STM32 и дисплей. Использование FSMC.

Posted on 30.04.2013 by Aveal

Пришло время обсудить замечательную плюшку микроконтроллеров STM32 – а именно модуль FSMC. Это практически незаменимая вещь при работе с внешней памятью, либо, например, с графическим дисплеем. Собственно, с дисплеем то мы и будем играться, разбираясь с FSMC.

Но для начала, как обычно, немного теории. Итак, FSMC реализует параллельный интерфейс обмена данными между различными устройствами. Короче говоря — просто параллельная шина ☺ Используя FSMC при работе с внешней памятью, мы получаем возможность включить внешнюю память в адресное пространство микроконтроллера. Что это дает? А то, что обращение к внешней памяти значительно упрощается — необходимо просто обращаться к ОЗУ микроконтроллера по определенным заданным адресам. То есть все ритуальные танцы с временными диаграммами, таймингами и прочим модуль FSMC берет на себя. Мы просто пишем данные по адресу — а FSMC дергает линии данных, полностью осуществляя непосредственную работу с подключаемым устройством.

Схожим образом работает все это дело и при подключении дисплеев. Но все-таки тут все несколько иначе. Пусть у нас есть дисплей, у которого есть следующие выводы:

DB[17:0] – 18 линий для передачи данных (не забываем, что тут у нас параллельная передача данных, а не последовательная)

Также есть выводы для разрешения записи/чтения — туда мы должны выдавать строб-импульсы в определенной последовательности

Кроме того, у дисплея есть выводы chip select'a, reset, полно всего короче)

И всем этим хозяйством нужно рулить) Вот тут то нам и поможет FSMC. И не просто поможет, а всю работу возьмет на себя! Итак, пусть мы уже подключили дисплей как надо, написали программу для FSMC STM32. Как же нам обращаться с дисплеем то? А опять все очень просто. Точно также, как с внешней памятью, мы будем всего лишь пихать байты по определенному адресу. А FSMC будет в это время лихорадочно дергать линиями данных, следить за временными интервалами, передавать все остальные нужные дисплею сигналы. А чтобы разделить передачу данных и команд мы должны записывать байты по разным адресам. И тут же встает вопрос – а какие именно адреса, и чем они определяются.

У дисплея есть вывод для выбора — данные/команда. То есть по состоянию этого вывода дисплей решает, что именно сейчас к нему прилетит. А у FSMC есть шина адреса и шина данных. Так вот этот вывод дисплея подключается к какому-нибудь пину шины адреса. И если мы подключили его, например, к 16 выводу шины адреса, то записав какой-нибудь байт по любому(!) адресу с нулевым 16 битом мы подадим дисплею команду. Вот, смотрите, пример небольшой.

Пусть как уже решили подключен 16 бит шины адреса. Берем адрес из доступных FSMC – 0x60000000. Видим, что в этом значении 16 бит равен нулю, а значит, когда мы запишем значение по этому адресу, FSMC подаст дисплею сигнал на запись, также сообщит дисплею, что сейчас будет команда, ну и, конечно же, выдаст на шину сами данные. А если мы запишем значение по адресу 0x60010000 (16 бит — единица!) то FSMC все разрулит и передаст дисплею данные. Вот так все просто)

Кстати, очень важный момент. В 16-битном режиме работы FSMC 16 бит шины адреса соответствует 17-му биту адреса (то есть 0x60020000). Ну естественно, все остальные адреса также смещены на 1 бит в этом режиме.

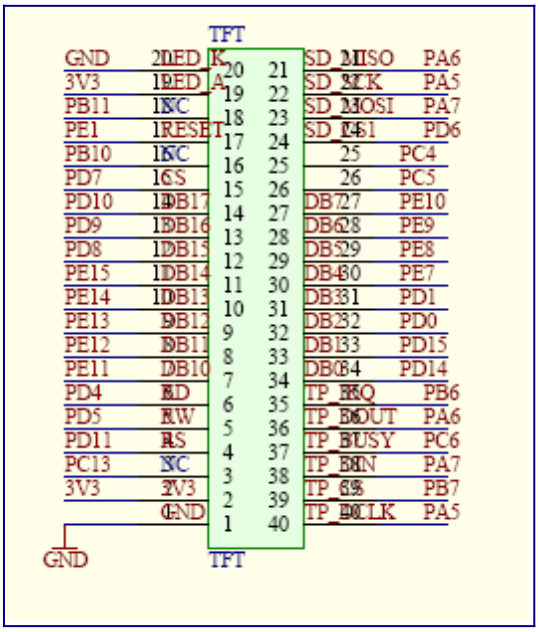
По идее работа с дисплеем и с внешней памятью с точки зрения программиста выглядит одинаково, но на деле все не так. При работе с дисплеем никакая память никуда не «проецируется», мы условно пишем данные по адресам, но это всего лишь дает FSMC сигнал о том, что пора начинать действовать ☺

Давайте потихоньку переходить к делу.

Для работы с FSMC будем по традиции использовать Standard Peripheral Library. Там все аналогично любой другой периферии, разве что настроек побольше) Так что об этом не будем особо разговаривать — лучше на практике при написании программы посмотрим как и что там настраивается.

Что у нас в плане железа…

Испытывать FSMC я буду при помощи платы MiniSTM32 (про нее была уже статейка). Там уже установлен дисплей, так что никаких лишних телодвижений не потребуется. Вот как дисплей подключен:

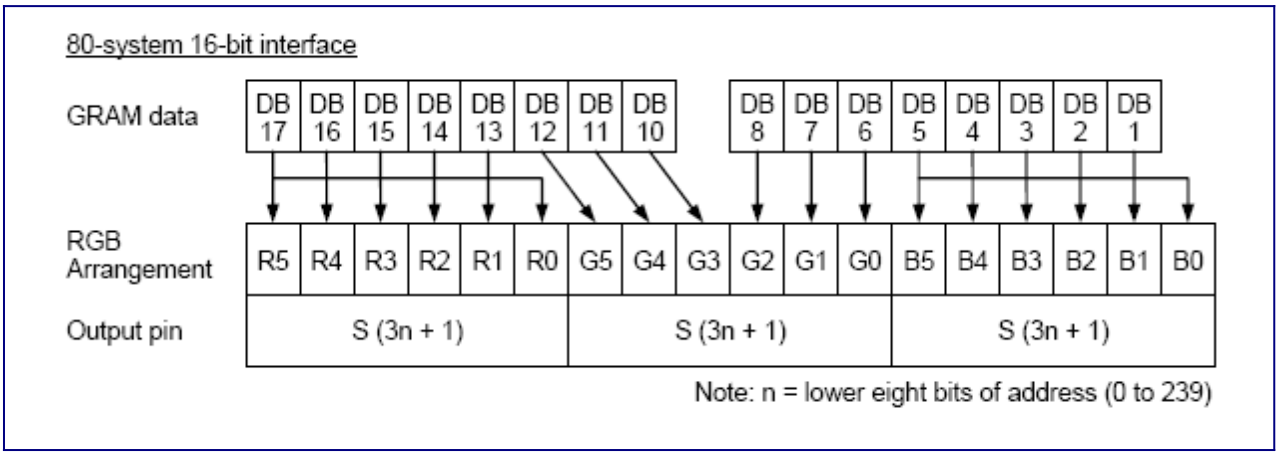


Но вообще есть серьезные опасения, что китайцы нарисовали схему коряво, поскольку тут можно найти явно абсурдные вещи)

Вообще, если посмотреть на распиновку конкретного дисплея и на выводы FSMC микроконтроллера, то там очень хорошо видно, что они довольно точно соответствуют друг другу.

Итак, что же будем делать то в качестве примера…Давайте разберемся как окрашивать дисплей в определенный цвет. Сначала зальем экран красным, потом зеленым и потом синим (RGB). Короче получить мы должны мигающий дисплей, на котором сменяют друг друга три цвета ☺

Данные будем передавать в 16-битном режиме.



Красному цвету соответствует – 111111 000000 000000

Зеленому – 000000 111111 000000

Синему – 000000 000000 111111

Как мы тут видим на 18 бит цвета приходятся 16 бит данных, в итоге получаем следующее:

Для красного – 0xF800

Для зеленого – 0x07E0

Для синего – 0x001F

Все очень просто, давайте напишем программу. Сразу скажу, шаманская инициализация дисплея взята из кошмарных китайских примеров программ, которые шли вместе с платой ☺

```
/*****************************************************************/
#include "stm32f10x_gpio.h"
#include "stm32f10x_fsmc.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x.h"
 /*****************************************************************/
// Определяем адреса, по которым будем записывать данные
// Для записи данных
#define LCD_DATA                          ((uint32_t)0x60020000)
// Для записи команд
#define LCD_REG                           ((uint32_t)0x60000000)
```

```
 /*****************************************************************/
// Простенькая функция задержки
void delay(uint32_t delayTime)
{
  uint32_t i;
  for(i = 0; i < delayTime; i++);
}
 /*****************************************************************/
 // Так мы будем писать команды в регистры LCD
void writeLCDCommand(unsigned int reg,unsigned int value)
{
  *(uint16_t *) (LCD_REG) = reg;
  *(uint16_t *) (LCD_DATA) = value;
}

/*****************************************************************/
// А так данные..
void writeLCDData(unsigned int data)
{
  *(uint16_t *) (LCD_DATA)= data;
 }

/*****************************************************************/
void initAll()
{
  FSMC_NORSRAMInitTypeDef fsmc;
  FSMC_NORSRAMTimingInitTypeDef fsmcTiming;
  GPIO_InitTypeDef gpio;

  // Включаем тактирование портов
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB |RCC_APB2Periph_GPIOC | RCC_APB2Periph_GPIOD | RCC_APB2Periph_GPIOE , ENABLE);
  // И тактирование FSMC
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_FSMC, ENABLE);

  // Инициализация пинов, задействованных в общении по FSMC
  gpio.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_4 | GPIO_Pin_5 | GPIO_Pin_8 | GPIO_Pin_9 | GPIO_Pin_10 | GPIO_Pin_14 | GPIO_Pin_15;
  gpio.GPIO_Mode = GPIO_Mode_AF_PP;
  gpio.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(GPIOD, &gpio);

  gpio.GPIO_Pin = GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9 | GPIO_Pin_10 | GPIO_Pin_11 | GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15;
  gpio.GPIO_Mode = GPIO_Mode_AF_PP;
  gpio.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(GPIOE, &gpio);

  gpio.GPIO_Mode = GPIO_Mode_Out_PP;
  gpio.GPIO_Pin = GPIO_Pin_6;
  GPIO_Init(GPIOD, &gpio);

  // Здесь у нас Reset
  gpio.GPIO_Pin = GPIO_Pin_1 ;
  GPIO_Init(GPIOE, &gpio);

  // CS
  gpio.GPIO_Mode = GPIO_Mode_AF_PP;
  gpio.GPIO_Pin = GPIO_Pin_7;
  GPIO_Init(GPIOD, &gpio);

  // RS
  gpio.GPIO_Pin = GPIO_Pin_11 ;
  GPIO_Init(GPIOD, &gpio);

  // CS -> 1
  // Reset -> 0
  // RD -> 1
  // RW -> 1

  GPIO_SetBits(GPIOD, GPIO_Pin_7);
  GPIO_ResetBits(GPIOE, GPIO_Pin_1);
  GPIO_SetBits(GPIOD, GPIO_Pin_4);
  GPIO_SetBits(GPIOD, GPIO_Pin_5);

  // Настройка FSMC
  fsmcTiming.FSMC_AddressSetupTime = 0x02;
  fsmcTiming.FSMC_AddressHoldTime = 0x00;
  fsmcTiming.FSMC_DataSetupTime = 0x05;
  fsmcTiming.FSMC_BusTurnAroundDuration = 0x00;
  fsmcTiming.FSMC_CLKDivision = 0x00;
  fsmcTiming.FSMC_DataLatency = 0x00;
  fsmcTiming.FSMC_AccessMode = FSMC_AccessMode_B;

  fsmc.FSMC_Bank = FSMC_Bank1_NORSRAM1;
  fsmc.FSMC_DataAddressMux = FSMC_DataAddressMux_Disable;
  fsmc.FSMC_MemoryType = FSMC_MemoryType_NOR;
  fsmc.FSMC_MemoryDataWidth = FSMC_MemoryDataWidth_16b;
  fsmc.FSMC_BurstAccessMode = FSMC_BurstAccessMode_Disable;
  fsmc.FSMC_WaitSignalPolarity = FSMC_WaitSignalPolarity_Low;
  fsmc.FSMC_WrapMode = FSMC_WrapMode_Disable;
```

```
  fsmc.FSMC_WaitSignalActive = FSMC_WaitSignalActive_BeforeWaitState;
  fsmc.FSMC_WriteOperation = FSMC_WriteOperation_Enable;
  fsmc.FSMC_WaitSignal = FSMC_WaitSignal_Disable;
  fsmc.FSMC_ExtendedMode = FSMC_ExtendedMode_Disable;
  fsmc.FSMC_WriteBurst = FSMC_WriteBurst_Disable;
  fsmc.FSMC_ReadWriteTimingStruct = &amp;fsmcTiming;
  fsmc.FSMC_WriteTimingStruct = &fsmcTiming;

  FSMC_NORSRAMInit(&fsmc);
  FSMC_NORSRAMCmd(FSMC_Bank1_NORSRAM1, ENABLE);
}

/*****************************************************************/
void initLCD()
{
  // Глобальный Reset дисплея
  GPIO_ResetBits(GPIOE, GPIO_Pin_1);
  delay(0x0FFFFF);
  GPIO_SetBits(GPIOE, GPIO_Pin_1 );
  delay(0x0FFFFF);

  // Пляски с  бубном от китайских товарищей
  writeLCDCommand(0x0000,0x0001);
  delay(10);

  writeLCDCommand(0x0015,0x0030);
  writeLCDCommand(0x0011,0x0040);
  writeLCDCommand(0x0010,0x1628);
  writeLCDCommand(0x0012,0x0000);
  writeLCDCommand(0x0013,0x104d);
  delay(10);
  writeLCDCommand(0x0012,0x0010);
  delay(10);
  writeLCDCommand(0x0010,0x2620);
  writeLCDCommand(0x0013,0x344d);
  delay(10);

  writeLCDCommand(0x0001,0x0100);
  writeLCDCommand(0x0002,0x0300);
  writeLCDCommand(0x0003,0x1030);
  writeLCDCommand(0x0008,0x0604);
  writeLCDCommand(0x0009,0x0000);
  writeLCDCommand(0x000A,0x0008);

  writeLCDCommand(0x0041,0x0002);
  writeLCDCommand(0x0060,0x2700);
  writeLCDCommand(0x0061,0x0001);
  writeLCDCommand(0x0090,0x0182);
  writeLCDCommand(0x0093,0x0001);
  writeLCDCommand(0x00a3,0x0010);
  delay(10);

  // Настройки гаммы
  writeLCDCommand(0x30,0x0000);
  writeLCDCommand(0x31,0x0502);
  writeLCDCommand(0x32,0x0307);
  writeLCDCommand(0x33,0x0305);
  writeLCDCommand(0x34,0x0004);
  writeLCDCommand(0x35,0x0402);
  writeLCDCommand(0x36,0x0707);
  writeLCDCommand(0x37,0x0503);
  writeLCDCommand(0x38,0x1505);
  writeLCDCommand(0x39,0x1505);
  delay(10);

  // Включение дисплея
  writeLCDCommand(0x0007,0x0001);
  delay(10);
  writeLCDCommand(0x0007,0x0021);
  writeLCDCommand(0x0007,0x0023);
  delay(10);
  writeLCDCommand(0x0007,0x0033);
  delay(10);
  writeLCDCommand(0x0007,0x0133);
}

/*****************************************************************/
int main()
{
  initAll();
  initLCD();

  while(1)
  {
    int i;

    // Начальный и конечный адреса по горизонтали
    writeLCDCommand(0x0050, 0);
```

```
    writeLCDCommand(0x0051, 239);
    // Начальный и конечный адреса по вертикали
    writeLCDCommand(0x0052, 0);
    writeLCDCommand(0x0053, 319);

    writeLCDCommand(32, 0);
    writeLCDCommand(33, 0);
    *(uint16_t *) (LCD_REG) = 34;

    // Красный
    for (i = 0; i < 76800; i++) writeLCDData(0xF800);
    delay(0x0FFFFF);
    // Зеленый
    for (i = 0; i < 76800; i++) writeLCDData(0x07E0);
    delay(0x0FFFFF);
    //Синий
    for (i = 0; i < 76800; i++) writeLCDData(0x001F);
    delay(0x0FFFFF);
  }
}
/*****************************************************************/
```

Магическое число 76800 – количество точек дисплея. Все остальное вроде бы понятно, с настройками FSMC тоже ясно, как всегда в SPL все поля структуры названы логично и адекватно их функции.

После прошивки программы в контроллер дисплей начинает исправно подмигивать. Чего, собственно, и добивались, так что на этом все, скоро попробуем залить в дисплей какое-нибудь изображение)