Лабораторная работа № 8. Моделирование схем с микроконтроллерами в программе Proteus.Isis

Задание 1. Микроконтроллер + Дешифратор.

Часть 1 – Создание схемы

Функции дешифраторов и шифраторов понятны из их названий. Дешифратор преобразует входной двоичный код в номер выходного сигнала (дешифрирует код), а шифратор преобразует номер входного сигнала в выходной двоичный код (шифрует номер входного сигнала). Количество выходных сигналов дешифратора и входных сигналов шифратора равно количеству возможных состояний двоичного кода (входного кода у дешифратора и выходного кода у шифратора), то есть 2^n, где n — разрядность двоичного кода. Микросхемы дешифраторов обозначаются на схемах буквами DC (от английского Decoder), а микросхемы шифраторов — CD (от английского Coder).

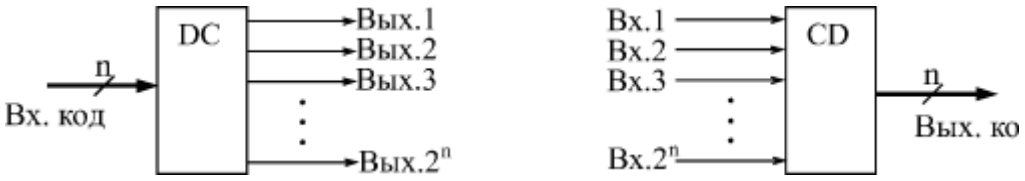

Рис. 1. Функции дешифратора (слева) и шифратора (справа)

На выходе дешифратора всегда присутствует только один активный сигнал, причем номер этого сигнала однозначно определяется входным кодом. Выходной код шифратора однозначно определяется номером входного сигнала. Выходы дешифраторов обычно инверсные, т.о. активным уровнем выхода является логический ноль.

Если нужно дешифровать код с большим числом разрядов (чем способен имеющийся тип дешифратора), то можно объединить несколько микросхем дешифраторов.

Смоделируем дешифратор 3→8 на трёх дешифраторах 2→4 на ИМС К155ИД4 (представляющих собой сдвоенные дешифраторы 2→4).

Соберём в программе Proteus.Isis следующую схему:

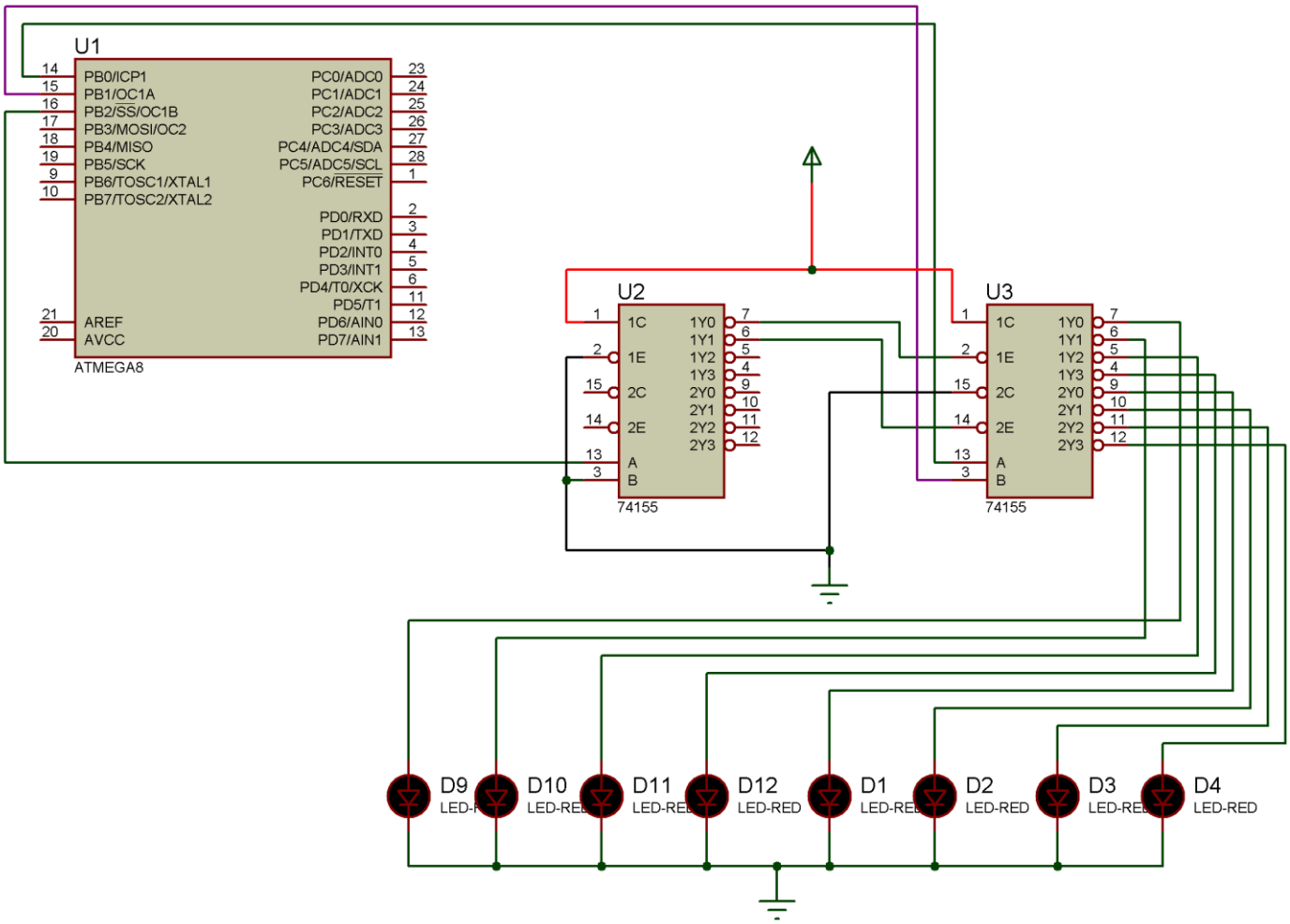

Рис. 2 – Схема масштабированного дешифратора в Proteus.Isis

Часть 2 – создание программы

Для того, чтобы схема на микроконтроллере заработала, ей необходимо подать программу – «прошивку».

Воспользуемся следующим программным кодом (на языке C++), созданным в программе CodeVisionAVR:

```
#include <io.h>
#include <delay.h>
void main(void)
{
    DDRB =255;
    PORTB = 0;
    while (1)
    {
       PORTB++;
       if (PORTB>7) PORTB=0;
      delay_ms(500);
    }
}
```

Скомпилированный hex-файл прошивки имеет следующий код:

```
:0200000017C027
:10000200FECFFDCFFCCFFBCFFACFF9CFF8CFF7CFA2
:10001200F6CFF5CFF4CFF3CFF2CFF1CFF0CFEFCFD2
:10002200EECFEDCF08000100050026000000F89495
:10003200EE27ECBBF1E0FBBFEBBFE5BFF8E1F1BDA2
:10004200E1BD8DE0A2E0BB27ED938A95E9F780E060
:1000520094E0A0E6ED930197E9F7E8E2F0E08591FC
:1000620095910097 61F0A591B59105901590BF010A
:10007200F00105900D920197E1F7FB01F0CFEFE55A
:10008200EDBFE4E0EEBFC0E6D1E000C0EFEFE1BBC0
:10009200E0EEE2BBEFEFE7BBE0E0E8BBEFE3E4BB9F
:1000A200E0E0E5BBE2B3EF5FE2BBE8B3EF5FE8BBE2
:1000B200E5B3EF5FE5BBE2B3F0E0E05EF0403A9714
:1000C20014F0E0EEE2BBE8B3E03110F0E0E0E8BBB0
:1000D200E0E8E51511F4E8E05E2E550CE8B3E529F9
:1000E200E8BBE5B3E03110F0E0E0E5BBA4EFB1E03E
:1000F20005D0E8B3E525E8BBD5CFFFCF109639F0A0
:1001020080ED97E00197F1F7A8951197C9F7089547
:00000001FF
```

Создайте в папке с проектом файл codesh.hex, откройте его в блокноте и вставьте этот код.

«Прошьём» микроконтроллер в Proteus следующим образом – откройте настройки ATMega8 и задайте следующие параметры:

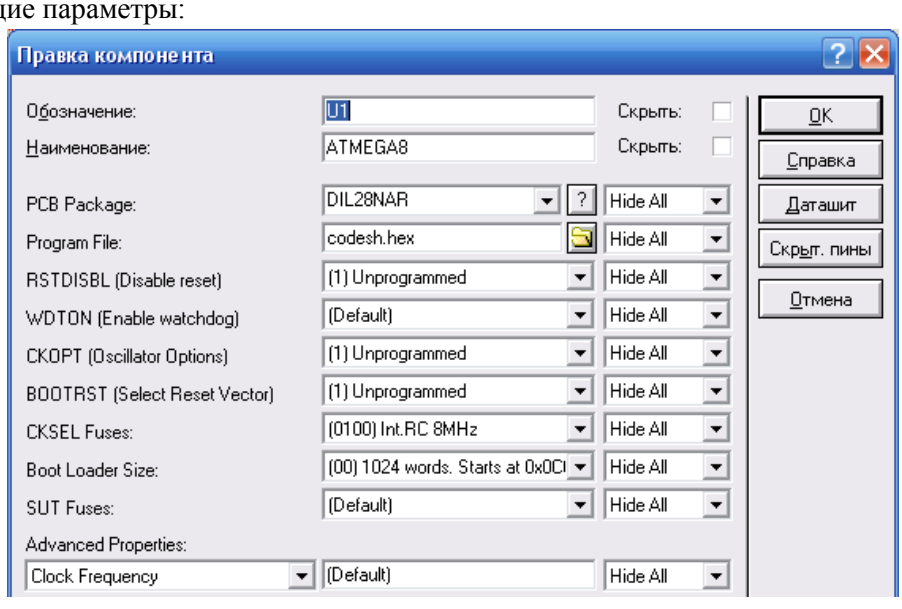

Рис. 3 – Настройка МК

Всё, схема должна заработать – поочерёдно тухнет один из восьми светодиодов (остальные горят). Т.е. мы получили дешифратор 3→8, имея в распоряжении три дешифратора 2→4 (в составе двух ИМС).

Задание 2. Микроконтроллер + Терминал

Смоделируем схему на микроконтроллере AT89C51, которая будет отсылать в окно терминала цифры по нажатиям кнопок, приделанных к портам контроллера.

Терминал и осциллограф находятся в разделе виртуальных приборов.

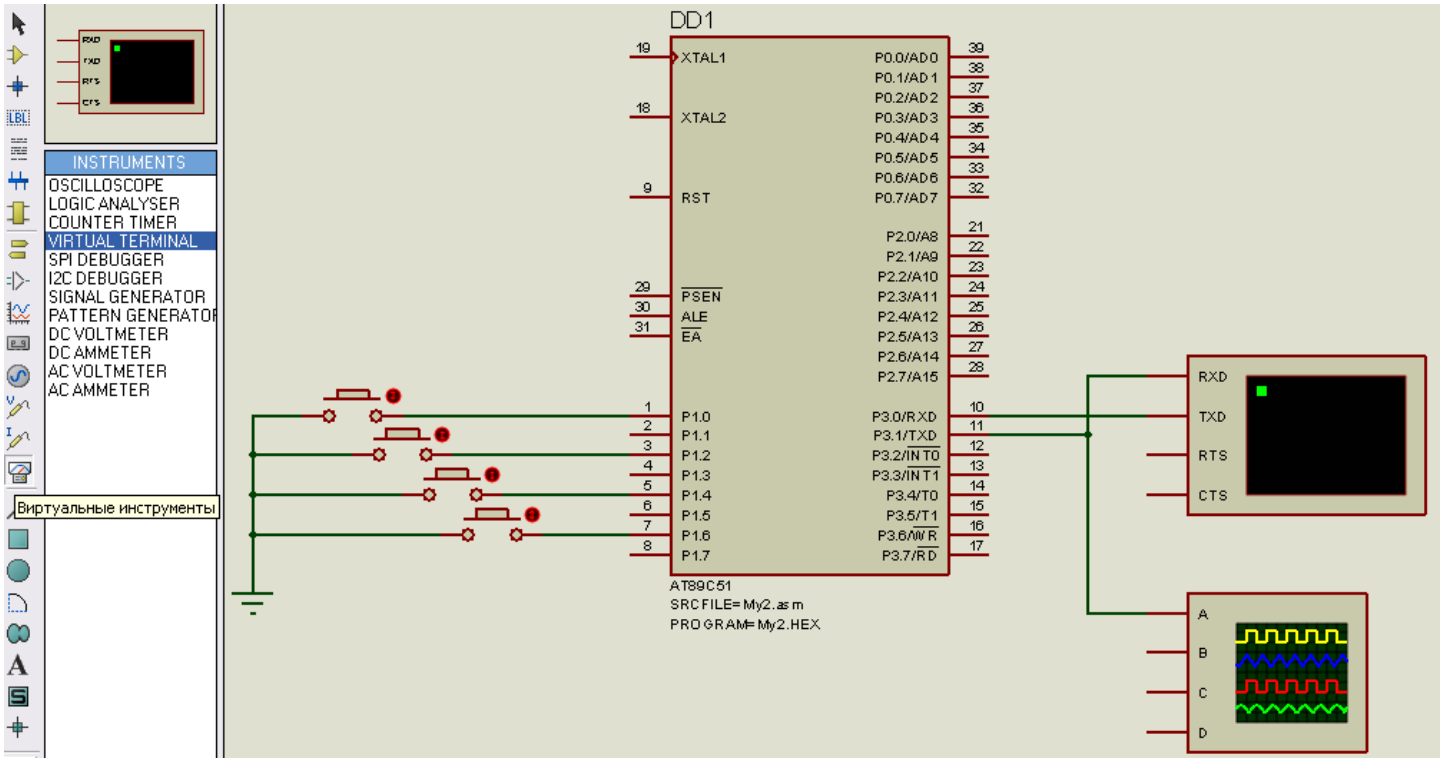

Рис.4 – Схема задания 2

Аппаратная часть готова. Перейдём к программированию и настройке.

Часть 2. Прошивка МК

Создаём исходник

Создайте в папке с проектом (желательно использовать простые пути, без пробелов и русских букв, но это если схема не будет работать, для начала попробуйте в своей обычной папке) файл My2.asm и с помощью блокнота вставьте в него код из приложения.

Добавляем исходник

В меню в пункте Source (Исходник) выбираем Add/Remove source (Добавить/удалить исходник) и добавляем файл My2.asm. Добавляя исходник, укажите компилятор, которым его надо будет компиллировать. Для данного случая в “Code generation tools” надо указать “ASEM51”, то есть компилятор архитектуры MCS-51.

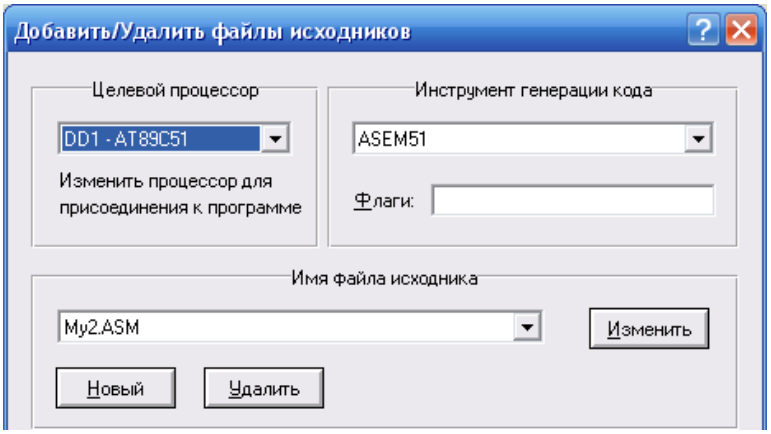

Рис.5 – Установка исходника программы

Жмите ОК и в меню Source появится еще один пункт – добавленный исходный файл, при выборе которого открывается редактор и можно подправить текст программы.

Настройка компилятора

Данный пункт для начала, если следующий кончится успехом, можно пропустить.

В меню Source найдите пункт “Define Code Generation Tools” (Определить инструменты генерации кода) – это опции компилятора. Изначально они могут быть настроены «криво» – в разделе “Make rules” в строке “Command Line” (Командная строка) удалите весть текст, что там есть. Оставь только “%1” без кавычек. ASEM51 сам добавит нужные файлы с описаниями регистров и переменных.

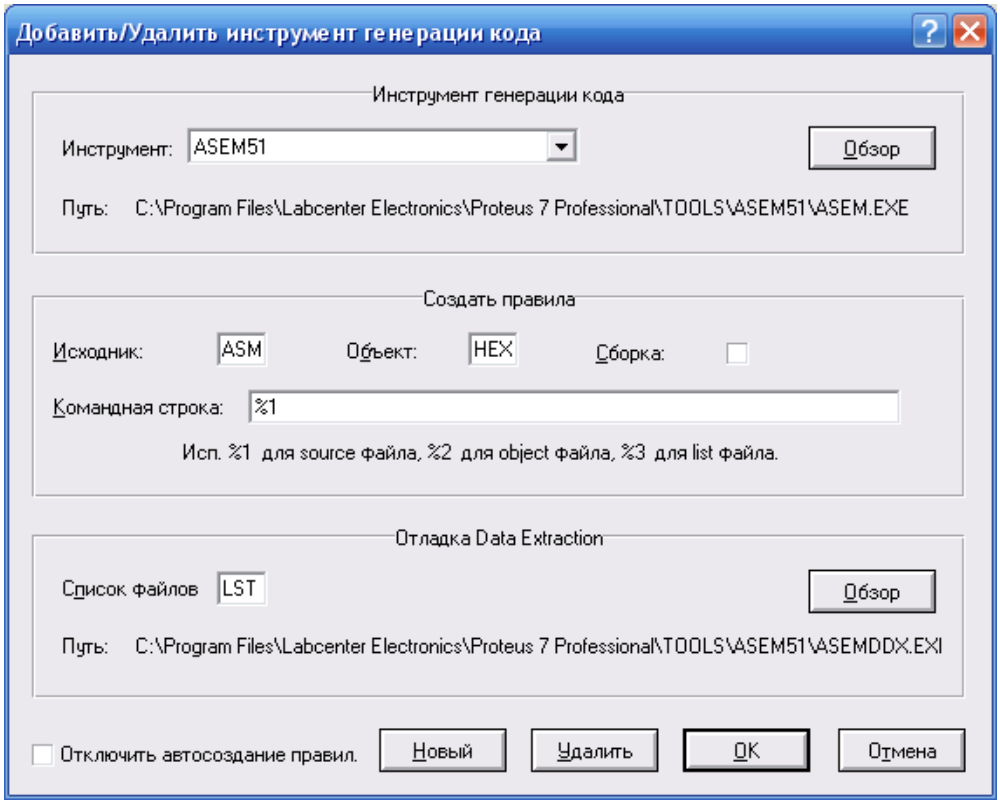

Рис.6 –Настройка компилятора

Компиляция

Нажмите в том же меню Source пункт Build All (Компоновать всё) и получите на выходе hex-файл, созданный «на месте». Откроется окно компилятора, в котором будут сведения о возможных ошибках и ряд служебных данных.

Отладка

Запустите схему кнопкой Play (Воспроизвести) в нижней панельке и сразу же нажмите либо паузу, либо пошаговый режим (Шаг). Сразу же должно открыться окно с кодом программы как в отладчике. Если не открылось, его можно найти в меню Отладка: Debug→8051CPU→Source Code–DD1. Там же будет ряд других полезных вещей, как, например, содержимое регистров процессора или памяти программ/данных.

Запуск моделирования

Запустите схему кнопкой Play и нажимайте поочерёдно кнопки – вы увидите в окне терминала соответствующие кнопкам цифры, в окне осциллографа – временную диаграмму импульсного сигнала – двоичного кода номера кнопки. Если вдруг по какой-либо причине окна визуализации инструментов не открылись, то в процессе выполнения программы откройте контекстное меню инструмента и поставьте галочку в нижнем пункте. Чтобы вместо непонятных символов терминал отображал правильные цифры (1..4), выставьте в настройках AT89C51 частоту 20MHz.

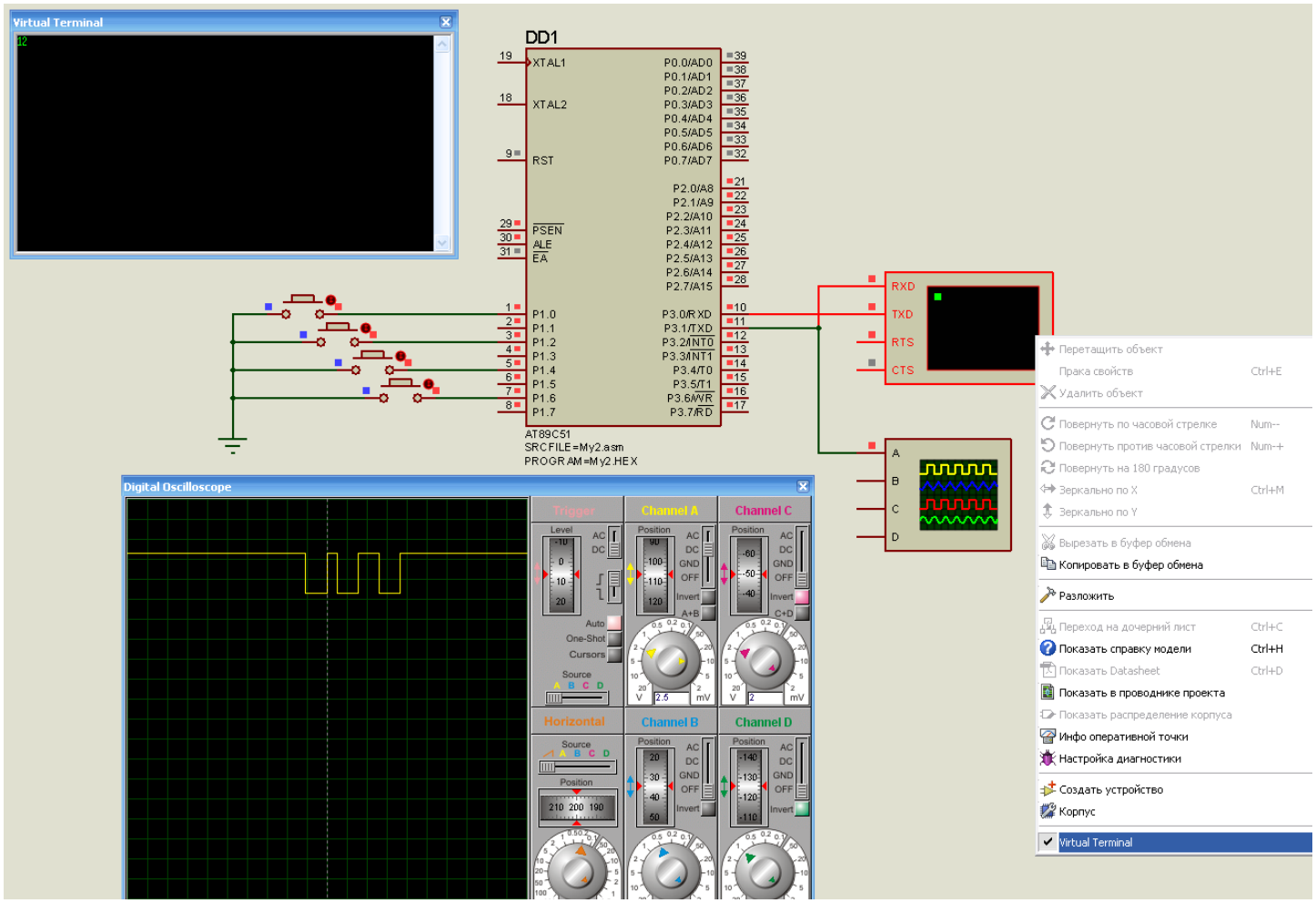

Рис.6 –Работа схемы (моделирование)

Задание 3

В Протеусе всё уже прекрасно работает (должно). Но модели МК в Proteus несколько упрощенные – они они не требуют наличия в виртуальной схеме кварцевого резонатора, системы сброса (подтяжка RESET до нужного уровня), наличия сигнала на использования внутренней памяти (+5 на EA, особенность МК С51, умеющих работать от внешней ПЗУ) и об этом не стоит забывать, когда в итоге будем делать реальную схему, а то, в итоге, искать причину неработающей схемы можно очень долго.

Добавим детали «обвески». Кварц в библиотеке называется «crystal». Какой-нибудь SMT конденсатор (Capacitor) емкостью порядка 33pF. Землю ищите в левой панели инструментов кнопку Терминал (Terminal mode). По нажатию на неё откроется панелька, где нужно выбрать строку GROUND. Power там же – это напряжение питания схемы.

Далее надо собрать схему сброса. Протеусу это не требуется, он и так будет нормально отрабатывать, но реальной схеме это нужно. Делается это просто. Ставим резистор и конденсатор. При включении, когда конденсатор не заряжен, то его сопротивление равно нулю и на вывод RST подается +5 вольт, т.е. логическая 1, а как только конденсатор зарядится (произойдет это через пару миллисекунд) то ножка через резистора будет лежать на земле (логический нуль) и МК запустится в штатном режиме.

Ну и поставьте еще светодиод на порт P2. Как подключать светодиоды к портам проца? Вешаем анод светодиода на питание, а катод – на резистор, а этот резистор уже на выход МК. Чтобы зажечь диод, надо на эту ногу выдать 0. Тогда разница напряжений между напряжением питания и напряжением нуля на ножке будет максимальной и диод будет гореть.

Итак, полная схема имеет вид (см. рис.7)

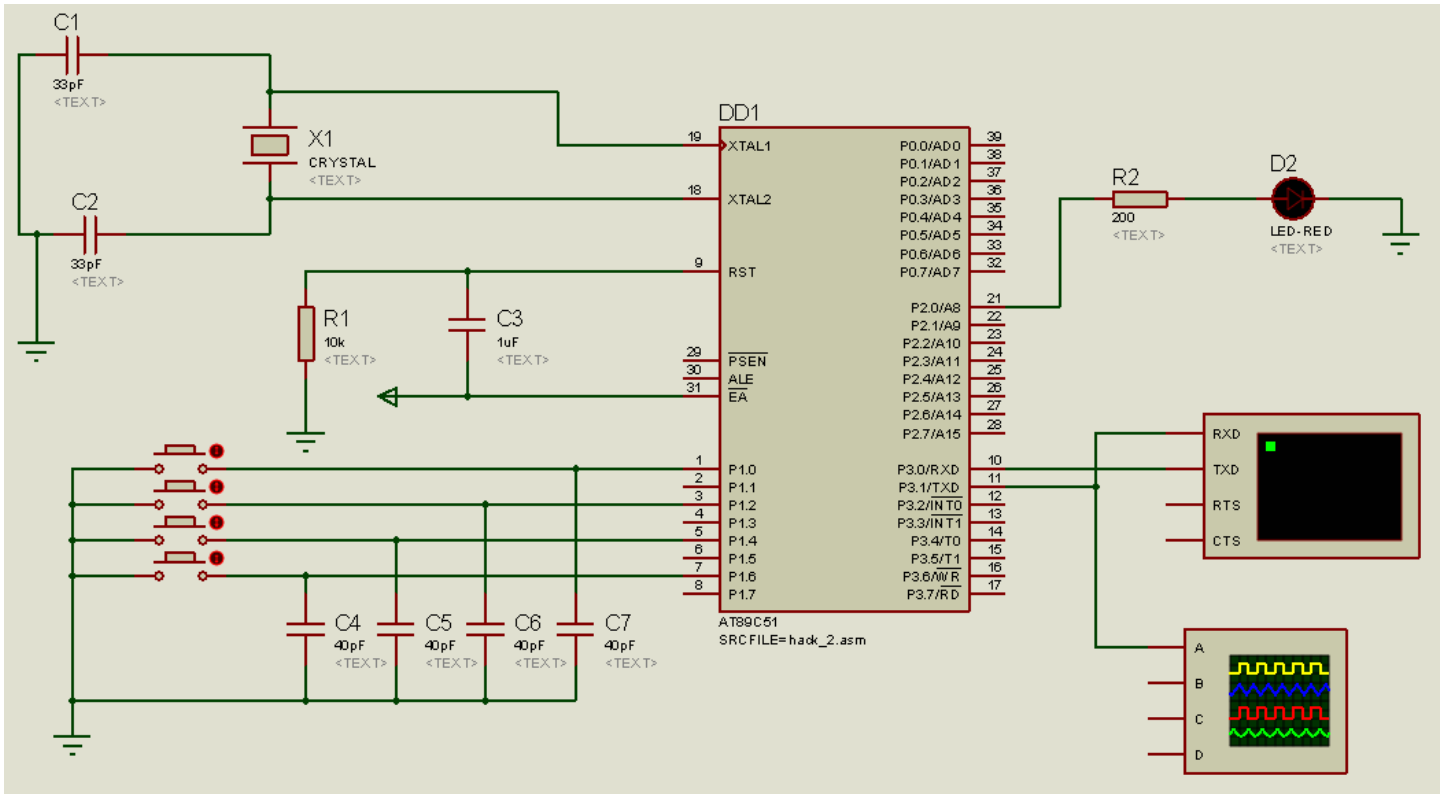

Рис.7 – Полноценная схема, годная для физической реализации

Приложение. Исходный код программы (файл My2.asm)

```
CR      EQU     0DH         ;Код возврата каретки (ASCII)
LF      EQU     0AH         ;Код перевода строки
ESC     EQU     1BH         ;Код операции ESC
        ORG 0000h
            JMP start
        ORG 0023h
        RETI

            ORG 0100h

; Инициализация UART тут:
start:  MOV P2,#255         ; Все выводы на высокий уровень
        MOV IE,#0           ; Прерывания отключаем нафиг, чтобы не мешались (далеко не везде и не всегда надо так!!!)
            CLR EA          ; До кучи вообще запрещаем все прерывания.
        SETB ES             ; Но оставляем прерывание от UART (на всякий, один фиг оно пока не работает - EA Сброшен)

        MOV PCON,#80h       ; Устанавливаем бит удвоенной скорости, он расположен в регистре PCON
        CLR TR1             ; Сбрасываем флаг таймера = таймер готов к работе
        MOV TMOD,#20h       ; Настраиваем таймер 1 в режим 2 (автоперезагрузка значения. Регистр TMOD определяет параметры таймера)
        MOV TH1,#0F5h       ; 9600 @ 20MHz  Прескалер таймера, определяющий, что порт будет выдавать 9600 бод на 20мгцовом тактовом кварце
        SETB TR1            ; Установили бит TR1 = запустили таймер
        MOV SCON,#52h       ; Uart Init and UART Ready - инициализация параметров UART

;       тут мы просто крутимся, обрабатывая кнопочки.
```

```
LOOP:   JB   P1.0,nxt1   ; Проверяем наличие бита на ноге 1.0. Если бита нет, значит эта нога кнопкой прижата к земле = 0
        MOV  A,#'1'      ; Кидаем код 1ки в аккумулятор и
        CALL UART        ; вызываем процедуру выдачи в УАРТ, которая отрыгнет этот код по порту.
        JNB  P1.0,$      ; а тут мы находимся до тех пор, пока не отпустят кнопку.

nxt1:   JB   P1.2,nxt2   ; аналогично, но для другой ножки порта.
        MOV  A,#'2'
        CALL UART
        JNB  P1.2,$

nxt2:   JB   P1.4,nxt3
        MOV  A,#'3'
        CALL UART
        JNB  P1.4,$

nxt3:   JB   P1.6,LOOP
        MOV  A,#'4'
        CALL UART
        JNB  P1.6,$


        JMP  LOOP
;======= Proc ========================================
; Out Byte in UART

UART:  JNB  TI,$      ;Ожидание готовности
       CLR  TI        ; сброс флага готовности
       MOV  SBUF,A    ;Выдача символа в буфер UART
       CALL DELAY     ; вызов задержки
       CALL DELAY     ; вызов задержки
       RET
;=====================================================
; Stupid Delay   Тупая регистровая задержка на куче вложенных пустых циклов.
DELAY:  MOV R7,#255
ZD0: MOV R6,#255
       MOV R5,#255
ZD1: DJNZ R6,ZD1
ZD2: DJNZ R5,ZD2
      DJNZ R7,ZD0
     RET
     END
```