Управление контроллером графического ЖКИ индикатора T6963C на примере индикатора PG24064-E/F.

В данной статье рассмотрены основные команды контроллера T6963C и показаны примеры их применения, написанные на языке C. Примеры были опробованы на микроконтроллере (далее МК) Fujitsu MB90598G, однако вследствие универсальности C они должны подойти для любого микроконтроллера. Статья не является переводом оригинальной документации на T6963C, здесь рассмотрены лишь основные моменты его программирования.

Графический ЖКИ индикатор состоит из трех основных логических узлов: LCD панели, LCD контроллера и ОЗУ. Для контроллера T6963C максимальный размер LCD панели может составлять 240x128 точек, максимальный объем ОЗУ 64 кБ. У индикатора PG24064-E/F размер панели равен 240x64 точек, емкость ОЗУ 8 кБ. В ОЗУ хранится информация, которая считывается контроллером и выводится на LCD панель в соответствии с заданным режимом индикации. Поэтому задача программиста сводится к правильному заполнению ОЗУ и заданию режимов работы контроллера.

Управление контроллером осуществляется по следующим линиям:

1. Двунаправленная шина данных D0... D7;
2. Линии разрешения чтения, разрешения записи, выбора кристалла и выбора типа пересылаемых данных (команда или данные);
3. Линия выбора размера шрифта (6x8 или 8x8 точек);
4. RESET.

На C определения этих линий для приводящихся ниже примеров выглядят так (для применения в какой-либо конкретной схеме или для другого микроконтроллера их необходимо изменить):

```
#define WR PDR6_P67//линия разрешения записи
#define RD PDR6_P66//линия разрешения чтения
#define CE PDR6_P65//Выбор кристалла
#define CD PDR6_P64//выбора типа пересылаемых данных
#define RESET PDR5_P57//сброс
#define FS PDR5_P56//выбора размера шрифта
#define LCD_PORT PDR2//Шина данных
#define LCD_PORT_DIR DDR2//Байт направления шины данных
```

Существуют четыре типа обмена данными между МК и T6963C: запись данных, чтение данных, запись команды и чтение байта состояния контроллера. Перед любой записью или чтением необходимо считать байт состояния и произвести проверку его битов для того, что бы убедиться в готовности контроллера к дальнейшей работе. Даже в том случае, если в последнее время к контроллеру обращения не производилось, байт состояния обязательно должен быть считан, т.е. простого введения задержки между двумя командами недостаточно. Для получения байта состояния небоходимо на линиях CD и WR выставить единицы, а на CE и RD нуль. Примерно через 200нс на шине данных будет байт состояния. На C эта функция выглядит так:

```
unsigned char GetStatusLCD(void)
{
unsigned char i;

LCD_PORT_DIR=0;//Переключение направления шины на вход

CD=1;
WR=1;
CE=0;
RD=0;

for (i=0;i<2;i++);//Задержка

i=LCD_PORT;//Чтение байта состояния

WR=1;
RD=1;
CE=1;
CD=1;

LCD_PORT_DIR=0xff;//Переключение направления шины на выход

return i;//Возврат байта состояния
}
```

Ниже приводится байт состояния побитно:

D7	D6	D5	D4	D3	D2	D1	D0
STA7	STA6	STA5	X	STA3	STA2	STA1	STA0

1. STA0 - 0=команда в процессе обработки, 1=контроллер готов получить новую команду или данные;
2. STA1 - 0=производится внутреннее чтение запись (контроллер занят), 1=контроллер готов получить новую команду или данные;
3. STA2 - 0=производится чтение в режиме "Auto Read", 1=контроллер свободен для режима "Auto Read";
4. STA3 - 0=производится запись в режиме "Auto Write", 1=контроллер свободен для режима "Auto Write";
5. X - не используется
6. STA5 - 0=контроллер не работает, 1=контроллер работает нормально.
7. STA6 - 0=указатель адреса находится в области памяти, выделенной для графики, 1=указатель находится за пределами этой области
8. STA7 - 0=LCD панель выкл.,1=LCD панель вкл.

Таким образом, перед записью/чтением необходимо проверить биты STA0 и STA1 на равенство единице. Так же иногда используются биты STA2 и STA3, которые проверяются перед выполнением автоматического чтения/записи (об этом режиме ниже). Использовать остальные биты автору данной статьи не приходилось. Пример проверки битов STA0 и STA1:

```
while ((GetStatusLCD()&0x03)!=0x03);//Ждем равенства STA0 и STA1 единице
```

Как уже упоминалось выше, обмен данными с T6963C производится либо с помощью команд, которые передаются из МК в T6963C и исполняются непосредственно самим контроллером либо данными, которые записываются в ОЗУ или читаются из него и представляют информацию, выводимую на панель LCD. Для передачи в контроллер команды необходимо на шину выставить саму команду, на линию CD единицу, на WR нуль, а на CE подать нулевой импульс длительностью более 100нс. Запись данных производится аналогично, только на линию CD должен быть подан нуль. Для чтения данных на CD, RD, CE выставляется нуль,после 200нс читается байт с шины данных, затем а CE выставляется единица. Ниже приводится функция записи команды или данных. Если переменная cmd=0, то записываются данные, в противном случае - команда. В переменной bt находится записываемый байт:

```
void Send8LCD(unsigned char cmd, unsigned char bt)
{
char i;

if (cmd)
CD=1;
else
CD=0;

for (i=0;i<2;i++);//задержка

CE=0;
WR=0;
LCD_PORT=bt;

for (i=0;i<2;i++);//задержка

CE=1;
WR=1;
CD=1;
}
```

Функция чтения данных будет выглядеть аналогично и вследствие своей простоты здесь не приводится. Желающие легко смогут написать ее самостоятельно.

Команды, передаваемые в контроллер могут состоять как собственно только из самой команды, так и содержать до двух байт данных. Полное описание команд смотрите в документации производителя. Если команда содержит данные, то они передаются перед ней, младшим байтом вперед.

Пример передачи одиночной команды:

```
while ((GetStatusLCD()&0x03)!=0x03); //Проверка битов состояния STA0 и STA1
Send8LCD(1,0x97); //Передача команды включения дисплея
```

Пример передачи команды с двумя байтами данных:

```
while ((GetStatusLCD()&0x03)!=0x03); //Проверка битов состояния STA0 и STA1
Send8LCD(0,0x01); //Передача младшего байта
while ((GetStatusLCD()&0x03)!=0x03); //Проверка битов состояния STA0 и STA1
Send8LCD(0,0x02); //Передача старшего байта
while ((GetStatusLCD()&0x03)!=0x03); //Проверка битов состояния STA0 и STA1
Send8LCD(1,0x24); //Команда
```

Далее можно перейти непосредственно к программированию контроллера. Перед его инициализацией желательно произвести сброс, выставив на линию RESET нуль:

```
RESET=0;
for (i=0;i<50;i++);//Задержка
RESET=1;
```

Затем необходимо выбрать один из режимов индикации графики и текста. Текст с графикой могут быть объеденены для воспроизведения на LCD панели в одном из трех логических вариантов: "OR", "EXOR" и "AND". К тому же текст может отображаться с использованием встроенного знакогенератора (128 знаков плюс 128 знаков, загружаемых пользователем), либо при помощи только пользовательских шрифтов. Так же имеется режим текстовых аттрибутов, при котором графика отключена, а каждому знаку текста можно задать специальный режим отображения. Зададим режим "OR" со встроенным знакогенератором:

```
while ((GetStatusLCD()&0x03)!=0x03);
Send8LCD(1,0x80);
```

Теперь необходимо разбить ОЗУ на две части для хранения текстовой и графической информации. Это разбиение осуществляется указанием начального адреса каждой из частей (в документации к контроллеру эти значения называются "Text home" и "Graphic home"). Помимо указания начального адреса для каждой части необходимо задать число байт, соответствующих одной строке показываемой информации (эти значения называются "Text area" и "Graphic area"). Например на дисплее длиной 240 точек в одной строке при ширине знака 6 точек поместится 40 знаков. Каждый знак соответствует одному байту и, следовательно необходимо задать 40 байт для текста. Для графики это число так же равно 40, т.к. вследствие того, что ширина текста равна 6 точкам, одним байтом можно вывести на индикацию так же шесть точек (старшие два бита не участвуют). Вывод графической информации производится горизонтально и поэтому на вывод одной линии потребуется так же 40 байт. Если это значение указать больше, то информация после сорокового байта будет выводиться за пределы экрана. Если меньше, то наоборот на экран выведется большее число символов, чем было задано. Определим начальный адрес графической области как 0x0000, текстовой как 0x1700 и отправим все эти значения в контроллер:

```
while ((GetStatusLCD()&0x03)!=0x03);
Send8LCD(0,00);
while ((GetStatusLCD()&0x03)!=0x03);
Send8LCD(0,00);
while ((GetStatusLCD()&0x03)!=0x03);//GRPHIC HOME
Send8LCD(1,0x42);

while ((GetStatusLCD()&0x03)!=0x03);
Send8LCD(0,0x28);
while ((GetStatusLCD()&0x03)!=0x03);
Send8LCD(0,0x00);
while ((GetStatusLCD()&0x03)!=0x03);//GRPHIC AREA
Send8LCD(1,0x43);

while ((GetStatusLCD()&0x03)!=0x03);
Send8LCD(0,0x00);
while ((GetStatusLCD()&0x03)!=0x03);
Send8LCD(0,0x17);
while ((GetStatusLCD()&0x03)!=0x03);//TEXT HOME
Send8LCD(1,0x40);

while ((GetStatusLCD()&0x03)!=0x03);
Send8LCD(0,0x28);
while ((GetStatusLCD()&0x03)!=0x03);
Send8LCD(0,0x00);
while ((GetStatusLCD()&0x03)!=0x03);//TEXT AREA
Send8LCD(1,0x41);
```

И последее, что необходимо сделать при инициализации, так это включить индикацию содержимого ОЗУ (после сброса дисплей выключен). Включать можно отдельно графику, текст, текстовый курсор. Следующая команда включит текст и графику, курсор будет выключен:

```
while ((GetStatusLCD()&0x03)!=0x03);
Send8LCD(1,0x9c);
```

По исполнению этой команды дисплей начнет отображать содержимое ОЗУ, в котором на данный момент находятся случайные значения, т.к. произведенный перед инициализацией сброс на ОЗУ не действует. Для того, что бы что-нибудь показать, необходимо заполнить ОЗУ. Выведем для примера на дисплей цифру "4" в третий столбец четвертой строки. Код этой цифры в знакогенераторе контроллера будет 0x14. Записать его нужно в ОЗУ по адресу 0x1700+0x3*0x28+0x2=0x177a. Сделать это можно следующим образом:

```
while ((GetStatusLCD()&0x03)!=0x03); //Поместить в указатель адреса значение 0x177a
Send8LCD(0,0x7a); //младший байт
while ((GetStatusLCD()&0x03)!=0x03);
Send8LCD(0,0x17); //старший байт
while ((GetStatusLCD()&0x03)!=0x03);
Send8LCD(1,0x24); //команда

while ((GetStatusLCD()&0x03)!=0x03); //Вывод знака
Send8LCD(0,0x14); //запись кода числа "4"
while ((GetStatusLCD()&0x03)!=0x03);
Send8LCD(1,0xC0); //команда
```

Как видно, для записи одного знака требуется сначала установить указатель адреса на требуемую позицию, а затем вывести сам знак. После этого при использовании данной команды указатель адреса будет инкрементирован. Такая возможность позволяет последовательно заполнять ОЗУ данными без установки указателя адреса (например, производить вывод строк символов). Так же существуют команды записи байта с декрементом или без изменения указателя.

В том случае, если требуется вывести относительно большой массив данных, то при использования вышеприведенных команд после каждого байта будет необходимо передавать команду записи, что значительно замедляет вывод. Поэтому в контроллере существует режим автозаписи (авточтения), позволяющий один раз перейти в этот режим, а затем производить непрерывную передачу (прием) данных без использования каких-либо команд. Ниже приведен пример вывода строки "0123456789" в этом режиме. Строка выводится, начиная с первого столбца первой строки (адрес в ОЗУ 0x1700).

```
while ((GetStatusLCD()&0x03)!=0x03); //Поместить в указатель адреса значение 0x1700
Send8LCD(0,0x00); //младший байт
while ((GetStatusLCD()&0x03)!=0x03);
Send8LCD(0,0x17); //старший байт
while ((GetStatusLCD()&0x03)!=0x03);
Send8LCD(1,0x24); //команда

while ((GetStatusLCD()&0x03)!=0x03); //Включение режима автозаписи
Send8LCD(1,0xb0);

//Далее запись символов
while ((GetStatusLCD()&0x08)!=0x08);
Send8LCD(0,0x10);//"0"
while ((GetStatusLCD()&0x08)!=0x08);
Send8LCD(0,0x11);//"1"
while ((GetStatusLCD()&0x08)!=0x08);
Send8LCD(0,0x12);//"2"
while ((GetStatusLCD()&0x08)!=0x08);
Send8LCD(0,0x13);//"3
while ((GetStatusLCD()&0x08)!=0x08);
Send8LCD(0,0x14);//"4
while ((GetStatusLCD()&0x08)!=0x08);
Send8LCD(0,0x15);//"5
while ((GetStatusLCD()&0x08)!=0x08);
Send8LCD(0,0x16);//"6
while ((GetStatusLCD()&0x08)!=0x08);
Send8LCD(0,0x17);//"7
while ((GetStatusLCD()&0x08)!=0x08);
Send8LCD(0,0x18);//"8
while ((GetStatusLCD()&0x08)!=0x08);
Send8LCD(0,0x19);//"9

while ((GetStatusLCD()&0x08)!=0x08);//Выключение режима автозаписи
Send8LCD(1,0xb2);
```

Как было указано выше, графическая информация при FS=1 выводится по 6 точек в байте. Младший бит соответствует левой точке, установленный бит - включенной точке. Запись байта в графическую область производится точно также как и в текстовую. Поэтому, если в приведенном выше примере вывода символа "4" указатель адреса установить в графической области ОЗУ (например 0x0000), то вместо символа "4" появится линия из шести точек, которые будут включены или погашены в следующем порядке (слева направо): погашена, погашена, включена, погашена, включена, погашена.

Помимо вывода графики побайтно также существует возможность установки или сброса одного бита в байте по заданному адресу. Это позволяет управлять отдельными точками на экране. Ниже приведена функция, позволяющая засвечивать или гасить отдельный бит по заданному адресу.

```
void PutPixelXYLCD(unsigned char X,unsigned char Y,unsigned char bt)
{
unsigned int i;

i=(unsigned int)X/6;
i=(unsigned int)(LCD_GraphicHome)+(unsigned int)X/6+(unsigned int)Y*0x28;
SetAddrLCD((unsigned int)(LCD_GraphicHome)+(unsigned int)X/6+(unsigned int)Y*0x28);//Установка адреса

while ((GetStatusLCD()&0x03)!=0x03);
//Вывод точки.
if (bt)
Send8LCD(1,(0xf8|(5-(X-(X/6)*6))));
else
Send8LCD(1,(0xf0|(5-(X-(X/6)*6))));
}
```

В этой функции используется функция установки указателя адреса, позволяющая выполнять эту операцию в одной строке, что облегчает читаемость программы. Вот ее текст:

```
void SetAddrLCD(unsigned int addr)
{

while ((GetStatusLCD()&0x03)!=0x03);
Send8LCD(0,(unsigned char)addr);
while ((GetStatusLCD()&0x03)!=0x03);
Send8LCD(0,(unsigned char)(addr>>8));
while ((GetStatusLCD()&0x03)!=0x03);
Send8LCD(1,0x24);
}
```

Для применения функции PutPixelXYLCD(X, Y, bt) в первой переменной необходимо задать столбец и строку точки (нумерация с нуля) и, в том случае, если точку необходимо погасить - нуль, а если засветить - не нуль. Ниже приводится еще одна функция, позволяющая аналогично выводить текст:

```
void SendCharXYLCD(unsigned char X,unsigned char Y,unsigned char bt)
{

SetAddrLCD((unsigned int)(LCD_TextHome)+(unsigned int)X+(unsigned int)Y*0x28);
while ((GetStatusLCD()&0x03)!=0x03);
Send8LCD(0,bt);
while ((GetStatusLCD()&0x03)!=0x03);
Send8LCD(1,0xC4);
}
```

Во всех вышеприведенных примерах индикатор использовался одновременно как в текстовом, так и в графическом режиме. Однако, если графика не требуется, то ее можно отключить, а вместо нее использовать так называемый режим текстовых аттрибутов. В этом режиме для каждого символа текста можно задать один из нескольких способов его отображения. Для включения этого режима необходимо командой "MODE SET" задать режим "Text Attribute", а командой "DISPLAY MODE" - включить как текст, так и графику. Чтобы задать текстовый аттрибут для символа, находящегося в текстовой области ОЗУ, необходимо записать соответствующий байт в графическую область ОЗУ с тем же смещением. Например, выведем негативную мигающую цифру "7" по адресу 0x1710. Для того, что бы она мигала необходимо по адресу 0x0010 записать значение 0x0d.

```
while ((GetStatusLCD()&0x03)!=0x03);//Задаем режим текстовых аттрибутов
Send8LCD(1,0x84);
while ((GetStatusLCD()&0x03)!=0x03);//Задаем режим отображения графики и текста
Send8LCD(1,0x9f);
```

```
SetAddrLCD(0x1710);
while ((GetStatusLCD()&0x03)!=0x03); //Вывод знака
Send8LCD(0,0x17); //запись кода числа "7"
while ((GetStatusLCD()&0x03)!=0x03);
Send8LCD(1,0xC0); //команда

SetAddrLCD(0x0010);
while ((GetStatusLCD()&0x03)!=0x03); //Вывод аттрибута
Send8LCD(0,0x0D);
while ((GetStatusLCD()&0x03)!=0x03);
Send8LCD(1,0xC0); //команда
```

И последнее, что будет рассмотрено в данной статье - это способы загрузки пользовательских шрифтов. Шрифты загружаются в текстовую область ОЗУ. Начальный адрес для загрузки шрифтов не должен перекрывать область вывода текстовой информации (в нашем случае это 0x1700+0xff). Пусть для примера начальным адресом будет 0x1c00. Перед загрузкой шрифтов необходимо задать так называемое смещение, которое зависит от области памяти, в которую выводятся шрифты. Ниже приведена таблица зависимости смещения от адресов в памяти.

Смещение	Адреса ОЗУ
0x00	0x0000 - 0x07ff
0x01	0x0800 - 0x0fff
0x02	0x1000 - 0x17ff
0x03	0x1800 - 0x1fff
...	...
0x1f	0xf800 - 0xffff

Как видно из таблицы, в рассматриваемом примере смещение должно быть равно 0x03. Затем, после установки смещения задается начальный адрес (0x1c00) и загружаются знаки. В том случае, если командой "MODE SET" был выбран режим "Internal CG ROM mode", можно будет ввести только 128 знаков, которые будут доступны начиная с 0x80 адреса знакогенератора, а по адресам 0x00-0x7f будут находиться знаки ПЗУ контроллера. В этом случае байту 0x80, записанному в текстовую область контроллера будет соответствовать знак, находящийся по адресу 0x1C00. При выборе режима "External CG RAM mode" пользователь может задать 256 символов, которые полностью заменят знаки, находящиеся в ПЗУ контроллера. Для примера загрузим в контроллер букву "Д" в режиме "Internal CG ROM mode". В двоичном виде она будет выглядеть так:

```
B'00001111
B'00000101
B'00000101
B'00001001
B'00001001
B'00010001
B'00011111
B'00010001
```

Загрузка начинается с верхней строки. Далее приводится текст примера загрузки и вывода на индикацию буквы "Д".

```
while ((GetStatusLCD()&0x03)!=0x03);//Задаем смещение
Send8LCD(0,0x03);
while ((GetStatusLCD()&0x03)!=0x03);
Send8LCD(0,0x00);
while ((GetStatusLCD()&0x03)!=0x03);
Send8LCD(1,0x22);

SetAddrLCD(0x1c00);//Устанавливаем начальный адрес для загрузки шрифта

SendByteIncLCD(0x0f);//Загрузка буквы
SendByteIncLCD(0x05);
SendByteIncLCD(0x05);
SendByteIncLCD(0x09);
SendByteIncLCD(0x09);
SendByteIncLCD(0x11);
SendByteIncLCD(0x1f);
SendByteIncLCD(0x11);

SendCharXYLCD(0,0,0x80);//Вывод буквы в нулевой столбец нулевой стоки
```

где SendByteIncLCD(bt) - функция загрузки байта с инкрементом указателя адреса:

```
void SendByteIncLCD(unsigned char bt)
{
while ((GetStatusLCD()&0x03)!=0x03);
Send8LCD(0,bt);
while ((GetStatusLCD()&0x03)!=0x03);
Send8LCD(1,0xC0);
}
```

На этом рассмотрение основных функций контроллера T6963C можно считать законченным, хотя помимо описанных у него имеются еще и другие возможности. Однако на взгляд автора давать их полное описание смысла не имеет, все можно найти в документации.
