UG

HPM FEMC 驱动 8080 屏开发指南

适用于上海先楫半导体 HPM 系列高性能微控制器

目录

版本：

日期	版本号	说明
2023-8-2	1.1	初版

1.简介

HPM6000 系列 MCU 是来自上海先楫半导体科技有限公司的高性能实时 RISC-V 微控制器，为工业自动化及边缘计算应用提供了极大的算力、高效的控制能力。上海先楫半导体目前已经发布了如 HPM6700/6400、HPM6300、HPM6200 等多个系列的高性能微控制器产品。

在 HPM6700/6400、6300 系列微控制器上均带了多功能外部存储器 FEMC 控制器。FEMC 可用来挂载外部 SDRAM、SRAM 或兼容 SRAM 接口的器件，以此来解决用户不同的 RAM 大小的需求。FEMC 其高效的高带宽访问速率及灵活的地址映射配置等特性，除了挂载 RAM 外，通常也会用来挂载 FPGA 等器件，把 FPGA 等器件当做 RAM 访问，通过地址读写即可高效灵活的访问 FPGA 等外设。

本文介绍了如何用 FEMC 控制器驱动 8080 屏。将 8080 屏通过 FEMC 控制器挂载，像访问 RAM 操作，通过读写对应地址即可轻松驱动点亮 8080 屏，并实现高速的刷新率。本文只介绍 FEMC 驱动 8080 屏的部分，针对 8080 屏命令和数据配置不做介绍，命令及数据配置详细查看 8080 对应屏的手册。

2.FEMC 介绍

FEMC(Flexible External Memory Controller)，多功能外部存储控制器。FEMC 包含 DRAM 控制器和 SRAM 控制器 2 套外部存储器模块。

FEMC 特性：

● DRAM 控制器，支持连接外部 SDRAM

– 支持 8/16/32 位数据模式

– 可以使用高 16 位数据线访问 16 位 SDRAM(当低 16 位数据线的 IO 被其他功能占用时)

– 最大支持 166MHz 工作频率

– 支持通过 APB 总线往 SDRAM 发命令 (READ, WRITE, MODE_SET, AUTO_REFRESH 等)

– 32 位 AXI 总线，内部读写数据 buffer，最大 outstanding 支持 8 级读和 8 级写

– auto-refresh 时间, bank 数, CAS 延迟, column 地址位数, burst 长度可配

● SRAM 控制器，支持连接外部 SRAM 或者兼容 SRAM 访问接口的器件

– 支持异步访问

– 支持数据地址复用模式 (ADMUX) 或者非复用模式 (Non-ADMUX)

SRAM 信号说明：

- D0~D15/AD0~AD15：在数据和地址非复用 AD Non-MUX 模式下，为数据输入和输出信号 0~15；在数据和地址复用 ADMUX 模式下，地址信号 0~15 和数据信号 0~15 共享信号 AD0~AD15；
- A0~A7/A16~A23：在数据和地址非复用 AD Non-MUX 模式下，为地址信号 0~7；在数据和地址复用 ADMUX 模式下，为地址信号 16~23；
- NWE：写使能信号 (Write Enable)，低电平有效；
- NOE：读使能信号 (Read Enable)，低电平有效；
- NCE：片选信号 (Chip Enable)，低电平有效；
- NLB：低字节控制 (Lower-Byte Control)，低电平有效；
- NUB：高字节控制 (Upper-Byte Control)，低电平有效；
- NADV：在数据和地址复用 ADMUX 模式下，为地址/数据有效信号 (address/data valid)，可以用来锁存地址；

SRAM 数据地址非复用模式下的时序图如下：

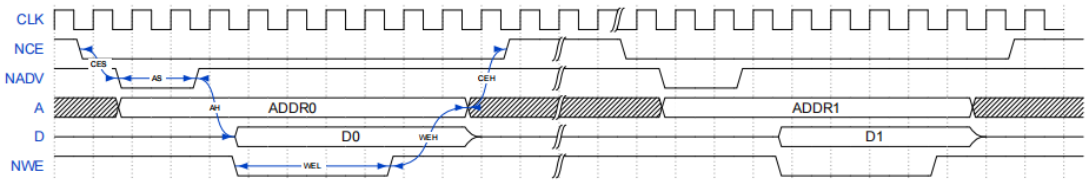

图 1 SRAM 非复用模式下写操作时序

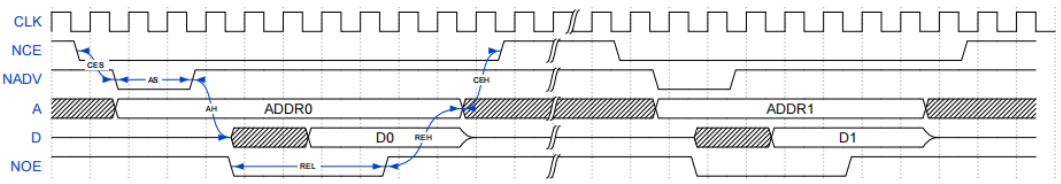

图 2 SRAM 非复用模式下读操作时序

3.8080 屏介绍

8080 屏特指的是支持 8080 并口协议的 LCD 显示屏。8080 协议是由英特尔公司提出的一种并口协议。8080 并行总线协议，在扩展方面及数据传输方面的优越性，在追求高速、近距离的接口和传输方式上应用非常广泛。

8080 并口信号说明：

- DB[0:15]：数据信号
- RDX：读数据控制信号，低电平有效
- DCX(RS)：数据/命令选择信号 ，低：命令；高：数据；
- RESET：复位信号
- WRX：写数据控制信号，低电平有效
- CSX：片选信号，低电平有效

8080 并口协议时序图如下：

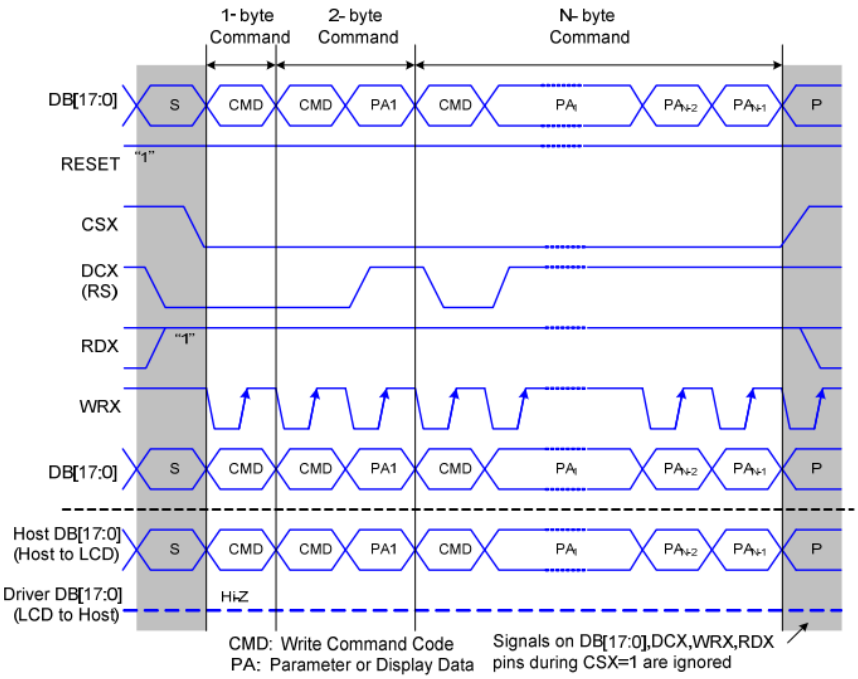

图 3 8080 并口协议写时序

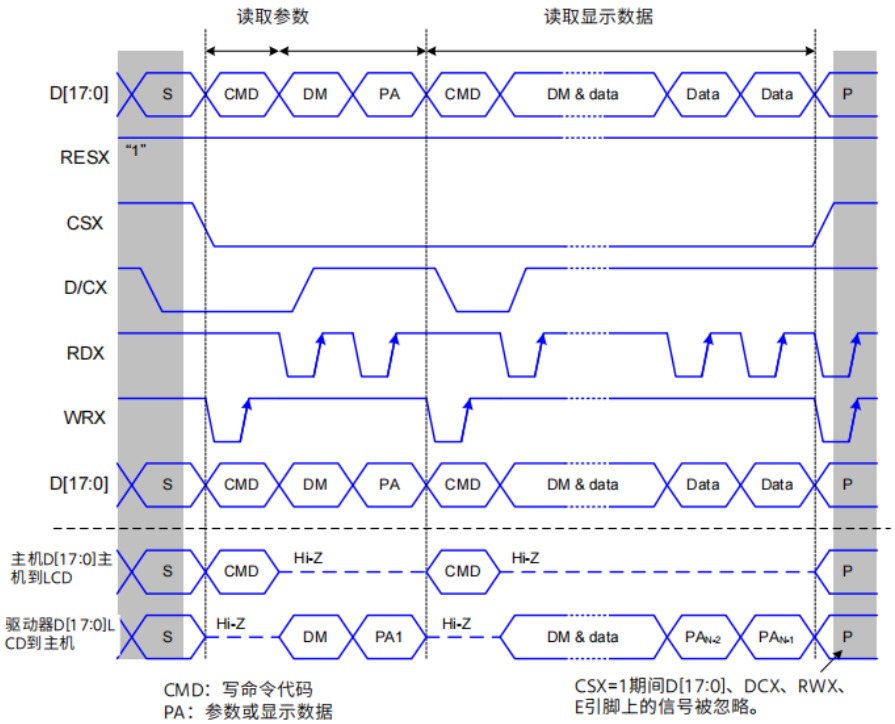

图 4 8080 并口协议读时序

4.FEMC 驱动 8080 屏

从上述 8080 协议时序图和 SRAM 时序图对比可知，8080 并口时序和 SRAM 时序基本一致。因此，我们可以使用 FEMC 控制器来驱动 8080 屏。

注意：A.对于 RESET、背光 PWR 等引脚，完全使用普通 GPIO 控制即可；RESET 时序等本文不做介绍，详细查阅 8080 屏手册。

B. CPU 访问 FEMC 挂载的外设或存储器时内部 cache 是有效的。因此，对 FEMC 挂载的 8080 屏映射地址范围需设置为 nocache 区。

从信号分配上来看：

8080 并口信号和 FEMC SRAM 信号对应如下：

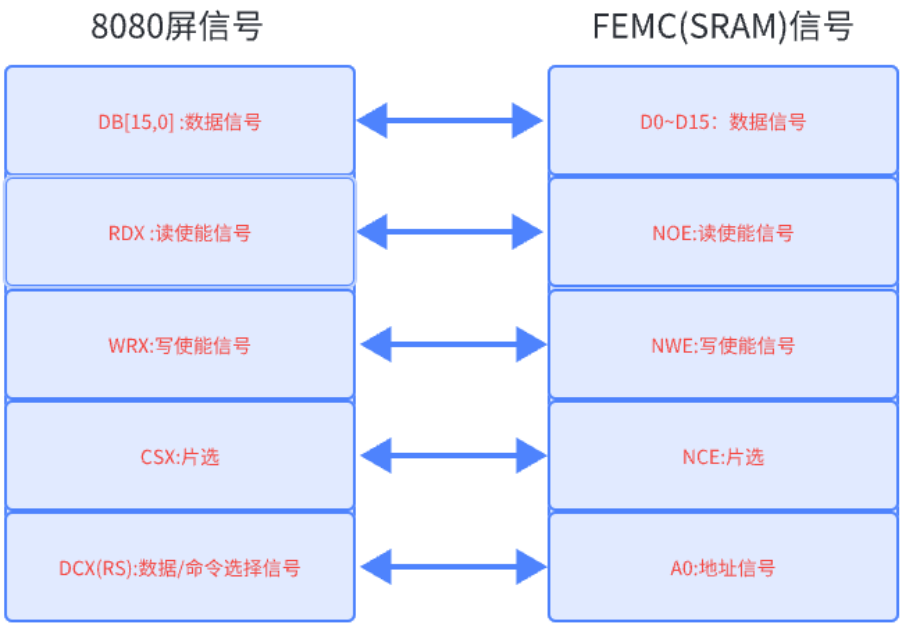

图 5 8080 信号和 FEMC 信号连接图

从时序和信号分配上发现 8080 屏 DCX(RS)脚：数据/命令选择信号和 FEMC 信号不匹配；在这里，我们使用一根地址线来接 DCX(RS)脚。可以通过操作对应的地址来控制地址线 DCX(RS)的高低，以此来区分是读写命令或读写数据；

根据手册查阅，FEMC 内存映射地址从 0x40000000~0x4FFFFFFF 共 256MB。从时序图上看 8080 并口时序和 FEMC SRAM 非复用地址模式时序一致。在这里，

我们设置 FEMC SRAM 基地址为 0x48000000，使用 A0(bit0)地址线接 DCX，因此当操作 0x48000001 地址时，A0 地址线会被拉高；当操作 0x48000000 地址时，A0 地址线会输出低；故，在硬件连接好、配置好 FEMC 为 SRAM 非复用模式下，通过读写以上两个地址，8080 屏读写时序就会被 FEMC 自动完成。

以下开发方案对应型号 ST7780V2 TFT-LCD 显示屏，其它 8080 屏驱动方案除一些初始参数不同外其它基本一致，详细查看对应型号屏的手册。

本文应用的 8 位并口数据总线，硬件上的连线如下：

DB[0,7] —— D0~D7

DCX(RS) —— A0

RDX —— OE

WRX —— WE

CSX —— CE

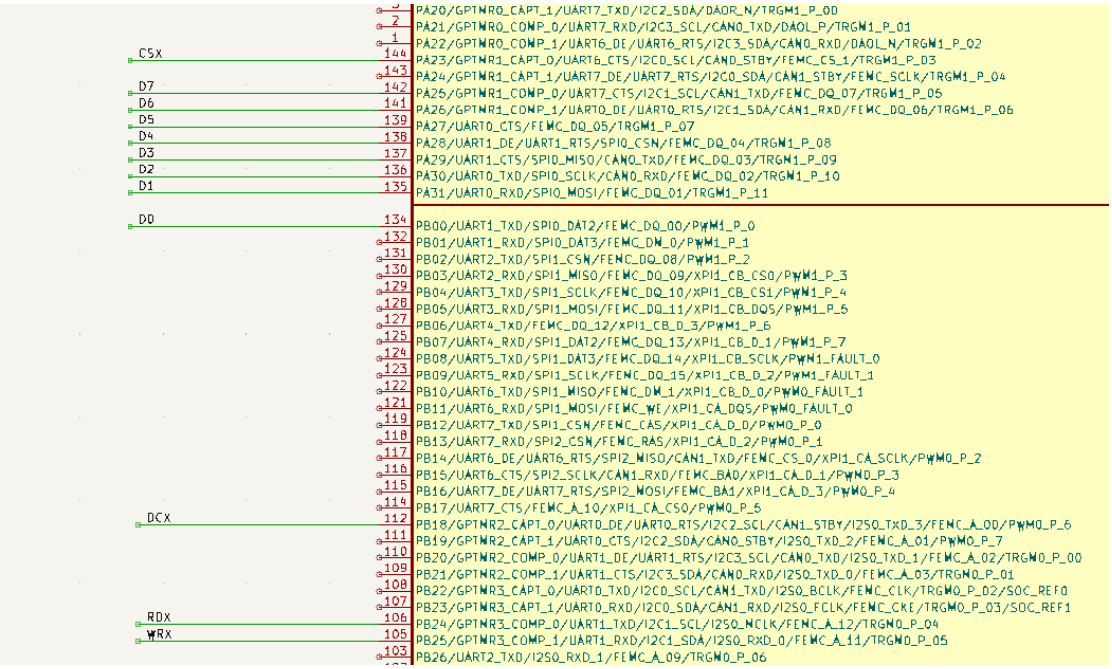

图 6 8080 屏和 HPM6300 系列硬件原理图

a) . 设置 8080 屏映射地址范围为 nocache 区

Linker 文件修改：

lcd8080.c board.c flash_xip_8080.icf

```
define region XPI0 = [from 0x80003000 size _flash_size - 0x3000 ];   /* XPI0 */
define region ILM = [from 0x00000000 size 128k];    /* ILM */
define region DLM = [from 0x00080000 size 128k];    /* DLM */
define region AXI_SRAM  = [from 0x01080000 size 256k];
define region NONCACHEABLE_RAM = [from 0x010C0000 size 256k];
define region LCD8080_RAM = [from 0x48000000 size 256k];
define region AHB_SRAM = [from 0xF0300000 size 32k];

assert (__STACKSIZE__ + __HEAPSIZE__) <= 128k with error "stack and heap total size larger than 128k";

/* Blocks */
define block vectors with fixed order       { section .vector_table, section .isr_vector };
define block vectors_s with fixed order     { section .vector_s_table, section .isr_s_vector };
define block ctors                          { section .ctors,     section .ctors.*, block with         alphab
define block dtors                          { section .dtors,     section .dtors.*, block with reverse alphab
define block eh_frame                       { section .eh_frame,  section .eh_frame.* };
define block tbss                           { section .tbss,      section .tbss.*  };
define block tdata                          { section .tdata,     section .tdata.* };
define block tls                            { block tbss, block tdata };
define block tdata_load                     { copy of block tdata };
define block heap  with size = __HEAPSIZE__,  alignment = 8, /* fill =0x00, */ readwrite access { };
define block stack with size = __STACKSIZE__, alignment = 8, /* fill =0xCD, */ readwrite access { };
define block cherryusb_usbh_class_info with alignment = 8    { section .usbh_class_info };
define block framebuffer with alignment = 8                  { section .framebuffer };
define block boot_header with fixed order                    { section .boot_header, section .fw_info_table,

/* Symbols */
define exported symbol __nor_cfg_option_load_addr__ = start of region NOR_CFG_OPTION;
define exported symbol __boot_header_load_addr__ = start of region BOOT_HEADER;
define exported symbol __app_load_addr__ = start of region XPI0;
define exported symbol __app_offset__ = __app_load_addr__ - __boot_header_load_addr__;
define exported symbol __boot_header_length__ = size of block boot_header;
define exported symbol __fw_size__ = 0x1000;

define exported symbol __noncacheable_start__  = start of region NONCACHEABLE_RAM;
define exported symbol __noncacheable_end__    = end of region NONCACHEABLE_RAM + 1;
define exported symbol __8080_start__   = start of region LCD8080_RAM;
define exported symbol __8080_end__   = end of region LCD8080_RAM + 1;
define exported symbol _stack_safe = end of block stack + 1;
```

图 7 8080 映射地址 nocache linker 文件修改

pmp 初始化修改

```
void board_init_pmp(void)

{

  extern uint32_t __noncacheable_start__[];

  extern uint32_t __noncacheable_end__[];

  extern uint32_t __8080_start__[];

  extern uint32_t __8080_end__[];


  uint32_t start_addr = (uint32_t) __noncacheable_start__;

  uint32_t end_addr = (uint32_t) __noncacheable_end__;

  uint32_t length = end_addr - start_addr;

  uint32_t start_addr1 = (uint32_t) __8080_start__;

  uint32_t end_addr1 = (uint32_t) __8080_end__;

  uint32_t length1 = end_addr1 - start_addr1;

  if (length == 0) {

    return;

  }

  if (length1 == 0) {
```

```
    return;
  }
  /* Ensure the address and the length are power of 2 aligned */
  assert((length & (length - 1U)) == 0U);
  assert((start_addr & (length - 1U)) == 0U);
  assert((length1 & (length1 - 1U)) == 0U);
  assert((start_addr1 & (length1 - 1U)) == 0U);
  pmp_entry_t pmp_entry[4] = {0};
  pmp_entry[0].pmp_addr = PMP_NAPOT_ADDR(0x0000000, 0x80000000);
      pmp_entry[0].pmp_cfg.val = PMP_CFG(READ_EN, WRITE_EN, EXECUTE_EN, ADDR_MATCH_NAPOT, REG_UNLOCK);


  pmp_entry[1].pmp_addr = PMP_NAPOT_ADDR(0x80000000, 0x80000000);
  pmp_entry[1].pmp_cfg.val = PMP_CFG(READ_EN, WRITE_EN, EXECUTE_EN, ADDR_MATCH_NAPOT, REG_UNLOCK);


  pmp_entry[2].pmp_addr = PMP_NAPOT_ADDR(start_addr, length);
  pmp_entry[2].pmp_cfg.val = PMP_CFG(READ_EN, WRITE_EN, EXECUTE_EN, ADDR_MATCH_NAPOT, REG_UNLOCK);
  pmp_entry[2].pma_addr = PMA_NAPOT_ADDR(start_addr, length);
  pmp_entry[2].pma_cfg.val = PMA_CFG(ADDR_MATCH_NAPOT, MEM_TYPE_MEM_NON_CACHE_BUF, AMO_EN);


  pmp_entry[3].pmp_addr = PMP_NAPOT_ADDR(start_addr1, length1);
  pmp_entry[3].pmp_cfg.val = PMP_CFG(READ_EN, WRITE_EN, EXECUTE_EN, ADDR_MATCH_NAPOT, REG_UNLOCK);
  pmp_entry[3].pma_addr = PMA_NAPOT_ADDR(start_addr1, length1);
  pmp_entry[3].pma_cfg.val = PMA_CFG(ADDR_MATCH_NAPOT, MEM_TYPE_MEM_NON_CACHE_BUF, AMO_EN);
  pmp_config(&pmp_entry[0], ARRAY_SIZE(pmp_entry));
}
```

b) . PINMUX 设置

```
void init_femc_lcd8080_pins(void)
{
  /* Non-MUX */                                  /* MUX */
  HPM_IOC->PAD[IOC_PAD_PB18].FUNC_CTL = IOC_PAD_FUNC_CTL_ALT_SELECT_SET(12); /* A0 */ /* A16 */


  HPM_IOC->PAD[IOC_PAD_PB00].FUNC_CTL = IOC_PAD_FUNC_CTL_ALT_SELECT_SET(12); /* D0 */ /* AD0 */
  HPM_IOC->PAD[IOC_PAD_PA31].FUNC_CTL = IOC_PAD_FUNC_CTL_ALT_SELECT_SET(12); /* D1 */ /* AD1 */
  HPM_IOC->PAD[IOC_PAD_PA30].FUNC_CTL = IOC_PAD_FUNC_CTL_ALT_SELECT_SET(12); /* D2 */ /* AD2 */
  HPM_IOC->PAD[IOC_PAD_PA29].FUNC_CTL = IOC_PAD_FUNC_CTL_ALT_SELECT_SET(12); /* D3 */ /* AD3 */
  HPM_IOC->PAD[IOC_PAD_PA28].FUNC_CTL = IOC_PAD_FUNC_CTL_ALT_SELECT_SET(12); /* D4 */ /* AD4 */
  HPM_IOC->PAD[IOC_PAD_PA27].FUNC_CTL = IOC_PAD_FUNC_CTL_ALT_SELECT_SET(12); /* D5 */ /* AD5 */
  HPM_IOC->PAD[IOC_PAD_PA26].FUNC_CTL = IOC_PAD_FUNC_CTL_ALT_SELECT_SET(12); /* D6 */ /* AD6 */
  HPM_IOC->PAD[IOC_PAD_PA25].FUNC_CTL = IOC_PAD_FUNC_CTL_ALT_SELECT_SET(12); /* D7 */ /* AD7 */


  HPM_IOC->PAD[IOC_PAD_PA23].FUNC_CTL = IOC_PAD_FUNC_CTL_ALT_SELECT_SET(12); /* #CE */
  HPM_IOC->PAD[IOC_PAD_PB24].FUNC_CTL = IOC_PAD_FUNC_CTL_ALT_SELECT_SET(12); /* #OE */
  HPM_IOC->PAD[IOC_PAD_PB25].FUNC_CTL = IOC_PAD_FUNC_CTL_ALT_SELECT_SET(12); /* #WE */
```

```
  HPM_IOC->PAD[IOC_PAD_PA20].FUNC_CTL = IOC_PA20_FUNC_CTL_GPIO_A_20;
  HPM_IOC->PAD[IOC_PAD_PB20].FUNC_CTL = IOC_PB20_FUNC_CTL_GPIO_B_20;
  gpio_set_pin_output(HPM_GPIO0, GPIO_DO_GPIOA, 20);
  gpio_write_pin(HPM_GPIO0, GPIO_DO_GPIOA, 20, 0);
  gpio_set_pin_output(HPM_GPIO0, GPIO_DO_GPIOB, 20);
  gpio_write_pin(HPM_GPIO0, GPIO_DO_GPIOB, 20, 0);
}
```

c) .FEMC 驱动 8080 屏配置

```
#define SRAM_BASE_ADDR 0x48000000U
#define SRAM_SIZE 2
#define TFTReadData() (*(uint8_t *)0x48000001U)
#define Bank1_LCD_DATA ((uint32_t)0x48000001U)
#define Bank1_LCD_REG ((uint32_t)0x48000000U)
/* LCD write data and register */
#define LCD_WR_DATA(value) ((*(__IO uint8_t *)(Bank1_LCD_DATA)) = ((uint8_t)(value))) // 写数据寄存器
#define LCD_WR_REG(index) ((*(__IO uint8_t *)(Bank1_LCD_REG)) = ((uint8_t)index))   // 写命令寄存器

void femc_lcd8080_config(void)
{
  uint32_t femc_clk_in_hz = board_init_femc_clock();
  femc_config_t config = {0};
  femc_sram_config_t sram_config = {0};
  femc_default_config(HPM_FEMC, &config);
  config.dqs = FEMC_DQS_INTERNAL;
  femc_init(HPM_FEMC, &config);
  femc_get_typical_sram_config(HPM_FEMC, &sram_config);
  sram_config.oeh_in_ns = 100;
  sram_config.oel_in_ns = 70;
  sram_config.weh_in_ns = 35;
  sram_config.wel_in_ns = 35;
  sram_config.ah_in_ns = 50;
  sram_config.as_in_ns = 10;
  sram_config.ceh_in_ns = 10;
  sram_config.ces_in_ns = 15;
  sram_config.base_address = SRAM_BASE_ADDR;
  sram_config.size_in_byte = SRAM_SIZE;
  sram_config.port_size = FEMC_SRAM_PORT_SIZE_8_BITS;
  femc_config_sram(HPM_FEMC, femc_clk_in_hz, &sram_config);
}
```

d) .8080 屏读写控制

```
void lcd8080_read_write_test(void)
{
  for(uint8_t i = 0; i < 0xFF; i++)
  {
    LCD_WR_DATA(i);
  }
  board_delay_ms(100);
  for(uint8_t i = 0; i < 0xFF; i++)
  {
    LCD_WR_REG(i);
  }
  board_delay_ms(100);
  for(uint8_t i = 0; i < 0xFF; i++)
  {
    printf("read:0x%02X\r\n", TFTReadData());
  }
  board_delay_ms(100);
}
```

e) .实测 FEMC 8080 写操作波形

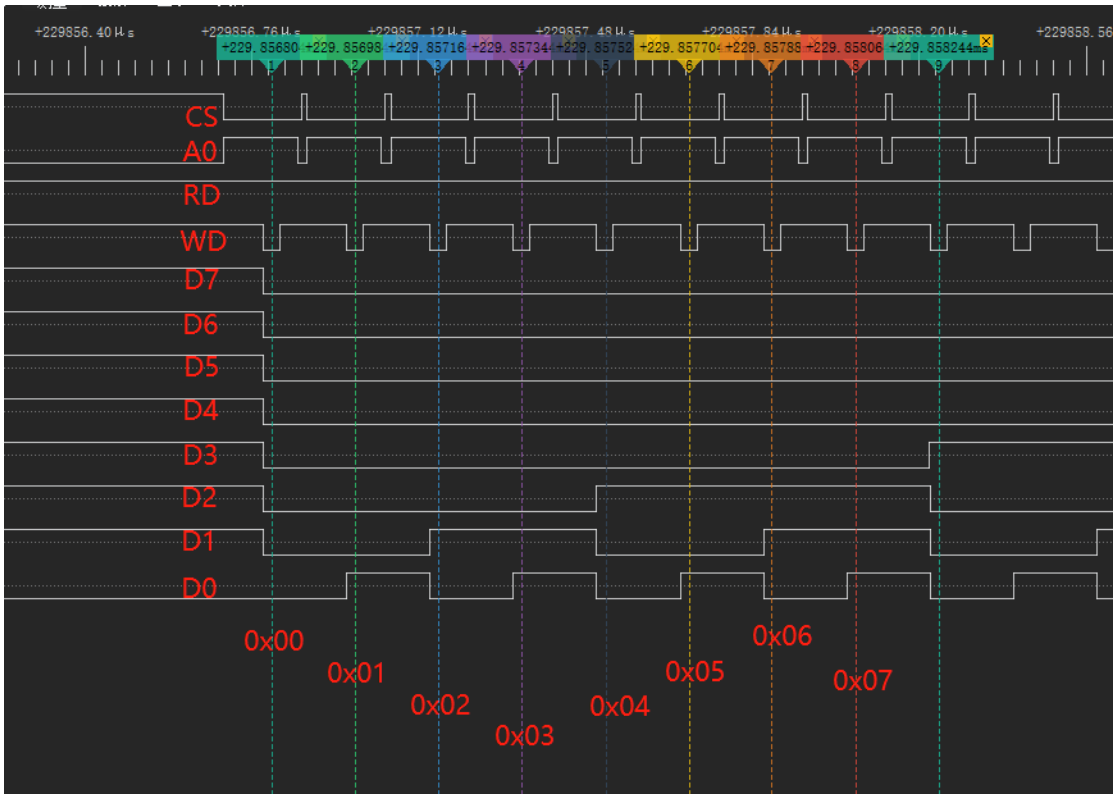

图 8 实测 FEMC 8080 写时序

f) .实测 FEMC 8080 读波形

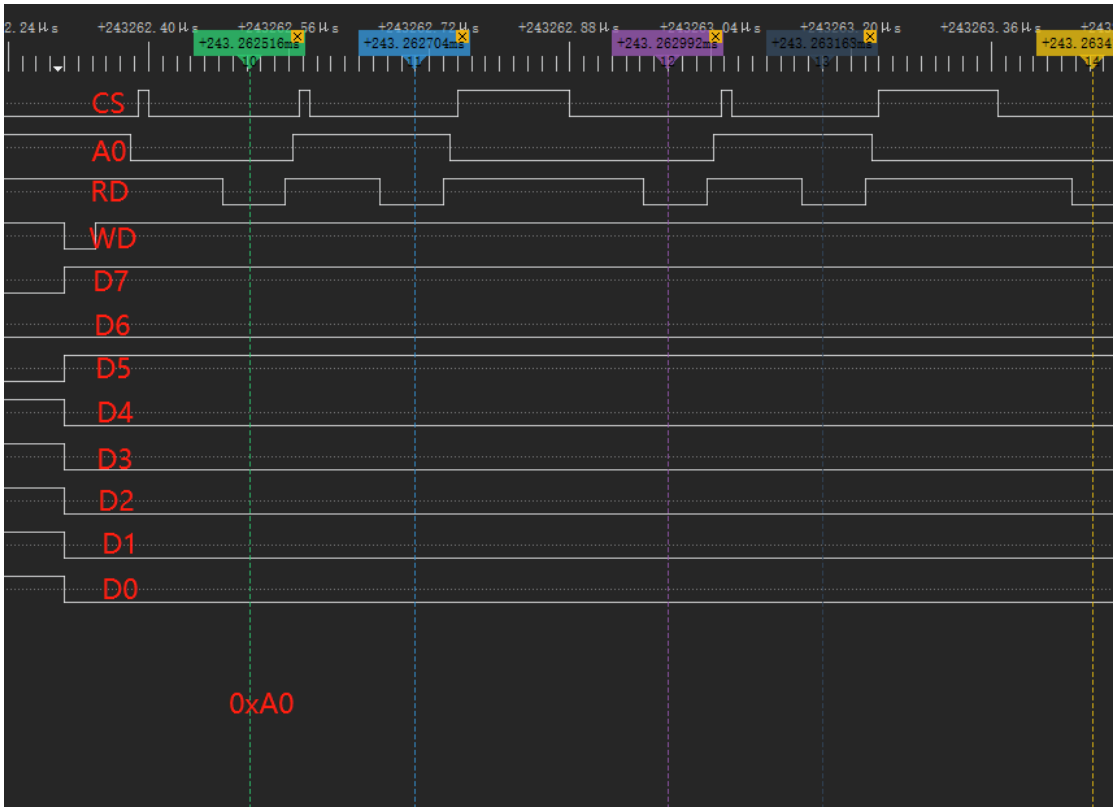

图 9 实测 FEMC 8080 读时序

FEMC 驱动 8080 屏点亮效果图：

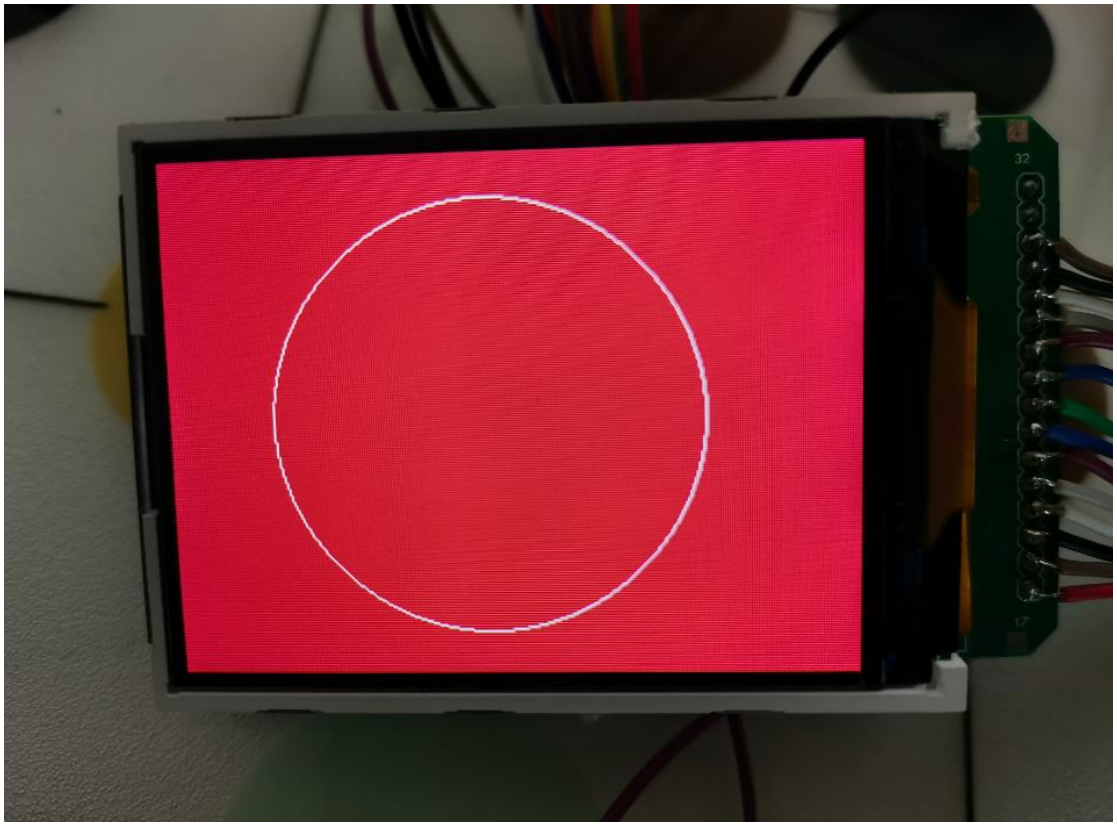

图 10 FEMC 驱动 8080 屏效果图

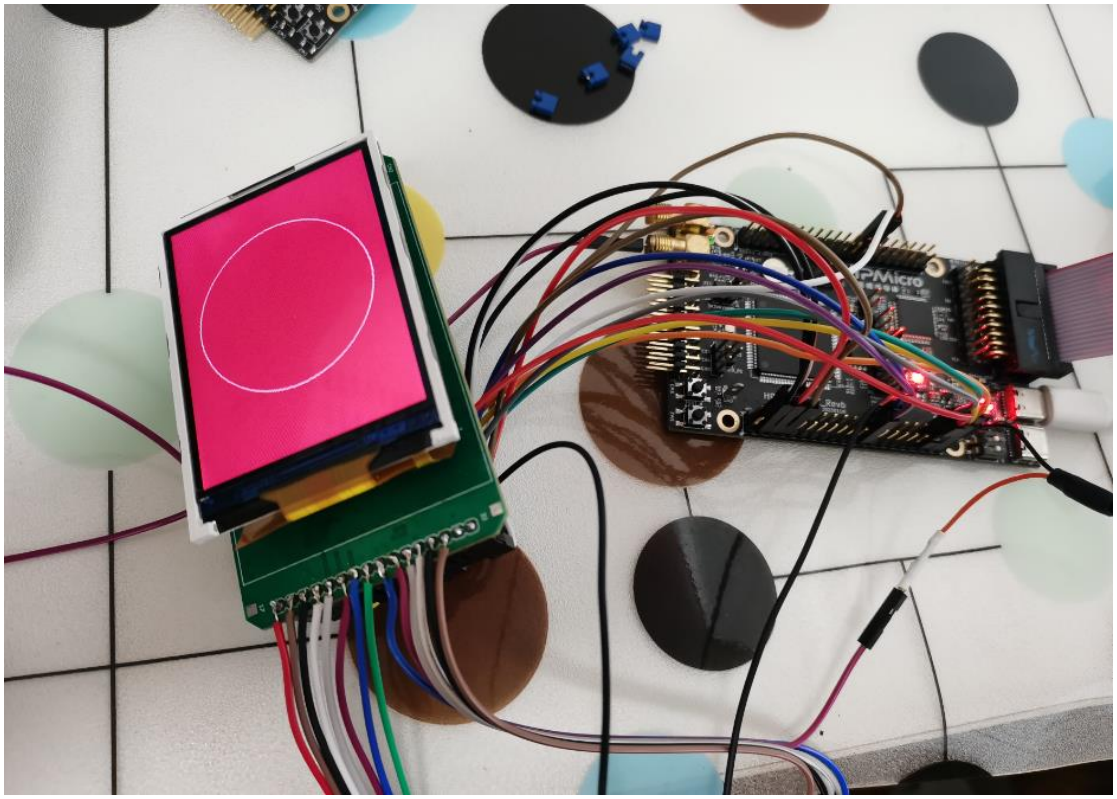

图 11 HPM6300evk 接 8080 屏实物图 (注：hpm6200evk 上贴 HPM6300 系列 MCU)

5.总结

本文主要介绍了 HPM6000 系列高性能微控制器上多功能存储器 FEMC 功能和特性，以及使用 FEMC 控制器驱动 8080 屏的方法。

当 FEMC 挂载 FPGA 或 8080 屏等兼容 SRAM 接口的器件时，可像访问 RAM 一样，通过读写指定地址即可访问操作此器件。

需要强调的是，CPU 访问 FEMC 挂载的外设时其 cache 是有效的。因此，对 FEMC 挂载外设的映射地址需设置为 nocahce 区或关闭 dcache。否则会因为 cache 影响导致实际外设并没有操作。