01.01.17

Проектирование ЭВМ

Целью проекта является знакомство, освоение и участие в проектировании цифровых устройств уровня ЭВМ на примере синтеза схем для микроэвм mcs51.
Проектирование включает
- разработку теста для контроля функциональности заданных команд,
- разработку функциональных микропрограмм и формирование двоичных кодов в Borland C для схемы микропрограммного управления,
- тестирование функциональных микропрограмм в IDE **Микро51,**

--проектирование блочно-структурной схемы для исполнения одной из команд теста и описание этой схемы в AHDL в **MaxPlus2**

IDE Микро51 разработана для целей образования и представляет собой открытую программу в Borland C++ с визуальным отображением результатов в Программной модели. Результат двоичного кодирования микропрограммы
формируется в виде файлов загрузки управляющей памяти в проекте MaxPlus2.

Система Микро51 настраивается в Borland C++ (или в другой функционально-совместимой системе программирования) для кодирования и исполнения микропрограмм заданных команд.
Предполагается знание программирования в C++ и схемотехники на уровне графического и текстового редактора в MaxPlus2 (Quartus2)

Содержание

1. Задание – обзор системы команд,
2. Структура ЭВМ
3. Функциональное тестирование в Ассемблере
   - тестовая программа
   - функциональная модель в Кейл
   -
4. Микропрограммирование
   - Исполнение команд с однотактным микропрограммным управлением
   -Функциональная микропрограмма – исполнение в Си

   - Структурная микропрограмма – кодирование микропрограмм
5. Схема блока в GDF и на языке AHDL
6. Отладка схемы на основе временных диаграмм

II. Задание включает список команд из системы команд mcs51

Требуется :
1) оформить задание , привести спецификацию(описание) команд задания из Кейл.Help, включая кодирование и принцип исполнения
2) разработать тест для заданной системы команд, отладить и компилировать в Keil и выполнить моделирование
3) разработать функциональные микропрограммы для команд в системе **Микро51** на языке Borland C++, используя общую структурную схему ЭВМ

Система команд

Ljmp start | 02 | A15… A8 | A7…… .A0 |

Содержание команды Code(A[15..0]) → PC , PC+3, PSW не меняется

Sjmp ii0 | 80 | rel |

PC+Code(rel) → PC, PC+2 PSW не меняется

Inc a | 04 |

a +1 → a, Ram(Acc), PSW(P)

Ret | 22 |

Ram[SP -] → PC[16..8]
Ram[SP -] → PC[7..0]
SP--> Ram(Sp)

Add a,#Const | 24 | # Const |

a + Const → a, PSW(C, OV, P)

Add a,Ri | 0x28 | Ri |

a + Ram (psw[rs]<<3 + Ri) → a, PSW(C, OV, P)

Nop | 00 |

PC++

Anl c,bit | 0x82 | bit |

if(psw(C)& Ram[bit]) → psw(C)

Acall Mm | $A_{10}A_9A_8$10001 | A7….A0 |

PC+2 → Ram(Sp++)
A10...A0 →PC

Inc Ri | 0x8 Ri |

Ram [psw(rs) + Ri] +1

reti	32	Ram[SP -] → PC[16..8] Ram[SP -] → PC[7..0 Разрешение прерывания

III Тестовая программа

Цель тестирования:

- контроль исполнения команд и работы программного управления

Сложность и непреодолимые трудности обоснования достоверно полного тестирования ограничивают его **функциональным** тестированием с визуализацией результатов контрольного исполнения тестовой программы.

Таким образом, с позиций разработчика компьютера проверяем на конечном числе примеров с конкретными исходными данными по исходной спецификации правильное исполнение доступа к данным в различных типах памяти и выполнения операций.

Каждое задание, по умолчанию, предполагает использование команд ljmp start, reti и вспомогательных команд, обеспечивающих функциональность основных команд (возврат из подпрограмм, прерываний и установку данных)

Команды типа

0) **ljmp start** - выборка и декодирование команды из памяти Code**,** изменение состояния регистра PC

1) Add a,#Const - контролируем следующую функциональность : доступ к регистру a/Acc, чтение Const из памяти Code, правильное исполнение сложения и формирования признаков результата в PSW.

2) Add a,Ri - доступ к регистрам общего назначения Ri

3) Anl c,bit - доступ к битам битового сегмента(f6), правильное исполнение логической операции с битами

4) Acall Mm - запись в Стек

5) Inc Ri - операция счета

6) Ret - чтение Стека

7) **sjmp met –**

Используя данные в одних командах изменяютс\ их значения в смежных командах

Определить оптимальный порядок размещения команд теста и выбирать значения для покрытия всех функциональностей

Порядок размещения команд 4 и 6 произвольный, а изменение значений в регистрах и памяти можно обеспечить их начальной установкой или дублированием этих команд

Команда 0 – начало теста.

```
; ACC=0x80  R0=0x85  PSW=0x00   SP=0x07

      Cseg at 0
       Ljmp start ;
Into:  reti

      Cseg  at  0x10
Mm:
       Inc  a    ; a/Acc=0x86
       Ret       ; PC=
```

```
    Cseg at  0x20
  Start:
     Add a,#0x80   ;Acc=0x80,  C=1,P=1,Ov=1
     Anl c,Acc.7   ;    C=0
    Add a,r0        ; Acc=0x85  C=1,P=0,Ov=0
    Inc r0          ;  R0=0x86
    Nop                                  ; задержка для формирования прерывания
     Acall Mm      ;  PC=   SP=
   end
```

Тест компилируется в Кейл51 , исполняется и оформляется в файле .LST

```
//----------------------------------------
 // Тест          память программ      исходное состояние памяти
                                      PC=00  SP=07 acc=  (r0)=
//-----------------------------------------------------------------
 //0:  ljmp start    0x02 00  0x23         Pc=0x23   SP=07
 //03: reti          0x32                  PC=0x26   sp=07
 //13: reti          0x32                  Pc=0x28   sp=07
 //mcal:
 //22: ret           0x22                   Pc=0x2a   sp=07
 //start:                                 acc=0x82  r0=0x21;
 //23: add a,#80     0x24 0x80              acc=0x02, PSW=0x81
 //25: nop           0x00
           //прерывание int0,            stack=0026, PC=03
           //прерывание int1,            stack=0026,PC=13)
 //26: add a,r0        0x28               acc=0x23,PSW=01
 //27: anl c,ACC.7   0x82 0xe7            PSW=01
 //29: acall mcal      0x11 0x22          PC=0x22, SP=0x09, Ram[sp]=00 2b
 //2b: nop           0x00                   //конец программы
```

IV.Схемотехника

4.1. Блок-схема.

Блок-схема ЭВМ -разделение схемы ЭВМ на функциональные блоки с возможным функциональным акцентом для упрощения последующей ее детализации в структурной схеме

16mem – блок 1 включает программную память Code типа ROM , память данных Xdata и соответствующие 16-разрядные адресные регистры PC(программный счетчик), Dptr(указатель) и логику адресации.

Control –блок 7 микропрограммного управления включает схему адресации микропрограммной памяти CU, микропрограммную память ROMM и логику управления адресацией, генератор синхросигналов GS

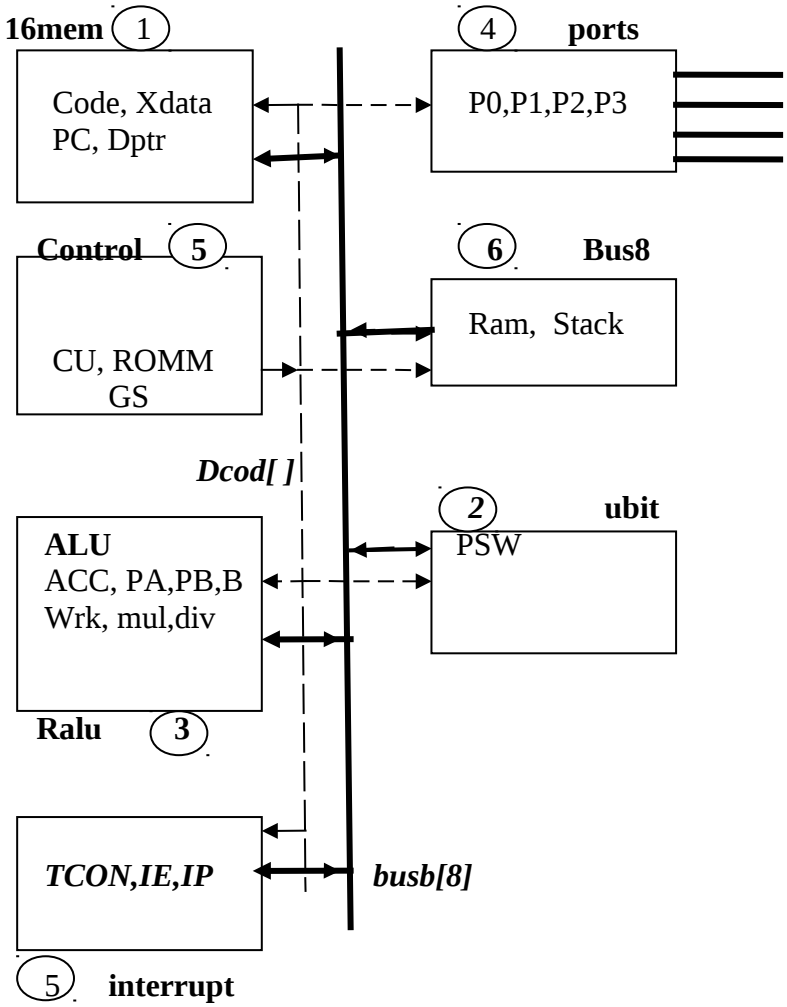
Bus8 – блок 6 включает память данных 256 байт, указатель стека SP и логику управления шиной данных **busb[8]**

Ports –блок 4 четыре 8-разрядных универсальных порта, включающих регистры и логику управления записью и чтением

Ralu – блок 3 регистровый арифметико-логический блок включает АЛУ, регистры временного хранения данных PA,PB,рабочий регистр Wrk , регистр –аккумулятор ACC, рабочий регистр B, модуль умножителя 8*8→16 **mul** и деления 8/8→8,8 **div**

Ubit – блок 2 выполнения битовых операций, регистр состояния PSW, схемы адресации битов

Interrupt- блок 5 управления прерываниями, регистры управления TCON, IE,IP, схему контроля запросов прерывания и выбора наиболее приоритетного запроса, схему формирования адрес-вектора.

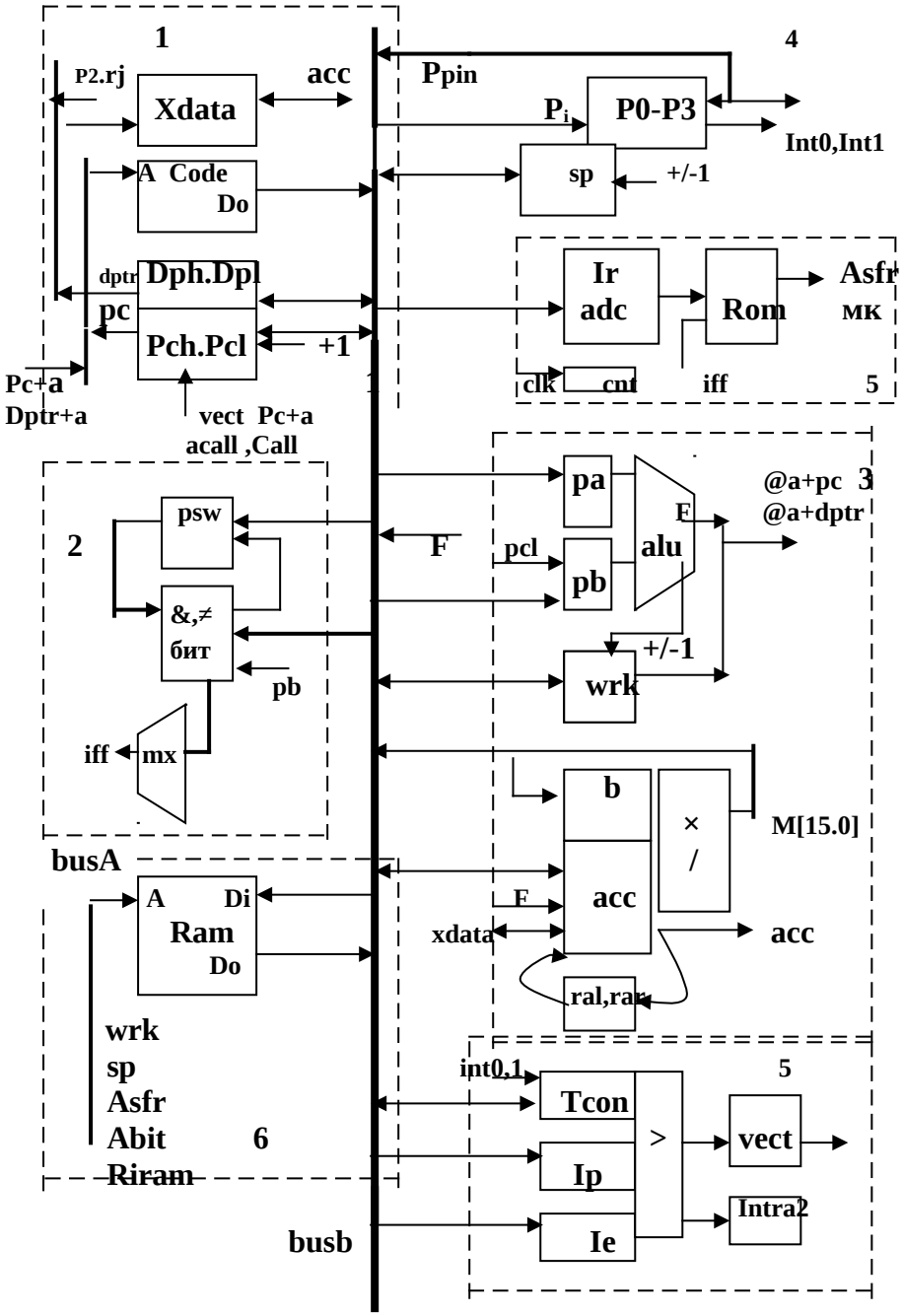


Предполагается одна 8-разрядная общая шина данных **busb[8]** и шина управления **Dcod[]** передает 8-разрядную микрокоманду для распределенного декодирования управляющих сигналов в каждом блоке.

Организация связей между блоками (внутренние интерфейсы в ЭВМ) при выполнении команд оформляется в виде графической **структурной схемы для двух проектов в графике и в** AHDL**.** Эти схемы являются основами для дальнейшей детализации в виде более подробных структурных схем отдельных блоков и функциональных схем в графике **.gdf (projectgraf в данном проекте)** и языке AHDL **.tdf (projectext в данном проекте**) в MaxPlus II.

4.2. Структурная схема ЭВМ

Структурная схема mcs51 выбирается для определения необходимых ресурсов памяти и проектирования функциональной микропрограммы для команд теста
В микропрограммировании необходимо ориентироваться всю структурную схему , детали организации схем блоков конкретизируются для конкретных команд теста



1) Задание ресурсов памяти программной модели в Микро51.

//ulong codDCM; //32-бит код микрокоманды в блоке
uchar CODE[2048];//программная память
//тест в кодах команд + таблица векторов и начало

uchar ADC[0x100]; //преобразование кода команды в адрес

```
uint ROMM[128];   //память микрокоманнд - (тест <10) + 3бита <128
//uchar WRSFR[128]; //декодер адресов в сигналы записи в регистры SFR

//   память данных ----------
 uchar
      Ram[256], Xdata[1000];
```

2)Формирование начального состояния памяти для тестирования

```
; ACC=0x80  R0=0x85  PSW=0x00   SP=0x07
//  исходные данные теста ============================
 for(k=0;k<0x100;k++)    //сброс декодера команд
            ADC[k]=0;
 uchar j=0;
//декодер команд ADC[IR]
 //начальный адрес j-ой микропрограммы  RAMK=(j<<3).Cnt=000, j=ADC[IR]
 //микропрограмма – не более 8 микрокоманд ограничен 3-битным счетчиком
 ADC[0]=j++; //команда NOP=j=0
 ADC[02]=j++; // команда ljmp ad   IR=02.j=1
 ADC[0x24]=j++; // add a,#d   IR=24, j=2
 ADC[0x22]=j++; // ret      IR=22, j=3

for(i=0x28;i<0x2f;i++) ADC[i]=j; j++;  //j=8 команда add a,ri  IR=28—2f
for(uchar i=0x11;i<0xF1;i=i+0x10) ADC[i]=j; j++;  // j=F команды acall met

ADC[0x82]=j++;  // команда anl c, bit
ADC[0x32]=j++; // reti   ;

//исходные состояния регистров и памяти
//------микропрограммная память
 for(k=0;k<128;k++)
 ROMM[k]=0;
//====================начальное состояние теневых и рабочих регистров
; ACC=0X80  R0=0X85  PSW=0X00   SP=0X07
 Ram[Sp]=SP=07;   //теневой = (прямой)=07
 Ram[0]=0x85; //значение R0
 Ram[Acc]=ACC=0x80;
 Ram[Psw]=PSW=0x00;
 Ram[Tcon]=TCON=0;
 P0=P1=P2=P3=0xff;//начальное состояние портов
 Ram[P0]=Ram[P1]=Ram[P2]=Ram[P3]=0xff;
//сброс декодеров микрокоманд
//---------------------------------------------
 for(i=0;i<16;i++) DCM1[i]=DCM2[i]=DCM3[i]=DCM4[i]=\
  DCM9[i]=DCM8[i]=0;
```

3) ==========**загрузка теста в программную память**

```
//if(CheckBox8->State==cbUnchecked)
{  for(i=0;i<100;i++) CODE[i]=0;//сброс программной памяти
 PC=0;       //программный код
CODE[PC++]=0x02; CODE[PC++]=0; CODE[PC++]=0x23; //ljmp 23
CODE[0x03]=0x32;  //reti
PC=0x22;
CODE[PC++]=0x22;  //ret
CODE[PC++]=0x24; CODE[PC++]=0x80; //add a,#10
CODE[PC++]=0x28;  //add a,r0
CODE[PC++]=0; //Nop для контроля прерываний
CODE[PC++]=0x82; CODE[PC++]=0xe7; //anl ACC.7
CODE[PC++]=0x11; CODE[PC++]=0x22; //acall 0x22
CODE[PC++]=0;    //конец теста
 }
 Instr->Clear();
  IE=0;      //сброс запросов прерываний
  EX1=EX0=EA=0;  //запрет прерываний
  //сброс масок прерываний на пульте
  CheckBox1->State=cbUnchecked;   //кодирование
  CheckBox2->State=cbUnchecked;   //EX1=0
  CheckBox3->State=cbUnchecked;   //EX0=0
  CheckBox4->State=cbUnchecked;    //EA=0

   ComboBox2->Clear();   //сброс окна регистров
   ComboBox6->Clear();   //сброс стека
  for(char i=0; i<8; i++)
            Reg(i); //исходные значения в регистрах !=0
  StateMCU(); //начальное состояние регистров
  i1=1; //начало кодов в декодерах DCMi  i1=0 - пустая МК
  CheckBox8->State=cbUnchecked;
  CheckBox9->State=cbUnchecked;
}
```

4) Функциональная Микропрограмма в Микро51

```
   switch(ADC[IR]);          //декодирование команды
      {
   //микропрограмма входа в прерывание
   //0 mk
   GoToInt();RAMK++; //схема регистрации запросов прерываний - запросы в ОКНЕ
   // 1 mk  вход в прерывание
   if (!intra2)  goto decoder;  //переход к исполнению  команды основной программы
   //2 mk
    SP++;RAMK++;
   //3 mk
   Ram[SP++]=PC;RAMK++;
   //4 mk
   Ram[SP]=PC>>8; PC=vect; ClearInt(); RAMK++;
   //5 mk Завершение входа в прерывание PC=vect
```

```
Ram[Sp]=SP; RAMK=0;

//-------------------------------------
decoder:   //исполнение  команд
  {IR=CODE[PC++];RAMK=(ADC[IR]<<3)+1;}
  Switch(IR){

     case 0:         //1.0  mk  Nop-
     {Instr->Text="Nop"; RAMK=0;
           goto finish;}

      case 1: //ljmp adr
          // 1   -чтение старшего байта  PCH адреса в Wrk
          { Wrk=CODE[PC++]; RAMK++;
          // 2  - чтение младшего байта  PCL и запись PCH.PCL в PC
            PC=CODE[PC]|(Wrk<<8); RAMK=0; }
                  Goto finish; //завершение команды – выборка следующей
       case 3: //ret
           // 1
           {  Wrk=Ram[SP--]; RAMK++;
           // 2   запись адреса из Стека в PC
             PC=(Wrk<<8)|Ram[SP--]; RAMK++; //PCL
           // 3  сохранить SP в Ram
             Ram[Sp]=SP; RAMK=0;
            }  goto finish;
     case 4:       // add a,ri
        // 1   чтение регистра
         { PB= Ram[(PSW&0x18)|(IR&0x3)];RAMK++;
        // 2 операция в АЛУ и формирование признаков и сохранение ACC в SFR
          ACC=ACC+PB, Ram[Acc]=ACC; RAMK++; Pswc(“add”);
        // 3 сохранение PsW в SFR
           Ram[Psw]=PSW; RAMK=0; }  goto finish;
   case 2:      // add a,#d
       // 1 -чтение операнда из  памяти Code
        { PB=CODE[PC++]; RAMK++;
       //2 - - операция в АЛУ и формирование признаков результата
         ACC=ACC+PB, RAMK++; Ram[Acc]=ACC; Pswc(“add”);
       //3-
         Ram[Psw]=PSW;  RAMK=0;  }  goto finish;
      case 5:        //  acall  met
       //1 - чтение второго байта команды и пре-инкремент указателя
         {Wrk=CODE[PC++]; SP++; RAMK++;
        //2- запись в Стек PC(7-0) и постинкремент указателя
           Ram[SP++]=PC;RAMK++;
        //3 –старший байт PC в стек и формирование PC
           Ram[SP]=PC>>8; RAMK++;
         PC=((PC&0xf800)|Wrk)|((IR&0xE0)>>5)<<8; }
         //4- сохранение продвинутого указателя в SFR
          Ram[Sp]=SP;  RAMK=0; }  goto finish;
case 6: // anl c,bit
        // 1- чтение адреса бита из второго байта команды
          {Wrk=CODE[PC++]; RAMK++;
```

```
        // 2 - чтение бит адресуемого байта из RAM
          if(W7) PB=Ram[Wrk&0xf8];
          else { PB=Ram[0x20|((Wrk&0x78)>>3)]; RAMK++; }
        // 3 - выполнение операции с битами
         If ((1<<(Wrk&0x7)&PB));
          else PSW=PSW&0x7f;
         RAMK++;
        // 4- сохранение PSW в SFR по адресу Psw
            Ram[Psw]=PSW;  RAMK=0; }   goto finish;
  case 7: //reti
         // 1
         {  Wrk=Ram[SP--]<<8; RAMK++;
         // 2
            PC=(Wrk<<8)|Ram[SP--]; RAMK++;
         // 3
            Ram[Sp]=SP;  RAMK=0; intra2=0; }

  //Вывод состояния регистров в HEX-коде
finish: //возврат к контролю прерывания и чтению команды-
       StateMCU( );
    }
```