EIE3810 Microprocessor System Design Laboratory

Lab 3. Flexible Static Memory Controller (FSMC)

School of Science and Engineering

The Chinese University of Hong Kong, Shenzhen

2023-2024 Term 1

1. Objectives

In Lab 3, we will spend the 2-week session to study the following:

- To study the interfacing of Flexible Static Memory Controller (FSMC) of Cortex-M3
- To interface a TFT-LCD with FSMC
- To study the control of TFT-LCD
- To study drawing lines/rectangles, and displaying seven-segment/alphabet on TFT-LCD

2. Basics

In a traditional embedded system built with a microprocessor, all peripherals and memory devices are interfaced with external system address, data, and control buses. They use many physical pins. Reducing the number of physical pins can reduce the cost of the microprocessor. A new generation of microcontrollers embed memories and peripherals, so buses may not be connected to physical pins. Address and data buses are just for internal connection inside the processor. If you want to interface the processor with external devices, a full set of address, data, and control buses will be employed again, and it will occupy many physical pins. Cortex-M3 has many input/output pins. All of them can be configured as a GPIO or Alternate Function (AF). AF can be used for address, data, and control buses, interrupt, timer, analog-to-digital inputs, and pre-defined communication protocol interfaces, etc.

In this lab, we will learn to use some I/Os as address, control, and data buses for external memory access outside MCU.

There are two kinds of Random Access Memories (RAM) for embedded systems, i.e., static (SRAM) and dynamic (DRAM). Dynamic memory is low-cost, but its data has to be refreshed periodically with a memory controller. The circuit for the DRAM controller is more complicated than SRAM's.

Cortex-M3's FSMC stands for Flexible Static Memory Controller. Flexible means that many parameters can be configured dynamically to fit different types of devices and speeds. When FSMC is employed, only a few I/O pins have to be set as memory interface; most of the pins can be kept as physical pins for I/O or other applications.

- **Memory address**

Cortex-M3 reserves 1GB of memory space for FSMC memory. The address is from 0x6000 0000 to 0x9FFF FFFF. There are 2 addresses to clarify: FSMC address and HADDR.

- *FSMC address*

FSMC address is from 0x6000 0000 to 0x9FFF FFFF. They are divided into 4 banks as shown below:

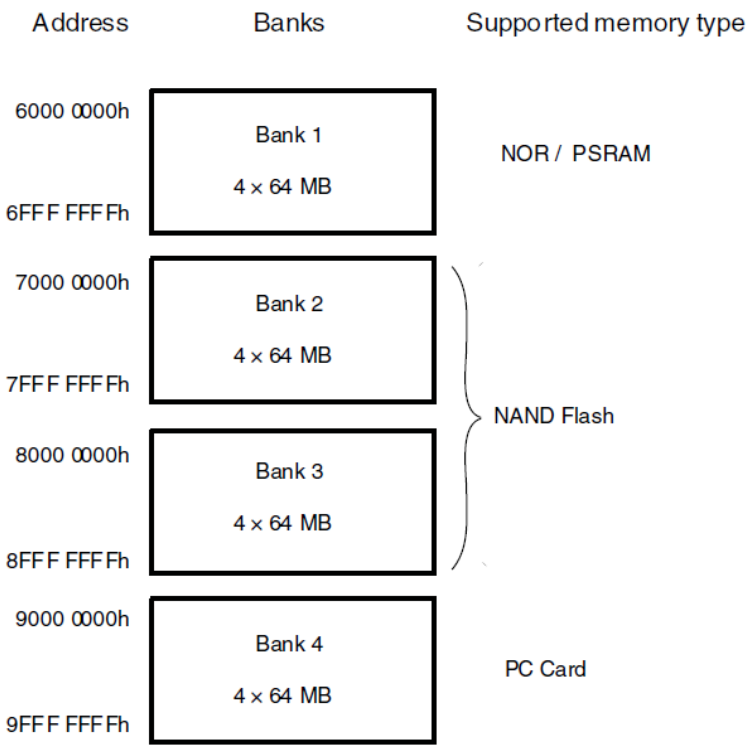


Fig. 1. FSMC memory bank

- Bank 1 used to address up to 4 NOR Flash or PSRAM (Pseudo Static RAM) memory devices. This bank is split into 4 NOR/PSRAM subbanks with 4 dedicated Chip Selects, as follows:
  - Bank 1 - NOR/PSRAM 1
  - Bank 1 - NOR/PSRAM 2
  - Bank 1 - NOR/PSRAM 3
  - Bank 1 - NOR/PSRAM 4
- Banks 2 and 3 used to address NAND Flash devices (1 device per bank)
- Bank 4 used to address a PC Card device

In this project board, it utilizes Bank 1 - NOR/PSRAM 4. The FSMC address is from 0x6C00 0000 to 0x6FFF FFFF.

- *HADDR*

HADDR are internal AHB address lines (28 bits), which are translated to external memory. To select Bank 1 - NOR/PSRAM 4, HADDR[27:26] should be 0b11. Table 1 shows the mapping. When HADDR[27:26]=0b11, the pin FSMC_NE4 will be "1". (In Fig. 2, NEx is low (x=4) for chip select, when FSMC_NE4=1.)

Table 1. NOR/PSRAM bank selection

HADDR[27:26](1)	Selected bank
00	Bank 1 - NOR/PSRAM 1
01	Bank 1 - NOR/PSRAM 2
10	Bank 1 - NOR/PSRAM 3
11	Bank 1 - NOR/PSRAM 4

The external memory width (e.g., the memory in the LCD module) is either 8-bit or 16-bit. However, FSMC can generally work with 8-bit memory cells with one address. For example, the address 0x6000 0000 points to an 8-bit memory cell by FSMC.

However, the external memory width of our TFT-LCD module is 16 bits (i.e., the data line of LCD is 16 bits). To deal with this inconsistency, based on *RM0008*, we can see that the 25-bit long HADDR[25:1] corresponds to FSMC address FSMC_A[24:0].

Table 2. External memory address

Memory width(1)	Data address issued to the memory	Maximum memory capacity (bits)
8-bit	HADDR[25:0]	64 Mbyte x 8 = 512 Mbit
16-bit	HADDR[25:1] >> 1	64 Mbyte/2 x 16 = 512 Mbit

1. In case of a 16-bit external memory width, the FSMC will internally use HADDR[25:1] to generate the address for external memory FSMC_A[24:0].
Whatever the external memory width (16-bit or 8-bit), FSMC_A[0] should be connected to external memory address A[0].

- **Read and write bus transactions**

In this lab, we will use a 4.3-inch, 800 (vertical) x 480 (horizontal) resolution TFT-LCD (thin-file-transistor liquid-crystal display) with full-color mode and touch screen function connected to the project board.

FSMC Bank-1 NOR/PSRAM 4 is connected to read from/write to the memory in the LCD module.

There are different modes for FSMC. For the TFT-LCD module, writing is faster than reading. So we will set FSMC into mode A, which can have different reading and writing DATAST delay time to wait for the data to be stable. Fig. 2 and 3 illustrate the read and write bus transactions

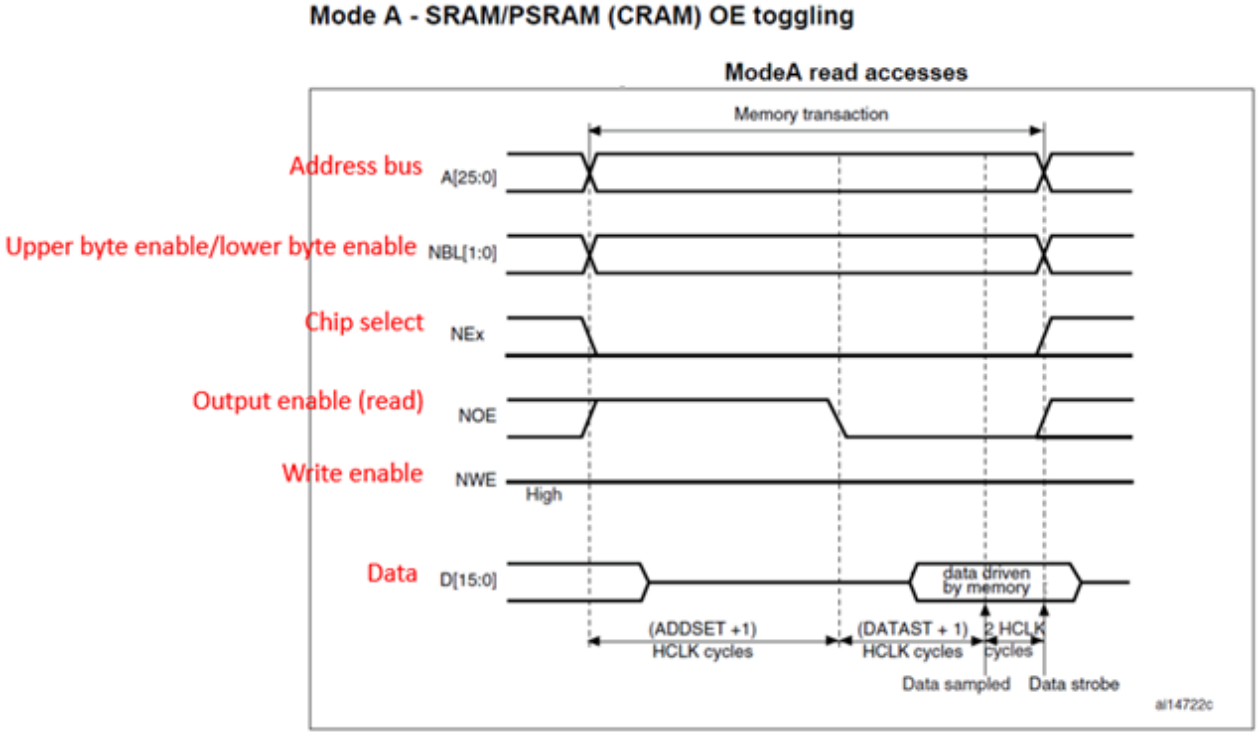


Fig. 2 Read bus

In this lab, we will not work on how to read from memory. In general, for the reading process, set ADDSET=0 and DATAST=15. ADDSET+1 is the number of HCLK cycles that signal on address bus should be stable, while DATAST+1 is the number of HCLK cycles for signals on the data bus.

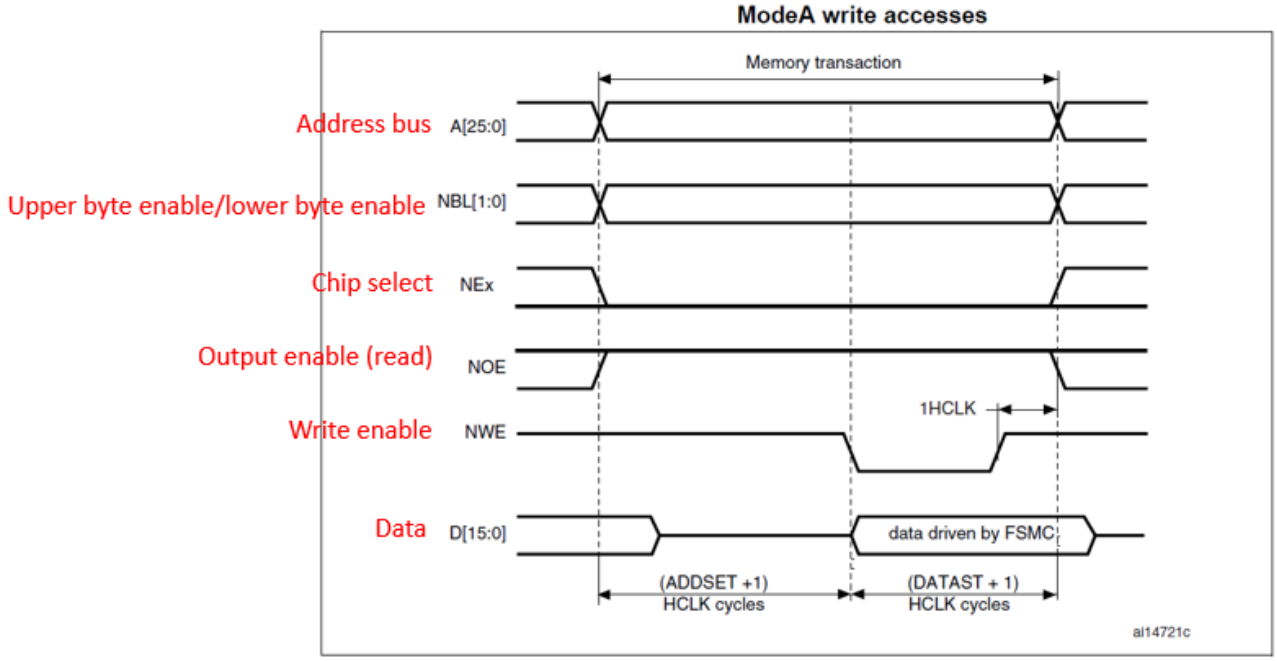


Fig. 3 Write bus

In this lab, we will focus on the writing process. ADDSET=0 and DATAST=3. The TFT-LCD module can perform faster in writing than reading.

- **Pin arrangement**

The LCD module has 34 pins. Within them, 20 pins are for FSMC, 1 for LCD backlight, and the rest will be omitted.

Table 3 illustrated the pin connection. The majority of the STM32 pins used will be configured into the alternative function. You have to turn on the backlight as what we did for Lab 1 GPIO pin. The reset pin of TFT-LCD is connected to the board's hardware reset, and it is not controlled by the microcontroller.

Table 3. Connections between TFT-LCD and STM32

TFT-LCD pin	TFT-LCD function	STM32 pin AF	STM32 pin
LCD_BL	Backlight	PB0	PB0
LCD_CS	LCD chip select	FSMC_NE4	PG12
LCD_RS	0=command / 1=data selection	FSMC_A10	PG0
LCD_WR	LCD write	FSMC_NWE	PD5
LCD_RD	LCD read	FSMC_NOE	PD4
LCD_D[15:0]	LCD 16-bit data bus	FSMC_D15~FSMC_D0	PD14, PD15, PD0, PD1, PE7 ~ PE15, PD8 ~ PD10
LCD_RST	LCD reset	* Board hardware reset	

The LCD pin LCD_RS specifies whether the signal on the data bus is command or data. It is connected to PG0, which is set as the alternative function FSMC_A10. This is the bit 10 of the address bus FSMC_A. From the previous information that HADDR[25:1] maps to FSMC_A[24:0], FSMC_A10 corresponds to HADDR[11].

As HADDR[11]=0 corresponds to command transmission, and HADDR[11]=1 is for data transmission, the address 0x6C00 0000 ~ 0x6C00 07FE is for a 16-bit command, and 0x6C00 0800 ~ 0x6C00 0FFE is for a 16-bit data.

- **How to access memory**

The memory cell in the TFT-LCD module for control can be accessed by its address. The next question is how to access that memory cell.

You can first define two macros, e.g., LCD_COMMAND and LCD_DATA, which are the addresses of the FSMC memory for command and data, respectively. Then write values of command and data to them. Fig. 5 illustrates some samples.

– EIE3810_TFTLCD_WrCmdData(u16 cmd, u16 data): write a 16-bit command and then a 16-bit data to the TFT-LCD
– EIE3810_TFTLCD_WrCmd(u16 cmd): write a 16-bit command to the TFT-LCD
– EIE3810_TFTLCD_WrData(u16 data) : write a 16-bit data to the TFT-LCD

```
#define LCD_COMMAND     ((u32) 0x6C000000)
#define LCD_DATA       ((u32) 0x6C000800)
```

Fig. 4 Macros for address of command and data

```
void EIE3810_TFTLCD_WrCmdData(u16 cmd,u16 data)
{
  * (u16 *) LCD_COMMAND = cmd;
  * (u16 *) LCD_DATA = data;
}

void EIE3810_TFTLCD_WrCmd(u16 cmd)
{
  * (u16 *) LCD_COMMAND = cmd;
}

void EIE3810_TFTLCD_WrData(u16 data)
{
  * (u16 *) LCD_DATA = data;
}
```

Fig. 5 Coding for memory access

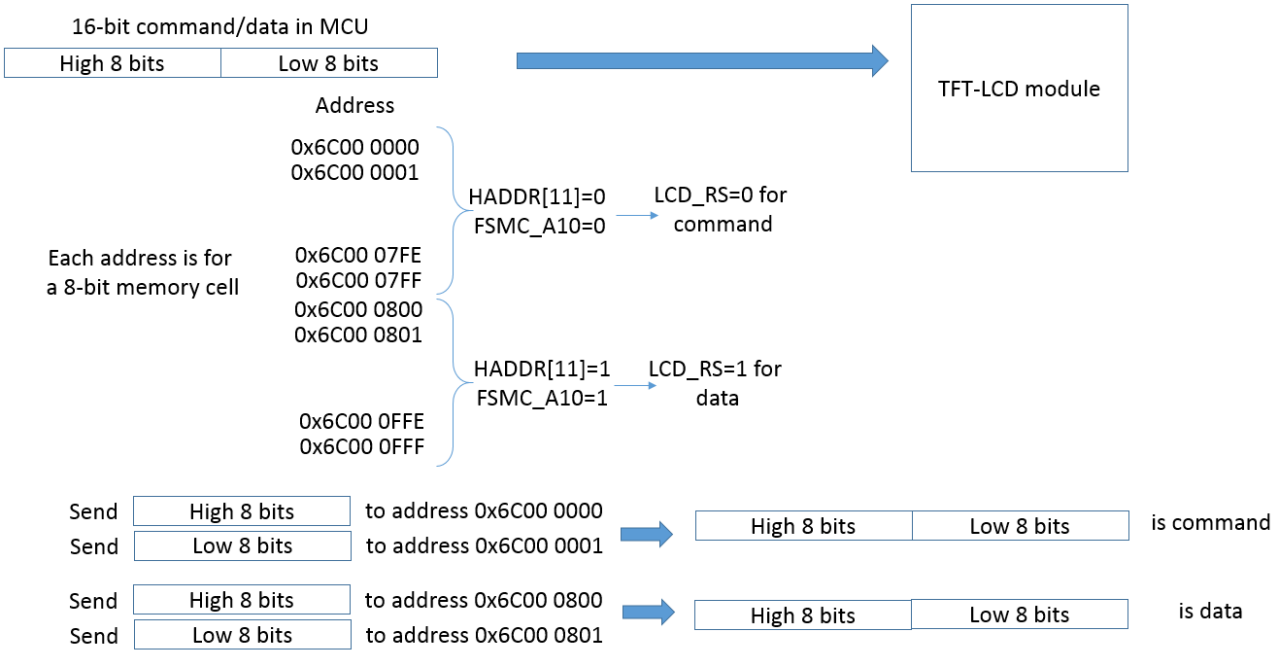


Fig. 6 Details to send a 16-bit command/data from MCU to TFT-LCD module by memory addressing

When STM32 visits the memory at the address 0x6C00 0000, FSMC_A10 will be set to “0” by FSMC as that bit in the address bus is “0”, and thus LCD will treat the value transmitted on the data as a command. When the memory cell at the address 0x6C00 0800 is accessed, FSMC_A10 will be “1”, and LCD will deem that the value in the data bus as data.

Each command/data here should be sent to some register inside the TFT-LCD module. We will explain this in the TFT-LCD command part.

- **TFT-LCD initialization**

Like using other peripherals, you have to create a function EIE3810_TFTLCD_Init() for TFT_LCD initialization first.

FSMC has a few access modes for different memory with different characteristics. The TFT-LCD uses mode-A (*RM0008*, page 519). Reading data from TFT-LCD is slower than writing data to it. Mode-A allows different timing for reading and writing processes. In the initialization, we will set

– FSMC_BCRx: SRAM/NOR-Flash chip-select control registers (x=1, 2, 3, 4)
– FSMC_BTRx: SRAM/NOR-Flash chip-select timing register (x=1, 2, 3, 4)
– FSMC_BWTRx: SRAM/NOR-Flash write timing registers (x=1, 2, 3, 4)

From the file stm32f10x.h, we have to know that these registers are given some alias names:

– FSMC_Bank1->BTCR[6]= FSMC_BCR4
– FSMC_Bank1->BTCR[7]= FSMC_BTR4
– FSMC_Bank1E->BWTR[6]= FSMC_BWTR4

To understand the setting of these registers, refer to *RM0008*, pages 540-547. The details have been marked in Figure 7. The source code is also provided in EIE3810_TFTLCD.c. Add comments to the source code after the lab, and upload it to Blackboard.

```
void EIE3810_TFTLCD_Init(void)
{
  RCC->AHBENR|=1<<8;          //Enable FSMC clock
  RCC->APB2ENR|=1<<3;         //Enable PortB clock
  RCC->APB2ENR|=1<<5;         //Enable PortD clock
  RCC->APB2ENR|=1<<6;         //Enable PortE clock
  RCC->APB2ENR|=1<<8;         //Enable PortG clock

  GPIOB->CRL&=0XFFFFFFF0;     //PB0 push-pull output for backlight
  GPIOB->CRL|=0X00000003;
  //PORTD
  GPIOD->CRH&=0X00FFF000;
  GPIOD->CRH|=0XBB000BBB;
  GPIOD->CRL&=0XFF00FF00;
  GPIOD->CRL|=0X00BB00BB;
  //PORTE
  GPIOE->CRH&=0X00000000;
  GPIOE->CRH|=0XBBBBBBBB;
  GPIOE->CRL&=0X0FFFFFFF;
  GPIOE->CRL|=0XB0000000;
  //PORTG12
  GPIOG->CRH&=0XFFF0FFFF;
  GPIOG->CRH|=0X000B0000;
  GPIOG->CRL&=0XFFFFFFF0;//PG0->RS
  GPIOG->CRL|=0X0000000B;

  //Set all registers to 0
  FSMC_Bank1->BTCR[6]=0X00000000;
  FSMC_Bank1->BTCR[7]=0X00000000;
  FSMC_Bank1E->BWTR[6]=0X00000000;

  FSMC_Bank1->BTCR[6]|=1<<12;   //Add comments
  FSMC_Bank1->BTCR[6]|=1<<14;   //Add comments
  FSMC_Bank1->BTCR[6]|=1<<4;    //Add comments
  FSMC_Bank1->BTCR[7]|=0<<28;   //Add comments
  FSMC_Bank1->BTCR[7]|=1<<0;    //Add comments
  FSMC_Bank1->BTCR[7]|=0XF<<8;    //Add comments
  FSMC_Bank1E->BWTR[6]|=0<<28;  //Add comments
  FSMC_Bank1E->BWTR[6]|=0<<0;   //Add comments
  FSMC_Bank1E->BWTR[6]|=3<<8;   //Add comments
  FSMC_Bank1->BTCR[6]|=1<<0;    //Add comments

  Delay(1000000);
  EIE3810_TFTLCD_SetParameter();
  LCD_LIGHT_ON;
}
```

Fig. 7 EIE3810_TFTLCD_Init()

EIE3810_TFTLCD_Init() calls a subroutine EIE3810_TFTLCD_SetParameter(). It has lots of steps for setting up the TFT_LCD. We have provided the coding in EIE3810_TFTLCD.c. For details of the setup process, reference can be made to NT35510_AN_for_CMI_4p02_IPS.pdf.

After setting up TFT-LCD, you can utilize the three functions in Fig.5 to develop your coding.

- **TFT-LCD command**

  There are a large number of commands for this LCD with NT35510 as the driver chip. You can take reference to its datasheet (NT35510.pdf). Each command is represented by a 16-bit number. The commands used in this lab include: 0x2A00~0x2A03, 0x2B00~0x2B03, 0x2C00, and 0x3600. Each command in this lab has some parameter(s). The rules to use each command are explained as below:

  – 0x2A00~0x2A03 (CASET: Column Address Set)

  This command is used to define the LCD columns (in pixels) that you are going to draw in. The format is: write 0x2A0X as a command and then write the corresponding parameter as data. The pixel number for each row is 480, so XS and XE should be within [0, 479].

CASET: Column Address Set (2A00h~2A03h)

Inst / Para	R/W	Address		Parameter								
		MIPI	Others	D[15:8] (Non-MIPI)	D7	D6	D5	D4	D3	D2	D1	D0
CASET	Write	2Ah	2A00h	00h	XS15	XS14	XS13	XS12	XS11	XS10	XS9	XS8
			2A01h	00h	XS7	XS6	XS5	XS4	XS3	XS2	XS1	XS0
			2A02h	00h	XE15	XE14	XE13	XE12	XE11	XE10	XE9	XE8
			2A03h	00h	XE7	XE6	XE5	XE4	XE3	XE2	XE1	XE0

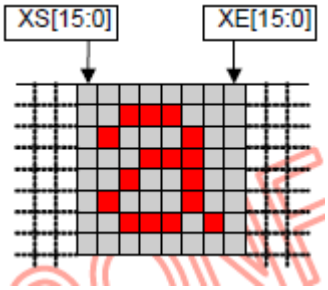


Fig. 8 CASET

– 0x2B00~0x2B03 (RASET: Row Address Set)

The pixel number for each column is 800, so YS and YE should be within [0, 799].

RASET: Row Address Set (2B00h~2B03h)

Inst / Para	R/W	Address		Parameter								
		MIPI	Others	D[15:8] (Non-MIPI)	D7	D6	D5	D4	D3	D2	D1	D0
RASET	Write	2Bh	2B00h	00h	YS15	YS14	YS13	YS12	YS11	YS10	YS9	YS8
			2B01h	00h	YS7	YS6	YS5	YS4	YS3	YS2	YS1	YS0
			2B02h	00h	YE15	YE14	YE13	YE12	YE11	YE10	YE9	YE8
			2B03h	00h	YE7	YE6	YE5	YE4	YE3	YE2	YE1	YE0

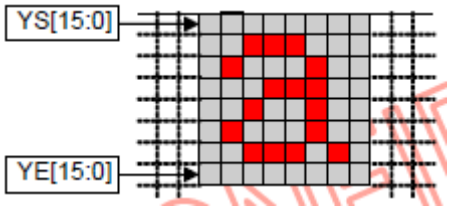


Fig. 9 RASET

– 0x2C00 (RAMWR: Memory Write)

RAMWR: Memory Write (2C00h)

Inst / Para	R/W	Address		Parameter							
		MIPI	Others	D[15:8] (Non-MIPI)	D7	D6	D5	D4	D3	D2	D1	D0
RAMWR	Write	2Ch	2C00h	D[15:8]	D7	D6	D5	D4	D3	D2	D1	D0
				D[15:8]	:	:	:	:	:	:	:	:
				D[15:8]	D7	D6	D5	D4	D3	D2	D1	D0

Fig. 10 RAMWR

The 16-bit color data adopts RGB 5-6-5 format, i.e. 5 bits for red, 6 bits for green, and 5 bits for blue. Data=0b1111 1000 0000 0000 for pure red; 0b0000 0111 1110 0000 for pure green; 0b0000 0000 0001 1111 for pure blue; 0b0000 0000 0000 0000 for black; and 0b1111 1111 1111 1111 for white.

After 0x2C00 as a command, you can repeatedly write 16-bit data, and the pixel will be automatically filled with colored based on memory access control, which is previously set by the command 0x3600.

– 0x3600 (MADCTL: Memory Data Access Control)

Inst / Para	R/W	Address		Parameter								
		MIPI	Others	D[15:8] (Non-MIPI)	D7	D6	D5	D4	D3	D2	D1	D0
MADCTL	Write	36h	3600h	00h	MY	MX	MV	ML	RGB	MH	RSMX	RSMY

Fig. 11 MADCTL

(MY, MX, MV) = (0, 0, 0): from left to right, from up to down
(MY, MX, MV) = (1, 0, 0): from left to right, from down to up
(MY, MX, MV) = (0, 1, 0): from right to left, from up to down
(MY, MX, MV) = (1, 1, 0): from right to left, from down to up
(MY, MX, MV) = (0, 0, 1): from up to down, from left to right
(MY, MX, MV) = (0, 1, 1): from up to down, from right to left
(MY, MX, MV) = (1, 0, 1): from down to up, from left to right
(MY, MX, MV) = (1, 1, 1): from down to up, from right to left

Bit	NAME	DESCRIPTION
MY	Row Address Order	These 3 bits controls interface to memory write/read direction. The behavior on display after pattern changed.
MX	Column Address Order	
MV	Row/Column Exchange	
ML	Vertical Refresh Order	TFT LCD Vertical refresh direction control. Immediately behavior on display.
RGB	RGB-BGR Order	Color selector switch control "0" = RGB color sequence, "1" = BGR color sequence Immediately behavior on display
MH	Horizontal Refresh Order	TFT LCD Horizontal refresh direction control Immediately behavior on display.
RSMX	Flip Horizontal	Flips the display image left to right. Immediately behavior on display.
RSMY	Flip Vertical	Flips the display image top to down. Immediately behavior on display.

Fig. 12 Bit description

- **A simple example: draw a dot**

As a simple example, draw a dot is not difficult to understand. It firstly sets column address to x, then row address to y, and finally sets the color to that pixel to the 16 bit "color". We have provided the main.c, EIE3810_TFTLCD.c, and EIE3810_TFTLCD.h. Try it out first.

```
void EIE3810_TFTLCD_DrawDot(u16 x, u16 y, u16 color)
{
  EIE3810_TFTLCD_WrCmd(0x2A00);
  EIE3810_TFTLCD_WrData(x>>8);
  EIE3810_TFTLCD_WrCmd(0x2A01);
  EIE3810_TFTLCD_WrData(x & 0xFF);

  EIE3810_TFTLCD_WrCmd(0x2B00);
  EIE3810_TFTLCD_WrData(y>>8);
  EIE3810_TFTLCD_WrCmd(0x2B01);
  EIE3810_TFTLCD_WrData(y & 0xFF);

  EIE3810_TFTLCD_WrCmd(0x2C00);
  EIE3810_TFTLCD_WrData(color);
}
```

Fig. 13 Draw dot function

3. Experiments

3.1 Experiment 1: Draw lines on the TFT-LCD with different colors

In this experiment, we will draw some lines on the screen with different colors.

Procedures:

1) Use 'for' loops to draw 5 lines with 20 dots each and the colors are black, white, green, red and blue color, with the lines starting at x=10, y=10, 20, 30, 40, and 50, respectively.
2) On your LCD, you may be able to see 5 short color lines. But the rest of the LCD pixels (i.e. background pixels) have random colors. Write the program to make the background pixels white. Hint: it is not wise enough to call EIE3810_TFTLCD_DrawDot(u16 x, u16 y, u16 color) for every pixel, as it writes a large number of commands. Try to write an efficient function to set all the pixel color to white.

[Demonstration] When you have completed 2), demonstrate to the instructor, TA, or technician. How many lines can you see? Why?

[In Report] Include your test result.

[Question] (1) After Experiment 1, you shall know where the origin (0, 0) is located on the screen. Where is it? Write the answer on your report. (2) In the defined macros (Fig. 4), what other values can be? If we set LCD_COMMAND as 0x6C0007FF, is it correct? Why or why not?

3.2 Experiment 2: Draw rectangle on the TFT-LCD

Write a subroutine to fill a rectangle with one color. One sample coding is given below.

```
void EIE3810_TFTLCD_FillRectangle(u16 start_x, u16 length_x,
                                  u16 start_y, u16 length_y, u16 color)
{
  u32 index =0;
  EIE3810_TFTLCD_WrCmd(0x2A00);
  EIE3810_TFTLCD_WrData(start_x >> 8);
  EIE3810_TFTLCD_WrCmd(0x2A01);
  EIE3810_TFTLCD_WrData(start_x & 0xFF);
  EIE3810_TFTLCD_WrCmd(0x2A02);
  EIE3810_TFTLCD_WrData((length_x + start_x - 1) >> 8);
  EIE3810_TFTLCD_WrCmd(0x2A03);
  EIE3810_TFTLCD_WrData((length_x + start_x - 1) & 0xFF);

  EIE3810_TFTLCD_WrCmd(0x2B00);
  EIE3810_TFTLCD_WrData(start_y >> 8);
  EIE3810_TFTLCD_WrCmd(0x2B01);
  EIE3810_TFTLCD_WrData(start_y & 0xFF);
  EIE3810_TFTLCD_WrCmd(0x2B02);
  EIE3810_TFTLCD_WrData((length_y + start_y - 1)>> 8);
  EIE3810_TFTLCD_WrCmd(0x2B03);
  EIE3810_TFTLCD_WrData((length_y + start_y - 1) & 0xFF);

  EIE3810_TFTLCD_WrCmd(0x2C00);
  for(index=0;index < length_x*length_y; index++)
  {
    EIE3810_TFTLCD_WrData(color);
  }
}
```

Fig. 14 Fill rectangle

Procedures:

1) Key in the subroutine for drawing a rectangle with input parameters, as shown in Fig. 14.
2) Use your subroutine to draw a rectangle at x=100, y=100, length, and width are 100 with yellow color on the screen.

[Demonstration] Demonstrate to instructor, TA, or technician that your program works.

[In Report] Include your test result.

3.3 Experiment 3: Draw a counting down digit by seven-segment on the TFT-LCD screen

You can use seven rectangles (as the seven-segment display) to form a digit from 0 to 9. The dimension of each segment is shown in Fig. 15.

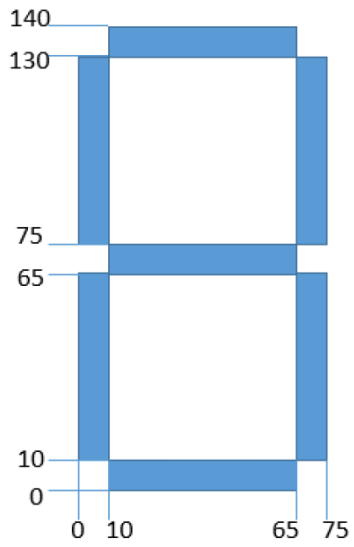


Fig. 15 Dimension of the seven-segment

Procedures:

1) Create a subroutine "EIE3810_TFTLCD_SevenSegment(u16 start_x, u16 start_y, u8 digit, u16 color)". In this subroutine, the parameter **color** is the color value of the segments. Parameters **start_x** and **start_y** are the starting position of the number (lower left corner). The **digit** is the number represented by the seven-segment.
2) Use your rectangle subroutine to draw digit "8" (Seven-segment format) at the center of the screen with your favorite color.
3) Write a program to display count-down digit from 9 to 0, periodically. After it comes to 0, go back to 9. The period of each count is around 1 second, which can be implemented with Delay().

[Demonstration] Demonstrate to instructor, TA, or technician that your program works.

[In Report] Include a flowchart of your coding and the test result.

3.4 Experiment 4: Show alphabets on the TFT-LCD

We can use the screen to show characters, which is a pattern or a group of pixels on the screen. The typical pattern size can be 6x12 or 8x16. The alphabets are grouped in a table with pre-defined order (0~127) named ASCII table (Fig. 16).

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Fig. 16 ASCII table

We provide Font.H, which contains a set of 8x16 size ASCII characters. The first 32 characters and the last character in this table are unprintable. So, the printable character sets start from the 32nd ASCII character, i.e., “space”. Thus, the table contains 95 character patterns only.

```
const unsigned char asc2_1608[95][16]={
{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},/*" ",0*/
{0x00,0x00,0x00,0x00,0x00,0x00,0x1F,0xCC,0x00,0x0C,0x00,0x00,0x00,0x00,0x00,0x00},/*"!",1*/
{0x00,0x00,0x08,0x00,0x30,0x00,0x60,0x00,0x08,0x00,0x30,0x00,0x60,0x00,0x00,0x00},/*"""",2*/
{0x02,0x20,0x03,0xFC,0x1E,0x20,0x02,0x20,0x03,0xFC,0x1E,0x20,0x02,0x20,0x00,0x00},/*"#",3*/
{0x00,0x00,0x0E,0x18,0x11,0x04,0x3F,0xFF,0x10,0x84,0x0C,0x78,0x00,0x00,0x00,0x00},/*"$",4*/
{0x0F,0x00,0x10,0x84,0x0F,0x38,0x00,0xC0,0x07,0x78,0x18,0x84,0x00,0x78,0x00,0x00},/*"%",5*/
{0x00,0x78,0x0F,0x84,0x10,0xC4,0x11,0x24,0x0E,0x98,0x00,0xE4,0x00,0x84,0x00,0x08},/*"&",6*/
{0x08,0x00,0x68,0x00,0x70,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},/*"'",7*/
{0x00,0x00,0x00,0x00,0x00,0x00,0x07,0xE0,0x18,0x18,0x20,0x04,0x40,0x02,0x00,0x00},/*"(",8*/
{0x00,0x00,0x40,0x02,0x20,0x04,0x18,0x18,0x07,0xE0,0x00,0x00,0x00,0x00,0x00,0x00},/*")",9*/
{0x02,0x40,0x02,0x40,0x01,0x80,0x0F,0xF0,0x01,0x80,0x02,0x40,0x02,0x40,0x00,0x00},/*"*",10*/
{0x00,0x80,0x00,0x80,0x00,0x80,0x0F,0xF8,0x00,0x80,0x00,0x80,0x00,0x80,0x00,0x00},/*"+",11*/
{0x00,0x01,0x00,0x0D,0x00,0x0E,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},/*",",12*/
{0x00,0x00,0x00,0x80,0x00,0x80,0x00,0x80,0x00,0x80,0x00,0x80,0x00,0x80,0x00,0x80},/*"-",13*/
{0x00,0x00,0x00,0x0C,0x00,0x0C,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},/*".",14*/
{0x00,0x00,0x00,0x06,0x00,0x18,0x00,0x60,0x01,0x80,0x06,0x00,0x18,0x00,0x20,0x00},/*"/",15*/
```

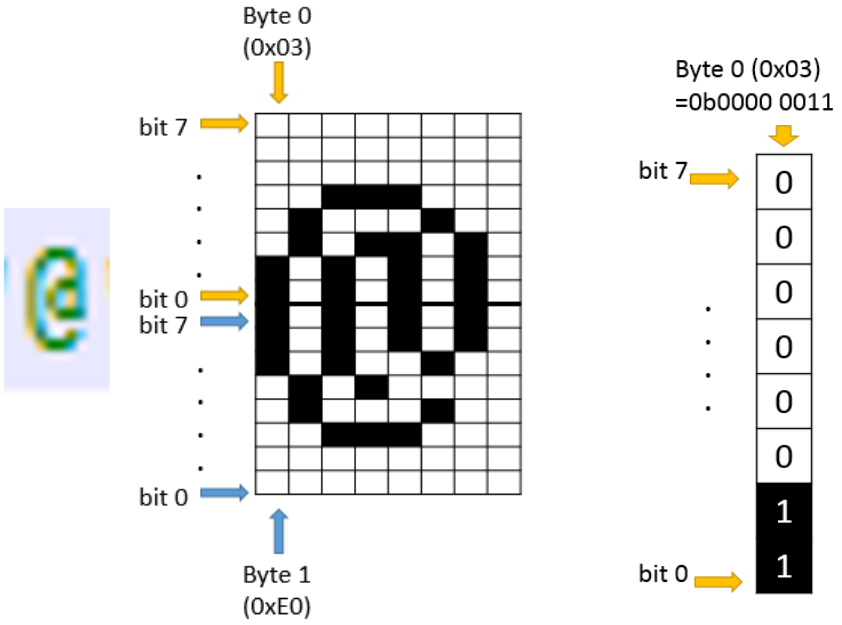
Fig. 17 Part of Font.H

Let's check one character as an example.

"@" is represented by

{0x03,0xE0,0x0C,0x18,0x13,0xE4,0x14,0x24,0x17,0xC4,0x08,0x28,0x07,0xD0,0x00,0x00}
/*"@",32*/

For byte 0 (asc2_1608[32][0]), Fig 18(b) shows the sequence of bits.



(a) "@" (b) byte 0 of "@"

Fig. 18 Arrangement of pixels of "@"

Procedures:

1) Download Font.H and add it into your "board" folder. Include it in the EIE3810_TFTLCD.c as below.

```
#include "stm32f10x.h"
#include "EIE3810_TFTLCD.h"
#include "Font.H"
```

Fig. 19 Header

2) Design the subroutine to show the character. The format is:
   void EIE3810_TFTLCD_ShowChar(u16 x, u16 y, u8 ASCII, u16 color, u16 bgcolor)
   x and **y** are the horizontal and vertical positions of the left-upper corner. **ASCII** is the index of **ASCII** code in Fig. 16. **ASCII** should be within [0, 127]. If it is an unprintable character, then do not print anything. **color** is the color of the character, and **bgcolor** is the background color in the other pixels in 8 x 16 rectangle.
3) Write the program that uses the subroutine in 2) to show your CUHKSZ ID on the screen. The background color should be any color other than white.
4) Include experiments 1-4 into the one main(), and show all the 4 experiments simultaneously. To make your coding clear, write 4 subroutines, e.g., exp_1(), exp_2(), exp_3(), and exp_4(), and call them in main().

[Demonstration] Show instructor, TA, or technician when you have completed 4).

[In Report] Include a flowchart of your EIE3810_TFTLCD_ShowChar() and the test result.

[Question] To show all the 4 experiments simultaneously, what consideration in exp1-4 do you need to make, and why?

4. Lab Report and Source Code

Submit the report softcopy and your code (complete project folder of each experiment) in zip format to Blackboard by the deadline below:

- Tuesday Class (L02): 15:00, Tuesday, October 31, 2023
- Friday Class (L01): 9:00, Friday, November 3, 2023

Each day of late submission will result in 10% deduction in the report and source code raw marks.