同济大学计算机系

数字逻辑课程综合实验报告

学　　号	1953610
姓　　名	孙久昌
专　　业	信息安全
授课老师	郭玉臣

一. 实验内容

名称

m1fpga-s - 基于现场可编辑逻辑门阵列（FPGA）的 Martin M1 制式慢扫描电视（SSTV）调制器。

用法

```
~/p/p/impl_1  file m1fpga-s.bit
m1fpga-s.bit: Xilinx BIT data - from top;UserID=0XFFFFFFFF;Version=2018.3 - for 7a100tcsg324 - bui
lt 2020/12/07(12:54:35) - data length 0x3a607c
```

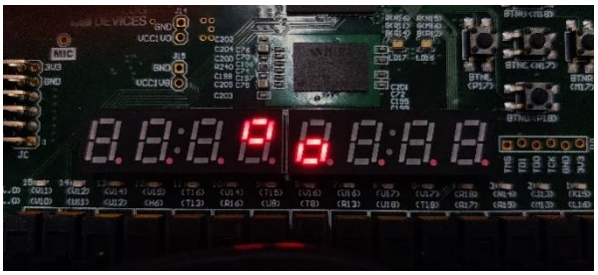
实验设计的成品是一个 Xilinx FPGA 位流文件，目标平台是 Nexys4 DDR 开发板。

开始前，需要准备一张已格式化为 FAT32 文件系统的 MicroSD 存储卡和一张图片。

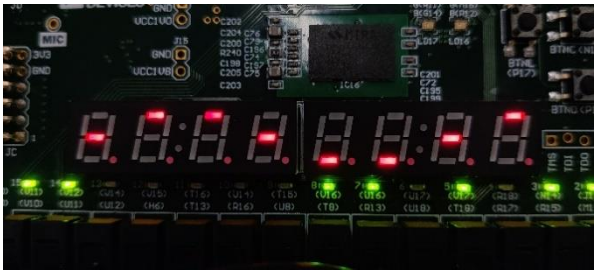
首先使用任意图像处理软件将原图缩放并裁切为宽 320 像素、高 256 像素的尺寸，保存为位偏移 `54` 字节，位深 24 的 `Windows 3.x 格式 PC 位图`格式。Windows 自带的画图工具保存的位图图像（`*.bmp`）就可满足要求。

将图片命名为 `sstv.bmp`，把它和位流文件一起放在存储卡的根目录下，然后将存储卡装入开发板。将开发板的音频接口通过音频线连接到带 3.5mm 线路输入接口的播放设备上，然后就可以打开开发板电源了。

程序载入之后，数码管上会显示方块移动的动画，同时下方 LED 灯会快速闪烁。这时图像正在加载。



图像加载完成后，可以听到规律性的脉冲声，数码管上显示正弦波形的动画，同时下方 LED 灯有规律地闪烁。这时图像正在通过音频信号播发。使用支持 Martin M1 制式的收录设备监听这段脉冲声，就可以接收到准备好的图像。

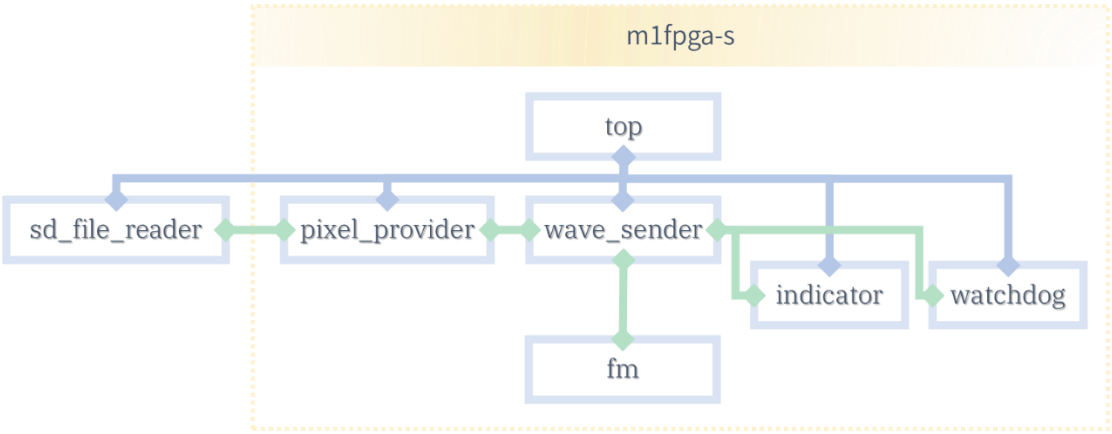


示例

见 § 实验结果。

二. Martin1 SSTV 调制器数字系统总框图

以下是本实验设计的系统总框图。



各子系统的功能和实现介绍如下。

- `top`：主模块

  负责系统输入输出的管理，例化各子模块并建立数据连接。

- `sd_file_reader`：基于 SPI 总线的 SD 卡文件读写模块

  从 FPGA 上可用的 SD 卡存储器中查找 FAT 文件系统下的 `sstv.bmp` 位图文件，然后将数据传送到存储模块。

- `pixel_provider`：存取图像像素信息的模块

  将从外部存储器载入的图像存储下来，再根据信号发送顺序依次读出并传送到调制模块。

- `wave_sender`：将图像调制为声波的模块

  基于图像的像素信息生成符合 Martin M1 制式的音频信号，并输出到板载单声道音频接口上。

- `indicator`, `watchdog`：显示设备运行状态的模块

指示模块 `indicator` 根据设备当前的运行状态在板上的七段数码管上显示动画。监测模块 `watchdog` 根据系统时钟判断文件载入过程是否正常完成，并在文件加载超时的情形下提示用户检查问题。

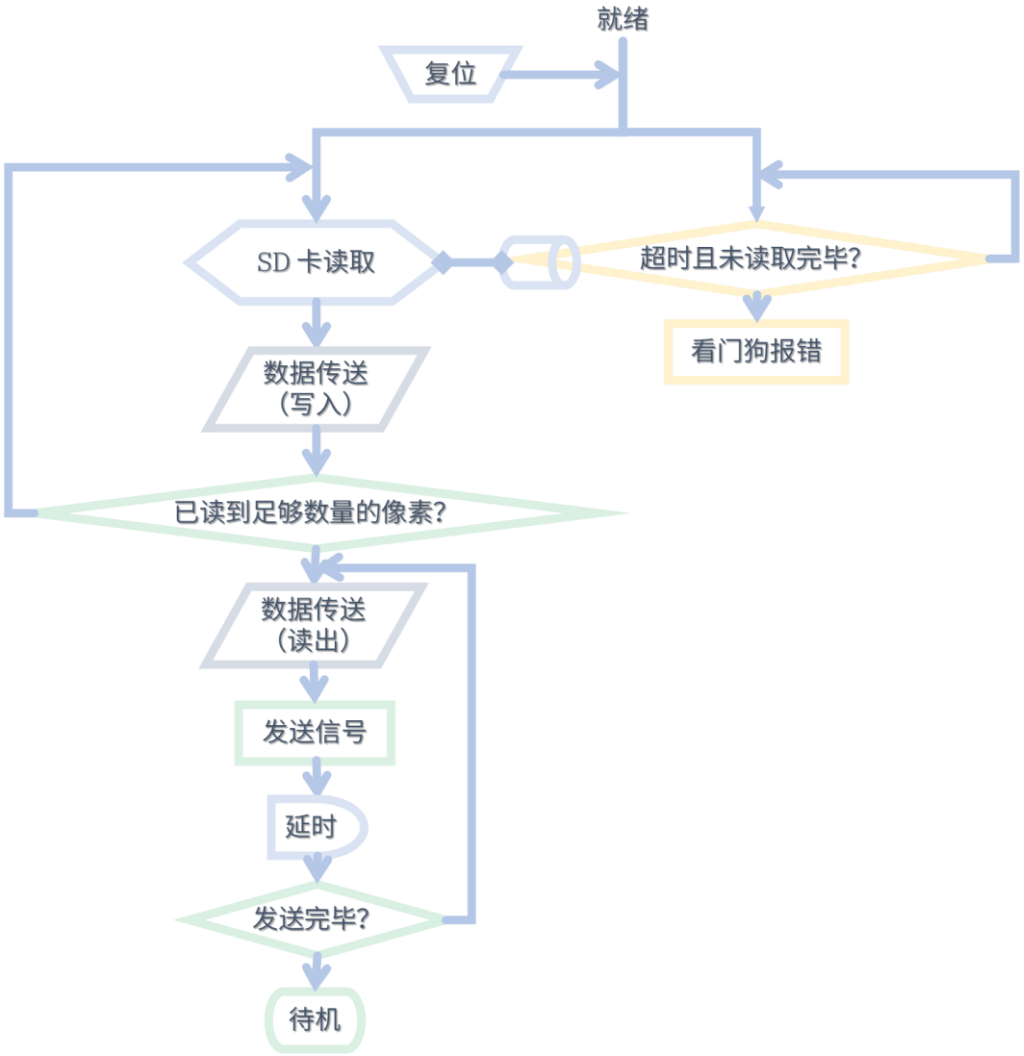
- `fm`：脉冲密度调制（PDM）信号的生成模块

类似一个动态可调分频倍率的分频器，可以生成脉冲密度调制的方波信号，用于形成指定频率的音频。

在以上各模块中，SD 读取模块和储存模块经由使能总线 `load_trigger` 和数据总线 `px_idata` 相连接，调制模块和储存模块经由使能总线 `load_done`, `send_trigger` 和数据总线 `px_odata` 相连接，指示模块和看门狗模块通过信号线 `err` 相连接。

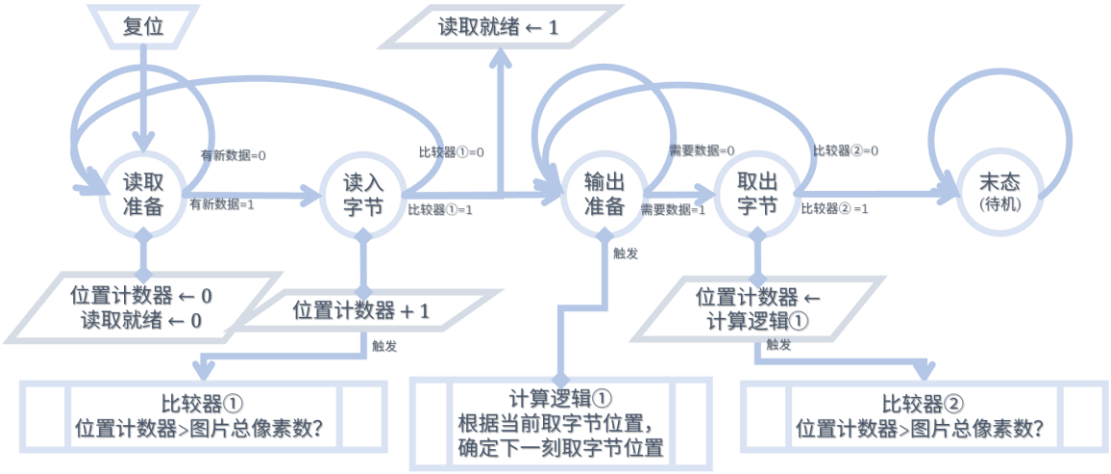
三. 系统控制器设计

本次所设计系统的 ASM 流程图如下。



考虑到本系统的主要状态逻辑是和数据的保存状态密切相关的，因此存储子模块和发波子模块分别包含了完整的主逻辑，并在运行期间通过信号线同步状态。状态的转移在存储子模块内部进行，而各时刻不同的状态信息经由信号线，仅传送给 需要根据这一状态的更新来切换输出 的其它模块。这样的做法降低了系统控制器的设计复杂度，并且符合了低内聚、高耦合的工程实践思路。

如下所示是存储子模块的状态图。



令$\{Q_2, Q_1, Q_0\} = \{000,001,010,011,110\}$分别代表读取准备到待机的五个状态，其中

$Q_2 =$ `wave_sender/done`，

$Q_1 =$ `pixel_provider/d_ok`，

$Q_0 =$ `pixel_provider/d_ok` ? `top/load_trigger` : `top/send_trigger`；

并且$\{R, Y_0, Y_1, Y_2, Y_3\} = \{$复位, 有新数据, 比较器①结果, 需要数据, 比较器②结果$\}$，

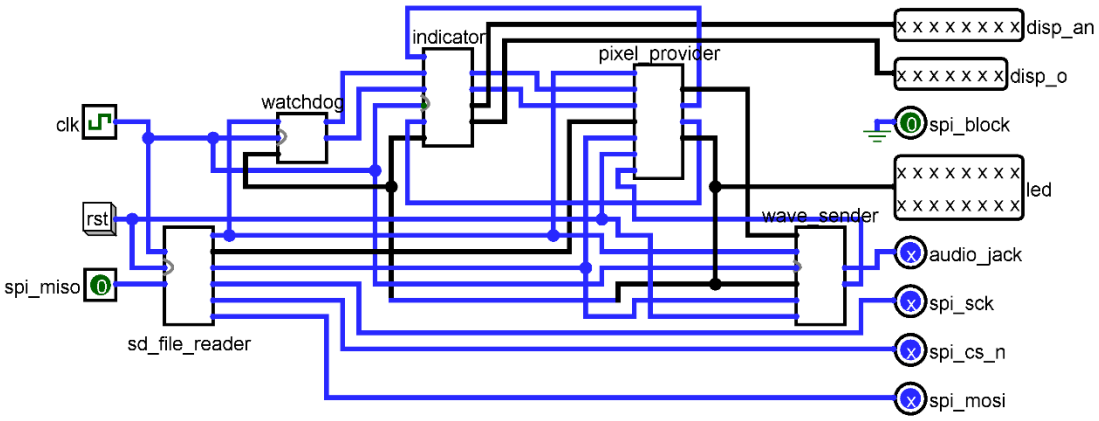
系统控制器的次态激励函数如下，可以根据这组方程设计出系统控制器的时序逻辑数字电路。

$$
\begin{aligned}
Q_0^n &= R\left(\overline{Y_0} Q_1^{n-1}(Y_2 + Y_1 Q_0^{n-1}) + Y_0 \overline{Q_0^{n-1}}\left(Y_2 + \overline{Q_1^{n-1}}\right)\right) \\
Q_1^n &= R(Q_1^{n-1} + Q_0^{n-1} Y_1) \\
Q_2^n &= R(Q_2^{n-1} + Y_3 Q_1^{n-1} Q_0^{n-1})
\end{aligned}
$$

整个系统的输入输出分别为：

- 带有 `spi_` 标记的五个接口分别经板载 SPI 总线连接到 SD 卡的触点上。
- `audio_jack` 接口连接到板载的 Sallen-Key 四阶巴特沃兹低通滤波器（Butterworth Low-pass 4th Order Filter）输入端上。
- `led` 接口分别连接到板载的 16 个 LED 灯上。
- 带有 `disp_` 标记的两个接口分别连接到板载的八位七段数码管控制器上。
- `clk` 是 100MHz 的系统时钟，`rst` 是带有“CPU reset”字样的复位按钮。

整个系统的逻辑方案图如下。

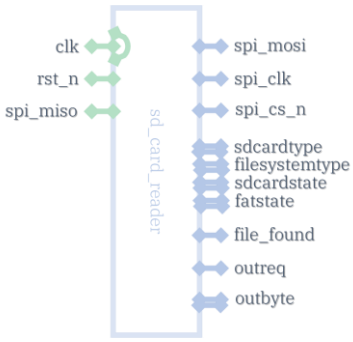


四. 子系统模块建模

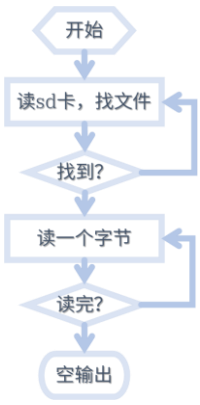
接下来分别是本实验设计的五个子系统的说明。限于篇幅，Verilog 源代码仅选取有说明价值的部分列出，完整的各模块源码已随报告一同提交。

1. `sd_file_reader`

接口定义



简略的功能框图



模块的输入包括时钟信号、复位信号和 *SPI 数据输入*信号。输出包括 *SPI 数据输出*、时钟、片选信号，四组调试信号，文件就绪信号，读取准备信号和待读出的数据信号。例化时，模块还接收两个参数，分别是待读取文件的名称、读写时钟基于主时钟的分频倍率。

为了降低实现难度，该模块使用了与其他部分不同的 System Verilog 语言实现，以利用这种语言数据类型完善、提供自动判断端口类型的 `logic` 关键字、支持通过循环语句写法缩短代码长度等优点。

在位文件被下载到 FPGA 上之后，板载的微控制器会重启 SD 插槽，将对总线的控制转交给程序。之后，FPGA 需要主动把 `SD_RESET` 信号保持为低电平，来给 microSD 卡插槽供电。

此模块与 SD 卡存储的数据交互是通过串行外设接口（SPI）协议[①]进行的，这是最简单的一种实现。基于 SPI 协议建立信道的部分逻辑在 `spi_session.sv` 中，如下所述。

> *基于 SPI 的通信要求有一个主设备和一个或多个从设备，在本实现中 FPGA 作为主设备，而 SD 作为从设备。*
>
> *主从设备之间通过至少 4 根线连接，分别是 SDI/MOSI (数据输入)、SDO/MISO(数据输出)、SCK/SCLK(时钟)、CS(片选)。主设备通过对 SCK 时钟线的控制，完成对通讯的控制。没有时钟跳变时，从设备就不采集或传送数据。*
>
> *SPI 通信有 4 种不同的实现模式，它们约定了时钟信号高低电平所对应的设备是空闲还是有效，数据是发送还是接收。本实现的工作模式是 CPHA（时钟相位）=1，CPOL（时钟极性）=1。即 SCLK 低电平有效，数据采集是在上升沿，数据发送是在下降沿。*

主机是通过由定义好的操作指令组成指令序列来控制 SD 卡内部状态机的。每个指令序列包括六段，分别是

- 2 位起始标志（一位起始位=`0` 和一位传输位=`1`）；
- 6 位命令索引；
- 32 位命令参数；
- 7 位 CRC7 校验码；
- 1 位结束标志。

以下状态图和流程图来自 SD 协会官网的技术规格文档[②]，说明了与 SD 卡在 SPI 模式下通讯的典型操作程序。这部分逻辑设计与 `sd_spi_sector_reader.sv` 相对应。

---

① Piyu Dhaker, Analog Devices: *AnalogDialogue:* ***Introduction to SPI Interface***, https://www.analog.com/media/en/analog-dialogue/volume-52/number-3/introduction-to-spi-interface.pdf

② SD Association: ***SD Specifications Part 1: Physical Layer Simplified Specification***, https://www.sdcard.org/downloads/pls/

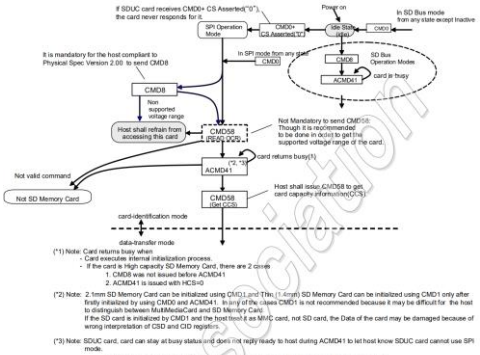


Figure 7-1 : SD Memory Card State Diagram (SPI mode)

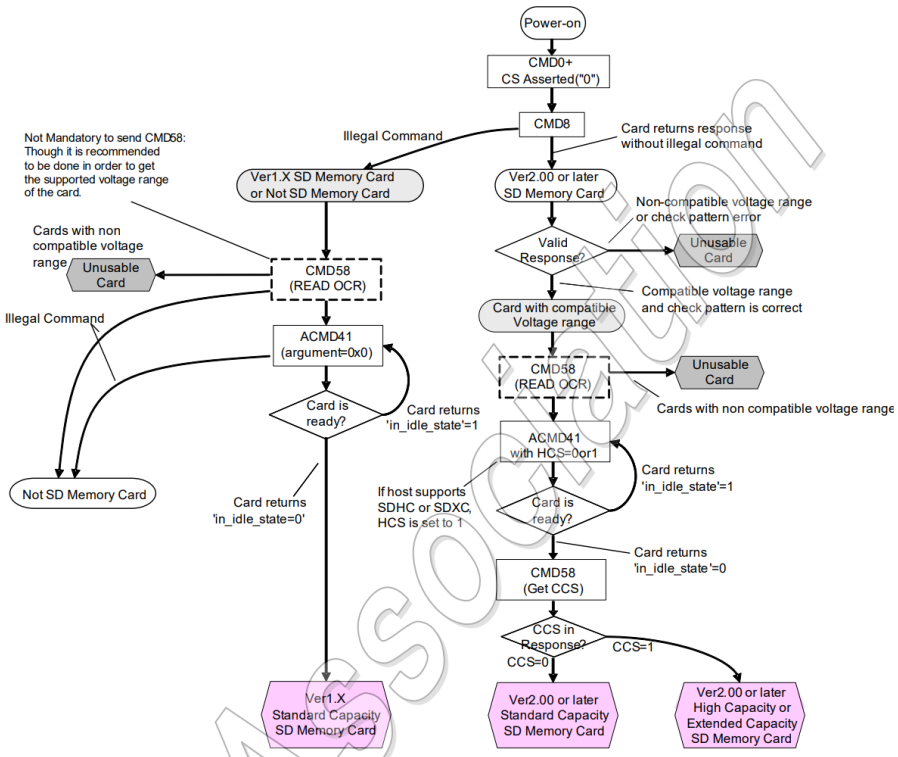


Figure 7-2 : SPI Mode Initialization Flow

SD 卡通电后，先要产生至少 *74* 个脉冲，让 *SD* 卡初始化完成。

之后循环发送 `CMD0` 复位指令，直到 *SD* 卡返回 `0x01`，表示状态就绪。在第一个 `CMD0` 复位信号到来时，由于 *SD* 卡上标号 *CS* 的触点未接通（低电位），*SD* 卡会自动进入 *SPI* 操作模式。

接下来发送 `CMD8` 指令，获取 *SD* 卡的版本信息。本模块未实现对 *1.0* 版本的支持，所以接下来的描述均假定 *SD* 卡版本是 *2.0* 以上的。

接下来发送 `ACMD41` 指令让卡片进入空闲状态，再发送 `CMD58` 指令鉴别卡的版本是标准版 *SDv2* 还是高速版 *SDHCv2*。

至此，*SD* 卡已准备就绪，可以开始读写其中内容了。*SD* 卡读单块和多块的命令分别为 `CMD17` 和 `CMD18`，它们的参数即要读的区域的开始地址。

> 因为考虑到一般 *SD* 卡的读写要求地址对齐，所以一般将地址转为块，并以扇区（块）（*512* 字节）为单位进行读写。

例子：本模块中 *SD* 卡初始化状态机的实现。

```
case(cardstate)
RESET : cardstate = CMD0;
CMD0  : if(cmdrsp==8'h01) cardstate = CMD8;
CMD1  : if(cmdrsp==8'h00) cardstate = IDLE;
CMD8  : if(cmdrsp==8'h01) begin  // SDv2.0
            if(cmdres[0+:8]==CMD8_VALID_RES)  // CMD8 success
                cardstate = ACMD41;
            else
                cardstate = CMD8FAILED;
        end else begin   // SDv1
            cardstate = CMD1;   // TODO: SDv1
            cardtype  = SDv1;
        end
ACMD41: if(cmdrsp==8'h01 && acmdrsp==8'h00) cardstate = CMD58;
CMD58 : if(~cmdrsp[7]) begin  // SDv2.0
            if((cmdres[3*8+:8]&8'hC0)==8'hC0)  // done initialize, SDHCv2
                cardtype  = SDHCv2;
            else                               // done initialize, SDv2
                cardtype  = SDv2;
            cardstate = CMD16;
        end
CMD16 :    if(cmdrsp==8'h00)  cardstate = IDLE;
IDLE  :    if( (~cmdrsp[7]) && rwrsp==8'hFE) begin cardstate = READING;            end
READING  : if( (~cmdrsp[7]) && rwrsp==8'hFE) begin cardstate = IDLE   ; done=1'b1; end
endcase
```

所有在外部存储器上的数据，都是以文件系统的形式组织的。出于实现简便考虑，本模块假定存储卡已经被格式化为 FAT 或 FAT32 文件系统[3]。FAT32 是 FAT 扩充了寻址空间的改进版，通常是同时提及的。这种文件系统实现简单，在磁盘操作系统（DOS）的时代就已经在软盘等介质上得到广泛应用，并且依然被今天的消费电子产品普遍支持。

读写 FAT 文件系统的逻辑设计位于模块本体 `sd_file_reader.sv` 中。

> *FAT* 文件系统通常与 *MBR* 磁盘分区表格式结合使用。
>
> 在 *MBR* 分区表格式下，存储器的头部区域保留有一个主引导记录区（*MBR*），它位于 `0` 柱面-`0` 磁道-`1` 扇区。通常它的大小是 *512* 字节，其中的前 *446* 字节

[3] Microsoft Corporation: ***Extensible Firmware Initiative FAT32 File System Specification*** - Microsoft Hardware White Paper, access from Florida State University https://www.cs.fsu.edu/~cop4610t/assignments/project3/spec/fatspec.pdf

> 保留给早期的旧式电脑用于储存开机引导程序，之后 64 字节保存了存储器的分区信息，最后是两个字节的结束标志 `0x55aa` 。
>
> 每个分区在分区表中都保存为一条记录。记录的第 1 个字节是分区活动标志，用来表示本分区是否为用来启动操作系统的分区。第 2,3,4 个字节是本分区开始位置的柱面、磁道、扇区编号。第 5 个字节是文件系统类型标记，典型取值例如 `06h`（FAT 文件系统），`0bh`（FAT32 文件系统），`07h`（NTFS 文件系统），`83h`（Linux 文件系统）等。第 6,7,8 字节是本分区结束位置的柱面、磁道、扇区编号。第 9-12 字节表示本分区之前已经用的扇区数，第 13-16 字节表示本分区的总扇区数，这两个数都按小端字节序编码保存。
>
> 分区内的数据存储方式即由文件系统决定。FAT32 文件系统将逻辑盘的空间划分为三部分，依次是引导区（BOOT 区）、文件分配表区（FAT 区）、数据区（DATA 区）。
>
> 引导区保存了该分区下每扇区字节数，每簇对应的扇区数等信息和供旧式电脑使用的启动引导记录。
>
> 文件系统对数据区的存储空间是按簇进行划分和管理的，簇是空间分配和回收的基本单位，即，一个文件总是占用若干个整簇，文件所使用的最后一簇剩余的空间就不再使用。文件分配表区保存两份互为备份的文件分配表，在其中按顺序依次记录了该盘各簇的使用情况。
>
> 根据引导记录区的数据找到根目录区，然后开始遍历根目录下的文件夹与文件，就可以根据文件之间的树状连接关系访问到任意文件。

解析文件路径、处理长文件名支持的逻辑位于 `dir_parser.sv` 中。

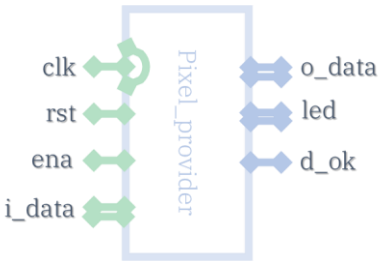
找到文件之后，就可以开始读取文件了。根据本项目的设计，本模块从头开始读取位图文件，并原样传送读到的字节。

例子：读取部分的实现。

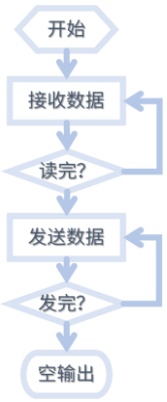
```
logic [31:0] fptr = 0;
initial {outreq,outbyte} = {1'b0,8'h0};

always @ (posedge clk or negedge rst_n)
    if(~rst_n) begin
        fptr = 0;
        {outreq,outbyte} = {1'b0,8'h0};
    end else begin
        if(rvalid && fat_state==READ_A_FILE && ~search_fat && fptr<file_size) begin
            fptr++;
            {outreq,outbyte} = {1'b1,rdata};
        end else
            {outreq,outbyte} = {1'b0,8'h0};
    end
```

2. `pixel_provider`

接口定义



简略的功能框图



模块的输入包括时钟信号、复位信号、使能信号和数据输入信号。输出包括数据输出信号、*LED* 信号、读取就绪信号。例化时，模块还接收三个参数，分别是图片的宽度、文件头长度、像素数据总数。

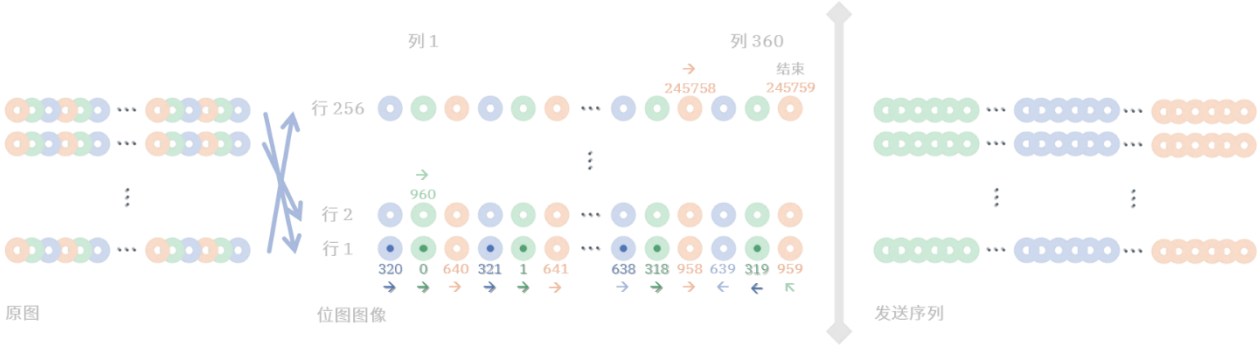
向该模块传送数据或从该模块获取数据的其他模块控制了该模块的使能信号，来让该模块的状态和外部同步。

和开发环境自带的 `Block Memory Generator` IP 核不同的是，本模块内部还集成了项目的状态逻辑和取像素时的数据定位逻辑。

本项目的对输入图片文件格式要求决定了所读取到数据的处理方式。24 位深的 PC 位图文件将图像上采样的每个像素点用红色、绿色、蓝色三个分量来表示，每个分量记录为一个字节，即取值为$\{(g,b,r)|\forall n\in\{r,g,b\}(n\in\mathbb{Z}\wedge n\in[0,256))\}$。图片的每一行内的像素点连续存储，图片中靠下的行存储靠前，每个像素占用连续的三个字节，先是蓝色分量、再是绿色分量、最后是红色分量。

接收数据时，数据被原样保存进元件内部的寄存器堆（实现中会被替换为动态随机存取存储器（DRAM））。

发送数据时，本模块会按照调制声波的需要先确定好取数据的位置，确保每次使能激活时模块输出的数据都是满足调制模块要求的正确数据。



具体地，调制声波的要求是将图像从左到右、从上到下地按行读取，每行先连续取出全部的绿色分量，再取出全部的蓝色分量，最后取出全部的红色分量（见信号产生模块部分的说明）。因此，在开始读取时，读取指针初始在存储区最靠后一行的第一个绿色分量位置。接着指针向前跳过三个字节取下一个像素的绿色分量，重复这一过程直到这一行所有的绿色分量读完为止。然后指针回到这一行的第一个蓝色分量位置读完，接下来读红色分量。这样，最后一行读完，指针回溯一行的长度，继续读下一行信息，并重复这个过程直到图片读完为止。

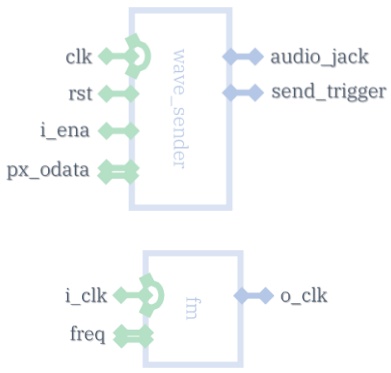
因为图像是从后往前读的，所以读到文件头部分时所有的像素已经传送完毕，设备已经进入待机状态，所以文件头不会被读出。

例子：提供像素数据部分的实现。

```
o_data = databuf[i];

if ((i + 3) % (scanline_width * 3) == 0)
    i = i - scanline_width * 3 + 5;
else if ((i + 2) % (scanline_width * 3) == 0)
    i = i - scanline_width * 3 + 2;
else if ((i + 1) % (scanline_width * 3) == 0)
    i = i - scanline_width * 3 + 2;
else
    i = i + 3;
```

3. `wave_sender`, `fm`

接口定义



简略的功能框图

信号产生模块的输入包括时钟信号、复位信号、使能信号和数据输入信号，输出包括音频输出信号、读取请求信号。调制模块是产生信号的核心部件，它的输入包括时钟信号、频率信号，输出是满足指定频率的波形信号。例化时，模块还接收两个参数，即图像的宽度、高度。

实际上调制模块的实现就是一个基于循环计数器的可调模数分频器，这正是 PDM 波的特点——生成简单，因此广泛用在广播发射设备的选型中。

调制后的信号是由许多段时长不同、频率确定的部分先后连接而成的。信号产生模块的任务就是循环判断接下来要发送多长时间、多少频率的信号。信号开始发送后，就激活数据提供模块准备好下一个字节信息，然后停机等待。信号发完后就再次判断并发送新的信号，直到图像传送完毕为止。

例子：信号状态转移逻辑的实现。

```
if (inlinepos == 0) begin
    linecur = linecur + 1;

    if (linecur >= scanline_num) begin
        done = 1;
    end
    else begin
        fm_diamter = 83333;  // 1200 Hz
        wait_counter = 486200; // 4.862 ms
    end
end
else begin
    fm_diamter = 100000000 / (px_odata * 800 / 255 + 1500);
    wait_counter = 45760;  // 0.4576 ms
    send_trigger = 1;
end

if ((inlinepos - 1) % scanline_width == 0) begin
    margin_trigger = 1;
end

inlinepos = (inlinepos + 1) % (3 * scanline_width + 1);
```

Martin M1 同步慢扫描系统[4]是 Martin Emmerson（电台代号 G3OQD）设计的。和更早期的 SC-1 等系统相比，它去除了冗余的同步信号，所以具有更高的信号质量。这种制式主要在欧洲被广泛使用。

以下内容来自加州伯克利大学 2014 春季数字信号处理课程（EE123）的大作业指导[5]，详细地说明了信号序列的组成。

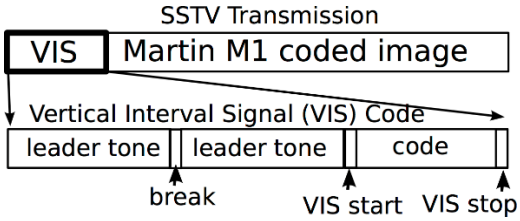
> ***VIS 代码***
>
> 所有标准的 *SSTV* 模式都使用一个独特的数字代码来让接收系统识别这种模式。这种代码称为 *VIS*（垂直间隔信号代码）。尽管通常整段校准标头信号都

[4] Martin Bruchanov (OK2MNM): *Image Communication on Short Waves: 4.* ***Formats of slow-scan TV transmission*** http://www.sstv-handbook.com/download/sstv_04.pdf

[5] Department of EECS at UC Berkeley: ***EE123: Digital Signal Processing Project information*** https://inst.eecs.berkeley.edu/~ee123/sp14/project.html

被叫做“*VIS* 代码”，但代码本身只是其中的一部分。七位代码首先传输最低有效位（*LSB*），遵循偶校验。*Martin M1* 的代码是 `44d`（十进制）。

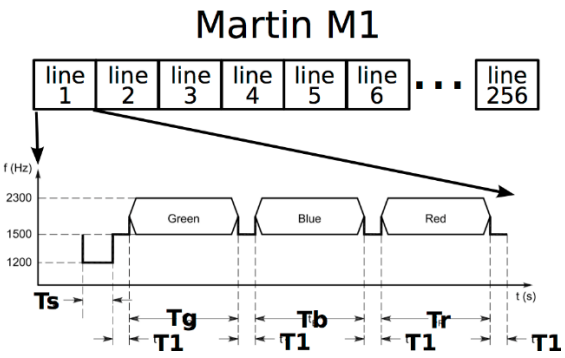
在发送图像信息前，需要先发送 *VIS* 数据包：



VIS 包的组成分别是 *300ms* 的 *1900Hz* 主音，*30ms* 的 *1200Hz* 间隔音，*300ms* 的另一个 *1900Hz* 主音，*30ms* 的 *1200Hz VIS* 起始位，*7* 个 *30ms* 的二进制位（*1100Hz="1", 1300Hz="0"*），*30ms* 的奇偶校验位（偶=*1300Hz*, 奇=*1100Hz*），*30ms* 的 *1200Hz VIS* 停止位。

***Martin M1* 模式：**

Martin M1 模式是相当直接的。它基于调频调制，用频率编码颜色分量的强度。彩色模式为 *RGB*，频率范围为 *1500-2300Hz*，用于编码亮度。一行扫描线有 *320* 像素，有 *256* 行。每行的扫描顺序由绿色、蓝色、然后是红色组成。下面是 *Martin M1* 的示意图：



每行以 *1200Hz* 的$T_s = 4.862ms$ 同步脉冲开始。接着是 *1500Hz* 的$T_1 = 0.572ms$ 间隔，*1500-2300Hz* 区间上的$T_g = 146.432$（*0.4576ms/*像素）绿色扫描行，*1500Hz* 的$T_1 = 0.572ms$ 间隔，*1500-2300Hz* 区间上的$T_b = 146.432$蓝色扫描行，*1500Hz* 的$T_1 = 0.572ms$ 间隔，*1500-2300Hz* 区间上的$T_r = 146.432$红色扫描行，*1500Hz* 的最后一个$T_1 = 0.572ms$ 间隔。总传输时间是 *114.3* 秒。

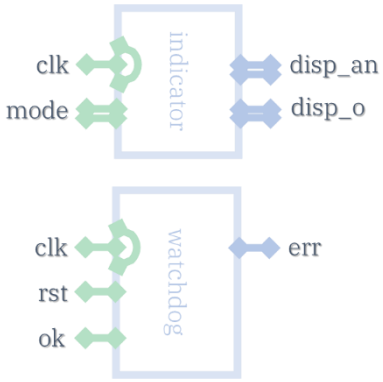
例子：*VIS* 包发送逻辑的实现。

```
if (vis_ptr == 0 || vis_ptr == 2) begin
    fm_diamter = 52631;      // 1900 Hz
    wait_counter = 30000000; // 300 ms
end
else if (vis_ptr == 1 || vis_ptr == 3 || vis_ptr == 12) begin
    fm_diamter = 83333;      // 1200 Hz
    wait_counter = 3000000;  // 30 ms
```

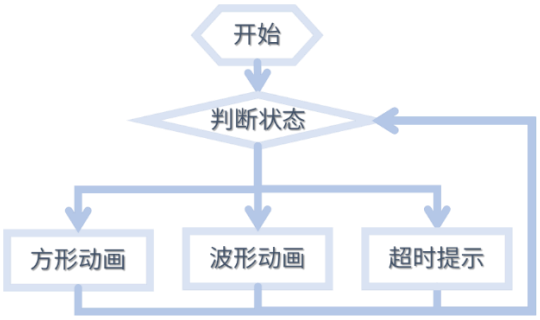
```
end
else if (vis_ptr >= 4 && vis_ptr <= 11) begin
    fm_diamter = martin_m1_id[vis_ptr - 4] ? 90909 : 76923;  // '0' = 1300 Hz, '1' = 1100
 Hz
    wait_counter = 3000000;  // 30 ms
end

vis_ptr = vis_ptr + 1;
```

4. `indicator`, `watchdog`

接口定义



简略的功能框图



指示器模块的输入包括时钟信号和模式选择信号，输出包括数码管使能信号和数码管图案信号。看门狗模块的输入包括时钟信号、复位信号、读写完成信号，输出是报错信号；例化时，模块还接收一个参数，即判断超时加载失败的延时长度。

八组七位数码管分别由八根正极线路控制开关，并且有七根控制所显示图案的线路作为负极同时连接八组数码管上。负极线路都是低电平时导通电路。而由于开发板使用了晶体管来给正极提供合适的电流，因此程序控制的正极线路也都是低电平有效。

模块通过在很短的时间内快速循环激活八组数码管并提供不同的图案信号，来达到让八个数码管同时显示出不同的图形的效果。

数码管共有三种显示：

- 移动的正方形（ ），
- 正弦波形 （ ），
- 读取超时提示（ ）。

动画的显示原理是使用一个计数器 `t`，每达到一定时钟间隔即增加计数器的值，进而更新屏幕显示的内容。

移动正方形的动画中，两个正方形先后分别从左侧的一个偏上和一个偏下的出发点出发，依相同的三次运动曲线 $x = \frac{(tick-8)^3+562}{125}$ 运动到右侧停止。这一曲线模拟了平滑的加速和减速过程。可以形象地向用户提示“加载”这一操作。

正弦波曲线动画基于以下等式。

$$\left[\sin\left(\frac{\pi}{3}t\right)\right]=\begin{cases}-1, & t \bmod 6 \in \{4,5\}\\ 0, & t \bmod 6 \in \{0,3\}\\ 1, & t \bmod 6 \in \{1,2\}\end{cases}$$

由上式易得动画的状态方程。

例子：正方形移动动画的实现。

```
wire [7:0] f, g;
assign f = ((t % 16 - 8) * (t % 16 - 8) * (t % 16 - 8) + 562) / 125;
assign g = (((t + 8) % 16 - 8) * ((t + 8) % 16 - 8) * ((t + 8) % 16 - 8) + 562) / 125;

// ...

{disp[7], disp[6], disp[5], disp[4], disp[3], disp[2], disp[1], disp[0]} <= {
    f == 0 ? 7'b0100011 : g == 0 ? 7'b0011100 : 7'b1111111,
    f == 1 ? 7'b0011100 : g == 1 ? 7'b0100011 : 7'b1111111,
    f == 2 ? 7'b0011100 : g == 2 ? 7'b0100011 : 7'b1111111,
    f == 3 ? 7'b0011100 : g == 3 ? 7'b0100011 : 7'b1111111,
    f == 4 ? 7'b0011100 : g == 4 ? 7'b0100011 : 7'b1111111,
    f == 5 ? 7'b0011100 : g == 5 ? 7'b0100011 : 7'b1111111,
    f == 6 ? 7'b0011100 : g == 6 ? 7'b0100011 : 7'b1111111,
    f == 7 ? 7'b0100011 : g == 7 ? 7'b0011100 : 7'b1111111
};
```

五. 测试模块建模

以下用到 testbench 文件进行测试的项目，仅列出测试过程块的代码。测试文件的其他部分是通过 VSCode 编辑器的 verilog-testbench-instance 插件脚本生成的。

5. `sd_file_reader` 测试

因为涉及到实际文件读写，因此若需要测试，直接在顶层模块中将本模块的调试输出连接到 led 灯输出上，然后下板观察指示灯即可。

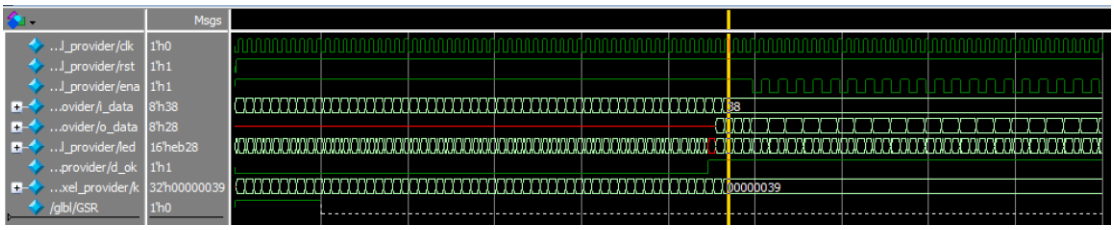
6. `pixel_provider` 测试

例化了一个参数为 `scanline_width = 6, size = 6 * 3 * 3;`的缩略版 `pixel_provider` 模块，周期性地写入数据，然后观察读取情况。

```
integer k;

initial
begin
    #(PERIOD/4)rst =    1;
```

```
            i_data =    0;

    for (k = 1; k <= 56; k = k + 1)
        #PERIOD i_data = k;

    #(PERIOD*70) $finish;
end

initial
begin
    #(PERIOD*239/4) ena = 0;
    forever #(PERIOD)  ena=~ena;
end
```

仿真波形如图，结果无误，符合预期。

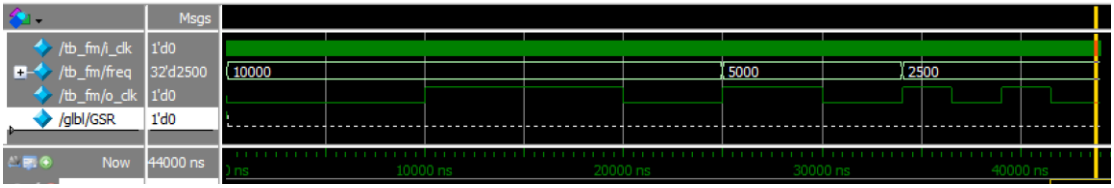


7. `fm` 测试

例化了一个 `fm` 模块，在不同时刻设定好不同的频率，观察输出波形。

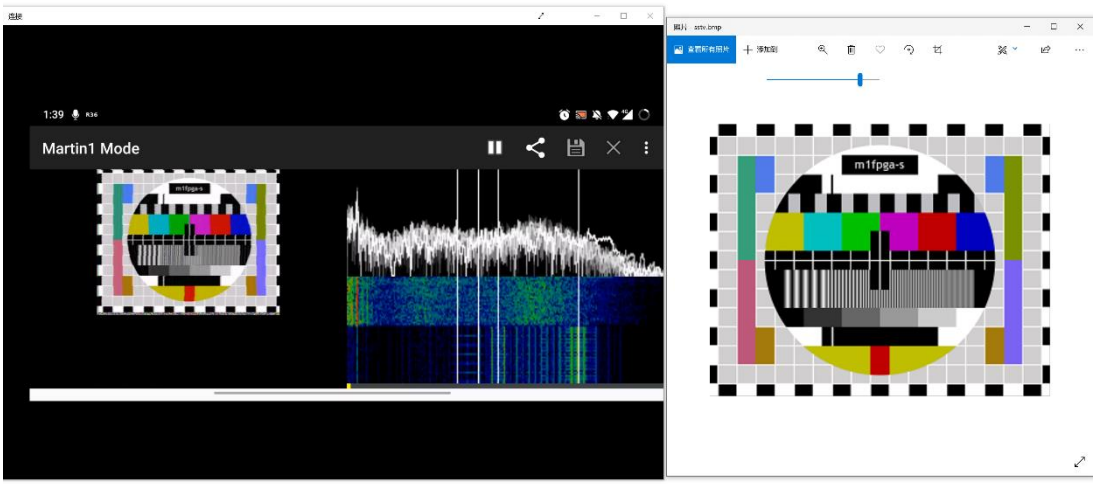
```
initial
begin
    freq = 10000;
    #(PERIOD * 12500) freq = 5000;
    #(PERIOD * 4500) freq = 2500;
    #(PERIOD * 5000) $finish;
end
```

结果无误。



8. `wave_sender` 测试

使用模拟图像信号调试常用的飞利浦 PM5544 测试卡进行收发实验，图像质量正常。



9. `indicator` 测试

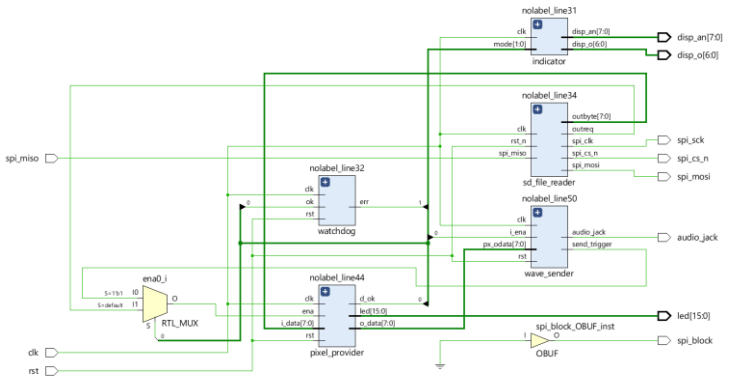
在 `indicator_test.v` 中，例化了一个 `indicator` 模块，周期性地切换显示图案。将该文件作为顶层模块，综合下板，即可对 `indicator` 的实现是否正确进行测试。

```
reg [9:0] span = 0;
reg [7:0] i = 0;

always @(clk) begin
    span = span + 1;
    if (span == 0) i = i + 1;
end

indicator(clk, i % 3, disp_an, disp_o);
```

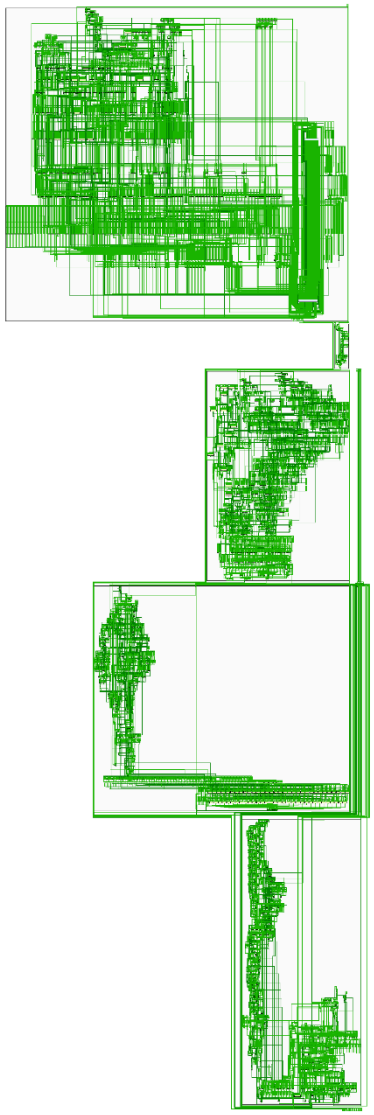
六. 实验结果

以下是本数字系统的寄存器转换级（RTL）电路逻辑图。

完整版已存为 schematic.pdf，随实验报告一同提交，可备参考。



右侧是系统实际实现的电路原理略图。



以下是综合日志给出的报告。

单元使用报告：

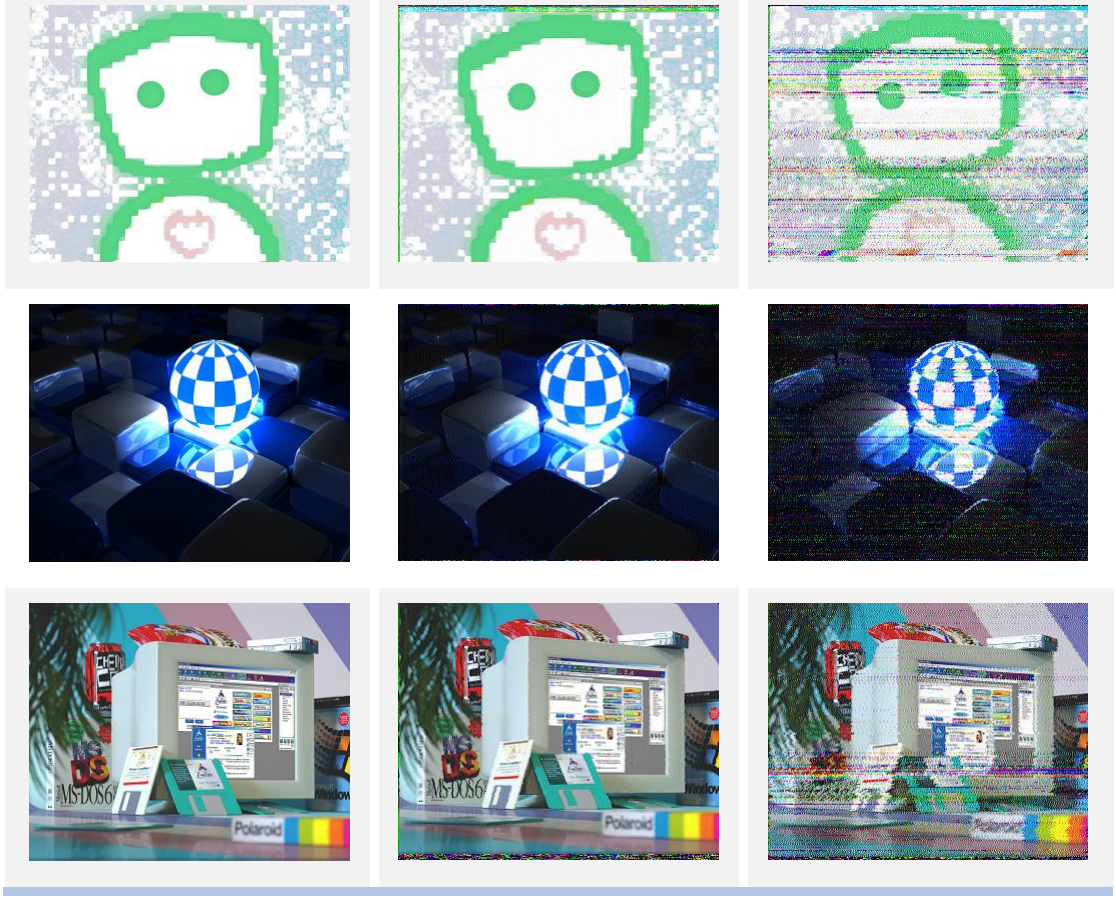
	Cell	Count
1	BUFG	1
2	CARRY4	598
3	DSP48E1	5
4	DSP48E1_1	13
5	DSP48E1_2	1
6	DSP48E1_3	1
7	DSP48E1_4	1
8	DSP48E1_5	3
9	LUT1	213
10	LUT2	882
11	LUT3	1051
12	LUT4	866
13	LUT5	773
14	LUT6	1156
15	MUXF7	7
16	RAMB36E1	32
17	RAMB36E1_1	32
18	FDCE	938
19	FDPE	13
20	FDRE	590
21	FDSE	31
22	IBUF	3
23	OBUF	36

实例区域报告：

	Instance	Module	Cells
1	top		7246
2	nolabel_line31	indicator	1592
3	nolabel_line32	watchdog	60
4	nolabel_line34	sd_file_reader	3431
5	root_dir_parser_inst	dir_parser	520
6	sd_sector_reader	sd_spi_sector_reader	1380
7	spi_session_for_sd_reader	spi_session	836
8	nolabel_line44	pixel_provider	905
9	nolabel_line50	wave_sender	1193
10	nolabel_line27	fm	111

下表是一些图片的发送实例。

原图	安静环境下接收效果	嘈杂环境下接收效果



七. 结论

在人类最早开始登月的上世纪中期，人们开始意识到通信事业和背后的信息技术隐藏的潜能。事实上，今日数字电路领域的一切，都还或多或少地体现着这个时代的印记。

1964 年，德州仪器发布了陶瓷半导体封装系列的最早一批原件，SN5400 系列；1966 年，更低成本的塑料半导体封装 SN7400 被推出，迅速占据了逻辑芯片市场的一半以上份额，最终成为了电子器件的事实标准。这就是今天依然广为人知的 7400 系列集成电路。

想象力并不会被技术条件限制。在更早的 1957 年到 1958 年，考普瑟·麦克唐纳（Copthorne "Coppie" MacDonald）研发了第一个使用摄像管和电子监视器的慢扫描电视系统，使用 120 行，120 列的分辨率在 3kHz 电话信道中传输黑白静态图片和音频信号。这是很直接又很理想化的想法：即使通信带宽很低，我们也能发送图像。

后来，SSTV 被用在登月的阿波罗计划中。更远的后来，信息时代的今天，SSTV 作为早期人们在通信技术领域探索的遗产之一，在无线电爱好者之间流传。

本次实验设计是将这项旧技术带进今天生活中的一个尝试。谨以本次作品，致以那个有着星辰大海理想的，我们不应忘记的却又已然逝去的时代——一个不到百年，技术却已经开始失传的时代。

八. 心得体会及建议

本次综合实验的完成目标是尽可能用简洁优雅的方式实现一个高可用性、功能完备的数字系统。

出于功能完备的考虑，技术选型上我们使用了今天依然在嵌入式开发等领域常用，而且依然在人们生活中随处可见的 FAT 文件系统和 BMP 位图格式。但因为这两者出现的历史条件距离今天也已经有一定距离了，它们到实现 SSTV 信号编码这一目标之间并没有复杂的处理过程、抽象层级或算力要求，这实际上相当程度地简化了设计，并且也没有给用户操作增加额外的复杂度成本。

本项目从开始编写代码到完成实验报告共花费了约一周时间。

开发和调试过程最主要的心得体会就是，硬件开发是相当需要耐心的一项工作。和软件开发中产品行为完全可预期、完善的错误处理使系统故障的影响范围被层层限制、相关开发实践高度成熟的条件相比，基于硬件的逻辑设计验证一次就要经过完整的综合下板步骤，耗费许多时间；错误现象不明显，需要敏锐的直觉判断出错概率最高的逻辑段；时序设计要求也更加精确。将这些种种问题都解决好的过程，对能力有相当程度的考验。

开发过程中遇到的错误。

但是，要让我们的设计达到高可用性，要做的实际上很简单，投入时间就够了。只要遇到错误，就修复。最后的某次修复之后，我们会忽然发现已经达到了错误被消灭一空的状态。

感谢开源项目 WangXuan95/FPGA-SDcard-Reader，提供了基于 FPGA 的 SD 卡读取的详细开发逻辑。本次项目将以 BSD 协议开源。

关于本课程的一些建议：

1. 课程助教可以主动联系一些课程进行中表现良好的同学，分享一些对本课程学习的心得体会，往往从学生的角度更能理解其他同学对于课程学习的困惑点在哪里。
2. 在满足教学需要的条件下，可以鼓励同学们使用新版的开发环境，这对开发效率有非常明显的提升。通常，新版开发环境在操作逻辑、报错信息、执行速度上相较旧版，都能得到更高程度的优化。有条件的话，可以以保证开发环境不占用同学设备过多资源为前提，向同学们提供三年内发行的较新版本的开发环境。

祝愿本课程越办越好。

2020 年 12 月 10 日