MCU 如何根据 LCD 时序图来写底层驱动

一般来说，LCD 模块的控制都是通过 MCU 对 LCD 模块的内部寄存器、显存进行操作来最终完成的；在此我们设计了三个基本的时序控制程序，分别是：

写寄存器函数（LCD_RegWrite）δ

δ 数据写函数（LCD_DataWrite）

数据读函数（LCD_DataRead）δ

这三个函数需要严格的按照 LCD 所要求的时序来编写，下面可以看看 MzL02 模块时序图：

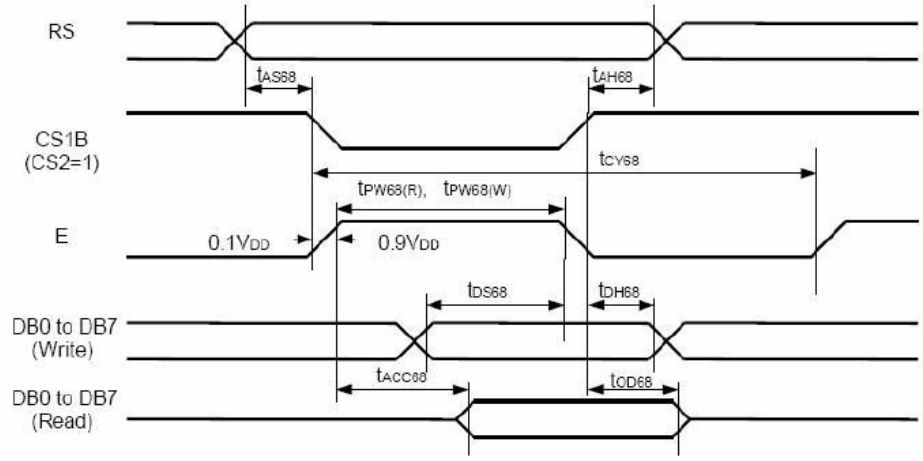

图 3.2 MzL02 模块的 6800 时序示意

注意：上图是该模块的控制 IC 资料中的原版时序图，其实有些示意不是太稳妥（少标出了 RW 线信号的要求），或者说是不太严谨，不过这些不作讨论，请看分析即可；而 EP 的有效触发沿在图中很有可能示意有误，实测为上升沿。图中 CS1B（CS2）的信号即为片选 CS，RS 即为数据/寄存器的选择端口 A0 信号，E 为 EP；当作写入寄存器数据操作时，首先要将 A0 置低，以通知 LCD 模块即将进行的是对寄存器的操作；而 RW 线需要置低，以示即将要进行的是写入的操作；然后片选 CS 信号置低，装载数据至总线，然后在 EP 线上产生一个上升沿以触发 LCD 模块将总线上的数据最终载入；在前面的操作完成后一般都会将各个信号线的状态恢复。而数据（显存）写入、数据读出的操作时序也比较类似，这里就不多作介绍，直接参考例程即可。

```
//=========================================================
//  函数: void LCD_RegWrite(unsigned char Command)
//  描述:  写一个字节的数据至 LCD 中的控制寄存器当中
//  参数: Command      写入的数据，低八位有效（byte）
//  返回:  无
//=========================================================
void LCD_RegWrite(unsigned char Command)
{
LCD_A0 = 0;                 //A0 置低，示意进行寄存器操作
LCD_RW = 0;                 //RW 置低，示意进行写入操作
LCD_EP = 0;                 //EP 先置低，以便后面产生跳变沿
LCD_CS = 0;                   //片选 CS 置低
```

```
DAT_PORT = Command;        //装载数据置总线
LCD_EP = 1;                //产生有效的跳变沿
LCD_CS = 1;                //片选置高
}
```

数据写入以及读出的函数源码如下：

```
//=============================================================
//  函数: void LCD_DataWrite(unsigned char Dat)
//  描述:  写一个字节的显示数据至 LCD 中的显示缓冲 RAM 当中
//  参数: Data  写入的数据
//  返回:  无
//=============================================================
void LCD_DataWrite(unsigned char Dat)
{
LCD_A0 = 1;        //A0 置高，示意进行显存数据操作
LCD_RW = 0;        //RW 置低，示意进行写入操作
LCD_EP = 0;        //EP 先置低，以便后面产生跳变沿
LCD_CS = 0;        //片选 CS 置低
DAT_PORT = Dat;    //装载数据置总线
LCD_EP = 1;        //产生有效的跳变沿
LCD_CS = 1;        //片选置高
}

//=============================================================
//  函数: unsigned char LCD_DataRead(void)
//  描述:  从 LCD 中的显示缓冲 RAM 当中读一个字节的显示数据
//  参数:  无
//  返回:  读出的数据，
//=============================================================
unsigned char LCD_DataRead(void)
{
unsigned char Read_Data;
DAT_PORT = 0xff;  //51 的端口想要输入前，要先给端口全置 1
LCD_A0 = 1;        //A0 置高，示意进行显存数据操作
LCD_RW = 1;        //RW 置高，示意进行读出操作
LCD_EP = 0;        //EP 先置低，以便后面产生跳变沿
LCD_CS = 0;        //片选 CS 置低
LCD_EP = 1;        //产生有效的跳变沿
LCD_EP = 0;
Read_Data = DAT_PORT;   //读出数据
LCD_CS = 1;          //片选置高
```

```
return Read_Data;      //返回读到的数据
}
```

以上便是要介绍的最基本的时序操作程序，它们几乎是整个 LCD 驱动程序当中与底层硬件打交道的代码了，这样的话，当要改变驱动 LCD 的 MCU 端口时或者换用别的 MCU 来驱动 LCD 时，基本上只需要在这些代码里作一下修改即可。

关于读 LCD 状态

而在一般的 LCD 模块当中，还有一个功能同样重要，就是读 LCD 状态；可以通过此操作获取当前 LCD 模块的忙状态以及一些相关的状态信息，当 LCD 模块正处于忙状态时，则不宜对它进行数据的写入或读出操作（有很多较老式的 LCD 控制器规定在忙的状态下时不允许写入或读出数据）。

所以在很多 LCD 的驱动程序当中，会在寄存器写入、数据写入/读出的操作前加入读取 LCD 状态并判别忙状态的代码；这点可以参考网上流传的很多 LCD 驱动程序。不过，对于 MzL02 这样的较新出的 LCD 控制器来说，已经对忙状态不是很在乎了，或者说影响已经很小甚至没有了；所以我们在前面的代码当中并没有加入这样的代码。至于有没有必要加读状态判忙的代码，要视具体的 LCD 控制器而定。

关于时序的时间要求

时序的一个非常重要的数据就是类似上图中标出的 tAS88 之类的时间长短要求，只是上图中并没有标出它们的具体最大最小值要求而已；但在编写这类的时序接口程序时它们还是非常重要的，当然还要看 MCU 的端口操作速度以及 MCU 的指令执行速度。打个比方，有的时序里就会有要求某些信号的电平保持最小宽度，而如果 MCU 的指令执行速度以及端口操作速度非常快的话，就需要酌情在连续操作端口的代码之间加入适量的延时（通用用空操作来代替，具体多少个多少时长视具体的 MCU 以及 LCD 控制器而定）以保证该信号的脉冲宽度满足要求。

在本文的所列出的源代码当中，并没有如前所述的为时序的要求而插入空操作或延时处理，因为 MCU 的速度并不是非常快，况且现在的 LCD 控制器的总线速度都挺快的了，没有必要加入而已。