2018《FPGA 应用实验》实验报告

实验编号： 03 实验时间： 2018.04.03

实验名称： LCD 显示字符控制模块设计

班级： F1503003 学号： 515030910067 姓名： 杨超琪

1、实验平台

采用 Xilinx 公司的 FPGA 集成开发环境 Xilinx ISE Design Suite 10.1 sp3，实验开发板为 Xilinx Spartan-3E FPGA Starter Kit。

2、实验设计要求：

功能描述：

设计 LCD 显示字符控制电路模块，使用在 Spartan－3E FPGA Starter Kit Board 上的 2X16 字符型 LCD 显示指定的字符串。

控制电路的功能和工作状态：

（0）LCD 显示两行字符串，其中：

第 1 行显示：Spartan-3E□FPGA

第 2 行显示：FPAG□Starter

这里，□表示空格。

（1）使用 BTN_WEST 做为复位键，当按下 BTN_WEST 时，LCD 复位并刷新重新显示两行字符串。

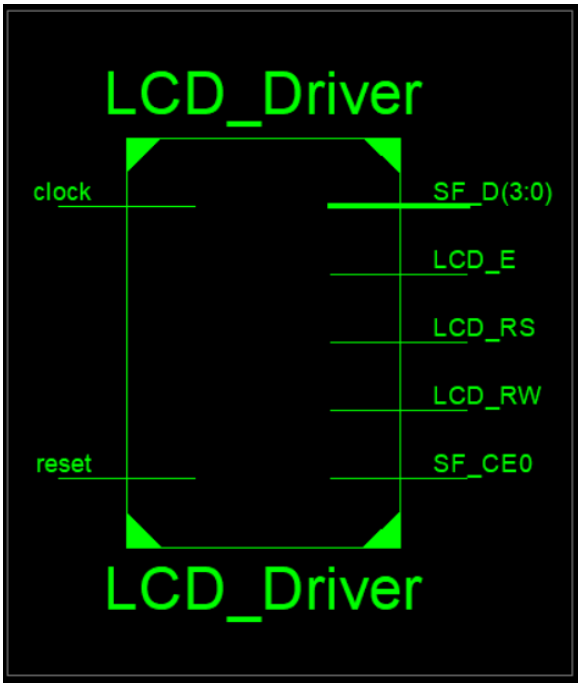
设计中需要解决的问题：

（1）LCD 控制器初始化；

（2）利用 4 位数据接口（4－bit data interface）向 LCD 控制器发送命令/数据。

3、模块设计框图

宏观顶层模块：



4、实验原理：

入门实验板显著的特征是 2 线 16 字符液晶显示器 LCD。尽管 LCD 支持 8 位的数据接口，为了与其它的 XILINX 的开发板保持兼容并且尽可能减少针脚数，FPGA 仅通过 4 位的数据接口线控制 LCD，如图 5.1 所示。

LCD 通过使用 ASCII 标准和自定义字符可以有效地显示多种信息。但是，这些显示速度并不是很快。每半秒扫描一次以测试实际清晰度的界限。与 50MHz 时钟频率相比，这样的显示速度是慢的。PicoBlaze 处理器可以有效地控制显示时间和显示内容。

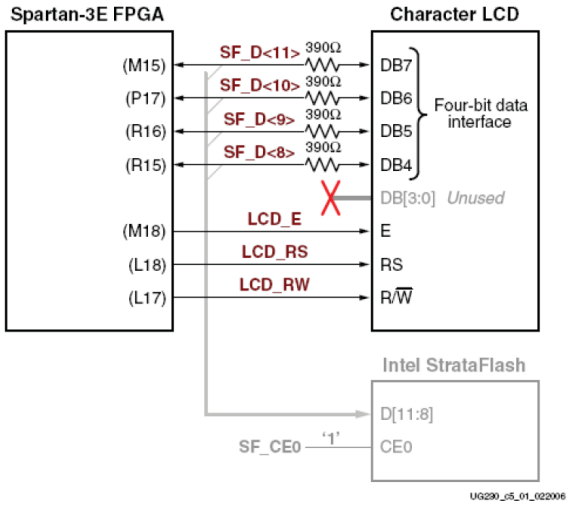


Figure 5-1: **Character LCD Interface**

LCD 通过使用 ASCII 标准和自定义字符可以有效地显示多种信息。但是，这些显示速度并不是很快。每半秒扫描一次以测试实际清晰度的界限。与 50MHz 时钟频率相比，这样

的显示速度是慢的。PicoBlaze 处理器可以有效地控制显示时间和显示内容。

如图 5.1 所示，4 根 LCD 数据线与 StrataFlash 数据线 SF_D<11:8>复用。LCD_E、LCD_RS、LCD_RW 分别表示响应功能的管脚信号。

Table 5-1: **Character LCD Interface**

Signal Name	FPGA Pin	Function	
SF_D<11>	M15	Data bit DB7	Shared with StrataFlash pins SF_D<11:8>
SF_D<10>	P17	Data bit DB6	
SF_D<9>	R16	Data bit DB5	
SF_D<8>	R15	Data bit DB4	
LCD_E	M18	Read/Write Enable Pulse 0: Disabled 1: Read/Write operation enabled	
LCD_RS	L18	Register Select 0: Instruction register during write operations. Busy Flash during read operations 1: Data for read or write operations	
LCD_RW	L17	Read/Write Control 0: WRITE, LCD accepts data 1: READ, LCD presents data	

控制器有三个内部存储空间，每个都有专门用途。送数据给这些空间之前必须初始化。

1）DD RAM

显示数据 RAM（DD RAM）存储字符编码。绝大多数应用中，都是与 DD RAM 相结合的。存储在 DD RAM 中的字符编码所涉及的特定的字符位图要么存在 CG ROM 字符设置中，要么存在用户自定义的 CG RAM 的字符设置中。

图 5.3 给出了显示器 32 位字符位置的默认地址。字符的最上行存储在地址 0X00 与 0X0F 之间。第二行的字符存储在地址 0X40 与 0X4F 之间。

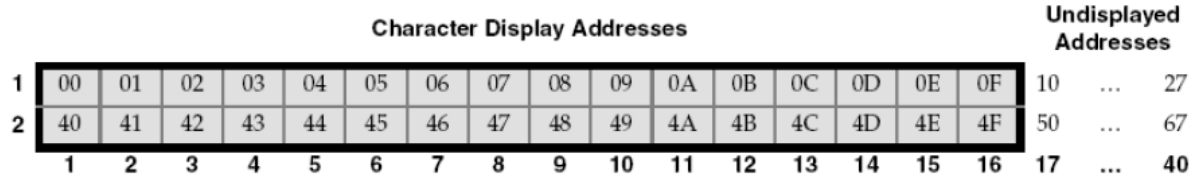


Figure 5-3: **DD RAM Hexadecimal Addresses (No Display Shifting)**

从物理上讲，DD RAM 一共有80 个字符位置，每行有40 个字符。位置0X10 到0X27 和0X50 到0X67 之间的地址用来存储其它非显示数据。此外，这些位置也可以存储只有使用控制器的显示移位功能才能显示的字符。

往 DD RAM 读或写之前，DD RAM 地址命令得初始化地址计数器。写DD RAM 数据使用写数据到CG RAM 或DD RAM 命令，读DD RAM 使用从CG RAM 或DD RAM 命令读数据。DD RAM 地址计数器要么在读或写之后保持常数，要么自动增加1 或自动减1。

2) CG ROM

字符产生器ROM（CG ROM）包括每个事先定好的字符的字体位图，这样LCD 屏才能显示，如图5.4。字符编码存储在DD RAM 中，每个字符的位置与CG ROM 的位置按顺序对应。例如，0X53 的一个16 进制的字符编码存储在DD RAM 中的位置显示字符是“S”。

0X53 最上面的轻咬位（高半位）等同于DB[7:4]=0101 和最低的轻咬位（低半位）等同于DB[3:0]=0011。

如图5.4 所示，字符“S”就显示在屏幕上了。英语/罗马字符存储在CG ROM 相应的ASCII 编码地址中。字符 ROM 存储ASCII 英文字符和日本字符。控制器同样提供了 8 位自定义字符位图，存储在CG RAM 中。这些8 位的自定义字符编码显示时存储在DD RAM 的0X00 与0X07 之间。

Figure 5-4: LCD Character Set

3）CG RAM

字符产生器RAM(CG RAM)提供空间给8 位的自定义字符位图。每个自定义字符位由8 行位图的5 个点组成，如图5.5 所示。

往 CG RAM 读或写之前CG RAM 地址命令得初始化地址计数器。写CG RAM 数据使用写数据到CG RAM 或DD RAM 命令，读CG RAM 使用从CG RAM 或DD RAM 命令读数据。CG RAM 地址计数器要么在读或写之后保持常数，要么自动增加1 或自动减1。

图 5.5 举了个例子，产生一个特殊的西洋跳棋盘字符。自定义字符存储在第四CG RAM 字符位置中，当DD RAM 的位置是0x03 时，其显示。写自定义字符时，使用设置CG RAM 地址命令初始化CG RAM 地址。前三行的地址位对应自定义字符位。后三行位对应字符地址的行地址。写数据到CG RAM 或DD RAM 命令用来写每个字符位行。“1”表示点亮。“0”表示熄灭。只有低5 位的数据被用到。高三位的数据与位置无关。第8 行的数据位一般为0 以适于指针之用。

						Upper Nibble				Lower Nibble			
						Write Data to CG RAM or DD RAM							
A5	A4	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0
Character Address			Row Address			Don't Care			Character Bitmap				
0	1	1	0	0	0	-	-	-	0	■	0	■	0
0	1	1	0	0	1	-	-	-	■	0	■	0	■
0	1	1	0	1	0	-	-	-	0	■	0	■	0
0	1	1	0	1	1	-	-	-	■	0	■	0	■
0	1	1	1	0	0	-	-	-	0	■	0	■	0
0	1	1	1	0	1	-	-	-	■	0	■	0	■
0	1	1	1	1	0	-	-	-	0	■	0	■	0
0	1	1	1	1	1	-	-	-	0	0	0	0	0

Figure 5-5: Example Custom Checkerboard Character with Character Code 0x03

表5.3 简要的说明了LCD 控制器的命令和位的定义。由于该显示屏是4 位操作，每8位命令被送到2 个4 位的轻咬位（2 个半位）。高半位先送，低半位后送。

Table 5-3: **LCD Character Display Command Set**

Function	LCD_RS	LCD_RW	Upper Nibble				Lower Nibble			
			DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Clear Display	0	0	0	0	0	0	0	0	0	1
Return Cursor Home	0	0	0	0	0	0	0	0	1	-
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S
Display On/Off	0	0	0	0	0	0	1	D	C	B
Cursor and Display Shift	0	0	0	0	0	1	S/C	R/L	-	-
Function Set	0	0	0	0	1	0	1	0	-	-
Set CG RAM Address	0	0	0	1	A5	A4	A3	A2	A1	A0
Set DD RAM Address	0	0	1	A6	A5	A4	A3	A2	A1	A0
Read Busy Flag and Address	0	1	BF	A6	A5	A4	A3	A2	A1	A0
Write Data to CG RAM or DD RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0
Read Data from CG RAM or DD RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0

1） 失能

如果 LCD_E 使能信号为低，所有其它输入 LCD 信号全被忽视。

2）清屏

清屏后指针返回到原始位置——最左上角。该命令写一个空白内容（ASCII/ANSI 字符编码为 0x20）给所有的 DD RAM 地址。DD RAM 的 0X00 地址计数器置 0。清除所有的选择设置。I/D 控制位置 1（地址自动增加模式）。执行时间：82us～1.64ms

3）返回指针原始位

指针返回原始位——最左上角。DD RAM 的内容不受影响。所有的显示被移到原始位，如图 5.3 所示。DD RAM 的 0X00 地址计数器置 0。如果移位，显示返回原始状态。指针或光标移到字符位的最左上角。执行时间：40us～1.6ms

4）进入模式设置

设置指针移动的方向，并规定是否移动显示。在读和写数据时，这些操作就完成了。执行时间：40us

Bit DB1: (I/D) Increment/Decrement

0	Auto-decrement address counter. Cursor/blink moves to left.
1	Auto-increment address counter. Cursor/blink moves to right.

在每次写数据给 CG RAM 或 DD RAM 或从 CG RAM 或 DD RAM 读数据后，该位的 DD RAM 和 CG RAM 地址计数器要么自动增加 1 要么自动减少 1.指针或光标的位置随之移动。

Bit DB0: (S) Shift

0	Shifting disabled
1	During a DD RAM write operation, shift the entire display value in the direction controlled by Bit DB1 (I/D). Appears as though the cursor position remains constant and the display moves.

1） 显示关/断

显示关或断，控制所有的字符、指针和指针位置的字符光标。

执行时间：40us

Bit DB2: (D) Display On/Off

0	No characters displayed. However, data stored in DD RAM is retained
1	Display characters stored in DD RAM

指针使用字符最底行的 5 个点。指针出现在显示字符的下面。

Bit DB1: (C) Cursor On/Off

0	No cursor
1	Display cursor

Bit DB0: (B) Cursor Blink On/Off

0	No cursor blinking
1	Cursor blinks on and off approximately every half second

2） 指针和显示移动

移动指针和显示并不改变 DD RAM 的内容。移动指针位置或显示往左或往右时并不需要写或读显示数据。

指针的位置功能是为了修改个别的字符，或向左或右滚动窗口来显示存储在 DD RAM 中的额外数据，可以移到每行的第 16 个符。当它移到第一行的第 40 个字符之处时，指针自动移到第二行。两行的显示移动在同一时间进行。

当显示数据重复移动时，两行水平移动。第二行不会移到第一行。

执行时间：40us

Table 5-4: **Shift Patterns According to S/C and R/L Bits**

DB3 (S/C)	DB2 (R/L)	Operation
0	0	Shift the cursor position to the left. The address counter is decremented by one.
0	1	Shift the cursor position to the right. The address counter is incremented by one.
1	0	Shift the entire display to the left. The cursor follows the display shift. The address counter is unchanged.
1	1	Shift the entire display to the right. The cursor follows the display shift. The address counter is unchanged.

3） 功能设置

设置接口数据的长度，每行显示的个数，字符的字体。入门实验板支持单功能设置，其值为 0X28。

执行时间：40us

7）设置 CG RAM 地址

设置 CG RAM 的初始地址。该命令后，以后所有往显示屏的读或写操作的数据来自或去往 CG RAM。

8）设置 DD RAM 地址

设置 DD RAM 的初始地址。该命令后，以后所有往显示屏的读或写操作的数据来自或去往 DD RAM。

执行时间：40us

9)读忙标志和地址

读忙标志（BF）用来判断内部操作是否在进行，并读当前地址计数器的内容。

BF＝1 说明内部操作在进行。下个指令不被接收直到 BF 被清 0 或直到当前指令达到最大的执行时间。

该命令返回当前地址计数器的值。地址计数器为 CG RAM 和 DD RAM 所用。具体内容取决于最新公布的设置 CG RAM 地址或设置 DD RAM 设置命令。

执行时间：1us

10）写数据给 CG RAM 或 DD RAM

要是该命令在设置 DD RAM 地址命令之后，则写数据给 DD RAM；或是该命令在设置 CGRAM 地址命令之后，则写数据给 CGRAM。根据进入设置模式，在写操作之后，地址自动加 1 或自动减 1。进入设置模式同样决定显示移动。

执行时间：40us

11）从 CG RAM 或 DD RAM 读数据

要是该命令在设置 DD RAM 地址命令之后，则从 DD RAM 读数据；或是该命令在设置 CGRAM 地址命令之后，从 CG RAM 读数据。根据进入设置模式，在写操作之后，地址自动加 1 或自动减 1。进入设置模式同样决定显示移动。执行时间：40us

该板使用 4 位数据接口给字符 LCD。图 5.6 说明了向 LCD 写操作建立、保持允许的最小时间以及使能脉冲对时钟（50MHz 或 T=20ns）的偏移时间。

DF_D<11:8>的数据值，寄存器选择信号（LCD_RS）以及读/写（LCD_RW）控制信号必须建立并在使能信号 LCD_E 转向高电平之前至少稳定 40ns。使能信号必须保留高电平 230ns 或更长时间——等于或超过 12 时钟周期（50MHz）。在许多应用中，LCD_RW 信号可以永远置低，因为 FPGA 一般不会从显示屏读取数据。

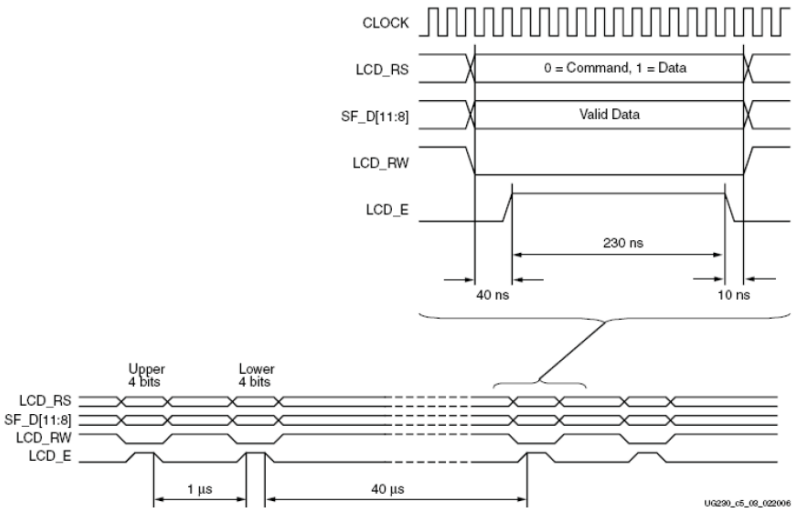


Figure 5-6: **Character LCD Interface Timing**

Transferring 8-Bit Data over the 4-Bit Interface

在初始化显示屏和建立通信之后，所有的命令和数据以 8 位形式传送给字符显示屏——用 2 个连续的 4 位传送。每个 8 位传送必须分为 2 个 4 位，其间隔时间至少 1us，如图 5.6 所示。先传高半位，再传低半位。一个 8 位的写操作在下个通信之前必须间隔至少 40us。在清屏命令之后，该延时必须增至 1.64ms。

Initializing the Display

上电后，显示屏必须初始化建立所需的通信协议。该初始化步骤简单，完全适合高效的 8 位 PicoBlaze 嵌入式控制器。初始化之后，PicoBlaze 控制器除了简单驱动显示屏外，还用来更为复杂的控制或计算。

1） 上电初始化

初始化的第一步骤是建立 FPGA 与 LCD 的 4 位的数据接口，具体如下：

A：等待 15ms 或更长，尽管 FPGA 完成配置后显示屏一般处于准备就绪状态。在 50MHz 时，15ms 时间等于 750000 时钟周期。

B：写 SF_D<11:8>=0x3，LCD_E 保持高电平 12 时钟周期。

C：等待 4.1ms 或更长，即在 50MHz 时，205000 时钟周期。

D：写 SF_D<11:8>=0x3，LCD_E 保持高电平 12 时钟周期。

E：等待 100us 或更长，即在 50MHz 时，5000 时钟周期。

F：写 SF_D<11:8>=0x3，LCD_E 保持高电平 12 时钟周期。

G：等待 40us 或更长，即在 50MHz 时，2000 时钟周期。

H：写 SF_D<11:8>=0x2，LCD_E 保持高电平 12 时钟周期。

I：等待 40us 或更长，即在 50MHz 时，2000 时钟周期。

2） 显示屏配置

上电初始化完成后，4 位的数据接口就建立了。下一步就是配置显示屏了：

A：发一个功能设置命令，0x28，配置显示屏。

B：发一个进入模式命令，0X06，设置显示屏自动增地址指针。

C：发一个显示开/断命令，0x0c，开显示屏并失能指针和光标。

D：最后，发清屏命令，此后等待至少 1.64ms（82000 时钟周期）。

Writing Data to the Display

写数据给显示屏，指定初始地址，紧接着是一个或多个数据值。写任何数据之前，发送一个设置 DD RAM 地址命令给 DD RAM 中指定的初始 7 位地址。见图 5.3 所示。

使用写数据给显示屏使用 CG RAM 或 DD RAM 命令。8 位数据值通过查表地址送给 CG ROM 或 CG RAM，如图 5.4 所示。CG ROM 或 CG RAM 中存储的位图驱动 5×8 点阵给相应的字符。

如果地址计数器配置为自动增 1，正如前面所说的，这种应用方法可以次序写多个字符编码，每个字符自动存储并显示在下个位置。

继续写字符，但是，在第一显示行的最后停止。剩余的数据不会自动在第二行显示，因为 DD RAM 的映射从第一行到第二行并不是连续的。

Disabling the Unused LCD

如果 FPGA 的应用不使用 LCD 显示屏，置低 LCD_E 管脚，失能它。同样，置低 LCD_RW 管脚，阻止 LCD 屏回传数据。

在这个基础上，我们对模版代码进行修改，传输 Spartan-3E□FPGA 与 FPAG□Starter 字符，并设置 BTN_WEST 做为复位，当按下 BTN_WEST 时，LCD 复位并刷新重新显示两行字符串。

5、Verilog 模块设计

源码：

```
module LCD_Driver( output SF_CE0,           // 4 位 LCD 数据信号与 StrataFlash 存储器共享数据线
SF_D<11:8>.
                                            // 当 SF_CE0 = High 时，禁用 StrataFlash 存储器，
                                            // 此时 FPGA 完全 read/write 访问 LCD.
                   output LCD_RW,           // Read/Write Control
                                            // 0: WRITE, LCD accepts SF_D
                                            // 1: READ, LCD presents SF_D

                   output LCD_RS,           // Register Select
                                            //  0:  Instruction  register  during  write
operations.
                                            //   Busy Flash during read operations
                                            // 1: Data for read or write operations

                   output [3:0] SF_D,       // Four-bit SF_D interface, Data bit DB7 ~ DB4,
                                            // Shared with StrataFlash pins SF_D<11:8>

                   output LCD_E,         // Read/Write Enable Pulse,
                                            //  0:  Disabled,  1:  Read/Write  operation
enabled

                   input clock,             // 连接 On-Board 50 MHz Oscillator CLK_50MHz
(C9)
                   input reset              // 使用按键 BTN West(D18)做为复位键

                  );

    ////////////////////////////////////////////////////////////////////////

    ////////////////////////////////////////////////
    // 定义 LCD 初始化和显示配置状态变量

    parameter   INIT_IDLE       = 4'h1,
                WAITING_READY   = 4'h2,

                WR_ENABLE_1     = 4'h3,
```

```
                WAITING_1       = 4'h4,
                WR_ENABLE_2     = 4'h5,
                WAITING_2       = 4'h6,

                WR_ENABLE_3     = 4'h7,
                WAITING_3       = 4'h8,
                WR_ENABLE_4     = 4'h9,
                WAITING_4       = 4'hA,

                INIT_DONE       = 4'hB;

// 保存 LCD 初始化状态变量: 位宽 3 bits
reg [3:0] init_state;

// 时序控制计数器
// The 15 ms interval is 750,000 clock cycles at 50 MHz.
// 750,000 (dec) = 1011_0111_0001_1011_0000(bin) 需要 20 bits
reg [19:0] cnt_init;

// 初始化状态标志
// 0: 初始化未完成
// 1: 初始化已完成
reg init_done;


parameter   DISPLAY_INIT    = 4'h1,

                FUNCTION_SET    = 4'h2,
                ENTRY_MODE_SET  = 4'h3,
                DISPLAY_ON_OFF  = 4'h4,
                DISPLAY_CLEAR   = 4'h5,
                CLEAR_EXECUTION = 4'h6,
                IDLE_2SEC       = 4'h7,

                SET_DD_RAM_ADDR = 4'h8,
                LCD_LINE_1      = 4'h9,
                SET_NEWLINE     = 4'hA,
                LCD_LINE_2      = 4'hB,
                DISPLAY_DONE    = 4'hC;
```

```
// 保存 LCD 显示配置状态变量: 位宽 3 bits
reg [3:0] ctrl_state;

// 时序控制计数器
// Clear the display and return the cursor to the home position, the top-left corner.
// Execution Time at least 1.64 ms (82,000 clock cycles)
// 82,000 (dec) = 1_0100_0000_0101_0000 (bin) 需要 17 bits
reg [16:0] cnt_delay;

// 控制初始化标志
// 1: 启动传输过程
// 0: 停止传输过程
reg init_exec;

// 复位后, 等待 2 sec, 运行在 50 MHz 时钟频率
// 等待 100,000,000(dec) = 101_1111_0101_1110_0001_0000_0000 (bin) (27 bits) 时钟周期
reg [26:0] cnt_2sec;

// 控制传输标志
// 1: 启动  涔  ?// 0: 停止传输过程
reg tx_ctrl;

// 传输序列状态
parameter   TX_IDLE     = 8'H01,
            UPPER_SETUP = 8'H02,
            UPPER_HOLD  = 8'H04,
            ONE_US      = 8'H08,
            LOWER_SETUP = 8'H10,
            LOWER_HOLD  = 8'H20,
            FORTY_US    = 8'H40;


// 保存传输序列状态: 位宽 7 bits
reg [6:0] tx_state;

// 传输控制时序计数器
// The time between successive commands is 40us, which corresponds to 2000 clock cycles
// 2000 (dec ) = 111_1101_0000 (bin) 需要 11 bits
reg [10:0] cnt_tx;
```

```
    // Register Select
    // 0: Instruction register during write operations. Busy Flash during read operations
    // 1: Data for read or write operations
    reg select;

    // The upper nibble is transferred first, followed by the lower nibble.
    reg [3:0] nibble;
    reg [3:0] DB_init;  // 用于初始化

    // Read/Write Enable Pulse, 0: Disabled, 1: Read/Write operation enabled
    reg enable;
    reg en_init;       // 用于初始化

    reg mux;             // 标志初始化过程，传输命令/数据
                         // 0: 初始化
                         // 1: 传输命令/数据

    // 向 LCD 传输的数据字节：位宽 8 bits
    reg [7:0] tx_byte;

    // 保存第 1 行显示输出的字符数据
    reg [7:0] tx_Line1;

    // 保存第 2 行显示输出的字符数据
    reg [7:0] tx_Line2;

    // 显示字符计数器
    reg [3:0] cnt_1 = 4'b0; // For Line 1
    reg [3:0] cnt_2 = 4'b0; // For Line 2

    ////////////////////////////////////////////////////////////////////////////////
//////
    // 禁用 Intel strataflash 存储器，将 Read/Write 控制设置为 Write，即：LCD 接收数据
    assign SF_CE0   = 1'b1; // Disable intel strataflash

    assign LCD_RW   = 1'b0;     // Write only

    assign LCD_RS   = select;

    assign SF_D = ( mux ) ? nibble : DB_init;
```

```
    assign LCD_E    = ( mux ) ? enable : en_init;

    always @(*)
    begin
        case ( ctrl_state )
            DISPLAY_INIT:       mux = 1'b0; // power on initialization sequence
            FUNCTION_SET,
            ENTRY_MODE_SET,
            DISPLAY_ON_OFF,
            DISPLAY_CLEAR,
            IDLE_2SEC,
            CLEAR_EXECUTION,
            SET_DD_RAM_ADDR,
            LCD_LINE_1,
            SET_NEWLINE,
            LCD_LINE_2:         mux = 1'b1;
            default:            mux = 1'b0;
        endcase
    end
    ////////////////////////////////////////////////////////////////////////////////
//////


    // The following "always" statements simplify the process of adding and removing states.

    //
    // refer to datasheet for an explanation of these values

    // 向 LCD 传输的命令字节: 位宽 8 bits
    // In Verilog-2001, you can initialize registers when you declare them.
    // Now Xilinx XST has supported to initialize registers
    always @( * ) begin
        case ( ctrl_state )
            FUNCTION_SET:       begin
                                    tx_byte = 8'b0010_1000;
                                    select = 1'b0;
                                end
            ENTRY_MODE_SET:     begin
                                    tx_byte = 8'b0000_0110;
                                    select = 1'b0;
```

```
                            end
        DISPLAY_ON_OFF:     begin
                                tx_byte = 8'b0000_1100;
                                select = 1'b0;
                            end
        DISPLAY_CLEAR:      begin
                                tx_byte = 8'b0000_0001;
                                select = 1'b0;
                            end
        SET_DD_RAM_ADDR:    begin
                                tx_byte = 8'b1000_0000;
                                select = 1'b0;
                            end
        ////////////////////////////////////////////////
        LCD_LINE_1:         begin
                                tx_byte = tx_Line1;
                                select = 1'b1;
                            end
        SET_NEWLINE:        begin
                                tx_byte = 8'b1100_0000;
                                select = 1'b0;
                            end
        ////////////////////////////////////////////////
        LCD_LINE_2:         begin
                                tx_byte = tx_Line2;
                                select = 1'b1;
                            end

        default:            begin
                                tx_byte = 8'b0;
                                select = 1'b0;
                            end
    endcase
end

always @(*)
begin
    case ( cnt_1 )
        0:      tx_Line1 = 8'b0101_0011;        // CHAR_S
        1:      tx_Line1 = 8'b0111_0000;        // CHAR_p
```

```
            2:      tx_Line1 = 8'b0110_0001;        // CHAR_a
            3:      tx_Line1    = 8'b0111_0010;     // CHAR_r
            4:      tx_Line1    = 8'b0111_0100;     // CHAR_t
            5:      tx_Line1    = 8'b0110_0001;     // CHAR_a
            6:      tx_Line1    = 8'b0110_1110;     // CHAR_n
            7:      tx_Line1    = 8'b0010_1101;     // CHAR_-
            8:      tx_Line1    = 8'b0011_0011;     // CHAR_3
            9:      tx_Line1    = 8'b0100_0101;     // CHAR_E
            10: tx_Line1    = 8'b0010_0000;     // CHAR_space1
            11:     tx_Line1    = 8'b0100_0110;     // CHAR_F
            12: tx_Line1 = 8'b0101_0000;        // CHAR_P
            13: tx_Line1 = 8'b0100_0111;        // CHAR_G
            14: tx_Line1 = 8'b0100_0001;        // CHAR_A
            default:tx_Line1    = 8'b0;             // NONE
        endcase
    end

    always @(*)
    begin
        case ( cnt_2 )
            0:      tx_Line2 = 8'b0100_0110;        // CHAR_F
            1:      tx_Line2 = 8'b0101_0000;        // CHAR_P
            2:      tx_Line2 = 8'b0100_0111;        // CHAR_G
            3:      tx_Line2 = 8'b0100_0001;        // CHAR_A
            4:      tx_Line2 = 8'b0010_0000;        // CHAR_space2
            5:      tx_Line2 = 8'b0101_0011;        // CHAR_S
            6:      tx_Line2 = 8'b0111_0100;        // CHAR_t
            7:      tx_Line2 = 8'b0110_0001;        // CHAR_a
            8:      tx_Line2 = 8'b0111_0010;        // CHAR_r
            9:      tx_Line2 = 8'b0111_0100;        // CHAR_t
            10:     tx_Line2 = 8'b0110_0101;        // CHAR_e
            11:     tx_Line2 = 8'b0111_0010;        // CHAR_r
            default:tx_Line2    = 8'b0;             // NONE
        endcase
    end

/*      上电后 LCD 初始化过程
        Power-On Initialization

        The initialization sequence first establishes that the FPGA application
```

```
       wishes to use the four-bit SF_D interface to the LCD as follows:

       (0) Wait 15 ms or longer, although the display is generally ready when the FPGA
finishes configuration.
          The 15 ms interval is 750,000 clock cycles at 50 MHz.

       (1) Write SF_D<11:8> = 0x3, pulse LCD_E High for 12 clock cycles.

       (2) Wait 4.1 ms or longer, which is 205,000 clock cycles at 50 MHz.

       (3) Write SF_D<11:8> = 0x3, pulse LCD_E High for 12 clock cycles.

       (4) Wait 100 µs or longer, which is 5,000 clock cycles at 50 MHz.

       (5) Write SF_D<11:8> = 0x3, pulse LCD_E High for 12 clock cycles.

       (6) Wait 40 µs or longer, which is 2,000 clock cycles at 50 MHz.

       (7) Write SF_D<11:8> = 0x2, pulse LCD_E High for 12 clock cycles.

       (8) Wait 40 µs or longer, which is 2,000 clock cycles at 50 MHz.
*/

   // Initializing the Display
   always @( posedge clock )
   begin
       if( reset ) begin
           init_state <= INIT_IDLE;

           DB_init <= 4'b0;
           en_init <= 0;

           cnt_init <= 0;

           init_done <= 0;
       end

       else begin
           case ( init_state )
               // power on initialization sequence
```

```
                INIT_IDLE:          begin
                                        en_init <= 0;

                                        if ( init_exec  )
                                            init_state <= WAITING_READY;
                                        else
                                            init_state <= INIT_IDLE;
                                    end

                WAITING_READY:      begin   // (0 )等待 15 ms 或更长, LCD 准备显示
                                        en_init <= 0;

                                        if ( cnt_init <= 750000 ) begin
                                            DB_init <= 4'h0;

                                            cnt_init <= cnt_init + 1;

                                            init_state <= WAITING_READY;
                                        end
                                        else begin
                                            cnt_init <= 0;

                                            init_state <= WR_ENABLE_1;
                                        end
                                    end

                WR_ENABLE_1:        begin
                                        DB_init <= 4'h3;            // Write SF_D<11:8> =
0x3

                                        en_init <= 1'b1;            //  Pulse  LCD_E  High
for 12 clock cycles.

                                        if ( cnt_init < 12 ) begin
                                            cnt_init <= cnt_init + 1;

                                            init_state <= WR_ENABLE_1;
                                        end
                                        else begin
                                            cnt_init <= 0;
```

```
                                        init_state <= WAITING_1;
                                    end
                                end

                WAITING_1:          begin   // Wait 4.1 ms or longer, which is 205,000
clock cycles at 50 MHz.
                                    en_init <= 1'b0;

                                    if ( cnt_init <= 205000 ) begin

                                        cnt_init <= cnt_init + 1;

                                        init_state <= WAITING_1;
                                end
                                else begin
                                        cnt_init <= 0;

                                        init_state <= WR_ENABLE_2;
                                    end
                                end
                WR_ENABLE_2:        begin
                                    DB_init <= 4'h3;            // Write SF_D<11:8> =
0x3
                                    en_init <= 1'b1;            //  Pulse  LCD_E  High
for 12 clock cycles.

                                    if ( cnt_init < 12 ) begin

                                        cnt_init <= cnt_init + 1;

                                        init_state <= WR_ENABLE_2;
                                end
                                else begin
                                        cnt_init <= 0;

                                        init_state <= WAITING_2;
                                    end
                                end
                                // Wait 100 µs or longer, which is 5,000 clock cycles
```

```
at 50 MHz.
                WAITING_2:          begin
                                        en_init <= 1'b0;

                                        if ( cnt_init <= 5000 ) begin

                                            cnt_init <= cnt_init + 1;

                                            init_state <= WAITING_2;
                                        end
                                        else begin
                                            cnt_init <= 0;

                                            init_state <= WR_ENABLE_3;
                                        end
                                    end

                WR_ENABLE_3:        begin   //  Write SF_D<11:8> = 0x3, pulse LCD_E High
for 12 clock cycles.
                                        DB_init <= 4'h3;            // Write SF_D<11:8> =
0x3
                                        en_init <= 1'b1;            //  Pulse  LCD_E  High
for 12 clock cycles.

                                        if ( cnt_init < 12 ) begin

                                            cnt_init <= cnt_init + 1;

                                            init_state <= WR_ENABLE_3;
                                    end
                                    else begin
                                            cnt_init <= 0;

                                            init_state <= WAITING_3;
                                        end
                                    end

                WAITING_3:          begin   //  Wait 40 us or longer, which is 2,000 clock
cycles at 50 MHz.
                                        en_init <= 1'b0;
```

```
                                    if ( cnt_init <= 2000 ) begin

                                        cnt_init <= cnt_init + 1;

                                        init_state <= WAITING_3;
                                end
                                else begin
                                        cnt_init <= 0;

                                        init_state <= WR_ENABLE_4;
                                    end
                                end

                WR_ENABLE_4:        begin   //  Write SF_D<11:8> = 0x2, pulse LCD_E High
for 12 clock cycles.
                                    DB_init <= 4'h2;            // Write SF_D<11:8> =
0x3
                                    en_init <= 1'b1;            //  Pulse  LCD_E  High
for 12 clock cycles.

                                    if ( cnt_init < 12 ) begin

                                        cnt_init <= cnt_init + 1;

                                        init_state <= WR_ENABLE_4;
                                end
                                else begin
                                        cnt_init <= 0;

                                        init_state <= WAITING_4;
                                    end
                                end

                WAITING_4:          begin   //  Wait 40 us or longer, which is 2,000 clock
cycles at 50 MHz.
                                    en_init <= 1'b0;



                                    if ( cnt_init <= 2000 ) begin
```

```
                                    cnt_init <= cnt_init + 1;

                                    init_state <= WAITING_4;
                                end
                                else begin
                                    DB_init <= 4'h0;        // Write SF_D<11:8> =
0x0
                                    cnt_init <= 0;

                                    cnt_init <= 0;

                                    init_done <= 1'b1;
                                    init_state <= INIT_DONE;
                                end
                            end

            INIT_DONE:      begin
                                init_state <= INIT_DONE;

                                DB_init <= 4'h0;
                                en_init <= 1'b0;

                                cnt_init <= 0;

                                init_done <= 1'b1;
                            end
            default:        begin
                                init_state <= INIT_IDLE;

                                DB_init <= 4'b0;
                                en_init <= 0;

                                cnt_init <= 0;

                                init_done <= 0;
                            end
        endcase
    end
end
```

```
always @( * )
begin
    case ( ctrl_state )
        DISPLAY_INIT:       tx_ctrl = 1'b0;
        FUNCTION_SET,
        ENTRY_MODE_SET,
        DISPLAY_ON_OFF,
        DISPLAY_CLEAR:      tx_ctrl = 1'b1;
        CLEAR_EXECUTION:    tx_ctrl = 1'b0;
        SET_DD_RAM_ADDR,
        LCD_LINE_1,
        SET_NEWLINE,
        LCD_LINE_2:         tx_ctrl = 1'b1;
        DISPLAY_DONE:       tx_ctrl = 1'b0;
        default:            tx_ctrl = 1'b0;
    endcase
end

// Main state machine
always @( posedge clock )
begin
    if( reset ) begin
        ctrl_state <= DISPLAY_INIT;

        cnt_delay <= 0;
        cnt_1 <= 0;
        cnt_2 <= 0;

        cnt_2sec <= 0;
    end

    else begin
        case ( ctrl_state )
            // power on initialization sequence
            DISPLAY_INIT:       begin   // (0 )等待 15 ms 或更长, LCD 准备显示
                                    init_exec <= 1;

                                    if ( init_done ) begin
                                        ctrl_state <= FUNCTION_SET;
```

```
                    cnt_1 <= 0;
                    cnt_2 <= 0;
                end
                else begin
                    ctrl_state <= DISPLAY_INIT;
                end
            end

        FUNCTION_SET:       begin
                // Wait 40 us or longer
                if ( cnt_tx <= 2000 ) begin
                    ctrl_state <= FUNCTION_SET;
                end
                else begin
                    ctrl_state <= ENTRY_MODE_SET;
                end
            end

        ENTRY_MODE_SET:     begin
                // Wait 40 us or longer
                if ( cnt_tx <= 2000 ) begin
                    ctrl_state <= ENTRY_MODE_SET;
                end
                else begin
                    ctrl_state <= DISPLAY_ON_OFF;
                end
            end

        DISPLAY_ON_OFF:     begin
                // Wait 40 us or longer
                if ( cnt_tx <= 2000 ) begin
                    ctrl_state <= DISPLAY_ON_OFF;
                end
                else begin
                    ctrl_state <= DISPLAY_CLEAR;
                end
            end

        DISPLAY_CLEAR:      begin
                // Wait 40 us or longer
```

```
                                    if ( cnt_tx <= 2000 ) begin
                                        ctrl_state <= DISPLAY_CLEAR;
                                    end
                                    else begin
                                        ctrl_state <= CLEAR_EXECUTION;

                                        cnt_delay <= 0;
                                    end
                                end

                CLEAR_EXECUTION:    begin
                                    // The delay after a Clear Display command is
1.64ms,
                                    // which corresponds to 82000 clock cycles.
                                    if ( cnt_delay <= 82000 ) begin
                                        ctrl_state <= CLEAR_EXECUTION;

                                        cnt_delay <= cnt_delay + 1;
                                    end
                                    else begin
                                        ctrl_state <= IDLE_2SEC;
                                        cnt_delay <= 0;

                                        cnt_2sec <= 0;
                                    end
                                end

                IDLE_2SEC:          begin // 清屏后, 等待 2 sec, 观察复位
                                    if ( cnt_2sec < 27'd100000000 ) begin
                                        ctrl_state <= IDLE_2SEC;
                                        cnt_2sec <= cnt_2sec + 1;
                                    end
                                    else begin
                                        ctrl_state <= SET_DD_RAM_ADDR;

                                        cnt_delay <= 0;
                                    end
                                end

                SET_DD_RAM_ADDR:    begin
```

```
                            // Wait 40 us or longer
                            if ( cnt_tx <= 2000 ) begin
                                ctrl_state <= SET_DD_RAM_ADDR;
                            end
                            else begin
                                ctrl_state <= LCD_LINE_1;
                                cnt_1 <= 0;
                            end
                        end

LCD_LINE_1:             begin
                            // Wait 40 us or longer
                            if ( cnt_tx <= 2000 ) begin
                                ctrl_state <= LCD_LINE_1;
                            end
                            else if ( cnt_1 < 14 ) begin
                                    ctrl_state <= LCD_LINE_1;

                                    cnt_1 <= cnt_1 + 1;
                                end
                                else begin
                                    ctrl_state <= SET_NEWLINE;

                                    cnt_1 <= 0;
                                end
                        end

SET_NEWLINE:            begin
                            // Wait 40 us or longer
                            if ( cnt_tx <= 2000 ) begin
                                ctrl_state <= SET_NEWLINE;
                            end
                            else begin
                                ctrl_state <= LCD_LINE_2;

                                cnt_2 <= 0;
                            end
                        end

LCD_LINE_2:             begin
```

```
                                    // Wait 40 us or longer
                                    if ( cnt_tx <= 2000 ) begin
                                        ctrl_state <= LCD_LINE_2;
                                    end
                                    else if ( cnt_2 < 11 ) begin
                                            ctrl_state <= LCD_LINE_2;

                                            cnt_2 <= cnt_2 + 1;
                                        end
                                        else begin
                                            ctrl_state <= DISPLAY_DONE;

                                            cnt_2 <= 0;
                                        end
                                end

            DISPLAY_DONE:       begin
                                    ctrl_state <= DISPLAY_DONE;
                                end
            default:            begin
                                    ctrl_state <= DISPLAY_INIT;

                                    cnt_delay <= 0;
                                    cnt_1 <= 0;
                                    cnt_2 <= 0;

                                    cnt_2sec <= 0;
                                end
        endcase
    end
end
/*
    Four-Bit Data Interface

    The board uses a 4-bit SF_D interface to the character LCD.
    The SF_D values on SF_D<11:8>, and the register select (LCD_RS) and the read/write
(LCD_RW)
    control signals must be set up and stable at least 40 ns before the enable LCD_E goes
High.
```

```
    The enable signal must remain High for 230 ns or longer-the equivalent of 12 or more
clock cycles at 50 MHz.

    In many applications, the LCD_RW signal can be tied Low permanently
    because the FPGA generally has no reason to read information from the display.

    Transferring 8-Bit Data over the 4-Bit Interface

    After initializing the display and establishing communication,
    all commands and SF_D transfers to the character display are via 8 bits,
    transferred using two sequential 4-bit operations.
    Each 8-bit transfer must be decomposed into two 4-bit transfers, spaced apart by at
least 1 µs.

    The upper nibble is transferred first, followed by the lower nibble.
    An 8-bit write operation must be spaced least 40 µs before the next communication.
    This delay must be increased to 1.64 ms following a Clear Display command.

    Note that the period of the 50MHz onboard clock is 20ns.

    The time between corresponding nibbles is 1us, which is equivalent to 50 clock cycles.

    The time between successive commands is 40us, which corresponds to 2000 clock cycles.

    The delay after a Clear Display command is 1.64ms, which corresponds to 82000 clock
cycles.

    Setup time ( time for the outputs to stabilize ) is 40ns, which is 2 clock cycles,
    the hold time ( time to assert the LCD_E pin ) is 230ns, which translates to roughly
12 clock cycles,
    and the fall time ( time to allow the outputs to stabilize) is 10ns, which translates
to roughly 1 clock cycle.
*/
    // specified by datasheet, transmit process
        // specified by datasheet, transmit process
    always @( posedge clock )
    begin
        if ( reset ) begin
            enable <= 1'b0;
            nibble <= 4'b0;
```

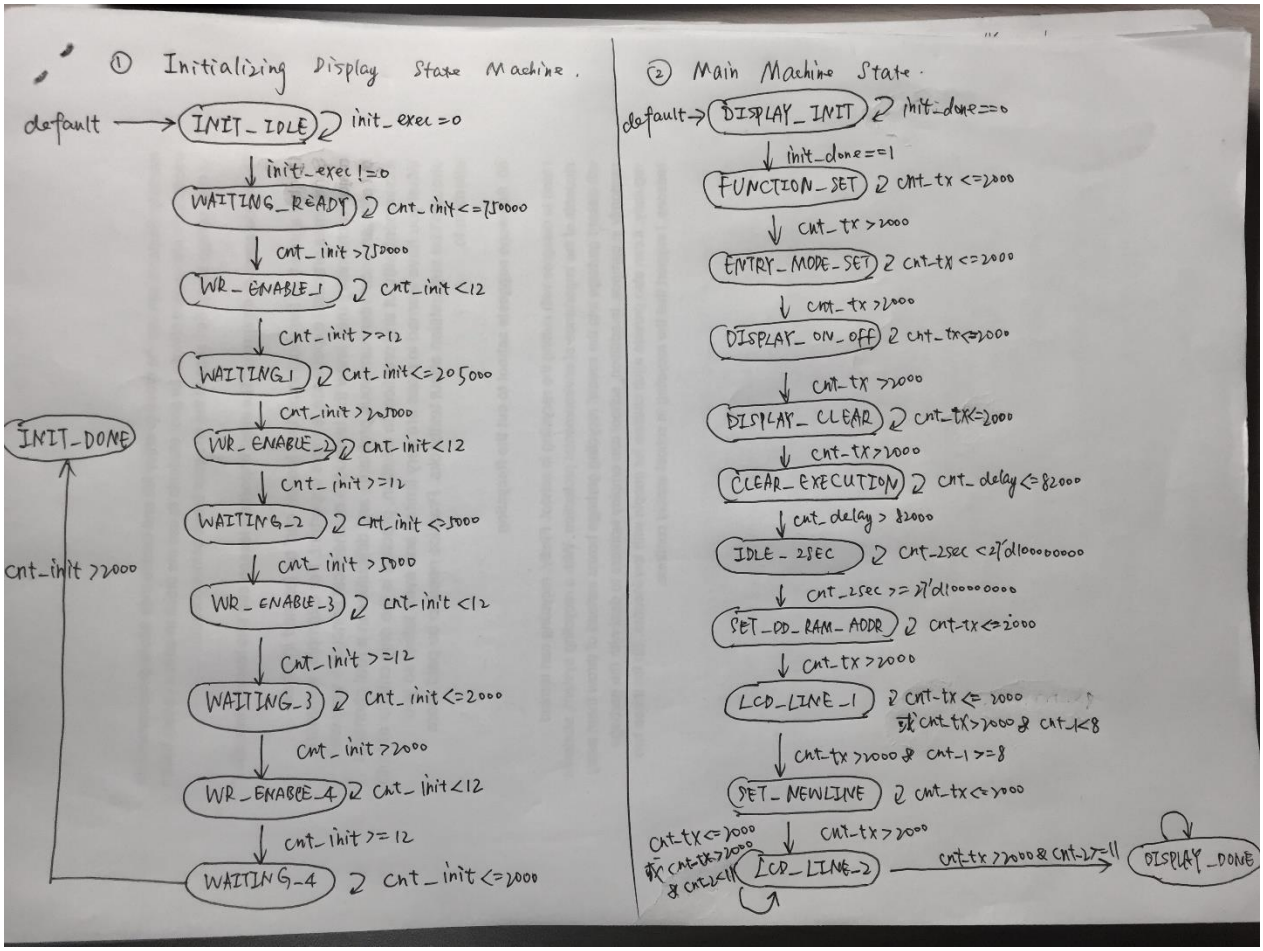
```
            tx_state <= TX_IDLE;
            cnt_tx <= 0;
        end
        else  begin
            case ( tx_state )
                TX_IDLE:            begin
                                        enable <= 1'b0;
                                        nibble <= 4'b0;
                                        cnt_tx <= 0;

                                        if ( tx_ctrl ) begin
                                            tx_state <= UPPER_SETUP;
                                        end
                                        else begin
                                            tx_state <= TX_IDLE;
                                        end
                                    end
                // Setup time ( time for the outputs to stabilize ) is 40ns, which is 2
clock cycles
                UPPER_SETUP:        begin
                                        nibble <= tx_byte[7:4];

                                        if ( cnt_tx < 2 ) begin
                                            enable <= 1'b0;

                                            tx_state <= UPPER_SETUP;

                                            cnt_tx <= cnt_tx + 1;
                                        end
                                        else begin
                                            enable <= 1'b1;

                                            tx_state <= UPPER_HOLD;
                                            cnt_tx <= 0;
                                        end
                                    end
                // Hold time ( time to assert the LCD_E pin ) is 230ns, which translates
to roughly 12 clock cycles
                UPPER_HOLD:         begin
```

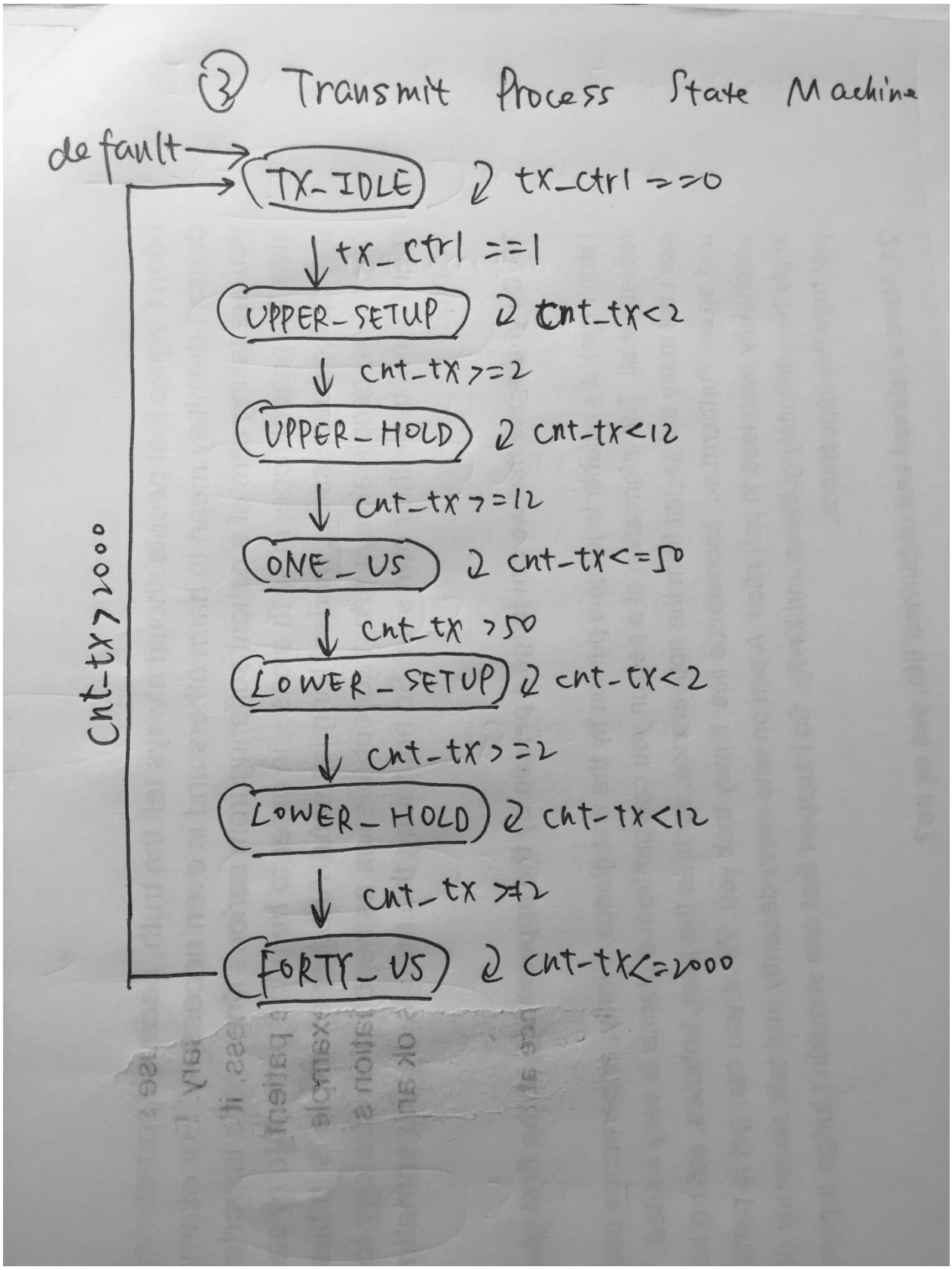
```
                                    nibble <= tx_byte[7:4];

                                    if ( cnt_tx < 12 ) begin
                                        enable <= 1'b1;
                                        tx_state <= UPPER_HOLD;
                                        cnt_tx <= cnt_tx + 1;
                                    end
                                    else begin
                                        enable <= 1'b0;
                                        tx_state <= ONE_US;
                                        cnt_tx <= 0;
                                    end
                                end
                // Each 8-bit transfer must be decomposed into two 4-bit transfers, spaced
apart by at least 1 µs.
                // The upper nibble is transferred first, followed by the lower nibble.
                // The time between corresponding nibbles is 1us, which is equivalent to
50 clock cycles.
                ONE_US:             begin
                                    enable <= 1'b0;

                                    if ( cnt_tx <= 50 ) begin
                                        tx_state <= ONE_US;
                                        cnt_tx <= cnt_tx + 1;
                                    end
                                    else begin
                                        tx_state <= LOWER_SETUP;
                                        cnt_tx <= 0;
                                    end
                                end
                // Setup time ( time for the outputs to stabilize ) is 40ns, which is 2
clock cycles
                LOWER_SETUP:        begin
                                    nibble <= tx_byte[3:0];

                                    if ( cnt_tx < 2 ) begin
                                        enable <= 1'b0;

                                        tx_state <= LOWER_SETUP;
                                        cnt_tx <= cnt_tx + 1;
```

```
                                    end
                                    else begin
                                        enable <= 1'b1;

                                        tx_state <= LOWER_HOLD;
                                        cnt_tx <= 0;
                                    end
                                end

            // Hold time ( time to assert the LCD_E pin ) is 230ns, which translates
to roughly 12 clock cycles
            LOWER_HOLD:         begin
                                    nibble <= tx_byte[3:0];

                                    if ( cnt_tx < 12 ) begin
                                        enable <= 1'b1;
                                        tx_state <= LOWER_HOLD;
                                        cnt_tx <= cnt_tx + 1;
                                    end
                                    else begin
                                        enable <= 1'b0;
                                        tx_state <= FORTY_US;
                                        cnt_tx <= 0;
                                    end
                                end

            // The time between successive commands is 40us, which corresponds to 2000
clock cycles.
            FORTY_US:           begin
                                    enable <= 1'b0;

                                    if ( cnt_tx <= 2000 ) begin
                                        tx_state <= FORTY_US;
                                        cnt_tx <= cnt_tx + 1;
                                    end
                                    else begin
                                        tx_state <= TX_IDLE;
                                        cnt_tx <= 0;
                                    end
                                end
```

```
                default:            begin
                                            enable <= 1'b0;
                                            nibble <= 4'b0;

                                            tx_state <= TX_IDLE;
                                            cnt_tx <= 0;
                                    end
            endcase
        end
    end
endmodule
```

状态机：




③ Transmit Process State Machine
default
TX_IDLE @ tx_ctrl ==0
tx_ctrl ==1
UPPER_SETUP @ cnt_tx<2
cnt_tx >=2
UPPER_HOLD @ cnt_tx<12
cnt_tx >=12
ONE_US @ cnt_tx<=50
cnt_tx >50
LOWER_SETUP @ cnt_tx<2
cnt_tx >=2
LOWER_HOLD @ cnt_tx<12
cnt_tx >=2
FORTY_US @ cnt_tx<=2000
cnt_tx>2000