HT48 & HT46 LCM 接口设计

文件编码：HA0013s

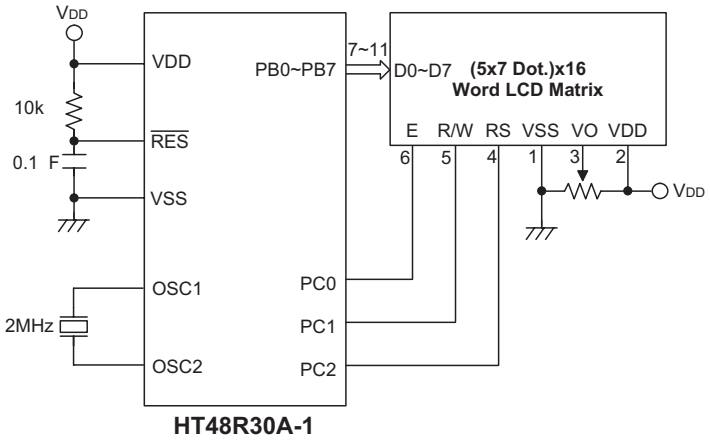
简介：

本文介绍利用 8 位微控制器控制 DV16100NRB 液晶显示驱动器的方法。该 LCM 由内置的 Hitachi HD44780 进行驱动及控制。本文应用中，着重考虑如何使微控制器产生正确的信号以符合 LCM 所需的时序。若要获得详细的时序及指令信息，请查阅 LCM 厂商的资料。

LCM 能以 4 位或 8 位模式工作。在 4 位模式下，传送一个字符或一条指令需两个传输周期完成。在 8 位模式下，同样操作仅需一个传输周期，但需要多占 4 个 I/O 端口。本文中可看到利用＃define，if，else 和 endif 编译程序的方法。

电路设计：

PB0～PB7 设为 I/O 端口用于数据传输， PC0～PC2 设为输出端口用于 LCM 控制。端口可重新设置以符合用户的需要。



程序：

```
;File name: lcm.asm
;作者：Chris
;说明：该程序是 LCM 控制主程序。
;=====================================================================
;选择 LCM 的工作模式
;four_bit＝0,LCM 以 8 位模式工作,four_bit＝1 LCM 以 4 位模式工作。
;=====================================================================
;#define four_bit 0                ;LCM 以 8 位模式工作
#define four_bit 1                 ;LCM 以 4 位模式工作

;=====================================================================
;以下部分不可修改
;=====================================================================
```

```
#include  ht48r30a-1.inc
;-------------------------------------------------------
;LCM.INC for DV-16100NRB
LCM_CLS          EQU    01H
CURSOR_HOME      EQU    02H
CURSOR_SR        EQU    14H
CURSOR_SL        EQU    10H
INCDD_CG_SHF_C   EQU    06H
TURN_ON_DISP     EQU    0FH
LCD_ON_CSR_OFF   EQU    0CH
;-------------------------------------------------------
lcm_data      equ    pb           ;PORT B 定义为 LCM 数据端口。
lcm_data_ctrl equ    pbc
lcm_ctrl      equ    pc           ;PORT C 定义为 LCM 控制端口。
ctrl_ctrl     equu   pcc

extern busy_chk:near
extern delay:near
extern write_char:near
extern snd_cmd:near


;LCM Display Commands and control Signal name.
E          equ    0
RW         equ    1
RS         equ    2

;-------------------------------------------------------
data    .section     'data'
coun0         db     ?
coun1         db     ?
coun2         db     ?
msg           db     ?
tmp           db     ?
;-------------------------------------------------------
code .section    at 0 'code'      ;程序部分。
   ORG    00h                    ;设置 ISR 地址。
   jmp    start
start:                           ;程序开始。
   clr    lcm_data_ctrl          ;将 LCM 数据端口设为输出状态。
   clr    ctrl_ctrl              ;将 LCM 控制端口设为输出状态。
   clr    lcm_data               ;LCM 数据端口清零。
   clr    lcm_ctrl               ;LCM 控制端口清零。
```

```
DISPLAY_INIT:
LCM_DELAY:                          ;循环延时程序，用于 LCM 控制。
    mov     a,0ffh
    mov     coun1,a
    mov     coun0,a
lp0:
    sdz     coun1
    jmp     lp0
    sdz     coun0
    jmp     lp0

    If  four_bit
        mov     a,28h               ;4 位工作模式。
    else
        mov     a,38h               ;8 位工作模式。
    endif
    call    snd_cmd                 ;执行写操作。
    MOV     A, 04H
    CALL    DELAY

CMD_SEQ:
    If  four_bit
           mov     a,28h            ;4 位工作模式。
    else
           mov     a,38h            ;8 为工作模式。
    endif
        call    snd_cmd             ;执行写操作。
        MOV     A, 04H
        CALL    DELAY

    if  four_bit
           mov     a,80h
        else
           mov     a, 38h
    endif
    call    snd_cmd                 ;执行写操作。
    MOV     A, 04H
    CALL    DELAY

    call    busy_chk                ;CALL"忙"状态检查程序。
    Mov     a,lcm_cls               ;将 LCM_CLS 指令赋予 ACC。
    call    snd_cmd                 ;CALL"写指令"程序。
    MOV     A, 04H
```

```
    CALL    DELAY

    call    busy_chk
    mov     a,TURN_ON_DISP          ;将 TURN_ON_DISP 指令赋予 ACC。
    call    snd_cmd
    MOV     A, 04H
    CALL    DELAY

    call    busy_chk
    mov     a,INCDD_CG_SHF_C        ;将 INCDD_CG_SHF_C 指令赋予 ACC。
    call    snd_cmd
    MOV     A, 04H
    CALL    DELAY
;-------------------------------------------------------
       ;inc addr of DD ram & shift
       ;the cursor to the right at
       ;the time of write to DD/CG
       ;RAM.
;-------------------------------------------------------
loop:
    call    busy_chk
    mov     a,lcm_cls
    call    snd_cmd
    MOV     A, 04H
    CALL    DELAY

    call    busy_chk
    mov     a,cursor_home           ;将 cursor_home 指令赋予 ACC。
    call    snd_cmd
    MOV     A, 04H
    CALL    DELAY

    call    busy_chk
    clr     tblp                    ;查表指针清零。

;initial display "HOLTEK 8 bit uC"
agn:
    tabrdl msg                  ;将从 last page 查表得到的
                                ;数据赋予 msg。
    mov     a,msg
    mov     tmp,a
    mov     a,24h
    xorm    a,msg
```

```
    sz      msg                     ;检查 msg 是否为零。
    jmp     agn1                    ;如果 msg≠0，跳至 agn1。
    jmp     secn_line               ;如果 msg＝0，
;跳至 secn_line。
agn1:
    call    busy_chk
    mov     a,tmp
    call    write_char              ;CALL"写数据"程序。
    inc     tblp
    jmp     agn
secn_line:
    inc     tblp                    ;查表指针加一。
    call    busy_chk
    mov     a,0c0h                  ;第二列的地址
    call    snd_cmd
    call    busy_chk
snd_line:
    tabrdl msg
    mov     a,msg
    mov     tmp,a
    mov     a,24h
    xorm    a,msg
    sz      msg
    jmp     snd_lin1
    mov     a,lcd_on_csr_off        ;将 lcd_on_csr_off 指令赋予 ACC。
    call    snd_cmd
    jmp     lp
snd_lin1:
    mov     a,tmp
    call    write_char
    call    busy_chk
    inc     tblp                    ;查表指针加一
    jmp     snd_line
lp:
;   jmp     start
    jmp     lp
;------------------------------------------------------
holtek_tbl .section at 700h 'code' ;table at last page
htk_tbl:
;   dc  4800h,4f00h,4c00h,5400h,4500h,4b00h,2000h,3800h,2400h
;   dc  2000h,6200h,6900h,7400h,2000h,7500h,4300h,2400h
```

```
;       dc      0048h,004fh,004ch,0054h,0045h,004bh,0020h,0038h,0024h
;   dc      0020h,0062h,0069h,0074h,0020h,0075h,0043h,0024h
;注意：以下是我的修改
    dc      0057h,0075h,0020h,004ah,0069h,006eh,0067h,0020h,0024h
    dc      0020h,004ah,0069h,006eh,0067h,0020h,0020h,0020h,0024h

    end
```

```
;File name: lcmglob.inc
;作者：Chris
;说明：定义全局常量
lcm_data          equ       pb
lcm_data_ctrl     equ       pbc
lcm_ctrl          equ       pc
ctrl_ctrl         equ       pcc
E                 equ       0
RW                equ       1
RS                equ       2
```

```
;File name: command.asm
;作者：Chris
;说明：LCM 的控制命令文件，作为外部程序供主程序 LCM.ASM 调用。编译时要和主程
;序一起加入到同一个 PROJECT 中。
;=====================================================================
;选择 LCM 的工作模式
;four_bit＝0,LCM 以 8 位模式工作,four_bit＝1 LCM 以 4 位模式工作。
;=====================================================================
;#define four_bit 0                ;LCM 以 8 位模式工作
#define four_bit 1                 ;LCM 以 4 位模式工作

;=====================================================================
;以下部分不可修改
;=====================================================================
;LCM command module begin at here
include ht48r30a-1.inc
include lcmglob.inc
public busy_chk
public delay
public write_char
```

```
public snd_cmd
;-------------------------------------------------------
scode .section   'code'                ;写 LCM 控制指令程序。
snd_cmd:
    if  four_bit                       ;如果为 4 位模式。
        Mov    dtmp,a
        and    a,0f0h                  ;将高 nibble  4 位输入 ACC。
    endif

    mov    lcm_data,a                  ;将 LCM 控制指令写入数据端口。
    clr    lcm_ctrl.rw                 ;设置"写指令"模式。
    Clr    lcm_ctrl.rs
    set    lcm_ctrl.e
    clr    lcm_ctrl.e                  ;将指令写入 LCM。

    if  four_bit
        swapa  dtmp
        and    a,0f0h                  ;将低 nibble  4 位输入 ACC。
        mov    lcm_data,a
        set    lcm_ctrl.e
        clr    lcm_ctrl.e
    endif
    ret
acode .section  'code'
busy_chk:
    set    lcm_data_ctrl               ;将 LCM 数据端口设为输入状态。
    clr    lcm_ctrl.rs                 ;设置 LCM 为检查;BUSY 模式。
    set    lcm_ctrl.rw
    set    lcm_ctrl.e
    mov    a,lcm_data                  ;将 LCM 数据端口值;赋予 ACC。
    if     four_bit
        and    a,0f0h                  ;4 位模式下，先将高 nibble 读入 ACC。
        mov    dtmp,a
        set    lcm_ctrl.e
        swapa  lcm_data
        clr    lcm_ctrl.e
        and    a,0fh                   ;将低 nibble 读入 ACC。
        or     a,dtmp                  ;组成 8 位。
    endif
        clr    lcm_ctrl.e
        sz     acc.7                   ;检查 LCM"忙"状态。
        jmp    busy_chk                ;若处于"忙",返回重新处理"忙"信号。
        clr    lcm_ctrl.rw
```

```
       clr    lcm_data_ctrl          ;若不处于"忙"状态,重新将 LCM 数据端口设为输出
状态。
       ret
;-------------------------------------------------------
wcode .section   'code'
write_char:
   if     four_bit
       mov    dtmp,a
       and    a,0f0h
   endif
       mov    lcm_data,a
       clr    lcm_ctrl.rw
       set    lcm_ctrl.rs
       set    lcm_ctrl.e
       clr    lcm_ctrl.e

   if     four_bit
       swapa   dtmp
       and    a,0f0h
       mov    lcm_data,a
       set    lcm_ctrl.e
       clr    lcm_ctrl.e
   endif
   ret
;-------------------------------------------------------
dsec .section   'data'
dtmp    db ?
dtmp2   db ?
;-------------------------------------------------------
code .section   'code'
delay:
   mov    dtmp,a
   set    dtmp2
drep:
   sdz    dtmp2
   jmp    drep
   sdz    dtmp
   jmp    drep

   ret
```

程序说明：

本文程序首先执行 include 内容，定义 PORT B 为 LCM 数据端口，定义 PORT C 为 LCM 控制端口。可通过 define 命令，将 LCM 程序编译为 4 位或 8 位工作模式。

电源接通时 LCM 会自动进行内部复位。但大多数可编程 LCM 仍利用软件进行初始化。本文例子中，从“START”开始，运行初始化的程序。依照 HD44780 规格，每下一个指令需要至少 4.5 毫秒的延迟时间。这即是 LCM_DELAY 程序的作用。LCM 完成初始化之前，是无法检查 LCM 是否在 BUSY 状态的。向 LCM 下的指令需依照 HD44780 定义的指令码。LCM.INC 中定义了一些常用的指令。在向 LCM 写指令之前，应先检查 LCM 是否处在 BUSY 状态。BUSY_CHK 程序作用就在于此。经确认 LCM 不在 BUSY 状态后，才能将数据传输至 LCM。将要显示的 ASCII 码存放在程序的最后一页，再通过查表方式将数据存入 ACC，然后利用 WRITE_CHAR 程序通过 LCM 完成显示。

RS，R/W 及 E 信号真值表：

RS	R/W	E	Operation
0	0		Write instruction code
0	1		Read busy flag & address counter
1	0		Write data
1	1		Read data