации синтезируемого описания схемы и т. д.

Создание модели устройства заключается в реализации и отладке некоторого (необходимого для данного устройства) подмножества процедур и функций интерфейса IModel.

```
IModel = interface(IUnknown)
   function Correct:boolean; stdcall;
   function Options:longint; stdcall;
   function MemorySize:longint; stdcall;
   procedure AutoStart; stdcall;
   procedure OnChanged; stdcall;
   function ExecAfter:comp; stdcall;
   function VHDLGetName: PChar; stdcall;
   procedure VHDLBuildDescribe; stdcall;
   procedure ShowForms; stdcall;
   function Version:integer; stdcall;
   procedure Syn(AParam: integer); stdcall;
   procedure OnDestroy; stdcall;
   procedure DizAssembler(Data:Pointer; var AsmLine, LabelLine,
DebugLine : PChar); stdcall;
   function GetInstrSize(Data : byte):integer; stdcall;
end;
```

Подробное описание назначения и параметров каждой из процедур и функций интерфейса IModel можно найти в статье на нашем сайте [5].

2.2.7. Интерфейс системы IHLCCAD

Интерфейс системы IHLCCAD — это фактически набор процедур и функций, который предоставляется модели для получения информации о результатах интерактивной работы пользователя в системе HLCCAD, а также о результатах симуляции.

Ниже представлен полный перечень этих процедур и функций.

```
Для Object Pascal (Delphi):

IHLCCAD = interface(IUnknown)
   function LinkFault:boolean; stdcall;
   procedure ErrorMsg (Code: integer; Param: pointer); stdcall;
   function TerminationOn:boolean; stdcall;
   function MainHandle: HWND; stdcall;
   function GetTime:comp; stdcall;
   Procedure GetDeviceParamValue (S: PChar; Var Data: Pointer; Var
Size: integer); stdcall;

   function GetContactName (Contact: pointer):PChar; stdcall;
   function GetContactCType (Contact: pointer):word; stdcall;
   function GetContactMType (Contact: pointer):word; stdcall;
   function GetContactPType (Contact: pointer):word; stdcall;
   function GetDeviceDelay:comp; stdcall;
   function GetContactDim (Contact: pointer):longint; stdcall;
   function GetContactCount:longint; stdcall;
   function GetContact (Name: PChar):pointer; stdcall;
   function GetContactAt (i: integer):pointer; stdcall;
   function GetContactValue (Contact: pointer):PChar; stdcall;
   function GetValidContactValue (Contact: pointer):PChar; stdcall;
   procedure SetContactValue (Contact: pointer; Value: PChar); stdcall;
   procedure SetZ (Contact: pointer); stdcall;

   Procedure ReadBytes (Addr, Count: integer; Var Buf); stdcall;
   Procedure WriteBytes (Addr, Count: integer; Var Buf); stdcall;
   function LoadMemoryFromTextFile (ACapacity: integer; AName:
PChar):integer; stdcall;

   Function CreateMemory (Size: integer; Name: PChar): integer; stdcall;
   Function mLoadMemoryFromTextFile (Index: integer; ACapacity:
integer; AName: PChar):integer; stdcall;
   Function mReadValue (Index, Addr, Count: integer):PChar; stdcall;
Procedure mWriteValue (Index, Addr: integer; Value: PChar); stdcall;
   Procedure mReadBytes (Index, Addr, Count: integer; Var Buf); stdcall;
Procedure mWriteBytes (Index, Addr, Count: integer; Var Buf); stdcall;
   function CreateValue (AName: PChar; AAddr, ASize, AType: integer):
Pointer; stdcall;
   function mCreateValue (Index: integer; AName: PChar; AAddr,
ASize, AType: integer): Pointer; stdcall;
   function ReadValue (Addr, Count: longint):PChar; stdcall;
   procedure WriteValue (Addr: longint; Value: PChar); stdcall;

   Procedure InsertVHDLString (S: PChar); stdcall;
   Function GetVHDLOptions: longint; stdcall;

   Function GetProjectPath: PChar; stdcall;
   Procedure SaveMemFileAsHex (SourceFileName, DestFileName: PChar);

   function GetPosition:integer; stdcall;
   procedure SetPosition(APos: integer); stdcall;
   function Seek(Offset: Longint; Origin: Word): Longint;
   procedure ReadBuffer(Dest: pointer; Count: integer); stdcall;
   procedure WriteBuffer(Source: pointer; Count: integer); stdcall;
   procedure SetModelOptions(AOptions: integer); stdcall;
   function AddEvent(ME : TModelEvent; ALevel: integer; Time : comp)
: THLCCADEvent; stdcall;
   procedure DelEvent(HE : THLCCADEvent); stdcall;
   procedure SetSubTime(ASubTime: comp); stdcall;
   procedure GetContactInterface (AContact: pointer; Var P:IContact);
stdcall;
   function GetViewTime:comp; stdcall;
   procedure SetViewTime (ATime: comp); stdcall;
   function GetModelPath:PChar; stdcall;
   function GetSourceProjectName: PChar; stdcall;

   Procedure SetDeviceParamValue(S : PChar; Value : Pointer; Size : integer);
stdcall;
   Procedure SetDeviceParamData(S : PChar; Data : Pointer; Size : integer);
stdcall;
   Procedure GetDeviceParamData(S : PChar; Var Data : Pointer; Var Size :
integer); stdcall;

   procedure Log(aMsg: PChar; aImmediately: boolean); stdcall;
   function GetHInstance:cardinal; stdcall;
   procedure Idle; stdcall;

   procedure MemNew (aName: PChar; aBitSize, aBitOffset, aBPW: in-
teger; aOptions: word; Var IMem : IMemory); stdcall;
   procedure MemReg (aName: PChar; aBuffer: pointer;
aBitSize,aBitOffset,aBPW: integer; aOptions: word; Var IMem :
IMemory); stdcall;

   function CheckFlag(cfFlag: integer):integer; stdcall;
   procedure SetDeviceDelay(const aDelay: comp); stdcall;
   function GetVHDLoutputDir:pchar; stdcall;
   function GetLanguage:pChar; stdcall;
   function GetCompileMode:integer; stdcall;
   function GetMTime : integer; stdcall;
   function RunInstruction(aInstrType : TInstrType) : boolean; stdcall;
   procedure AddMsg(MSGType : integer; Text : PChar); stdcall;
   function CompileWProject(FileName : PChar) : HResult; stdcall;
   function GetCorpusName : PChar; stdcall;
   function StopSimulateProcess : HResult; stdcall;
```

Описание назначения и параметров основных функций интерфейса можно найти в статье на нашем сайте [5].

2.2.8. Интерфейс работы с памятью IMemory

Процедуры и функции этого интерфейса обеспечивают разнообразные производительные функции работы с памятью устройств.

```
IMemory = interface(IUnknown)
   procedure Map(aName: PChar; aBitAddr, aBitSize, aBitOffset, aBPW:
integer; aOptions: word; Var IMem : IMemory); stdcall;
   function GetPtr:pointer; stdcall;
   procedure ReadBits (aBitAddr, aBitSize: integer; aDst: pointer); stdcall;
   procedure WriteBits (aBitAddr, aBitSize: integer; aSrc: pointer); stdcall;
   procedure WriteBytes(aByteAddr, aByteSize: integer; aSrc: pointer);
stdcall;
```

```
    procedure ReadStr(aBitAddr, aBitCount: integer; aDst: PChar); stdcall;
    procedure WriteStr(aBitAddr, aBitCount: integer; aSrc: PChar); stdcall;
    procedure ReadIn(aDst: pointer); stdcall;
    procedure Writeln(aSrc: pointer); stdcall;
    function LoadFromFile(aFileName: PChar):boolean; stdcall;
end;
```

Модель может указать имя файла, содержащего помощь по устройству (принцип работы, возможные ошибки и т. п.). Эту помощь можно будет использовать в системе при эксплуатации модели.

2.3. Пример текста простейшей высокоуровневой модели на Object Pascal

В качестве примера (см. врезку) рассмотрим статический регистр с двумя входами (In, Get) и одним выходом (Out). Контакты In и Out могут иметь размерность от 1 до 256. На выход OUT всегда подается значение из памяти. При 1 на GET содержимое IN запоминается в память.

Данный пример написан на Delphi 5.0. Модель написана с самого начала, чтобы нагляднее продемонстрировать работу с механизмами обмена информацией между моделью и HLCCAD. На практике удобнее создать шаблон (базовый обьект), в котором описаны высокоуровневые механизмы.

```
library Demo;
Uses
IntTypes;

Type
   TDemoReg = class(TInterfacedObject, IModel)
        cIN, cGET, cOUT : pointer; // ссылки на контакты
        pIN, pGET : pChar; // ссылки на значения входов
        Dim : integer; // размерность контактов IN,OUT
        strOUT : string; // для пересылки на OUT значения из памяти
        HLCCAD : IHLCCAD;
        Valid : boolean; // признак что модель создана без ошибок
        Mem : IMemory; // указатель
        PSelf : IModel;

        constructor Create(aHLCCAD: IHLCCAD);
        destructor Destroy; override;

        Function Correct:boolean; stdcall;
        Function Options:longint; stdcall;

        Procedure AutoStart; stdcall;
        procedure OnChanged; stdcall;

// эти методы здесь заполняться не будут
//(для нашей модели они не важны)
        Procedure AfterCreate; stdcall;
        Procedure OnDestroy; stdcall;
        Function MemorySize:longint; stdcall;
        Function Version:integer; stdcall;
        Function ExecAfter:comp; stdcall;
        Procedure ShowForms; stdcall;
        Function DizAssembler (Data: Pointer; Var AsmLine: PChar):
integer; stdcall;
        Function GetInstrSize (Data: Pointer):integer; stdcall;
        Function GetMaxInstrLength: integer; stdcall;
        Function IsInstructionRun: integer; stdcall;
        Function GetCodeMemoryWordSize: integer; stdcall;
        Procedure Save; stdcall;
        Procedure Load; stdcall;
        Function VHDLGetName: PChar; stdcall;
        Procedure VHDLBuildDescribe; stdcall;
        Procedure VHDLLibraryDescribe; stdcall;
        Procedure SaveEvent (ME: TModelEvent); stdcall;
        Function LoadEvent (HE: THLCCADEvent): TModelEvent; stdcall;
        procedure ExecEvent (ME: TModelEvent); stdcall;
        function GetExecutedInstructionCount:integer; stdcall;
end;

// конструктор обьекта-модели
constructor TDemoReg.Create(aHLCCAD: IHLCCAD);
begin
   inherited Create;
   PSelf := Self;
   Valid := false;
   HLCCAD := aHLCCAD;
   if (HLCCAD.GetContactCount<>3) then exit;
   cIN := aHLCCAD.GetContact('IN');
   cGET := aHLCCAD.GetContact('GET');
   cOUT := aHLCCAD.GetContact('OUT');
   if not assigned(cIN) or not assigned(cGET) or
      not assigned(cOUT) then exit;
   DIM := HLCCAD.GetContactDim(cIN);
   if (DIM<1) or (HLCCAD.GetContactDim(cOUT)<>DIM) or
                 (HLCCAD.GetContactDim(cGET)<>1) then exit;
   if HLCCAD.GetContactMType(cIN) <>mtIN then exit;
   if HLCCAD.GetContactMType(cGET)<>mtIN then exit;
   if HLCCAD.GetContactMType(cOUT)<>mtOUT then exit;
   if HLCCAD.GetContactCType(cIN) in [ctDOT, ctDOTBus] then exit;
   if HLCCAD.GetContactCType(cOUT) in [ctDOT, ctDOTBus] then exit;
   SetLength(strOUT, Dim);
   HLCCAD. MemNew ('Demo register', Dim, 0, Dim,
                   atRead+atWrite+atRegister, Mem);
   pIn := HLCCAD.GetContactValue(cIN);
   pGET := HLCCAD.GetContactValue(cGET);
   Valid := true;
end;

destructor TDemoReg.Destroy;
begin
   Mem := nil; // нужно освободить память модели!
   inherited;
end;

Function TDemoReg.Correct:boolean; // проверка, что корпус соответствует модели
begin
   Result := Valid;
end;

Function TDemoReg.Options:longint;
begin
   result := flOnChange; // вызывать модель при изменениях на
контактах
end;

procedure TDemoReg.OnChanged;
      // этот метод вызывается при изменениях на контактах
begin
   // считываем в память значение (если нужно)
   if pGET='1' then Mem.WriteStr(0, Dim, pIN);
   // подаем на выход значение из памяти
   Mem.ReadStr(0, Dim, pChar(strOUT));
   HLCCAD.SetContactValue(cOUT, pChar(strOUT));
end;

Procedure TDemoReg.AutoStart;
 // при инициализации нужно подать на выход значение
begin
 OnChanged;
end;

//======== Не реализовываются за ненадобностью
====================
Procedure TDemoReg.AfterCreate; begin end;
Procedure TDemoReg.OnDestroy; begin end;
Function TDemoReg.MemorySize:longint; begin result := 0; end;
Function TDemoReg.Version:integer; begin end;
Function TDemoReg.ExecAfter:comp; begin end;
Procedure TDemoReg.ShowForms; begin end;
Function TDemoReg.DizAssembler (Data: Pointer; Var AsmLine:
PChar): integer; begin end;
Function TDemoReg.GetInstrSize (Data: Pointer):integer; begin end;
Function TDemoReg.GetMaxInstrLength: integer; begin end;
Function TDemoReg.IsInstructionRun: integer; begin end;
Function TDemoReg.GetCodeMemoryWordSize: integer; begin end;
Procedure TDemoReg.Save; begin end;
Procedure TDemoReg.Load; begin end;
Function TDemoReg.VHDLGetName: PChar; begin end;
Procedure TDemoReg.VHDLBuildDescribe; begin end;
Procedure TDemoReg.VHDLLibraryDescribe; begin end;
Procedure TDemoReg.SaveEvent (ME: TModelEvent); begin end;
Function TDemoReg.LoadEvent (HE: THLCCADEvent): TModelEvent;
begin end;
Procedure TDemoReg.ExecEvent (ME: TModelEvent); begin end;
Function TDemoReg.GetExecutedInstructionCount:integer; begin end;
function Demo_Init(HLCCAD: PHLCCAD; Action: integer):PModel;
stdcall;
begin
   result := TDemoReg.Create(HLCCAD^).PSelf;
end;
exports
   Demo_Init;
end.
```

Заключение

На первый взгляд может показаться, что для реализации столь простой сущности, как статический регистр, исходный текст выглядит довольно сложно. Хотелось бы отметить следующие факты:

1) Все практически значимые простые сущности уже реализованы разработчиками HLCCAD в виде параметризованной библиотеки стандартных компонентов.
2) Более сложные сущности прежде всего предлагается создавать комплексированием простых сущностей. При этом система поддерживает иерархическое проектирование как «снизу-вверх», так и «сверху-вниз».
3) Существует огромное множество сложных сущностей (микроконтроллеры, например), реализация моделей которых с помощью языка программирования типа Object Pascal или C++ и разработанных интерфейсов во много раз проще, чем комплексированием цифровых узлов средней степени интеграции.

Ниже представляется множество реальных примеров эффективного использования высокоуровневых моделей в самых различных применениях. В частности, в практике эксплуатации HLCCAD с использованием языков программирования уже разработаны следующие модели:

- Модели устройств отображения информации (индикаторов).
- Модель кнопочной панели.
- Модель терминала для игрального автомата BlackJack.
- Модель сетевого компонента ввода-вывода информации.
- Модель параллельного порта.
- Библиотека моделей блоков цифровой обработки сигналов.
- Модели компонентов игрального автомата BlackJack.
- Модель для взаимодействия с универсальным эмулятором UniICS.
- Модель для взаимодействия с удаленным UniICS.
- Модели микроконтроллеров Intel 8051, Atmel AVR, MicrChip PIC, Motorola 68HC05/08, Texas Instruments TMS370, ARM ARM7TDMI.
- Модель виртуального процессора микропрограммных автоматов.
- Модель гетерогенной мультипроцессорной системы.
- Библиотека моделей компонентов SoC-платы E5 фирмы Triscend.

Литература

1. Долинский М. Концептуальные основы и компонентный состав IEESD-2000 — интегрированной среды сквозной совместной разработки аппаратного и программного обеспечения встроенных цифровых систем // Компоненты и технологии. 2002. № 8.
2. Долинский М., Литвинов В., Галатин А., Ермолаев И. HLCCAD — среда редактирования, симуляции и отладки аппаратного обеспечения // Компоненты и технологии. 2003. № 1.
3. Долинский М., Литвинов В., Толкачев А., Корнейчук А. Система высокоуровневого проектирования аппаратного обеспечения HLCCAD: тестирование // Компоненты и технологии. 2003. № 3.
4. http://NewIT.gsu.unibel.by.
5. http://NewIT.gsu.unibel.by/resources/articles/dolinsky/Russian/EmbeddedSystems/hlccad_m.doc.