# Система высокоуровневого проектирования аппаратного обеспечения HLCCAD:

## технология разработки потактовой модели микроконтроллера с использованием языка программирования высокого уровня на примере Intel 8051/Object Pascal

**Михаил Долинский,**
**Вячеслав Литвинов,**
**Алексей Федорцов,**
**Игорь Ермолаев**

dolinsky@gsu.unibel.by

### Введение

Важным достоинством системы HLCCAD является возможность симулировать и отлаживать схемы, включающие модели микропроцессоров (МП) и микроконтроллеров (МК). Еще более важным достоинством системы HLCCAD является предоставляемая пользователям возможность самостоятельно создавать модели МП/МК, пользуясь поставляемыми разработчиками HLCCAD технологиями:

1) С использованием языков программирования высокого уровня (Object Pascal, C++);
2) С использованием языка программирования ассемблер Intel 80386 и средств автоматизированной генерации моделей МП/МК.

К достоинствам первого подхода можно отнести существенно меньшие требования к квалификации программиста, создающего модель. К достоинствам второго — более высокую производительность (до 10 раз) исполнения созданной моделью прикладных программ. Данная статья посвящена систематическому изложению создания моделей МП/МК в рамках первого подхода. Для наглядности будет последовательно разрабатываться базовая модель МК семейства Intel 8051. При наличии полного исходного текста такой модели и понимании процессов разработки пользователю не составит большого труда скорректировать данную модель под требуемую модификацию конкретного МК семейства Intel 8051. Более того, аналогичным способом можно создавать модели практически любых МП/МК, реально используемых сегодня разработчиками цифровых систем.

### 1. Общая схема создания модели МП/МК

В HLCCAD создается условное графическое обозначение (УГО) корпуса МК Intel 8051, включающее следующие внешние контакты (рис. 1):

| | | |
|---|---|---|
| P0 | 8 бит | двунаправленный |
| P1 | 8 бит | двунаправленный |
| P2 | 8 бит | двунаправленный |
| P3 | 8 бит | двунаправленный |
| ALE | 1 бит | выходной |
| PME | 1 бит | выходной |
| RST | 1 бит | входной |
| EA | 1 бит | входной |
| XTAL1 | 1 бит | входной |
| XTAL1 | 1 бит | входной |

В параметрах корпуса указывается название модели (i8051) и префикс вызываемой при инициализации функции модели (CPU) (рис. 2). Теперь можно создавать схемы, включающие модель этого процессора и даже моделировать их без процессора (временно, до завершения создания первого варианта модели процессора, установив у корпуса МК флажок «не моделировать»).


**Рис. 1.** УГО корпуса микроконтроллера

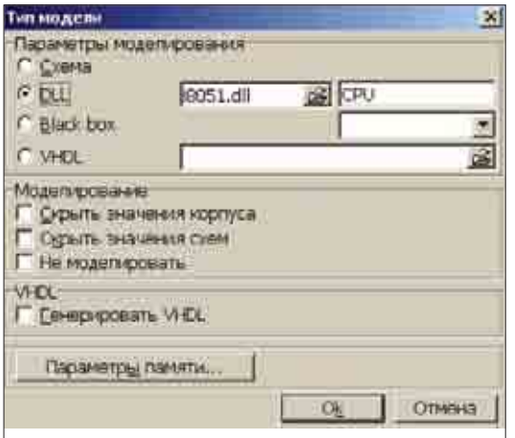


**Рис. 2.** Окно параметров корпуса микроконтроллера

Создав исходный текст модели МК Intel 8051 и скомпилировав его как DLL (Dynamic Link Library), можно отключить флажок «не моделировать» и тестировать разработанную модель МК в рамках подготовленных тестовых проектов.

## 2. Способы тестирования модели микроконтроллера

### *2.1. Тестирование симуляции инструкций*

Тестирование симуляции инструкций можно производить, указав флаг «Использовать внутреннюю память инструкций» и подготовив файл содержимого этой памяти. Корректно исполняющая инструкции модель МК должна правильно загружать, дешифрировать и исполнять подготовленные в файле инструкции. При ручном формировании этого файла для такого тестрования не требуется никаких дополнительных средств программирования.

В то же время для подготовки содержимого памяти программы можно использовать имеющиеся средства компиляции (на начальном этапе отладки модели — ассемблер, на завершающем — С-компилятор).

Для автоматизации тестирования модели можно использовать язык тестов [3], устанавливая до исполнения команд значения регистров и флагов, которые будут использоваться или анализироваться при исполнении, а после исполнения сверяя полученные результаты с заранее подготовленными эталонными.

Но наиболее удобным средством тестирования модели МК являются теневые команды. Этот вариант можно использовать, только если перед разработкой модели (или параллельно ей) ведется разработка встроенного ассемблера для данного МК. Комплекс IEESD [1] представляет для этого специальную инструментальную систему RtASM — ассемблер, настраиваемый на целевую архитектуру.

На сегодняшний день существуют следующие типы теневых инструкций:

- $E — сигнал об окончания автоматического тестирования либо безусловная точка останова при ручном тестировании;
- $R — сигнал обновления значений ресурсов МК (регистров, битов, памяти) в соответствующих окнах отображения среды отладки;
- $OK, $ERR — сигнализирует о правильности либо неправильности выполнения соответствующей инструкции, используется в основном для тестирования инструкций передачи управления;
- $S <ресурс> = <значение> — установка ресурса в указанное значение;
- $T <ресурс> <операция> <значение> — проверка значения ресурса;
- $O <значение> — проверка значения кодовой памяти;
- $B <ресурс> <операция> <значение> — условная точка останова.

В последних трех из вышеперечисленных типов теневых инструкций в качестве значения может выступать целочисленная, плавающая, строковая либо символьная константа.

Целочисленная константа может быть представлена в двоичной, восьмеричной, десятичной либо шестнадцатеричной системе счисления.

В качестве операции (для $T и $B) может использоваться: = (равно), < (меньше), > (больше), <= (меньше или равно), >= (больше или равно), <> (не равно).

В качестве ресурса может быть использовано имя бита, флага, регистра, памяти, переменной. Указание регистра, памяти и переменной может сопровождаться дополнительными атрибутами:

- «.bit» — выделение одного бита ресурса под номером bit (acc.3 — 3-й бит регистра-аккумулятора);
- «[offset]» — выделение одного слова ресурса со смещения offset (DATA[30h] — 48-е слово памяти данных);
- «[offset]:words» — выделение words слов ресурса со смещения offset (XDATA[100h]:4 — 4 слова внешней памяти данных со смещения 256);
- «[offset.bit]» — выделение одного бита ресурса под номером bit в слове со смещением offset (DPTR[1.5] — 5-й бит 1-го слова адресного регистра);
- «[offset.bit]:bits» — выделение bits бит начиная с бита bit в слове со смещением offset (IDATA[0B0h.2]:4 — выделение бит со 2-го по 5-й включительно в 176-м слове внутренней памяти данных).

Сценарий тестирования [3] может включать в себя последовательные ассемблирование и исполнение фрагментов типа:

Файл T_inc.a51 :

```
nop;        $S a:=5
inc         a
nop;        $T a=6
```

В этом примере с помощью теневой команды $S устанавливается значение регистра a, после выполнения инструкции инкремента с помощью теневой команды $T проверяется результат.

### *2.2. Тестирование корректности симуляции периферии*

В этом случае можно создать схему, в рамках которой проверять корректность функциональности модели МК, используя язык тестов, внешние высокоуровневые модели [4], внешнюю память программ, встроенную или внешнюю компиляцию. Некоторые виды периферии также можно протестировать без создания схем с помощью языка тестов. Пример тестирования таймера:

```
; Timer0 test
        org 0h
        jmp start           ; переход на начало программы

        org 0Bh
        jmp timer0          ; переход на обработчик прерывания

        org 30h
start:
        setb ea             ; разрешение прерываний
        setb et0            ; разрешение прерываний для таймера 0
        mov r0,#1
        mov th0,#0FFh;      начало тестирования режима 0
        setb tr0            ; запуск таймера
loop0:
        inc a
        inc r0
        djnz r0,loop0
        mov r0,#1           ;$T A = 8 (проверка результата)
        clr a
        mov tmod,#1         ; начало тестирования режима 1
        mov tl0,#0
        mov th0,#0FFh
        setb tr0
loop1:
        inc a
        inc r0
        djnz r0,loop1
        mov r0,#1           ;$T A = 40h (проверка результата)
        clr a
        mov tmod,#2         ; начало тестирования режима 2
        mov tl0,#080h
        mov th0,#080h
        setb tr0
loop2:
        inc a
        inc r0
        djnz r0,loop2
        mov r0,#1           ;$T A = 20h (проверка результата)
        clr a               ;$T TL0 = 81h
        mov tmod,#3         ; начало тестирования режима 3
        mov tl0,#080h
        mov th0,#080h
        setb tr0
loop3:
        inc a
        inc r0
        djnz r0,loop3
        mov r0,#1           ;$T A = 20h (проверка результата)
lend: jmp $                 ; конец

timer0:                     ; обработчик прерывания
        clr tf0             ; сброс флага прерывания
        clr tr0             ; останов таймера
        mov r0,#0           ; сброс контрольного регистра
        reti
```

В данном примере тестируются четыре возможных режима работы таймера 0. Для этого разрешаются прерывания, для каждого режима инициализируются переменные и выполняются циклы, в которых эти переменные модифицируются. После вызова прерывания цикл прекращается, и происходит проверка значений переменных.

## 3. Общая схема исходного текста модели МК на языке программирования высокого уровня

Приведем область объявлений описания микроконтроллера 8051:

```
library i8051;
uses
  SysUtils, Classes, Forms, IntTypes, BaseExtModel, SystemPlus,
  Constants, CPUModel, ModelLibrary, WinterSupport, WinterTypes,
  MemoryModel, DizAsm51, Describe51, UCPUOptionsForm,
  UExtParameterTypes;

Type
  T_i8051 = class(TBaseCPUModel)
  private
    Procedure  SetACC(Val: byte);
    Function   GetACC : byte;
    Procedure  SetSP(Val: byte);
    Function   GetSP : byte;
    Procedure  SetDPTR(Val: word);
    Function   GetDPTR : word;
    Procedure  SetB(I : integer; Val: byte);
    Function   GetB(I : integer) : byte;
    Function   GetP(I : integer) : byte;
    Function   GetPReg(I : integer) : byte;
    Procedure  SetDir(I : integer; Val: byte);
    Function   GetDir(I : integer) : byte;
    Procedure  SetInd(I : integer; Val: byte);
    Function   GetInd(I : integer) : byte;
  public
  // EnableInterrupts,EnableTimers,EnableUARTs : boolean;
    // Pins
    Pins : array[1..CountPins]of pointer;
    p_P0,p_P1,p_P2,p_P3,p_ALE,p_PME,p_RST,p_EA,p_XTAL1,
p_XTAL2:pointer;
    // Memory
    Memory : array[1..CountMemory]of TMemoryModel;
    IRAM,SFR,ROM,XRAM,PCMEM,R : TMemoryModel;
    SFRtemp : array[MinSFRAddress..MaxSFRAddress] of byte;
    PINtemp : array[0..3] of byte;
    R_Writer,R_Reader,Writer: boolean; // SFRWriter Enable
```

```
    // Events
    TimerEvent, pinT0Event, pinT1Event, pinINT0Event, pinINT1Event,
    UARTEvent, TransmissionEvent, ReceivingEvent: boolean;
    IdleMode, PowerDownMode : boolean;
    ModeTimer0, ModeTimer1, ModeUART : byte;
    CInstr: array[1..MaxInstructionSize]of byte; // Instruction code
    InstrIndex: byte; // Index of Instruction in array
    Cycle : byte; // Number cyrrent cycle
    LastRST : boolean; // Check rst
    LastX : boolean;
    // Support interrupts
    IntStack: array[0..CountInterrupts] of byte; // Interrupts stack
    // Support IDLE Mode and Power Down Mode
    ExternalCode : boolean;
    PowerReset : boolean;

    property ACC: byte read GetACC write SetACC;
    property SP : byte read GetSP write SetSP;
    property DPTR: word read GetDPTR write SetDPTR;
    property B[Index : integer] : byte read GetB write SetB;
    property DirMem[Index : integer] : byte read GetDir write SetDir;
    property IndMem[Index : integer] : byte read GetInd write SetInd;
    property P[Index : integer] : byte read GetP;
    property PReg[Index : integer] : byte read GetPReg;

    Constructor CreateHLCCAD(aAction : Integer; aHLCCAD: IHLC-
CAD);
    Constructor CreateWinter;
    Function GetPC : cardinal; override;
    Procedure SetPC(W: cardinal); override;
    Procedure InitVariable; // Init data
    function GetAddressROM:Word;
    function GetAddressXRAM:Word;
    procedure Interrupts;
    procedure Timers;
    procedure PinsProcessing;
    procedure CountersProcessing;
    Procedure SFRWriter(Sender : TMemoryModel; Addr,Size : integer);
    Procedure RWriter(Sender : TMemoryModel; Addr,Size : integer);
    Procedure RReader(Sender : TMemoryModel; Addr,Size : integer);
    Procedure InitProcessor; override;
    Procedure RunInstruction; override;
    Procedure RunLoadInstruction(CheckBreakPoints : boolean); override;
    Procedure RunTact; override;
    Function MaxInstructionSize : integer; override;
    Function MaxDisAsmTextSize : integer; override;
    Procedure AfterCreate; override;
    Function ExecAfter : comp; override;
    Procedure OnChanged; override;
    Function GetInstrSize(Data:Pointer):integer; override;
    Function DizAssembler(Data:Pointer; var AsmLine:PChar) : inte-
ger; override;
    Function ToBitAddress(Memory : TMemoryModel; Address,Size:
integer) : integer; override;
    Function GetInstrSizeFromPCForWinter : integer; override;
 End;
```

Здесь приведены объявления переменных, функций и процедур модели микроконтроллера. Описанию констант посвящен следующий раздел. Реализация основных процедур и функций описана далее в тексте статьи.

В общем случае исходный текст модели процессора должен включать описания следующих компонент:

- память, регистры, флаги, биты, контакты, счетчики;
- инструкции;
- циклы процессора;
- дизассемблирование;
- работа ядра;
- инициализация процессора;
- исполнение инструкций;
- работа периферийных устройств;
- прерывания;
- таймеры-счетчики;
- внутренние счетчики;
- взаимодействие с окружением.

Предлагаемая технология разработки исходного текста модели предполагает использование специальной нотации (в рамках возможностей, представляемых языком Object Pascal) для повышения понимаемости и готовности к модификации ее исходного текста.

## 4. Описание константной информации

```
Const
  CountMemory = 6;
Const
  vrMemory:array[1..CountMemory] of TCPUMemory=(
    (Title:'Code Memory'; MinSeg:0; MaxSeg:0; MinOfs:0; MaxOfs:$FFFF;
       BitPerWord:8; Point:'PC'; Feature:mfCode),
    (Title:'InternalDataMem.';MinSeg:0; MaxSeg:0; MinOfs:0; MaxOfs:$7F;
       BitPerWord:8; Point:'SP'; Feature:mfData+mfStack),
    (Title:'Spec.Func.Regs';MinSeg:0; MaxSeg:0; MinOfs:$80; MaxOfs:$FF;
       BitPerWord:8; Point:''; Feature:0),
    (Title:'Programm Counter';MinSeg:0; MaxSeg:0; MinOfs:$0;MaxOfs:$0;
       BitPerWord:16; Point:''; Feature:mfShadow),
    (Title:'Current Bank'; MinSeg:0; MaxSeg:0; MinOfs:$0; MaxOfs:$7;
       BitPerWord:8; Point:''; Feature:mfShadow),
    (Title:'eXt. DataMem'; MinSeg:0;MaxSeg:0;MinOfs:0; MaxOfs:$FFFF;
       BitPerWord:8; Point:'DPTR';Feature:mfData+mfExternal));

    memROM = 1; memIRAM = 2; memSFR = 3;
    memPC = 4; memR = 5; memXRAM = 6;
```

Фактически, приведенный выше фрагмент текста указывает на наличие 6 блоков памяти (от кодовой до внешней памяти данных), для каждого из них указываются минимальное и максимальное смещение (MinOfs, MaxOfs), количество бит в слове (BitPerWord), указатель адреса в этой памяти (Point: PC — в кодовой, SP — в стековой, DPTR — во внешней памяти данных), свойства (Feature: MfCode — кодовая память, MfData — память данных, MfStack — память стека, MfExternal — внешняя память, mfShadow — скрытая от пользователя память). Минимальное и максимальное значение сегмента (MinSeg, MaxSeg) в данном случае не используются, поскольку память МК Intel 8051 не является сегментируемой.

Кроме того, вводятся легко запоминаемые символические обозначения идентификаторов памяти (1 — кодовая (memROM), 2 — внутренняя память данных (memIRAM) и т. д.)

```
// Special Function Registers
    sfrP0    = $80;
    sfrSP    = $81;
    sfrDPL   = $82;
    ...
    sfrB     = $F0;
```

Приведенным выше образом вводятся читабельные и понятные символические константы, обозначающие адреса соответствующих регистров в специальной (SFR) памяти.

```
// Pins
    pinP0      =   1;
    pinP1      =   2;
    pinP2      =   3;
    pinP3      =   4;
    pinALE     =   5;
    pinPME     =   6;
    pinRST     =   7;
    pinEA      =   8;
    pinXTAL1   =   9;
    pinXTAL2   =  10;
```

Приведенным выше образом вводятся читабельные и понятные символические константы, обозначающие внешние контакты.

```
Const vrRegs:array[1..CountRegs] of TVariable=(
(Title:'R0';Size:8;  inMemory:memR;Address:0;  Feature:rfDefault),
(Title:'R1';Size:8;  inMemory:memR;Address:1;  Feature:rfDefault),
(Title:'R2';Size:8;  inMemory:memR;Address:2;  Feature:rfDefault),
(Title:'R3';Size:8;  inMemory:memR;Address:3;  Feature:rfDefault),
  ...
(Title:'R7';Size:8;  inMemory:memR;Address:7;  Feature:rfDefault),

(Title:'P0';Size:8; inMemory:memSFR;Address:sfrP0;Feature:0),
(Title:'SP';Size:8; inMemory:memSFR;Address:sfrSP;Feature:rfDefault),
  ...
```

Таким образом объявляются все поименованные регистры МК. Для каждого из них определяется отображаемое пользователям название (Title); память, в которой расположен регистр (InMemory), номер соответствующей памяти (MemR); адрес текущего регистра в памяти (Address); его свойства (Feature).

Аналогично можно задавать все внутренние переменные модели МК, которые можно делать как видимыми, так и невидимыми для пользователей.

```
vrPins:array[1..CountPins] of TPin=(
  (Title:'P0';      Size:8;  Options:mtBI),
  (Title:'P1';      Size:8;  Options:mtBI),
  (Title:'P2';      Size:8;  Options:mtBI),
  (Title:'P3';      Size:8;  Options:mtBI),
  (Title:'ALE';     Size:1;  Options:mtOUT),
  (Title:'PME';     Size:1;  Options:mtOUT),
  (Title:'RST';     Size:1;  Options:mtIN),
  (Title:'EA';      Size:1;  Options:mtIN),
  (Title:'XTAL1';   Size:1;  Options:mtIN),
  (Title:'XTAL2';   Size:1;  Options:mtIN));
```

Здесь объявляются все внешние контакты модели микроконтроллера. Необходимо обеспечить соответствие их количества, разрядности, названий и типов величинам, заданным в системе HLCCAD на УГО корпуса МК.

**Счетчики.** При описании модели периферийных устройств часто возникает необходимость отслеживать количество проходов через те или иные участки программы для корректной обработки. Например, при включенном таймере с предделителем нужно при выполнении каждого такта, машинного цикла или инструкции отслеживать состояние предделителя и таймера и в случае переполнения выполнять определенные действия. Таких устройств в МК может быть много. Для упрощения обработки параллельных процессов предусмотрен механизм счетчиков.

Для описания счетчиков используется текст следующего вида:

```
NumCounters = 6;
  cntPredCounter0= 0;
  cntPredCounter1= 1;
  cntTL0 = 2;
  cntTL1 = 3;
  cntTH0 = 4;
  cntTH1 = 5;
  cntExtClk = $FFFF;
  Counters: array[1..NumCounters] of TCounter = (
(Source:cntExtClk; Value:0; Limit:0; Stop:True; Full:False),
(Source:cntExtClk; Value:0; Limit:0; Stop:True; Full:False),
(Source:cntPredCounter0;Value:0; Limit:$FF; Stop:False; Full:False),
(Source:cntPredCounter1;Value:0; Limit:$FF; Stop:False; Full:False),
(Source:cntTL0; Value:0; Limit:$FF; Stop:False; Full:False),
(Source:cntTL1; Value:0; Limit:$FF; Stop:False; Full:False));
```

Массив записей Counters задает список счетчиков и их параметры, которые соответствуют полям записи:

- Source — идентификатор источника. Источниками могут быть как счетчики машинных циклов и тактов, так и другие счетчики. Это дает возможность создавать цепочки связанных друг с другом по переполнению счетчиков.
- Value — переменная, которая содержит текущее значение счетчика.
- Limit — значение переполнения счетчика.
- Stop — остановлен ли счетчик сейчас.
- Full — флаг переполнения. Устанавливается при переполнении счетчика, сбрасывается после прихода следующей команды

на изменение, при этом текущее значение счетчика сбрасывается в 0.

**Прерывания.** Описание количества и векторов прерываний:

```
CountInterrupts = 5;
InterruptVector: array[1..CountInterrupts] of byte = ($03,$0B,$13,$1B,$23);
```

**Описание машинных циклов.** Как известно, процессоры выполняют команды согласно различным типам машинных циклов, которые в свою очередь, состоят из тактов [7]. Машинные циклы различаются между собой действиями, которые происходят в те или иные моменты времени, а также состоянием внешних контактов во время их выполнения. Машинные циклы удобно описывать в виде наборов массивов. Количество и состав массивов в наборе определяется числом объектов, изменяющихся в процессе выполнения того или иного машинного цикла. Число элементов массива равно числу тактов, из которых состоит машинный цикл. Каждый элемент массива описывает то или иное действие над соответствующим объектом. Такая форма представления позволяет легко и наглядно описывать поцикловое и потактовое выполнение инструкций.

```
Const
CountTacts = 12;
Type
  ArrayTacts = array[1..CountTacts]of byte;
  ActivePinTacts = record
                     Core, P0, P2, ALE, PME, WR, RD : ArrayTacts
                   end;
Const
  n = 3;     // No Changed
  z = 4;     // Z-state Pin
  d = 5;     // Data
  a = 6;     // PC Address
  x = 7;     // External Data Address
  c = 8;     // Code Address
  g = 9;     // Read Instruction
  o = 10;    // Pins
  // Cycles
  clReadInstr    : ActivePinTacts = (
                   Core:(c,n,n,n,n,c,n,o,n,n,n);
                   P0 :(n,a,n,z,n,n,n,a,n,z,n,n);
                   P2 :(n,a,n,n,n,n,n,a,n,n,n,n);
                   ALE :(1,n,0,n,n,n,1,n,0,n,n,n);
                   PME :(1,n,n,0,n,n,1,n,n,0,n,n);
                   WR :(n,n,n,n,n,n,n,n,n,n,n,n);
                   RD :(n,n,n,n,n,n,n,n,n,n,n,n));;
  clRun          : ActivePinTacts = (
                   Core:(c,n,n,n,n,n,c,g,o,n,n,n);
                   P0 :(n,a,n,z,n,n,n,a,n,z,n,n);
                   P2 :(n,a,n,n,n,n,n,a,n,n,n,n);
                   ALE :(1,n,0,n,n,n,1,n,0,n,n,n);
                   PME :(1,n,n,0,n,n,1,n,n,0,n,n);
                   WR :(n,n,n,n,n,n,n,n,n,n,n,n);
                   RD :(n,n,n,n,n,n,n,n,n,n,n,n));;
  ClReadCode     : ActivePinTacts = (
                   Core:(c,n,n,n,n,d,c,n,o,n,n,n);
                   P0 :(n,c,n,z,n,n,n,a,n,z,n,n);
                   P2 :(n,c,n,n,n,n,n,a,n,n,n,n);
                   ALE :(1,n,0,n,n,n,1,n,0,n,n,n);
                   PME :(1,n,n,0,n,n,1,n,n,0,n,n);
                   WR :(n,n,n,n,n,n,n,n,n,n,n,n);
                   RD :(n,n,n,n,n,n,n,n,n,n,n,n);;
  clReadData     : ActivePinTacts = (
                   Core:(c,n,n,n,n,n,n,n,o,n,d,n);
                   P0 :(n,x,n,z,n,n,n,n,n,n,n,n);
                   P2 :(n,x,n,n,n,n,n,n,n,n,n,n);
                   ALE :(1,n,0,n,n,n,n,n,n,n,n,n);
                   PME :(1,n,n,n,n,n,n,n,n,n,n,n);
                   WR :(n,n,n,n,n,n,n,n,n,n,n,n);
                   RD :(n,n,n,n,n,0,n,n,n,n,n,1));
  clWriteData    : ActivePinTacts = (
                   Core:(c,n,n,n,n,n,n,n,o,n,n,n);
                   P0 :(n,x,n,d,n,n,n,n,n,n,n,n);
                   P2 :(n,x,n,n,n,n,n,n,n,n,n,n);
                   ALE :(1,n,0,n,n,n,n,n,n,n,n,n);
                   PME :(1,n,n,n,n,n,n,n,n,n,n,n);
                   WR :(n,n,n,n,n,0,n,n,n,n,n,1);
                  RD :(n,n,n,n,n,n,n,n,n,n,n,n));;
```

Приведенным выше образом определяются:

- Количество тактов в одном полном цикле процессора (12).
- Набор объектов, изменяющихся в процессе выполнения того или иного машинного цикла (в записи ActivePinTacts).
- Набор констант (0, 1, n, z, d, a, x, c, g, o), которые определяют, какие именно действия совершаются над объектом в том или ином такте:
  - 0 — установка в 0;
  - 1 — установка в 1;
  - n — ничего не делать;
  - z — установка в z-состояние;
  - d — прочитать и выставить на шину данные;
  - a — выставить на шину адрес памяти программ;
  - x — выставить на шину адрес памяти данных;
  - c — прочитать инструкцию;
  - g — выполнить инструкцию;
  - o — проверить значения на контактах.
- Циклы работы процессора.
- Чтение инструкций clReadInstr.
- Исполнение инструкций clRun.
- Чтение кода clReadCode.
- Чтение данных clReadData.
- Запись данных clWriteData.

Для каждого вида циклов процессора указывается, какая именно работа должна производиться на каждом из 12 тактов этого цикла.

```
Type
TInstMode =(b1c1, b1c2, b1c2c, b1c2r, b1c2w, b2c1, b2c2, b3c2, b1c4);
```

Приведенным выше образом определяются возможные типы инструкций процессора (длина в байтах и количество циклов).

```
Type TRunMode = (Instructions, Tacts);
Var RunMode : TRunMode;
```

Модель может исполнять программу с максимальной адекватностью (по тактам) или с максимальной быстротой (по инструкциям, игнорируя заданную информацию о потактовой работе МК). Способ симуляции (по тактам или по инструкциям) устанавливается пользователем в среде HLCCAD, как значение одного из параметров корпуса МК.

**Описание инструкций.**

Сначала описываются мнемонические константы для кодов инструкций:

```
ciNOP          = $00;
ciAJMPx0       = $01;
ciLJMP         = $02;
ciRR           = $03;
ciINC_A        = $04;
...
ciMOV_r5_A     = $FD;
ciMOV_r6_A     = $FE;
ciMOV_r7_A     = $FF;
```

Набор инструкций описывается при помощи массива записей, в которых определены следующие поля:

Code — идентификатор команды;

Mode — тип инструкции;

Format — формат команды в виде строки (введено для описания сложных инструкций с размером слова более 8 бит и может отсутствовать. Например, формат инструкции SLEEP процессора с ядром AVR выглядел бы как '1001 0101 100X 1000').

```
Type
TInstr = record
           Code : Byte; // Code
           Mode : TInstMode; // Mode of instruction
         end;
Const
 CountInstrs = $FF;
 Instrs : array[0..CountInstrs] of TInstr =
 (
  (Code:ciNOP; Mode:b1c1),
  (Code:ciAJMPx0; Mode:b2c2),
  (Code:ciLJMP; Mode:b3c2),
  ...
  (Code:ciMOV_r6_A; Mode:b1c1),
  (Code:ciMOV_r7_A; Mode:b1c1)
 );
```

Приведенным выше образом для каждой инструкции определяется ее код и тип (количество байтов и способ исполнения).

## 5. Информация для дизассемблирования

Модель должна иметь реализованную функцию Dizassembler следующего вида:

```
function DizAssembler(       data       : Pointer;
                       var AsmLine      : PChar;
                       var InstrSize    : byte) :boolean;
```

Здесь

Data — указатель на данные, подлежащие дизассемблированию;

AsMLine — результат дизассемблирования;

InstrSize — размер текущей инструкции.

```
Move(data^,Instr,SizeOf(Instr));
InstrSize:=ISize[Instr[1]];
Case Instr[1] of
  ciNOP       :AsmText:='nop';
  ciAJMPx0    :AsmText:='ajmp   x0'+gn2xh(Instr[2]);// ajmp address
  ciLJMP      :AsmText:='ljmp   '+gn4xh(Instr[2],Instr[3]);// ljmp address
  ciRR        :AsmText:='rr     A';      // RR ACC
  ciINC_A     :AsmText:='inc    A';      // inc ACC
  ciINC_d     :AsmText:='inc    '+gnSFR(Instr[2]);// inc ad
  ciINC_ar0   :AsmText:='inc    @R0';    // inc @r0
  ...
  ciMOV_r5_A  :AsmText:='mov    R5,A';
  ciMOV_r6_A  :AsmText:='mov    R6,A';
  ciMOV_r7_A  :AsmText:='mov    R7,A';
end;
AsmLine:=PChar(AsmText);
```

Выше приведен фрагмент текста функции Dizassembler с дизассемблированием нескольких инструкций. Можно заметить, что для формирования строки дизассемблирования используются вызовы специальных функций преобразования чисел в символьное представление в нужном формате (gn2xh, gn4xh и др.). При использовании функций gnSFR и gnBIT, в случае наличия поименованного ресурса по соответствующему адресу (например, регистра в области SFR), в дизассемблированном тексте появится его имя.

## 6. Общий подход к реализации модели МК

Для реализации модели МК, взаимодействующей с HLCCAD, необходимо унаследовать ее от поставляемого с HLCCAD объекта TBaseCPUModel:

```
Type
    T_i8051 = class(TBaseCPUModel)
    ...
```

и разработать методы модели, взаимодействующие с системой (вызываемые) HLCCAD: InitProcessor — для инициализации процессора; RunInstruction — для выполнения текущей инструкции.

Основной оператор исполнения инструкции процессором (в процедуре RunInstruction) выглядит следующим образом:

```
Case CInstr[1] of
      ciNOP :;
      ciAJMPx0 :PC:=(PC and $F800) or CInstr[2];
      ciLJMP :PC:=(CInstr[2] shl 8) or CInstr[3];
      ciRR :ACC:=(ACC shr 1)or((ACC and 1)shl 7);
      ciINC_A :ACC:=ACC+1;
      ciINC_d :DirMem[CInstr[2]]:=DirMem[CInstr[2]]+1;
      ...
      ciMOV_d_A :DirMem[CInstr[2]]:=ACC;
      ciMOV_ar0_A :IndMem[R[0]]:=ACC;
      ciMOV_ar1_A :IndMem[R[1]]:=ACC;
      ciMOV_r0_A..
      ciMOV_r7_A :R[CInstr[1] and 7]:=ACC;
   end;

...

if InterruptEvent and not IRetEvent then Interrupts;
if TimerEvent then Timers;
...
```

Здесь каждому идентификатору инструкции ставится то или иное действие, выполняемое при ее исполнении. Для доступа к ресурсам процессора используются идентификаторы ACC, PC и др. (для обращения к регистрам) и IntMem, DirMem и пр. (для обращения к памяти). После выполнения каждой инструкции обеспечивается также обработка прерываний (в процедуре Interrupts) и таймеров (в процедуре Timers).

Загрузка очередной инструкции при выборе режима исполнения по инструкциям выглядит так:

```
CInstr[1]:=ROM[PC_];
Case ISize[CInstr[1]] of
   2:CInstr[2]:=ROM[PC_+1];
   3:begin
        CInstr[2]:=ROM[PC_+1];     // 2nd code of instruction
        CInstr[3]:=ROM[PC_+2];     // 3td code of instruction
     end;
   end;
PC:=PC + ISize[CInstr[1]];          // set next PC
RunInstruction;
for i:=1 to ICycle[CInstr[1]] do CountersProcessing;
```

Последняя строка фрагмента, приведенного выше, обеспечивает обработку счетчиков после выполнения каждой инструкции.

Взаимодействие модели контроллера с остальной частью схемы осуществляется в момент вызова системой HLCCAD методов ExecAfter (после истечения указанного моделью МК модельного времени) и OnChanged (в момент изменения значения на любом из контактов МК).

При выборе потактового исполнения процедуры ExecAfter и OnChanged включают проверку отсутствия сигнала сброса:

```
   ...
If RST then InitProcessor;
...
```

и запуск процедуры исполнения модели по тактам:

```
...
   RunTact;
   ...
```

Для желающих глубже вникнуть в организацию потактового исполнения ниже приводится полный текст процедуры RunTact, описывающей потактовое исполнение модели МК Intel 8051.

```
Procedure T_i8051.RunTact;
const RData:boolean = false;
Begin
   if (PINb[p_EA]=0) or (PC>=MaxIROMAddress) then ExternalCode:=True;
   if not PowerDownMode and not IdleMode then begin
      Case CycleMode.Core[TactMode] of
         c : begin
            if (InstrIndex<=ISize[CInstr[1]]) and (Cycle<>0) then begin
               if ExternalCode // Read instruction code
                  then CInstr[InstrIndex]:=PINb[p_P0]
                  else CInstr[InstrIndex]:=ROM[PC];
               Inc(InstrIndex);
               PC:=PC+1;
            end;
            if (TactMode=1) then begin
               Case Cycle of
                     1: Case Instrs[CInstr[1]].Mode of
                           b1c1,b2c1,b1c2,b2c2,b1c4: CycleMode:=clRun;
                           b3c2: CycleMode:=clReadInstr;
                           b1c2c: CycleMode:=clReadCode;
                           b1c2r: CycleMode:=clReadData;
                           b1c2w: CycleMode:=clWriteData;
                        end;
                     2: Case Instrs[CInstr[1]].Mode of
                           b3c2: CycleMode:=clRun;
                        else CycleMode:=clReadInstr;
                        end;
                     3,4: CycleMode:=clReadInstr;
               end;
               RData:=(Instrs[CInstr[1]].Mode=b1c2r);
            end;
         end;
         g : RunInstruction;
         o : PinsProcessing;
         d : if ExternalCode or (Instrs[CInstr[1]].Mode=b1c2r)
            then Begin ACC:=PINb[p_P0]; CheckInstrRun:=True; End
            else ACC:=ROM[GetAddressROM];
      end;
      Case CycleMode.P0[TactMode] of
         z : if ExternalCode or RData then PINz[p_P0]:=True;
         a : if ExternalCode
            then PINb[p_P0]:=lo(PC)
            else PINb[p_P0]:=SFR[sfrP0];
         c : if ExternalCode then PINb[p_P0]:=lo(GetAddressROM);
         x : PINb[p_P0]:=lo(GetAddressXRAM);
         d : Begin PINb[p_P0]:=ACC; CheckInstrRun:=True; End;
      end;
      Case CycleMode.P2[TactMode] of
         a : if ExternalCode
            then PINb[p_P2]:=hi(PC)
            else PINb[p_P2]:=SFR[sfrP2];
         c : if ExternalCode then PINb[p_P2]:=hi(GetAddressROM);
         x : PINb[p_P2]:=hi(GetAddressXRAM);
      end;
      if CycleMode.ALE[TactMode]<>n then PINb[p_ALE]:=CycleMode.ALE
      [TactMode];
      if (CycleMode.PME[TactMode]<>n) and ExternalCode
         then PINb[p_PME]:=CycleMode.PME[TactMode];
      if CycleMode.WR[TactMode]<>n then B[bitWR]:=CycleMode.WR
      [TactMode];
      if CycleMode.RD[TactMode]<>n then B[bitRD]:=CycleMode.RD
      [TactMode];
      Inc(TactMode);
      if TactMode=CountTacts+1 then begin
         TactMode:=1;
         Inc(Cycle);
         Case Instrs[CInstr[1]].Mode of
            b1c1,b2c1: begin Cycle:=1; InstrIndex:=1; end;
            b1c2,b1c2c,b1c2r,b1c2w,b2c2,b3c2:
               if Cycle=3 then begin Cycle:=1; InstrIndex:=1; end;
            b1c4: if Cycle=5 then begin Cycle:=1; InstrIndex:=1; end;
         end;
      end;
   end;
   if IdleMode then begin
      PINb[p_ALE]:=1;
      PINb[p_PME]:=1;
      CountersProcessing;
      if InterruptEvent {and EnableInterrupts} then Interrupts;
   end;
   if PowerDownMode then begin
      PINb[p_ALE]:=0;
      PINb[p_PME]:=0;
   end;
End;
```

При выполнении процедуры потактового выполнения происходит вызов тех или иных действий, определенных константами (0, 1, n, z, d, a, x, c, g, o) для каждого из объектов, определенных в записи ActivePinTacts в соответствии с текущим циклом процессора. Здесь также происходит проверка наличия внешней памяти программ, смена циклов процессора, инкремент программного счетчика PC. При необходимости происходит переход в спящий режим или режим пониженного потребления энергии (IdleMode и PowerDownMode).

Внимательный читатель заметил, что среди вызываемых процедур имеются процедуры:
PinProcessing — процедура обработки контактов модели;
CountersProcessing — процедура обработки счетчиков;
RunInstruction — процедура, непосредственно обеспечивающая алгоритм исполнения инструкции.

Прерывания вызываются при выполнении определенных условий. Как правило, такими условиями являются изменения битов в определенных регистрах или приход на внешние контакты МК импульсов определенной полярности. При возникновении такого условия в модели активизируется событие и, после выполнения очередной инструкции, происходит вызов процедуры обработки прерываний.

```
procedure T_i8051.Interrupts;
Var NumInt:byte;
begin
   if pinINT0Event and (B[bitIT0]=1) then B[bitIE0]:=1;
   if pinINT1Event and (B[bitIT1]=1) then B[bitIE1]:=1;
   if (P[bitINT0]=0)and(B[bitIT0]=0)and(IntStack[IntStack[0]]<>1)
      then B[bitIE0]:=1;
   if (P[bitINT1]=0)and(B[bitIT1]=0)and(IntStack[IntStack[0]]<>3)
      then B[bitIE1]:=1;
   NumInt:=0;
   if B[bitEA]=1 then begin
      if IntStack[0]=0 then begin
         if (B[bitES]=1)and((B[bitTI]=1)or(B[bitRI]=1)) then NumInt:=5;
         if (B[bitET1]=1)and(B[bitTF1]=1) then NumInt:=4;
         if (B[bitEX1]=1)and(B[bitIE1]=1) then NumInt:=3;
         if (B[bitET0]=1)and(B[bitTF0]=1) then NumInt:=2;
         if (B[bitEX0]=1)and(B[bitIE0]=1) then NumInt:=1;
      end;
      if (IntStack[0]=0)or(B[sfrIP+IntStack[IntStack[0]]-1]=0) then begin
         if (B[bitPS]=1)and(B[bitES]=1)and((B[bitTI]=1)or(B[bitRI]=1)) then
         NumInt:=5;
         if (B[bitPT1]=1)and(B[bitET1]=1)and(B[bitTF1]=1) then NumInt:=4;
         if (B[bitPX1]=1)and(B[bitEX1]=1)and(B[bitIE1]=1) then NumInt:=3;
         if (B[bitPT0]=1)and(B[bitET0]=1)and(B[bitTF0]=1) then NumInt:=2;
         if (B[bitPX0]=1)and(B[bitEX0]=1)and(B[bitIE0]=1) then NumInt:=1;
      end;
   end;
   if (NumInt<>0)and(NumInt<>IntStack[IntStack[0]]) then begin
      IntStack[0]:=IntStack[0]+1;
      IntStack[IntStack[0]]:=NumInt;
      SP:=SP+1; IndMem[SP]:=lo(PC);
      SP:=SP+1; IndMem[SP]:=hi(PC);
      if IdleMode
         then PC:=InterruptVector[NumInt] — 1
         else PC:=InterruptVector[NumInt];
      IdleMode:=False;
      SFR[sfrPCON]:=SetBtB(SFR[sfrPCON],0,0);
      case NumInt of // clear inguairy flag
         1: if B[bitIT0]=1 then B[bitIE0]:=0;
         2: B[bitTF0]:=0;
         3: if B[bitIT1]=1 then B[bitIE1]:=0;
         4: B[bitTF1]:=0;
      end;
   end;
   pinINT0Event:=False;
   pinINT1Event:=False;
   InterruptEvent:=False;
end;
```

В теле процедуры происходит проверка истинности условия, идентификация прерывания, выбор прерывания исходя из приоритетов, сохранение определенных данных в стеке и переход к выполнению инструкции, на которую указывает вектор соответствующего прерывания.

Для обработки счетчиков служит процедура, которая вызывается при выполнении каждой инструкции или каждого машинного цикла. В процессе работы процедуры анализируется состояние каждого счетчика и, при выполнении определенных условий, инкрементируются активные в данный момент времени. Условием инкремента счетчика является переполнение его источника.

```
Procedure T_i8051.CountersProcessing;
var i: integer;
begin
  for i:=0 to NumCounters-1 do begin
    if (Counters[i].Source=cntExtClk) then IncCounter(i,1);
    if CounterFull(i) then
      case i of
        cntPredCounter0 :begin
                         IncCounter(cntTL0,1);
                         SFR[sfrTL0]:=Counters[cntTL0].Value;
                         if ModeTimer0=3 then begin
                           IncCounter(cntTH0,1);
                           SFR[sfrTH0]:=Counters[cntTH0].Value;
                         end;
                        end;
        cntPredCounter1 :begin
                         IncCounter(cntTL1,1);
                         SFR[sfrTL1]:=Counters[cntTL1].Value;
                        end;
        cntTL0          :case ModeTimer0 of
                         0,1:begin
                               IncCounter(cntTH0,1);
                               SFR[sfrTH0]:=Counters[cntTH0].Value;
                             end;
                         2  :begin
                               B[bitTF0]:=1;
                               SFR[sfrTL0]:=SFR[sfrTH0];
                             end;
                         3  : B[bitTF0]:=1;
                         end;
        cntTL1          :case ModeTimer1 of
                         0,1,3 :begin
                                  IncCounter(cntTH1,1);
                                  SFR[sfrTH1]:=Counters[cntTH1].Value;
                                end;
                           2 :begin
                                B[bitTF1]:=1;
                                SFR[sfrTL1]:=SFR[sfrTH1];
                              end;
                           end;
        cntTH0          :case ModeTimer0 of
                         0,1:B[bitTF0]:=1;
                         3 :B[bitTF1]:=1;
                         end;
        cntTH1          :case ModeTimer1 of
                         0,1 :B[bitTF1]:=1;
                         end;
    end;
  end;
end;
```

Здесь для каждого счетчика определен порядок действий, выполняемых при его переполнении. Для инкремента счетчика используется процедура инкремента IncCounter с параметрами идентификатора счетчика и приращения. При достижении лимита счетчика эта процедура автоматически устанавливает флаг переполнения.

## Заключение

Наличие разработанной технологии создания и отладки моделей микроконтроллеров позволяет получать модели в сроки от недели до месяца, в зависимости от сложности микроконтроллера, уровня подготовки разработчика в программировании и понимания им алгоритмов функционирования микроконтроллера.

## 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. Долинский М., Литвинов В., Галатин А., Шалаханова Н. Система высокоуровневого проектирования аппаратного обеспечения HLCCAD: открытый универсальный интерфейс моделируемых компонент. // Компоненты и технологии. 2003. № 4.
5. Федорцов А. О. Описание алгоритма работы встроенной периферии при создании программных моделей микроконтроллеров / Новые компьютерные технологии проектировании, производстве и научных исследованиях: Материалы III респ. науч.-техн. конф. студентов и аспирантов 13–18 марта 2000. Гомель: Гомельский гос. ун-т им. Ф. Скорины. 2000.
6. Федорцов А. О. Метод описания моделей процессоров / Сборник материалов респ. межвузовской науч.-тех. конф. студентов, аспирантов и магистрантов, посвященной 55-летию Победы в Великой Отечественной войне. 10–12 мая 2000. Гомель: ГГТУ. 2000.
7. Сташин В. В., Урусов А. В., Мологонцева О. Ф. Проектирование цифровых устройств на однокристальных микроконтроллерах. М.: Энергоатомиздат, 1990.
8. h ttp:// NewIT.gsu.unibel. by