# 本文档部分内容在开发早期写完后没有更新，系统工作流程中寄存器的描述是最新的，其他的不是，不一定能对上最终版本的系统实现。

# CIS控制器

## 模块简介

内部一条RGB导光棒，一组线性CMOS感光元件，控制接口一个12Pin的FFC连接器。

\*\*一些重要参数\*\*：

- 扫描长度：218mm （18.3mm \* 4chips \* 3out）

- 分辨率：有三种模式：1200dpi，600dpi，300dpi（dpi：每英寸像素点个数，1inch=2.54cm）

- 总像素：

计算方式：每个感光元件的长度 \* 分辨率 \* 感光元件个数

- 1200dpi：10368（1.83 / 2.54 \* 1200 \* 12 = 864\*12chips）

- 600dpi：5184（1.83 / 2.54 \* 600 \* 12 = 432\*12chips）

- 300dpi：2592（1.83 / 2.54 \* 300 \* 12 = 216\*12chips）

- 时钟频率：典型值是5Mhz，最大值是10Mhz

- 信号输出通道：3通道输出（这里不太明白为什么是三通道？每个通道输出4个chip的数据吗）

- Vout输出电压：

- 亮电压范围：1.0V ~ 2.0V

- 暗电压范围：0.8V ~ 1.0V

- 时钟信号输入电平：3.3V（CP & SP）

- CIS模块输入电压：3.3V

- LED输入电压：5.0V

从手册的描述猜测，单个感光元件的感光长度应该是18.3mm，所以从整个扫描长度和物理尺寸计算，模块内部应该包含12个感光元件，即12chips。

## 接口引脚

|序号|符号|功能|

|:--:|:--:|:--:|

|1|GND|传感器接地端|

|2|VOUT1|输出通道1|

|3|VOUT2|输出通道2|

|4|VOUT3|输出通道3|

|5|GND|传感器接地端|

|6|VDD|传感器正电源输入（3.3V）|

|7|SP|起始信号输入|

|8|CP|时钟信号输入|

|9|LEDG|绿光控制端|

|10|LEDR|红光控制端|

|11|LEDB|蓝光控制端|

|12|VLED|光源电源输入（5.0V）|

从FPGA角度，要输出一个时钟信号CP给到CIS，同时还有控制信号SP，LEDG，LEDR和LEDB，控制模块启动扫描和打开灯光。CIS输出的VOUT是模拟信号，要通过一个ADC转为数字信号，输入到FPGA，所以FPGA还要采集从ADC输入进来的数据（多bit的数字信号），同时还要输出一路时钟给到ADC作为它的工作时钟。

## VOUT信号时序

![vout时序图](./picture/vout\_timing.png)

扫描一线的过程如图所示，前面第一部分10个周期为分辨率选择；第二部分65个周期为准备时钟；第三部分17个周期为参考电平（Vref）；整个工作周期为`N = 10+65+17+n+m`，`n = 3456/1725/864，m>=2`。

## 导光棒控制

LEDG\R\B可能不能直接控制灯的开关，好像还要在外部设计一个开关电路，用三极管做数字开关，LEDG\R\B连接到三极管集电极，真正的控制引脚连接到基极，控制三极管的打开和关闭，高电平打开，灯点亮，低电平关闭，灯熄灭。

### 三极管选择

要注意三极管参数，负载电阻选择。

### LED驱动电流

负载电阻的选择影响到最终LED的工作电流，手册给出两种工作电流：

- 53mA：此时led的控制需要用33%占空比的pwm控制

- 20mA（Green，Blue），25mA（Red）：此时用100%占空比信号控制

为方便控制，应该选择合适的负载电阻，让led驱动电流满足第二种工作电流。

## LED信号时序

![led时序图](./picture/led\_timing.png)

LED在SP输入后的第75或之后的时钟周期点亮，在下一个SP出现前的2个以上时钟周期关闭。

# 设计方案

硬件方面：在已有soc平台上，设计一个用于控制CIS，解析，处理并存储CIS返回数据的硬件模块，在顶层封装AXI4接口，以便其作为soc的外设接入soc的AXI4总线。

软件方面：需要编写驱动程序，使得应用程序可以通过软件驱动初始化，配置和启动模块。

## 系统结构框图

FPGA内部设计一个CIS控制模块，输出CIS模块需要的时钟和控制信号；还有一个ADC控制模块，输出时钟给ADC，同时采集，处理ADC输入的数字信号；另外还需要Bram存储解析得到的像素数据。

![系统结构图](./picture/system\_fram.png)

Scanner Controller模块内部设计如下所示：

![模块内部结构图](./picture/module\_fram.png)

几个主要模块的功能：

- port\_register：控制寄存器，存放模块的配置和状态信息，通过axi接口读写该寄存器，可以配置和启动模块。

- clk\_gen：需要产生一个5Mhz的CP时钟，以及一个25Mhz的adc时钟。（5倍采样）

- counter：计时器，计数CP周期数，以便在指定周期启动相应阶段的工作。

- reset\_ctr：产生各个模块的复位信号。

- adc\_ctr：接收，处理ADC输入进来的数据，然后输出到fifo缓存。

- led\_ctr：led控制器，在指定时刻打开和关闭led。

- fifo：缓存数据，当数据量达到突发传输size后自动发起一次axi memory写操作，将数据写进预留内存。

## ADC

### 需求分析

- 输入的模拟信号：有三路VOUT模拟信号需要转换，且VOUT信号的电压范围是0.8V ~ 2.0V，所以需要一个支持三路模拟输入的ADC，或者用三块支持单路模拟输入的ADC来并行完成三路信号的转换。

- VOUT信号频率：VOUT的输出与CP同步，CP的经典值是5Mhz，通过在一个周期内采样5个点做平均应该可以保证数据准确性，而如果一个周期采样5个点，那就要求ADC的转换速率 >= 25MSPS。

\*\*可选ADC型号\*\*：

AD9280：1路输入，8位输出，max转换速率32MSPS

\*\*ADC采样\*\*：

- 采样原理：电容充放电。

- 采样时间：外部源电阻大，则充电时间长，采样频率过快会导致未充满而出现测量误差。

\*\*转换速率\*\*：

转换速率是指完成一次从模拟转换到数字的AD转换所需要的时间的倒数。积分型AD的转换时间是毫秒级属低速AD，逐次比较型AD是微秒级属中速AD，全并行/串并行型AD可达到纳秒级。采样时间则是另外一个概念，是指两次转换的间隔。为了保证转换的正确完成，采样速率（Sample Rate）必须小于或等于转换速率 。因此有人习惯上将转换速率在数值上等同于采样速率也是可以接受的

### AD9280：

供电电压：2.7~5.5V

\*\*模式控制\*\*：

内部电路主要包括模拟输入电路，参考电压缓冲器电路，参考电压电路。

工作模式通过几个引脚控制：

- VREF

- VREFSENSE

- REFTS

- REFBS

- MODE

1. 参考电压（对应设置引脚：VREF，VREFSENSE）

- 内部电压源：1V或者2V（图16d，e）

- 外部电压源：添加外置电阻，分压，最后应该是1点多V（图16f）

设置的是VREF

2. 参考电压缓冲器（对应设置引脚：MODE，REFTS，REFBS）

- Center Span模式：输入电压AIN为以中心点（midscale）为中间电压的范围波动。

- Top/Bottom模式：输入电压AIN范围上/下限为REFTS和REFBS输入的电压值。

3. 模拟输入电路（对应设置引脚：MODE，REFTS，REFBS）

- 差分输入模式：AIN作为一端，REFTS和REFBS短接作为另一端。

- 单端输入模式：单端输入，输入范围REFBS~REFTS

设置的是输入AIN的范围

参考别人的设计时，简单来说：

- 要知道VRE参考电压，看REFSENSE，是地的话就是内部2V，与VREF短接的话就是内部1V。否则就是外部参考电压。

- 要知道什么模式，就看MODE，如果MODE是AVDD就是单端输入，同时REFTS和REFBS没有短接就是top/bottom模式。这时REFTS和REFBS决定了AIN的电压范围。

### 用信号发生器测试AD采集电路时的阻抗匹配问题：

信号发生器产生信号，用示波器/AD测量时，需要设置信号源的负载阻抗与示波器/AD的输入阻抗匹配（不是输出阻抗），信号发生器的输出阻抗是固定的，一般是50欧姆。设置负载阻抗是为了告诉信号发生器后级电路的输入阻抗是多大，以便产生合适电压。说到底是软件设置问题。

### 一些深层次的问题：

一般高速ADC阻抗小？因为需要让内部采样电容快速充电？

信号源输出阻抗影响到分压，还是影响驱动能力？驱动能力怎么定义？

低速adc输入阻抗很大，所以分压后可以认为等于输入的测量电压；而高速adc输入阻抗很小，这时信号源的输出阻抗的分压不能忽略？

- AD9280数据手册

- [野火AD9280\_AD9708模块](https://doc.embedfire.com/products/link/zh/latest/module/adc\_dac/ad9280\_ad9708.html)

- [ZYNQ-FPGA-AD\DA（高速）](https://blog.csdn.net/weixin\_48444982/article/details/134361722?spm=1001.2014.3001.5502)

- [FPGA实现AD9708和AD9280波形收发输出](https://blog.csdn.net/qq\_41667729/article/details/128888017)

- [FPGA模块——AD高速转换模块（并行输出转换的数据）](https://blog.csdn.net/Treasureljc/article/details/134471345?spm=1001.2014.3001.5502)

- [【嵌入式】测量值与数字信号发生器输出值形成2倍关系的原因](https://blog.csdn.net/spiremoon/article/details/107580320?spm=1001.2014.3001.5502)

## 系统工作流程

在CIS控制器的硬件设计时设置四个内部寄存器，用户程序可以通过读写寄存器，设置和启动CIS。

### 控制寄存器 ctr\_reg(slv\_reg0)

|Bits|Name|Default Value|Description|

|:--|:--|:--|:--|

|9-31|Reserved|0h|预留位|

|8|data\_ready|0h|数据就绪中断，硬件置1，软件清零。为1表示缓冲区数据已填满，可以读取。|

|4-7|Reserved|0h|预留位|

|3|finish|0h|扫描结束中断，硬件置1，软件清零|

|2|start|0h|启动标志位，软件置1，硬件自动清零|

|0-1|mode|0h|模式设置。0：300dpi；1：600dpi；2：1200dpi|

### 地址寄存器 addr\_reg(slv\_reg1)

指定一个ddr地址作为缓冲区的首地址，CIS模块扫描完后会将数据直接送到以addr\_reg指定的地址为首地址的ddr地址空间。

### 缓存大小寄存器 size\_reg(slv\_reg2)

指定缓冲区的大小，单位为字节。

### 行数寄存器 line\_num\_reg(slv\_reg3)

指定要扫描的行数

### 间隔时间寄存器 interval\_time\_reg (slv\_reg4)

一行扫描完到下一行开始扫描之间的间隔时间，准确来说是间隔的cp时钟周期数，需要等cis移动到下一行位置。与移动速度有关。

### 软件控制：

- 然后可以初始化CIS的配置参数，用户程序首先写控制寄存器，先设置`mode`以及地址寄存器，缓存大小寄存器和用于指定待扫描行数的行数寄存器。

- 然后将`start`位置1（必须先进行前面的设置），启动设备。

- 用mmap尝试映射驱动中维护的就绪队列，如有可用数据则将其通过socket传输出去，循环此步骤，直至mmap返回结束标志。

- 最后一定要close(fd)，否则可能导致存在未释放的dma内存。

# 问题

## Vout输出

- 为什么是三通道输出，每个通道输出4chips的数据吗，vout1输出1 ~ 4chips，vout2输出5 ~ 8chips，vout3输出8 ~ 12chips？即如果扫描是从左到右，则vout1输出的是左边73.2mm的像素数据，vout2输出中间73.2mm的像素数据，vout3输出最右边73.2mm的像素数据？

- 答：是的，三通道并行输出，每个通道负责输出一段数据。

## 工作模式

- 因为是彩色扫描，所以对于同一行像素是要扫描三次吗？按照手册figure 8的时序图，vout输出的结果会延迟一个扫描周期（N个CP周期）？

- 答：是的，要按照手册说明完成一行扫描，即每行要扫四遍(RGBR)。

---

# zynq 7020上的实验

## Vivado工程移植

- 要将CIS Controller模块移植到zynq 7020上运行，先在viavdo 2020.2版本上创建工程，添加zynq的ip，然后完成配置，并将cis模块添加进去，之后导出xsa文件，用于制作系统镜像。

- 这里用vivado 2020版本是为了跟petalinux的版本对应，得到xsa之后就可以不需要在vivado 2020版本上做开发了，可以用自己熟悉的更高版本的vivado来调试，综合，生成bit文件，最后将生成的bit文件放到sd启动卡里就行。注意后续用高版本vivado构建的工程中，zynq的配置，以及axi接口的连接，地址，时钟频率等要与前面生成xsa文件时的配置一样。

## 制作petalinux镜像

这里参考\*\*正点原子启明星的嵌入式linux开发指南\*\*，按照第20章：“搭建驱动开发使用的zynq镜像”的做法，制作系统镜像。各软件之间有版本匹配要求，开发指南的示例用的是vivado 2020，petalinux 2020和ubuntu 18.04。

### 创建petalinux工程

- 首先将vivado工程导出的xsa文件复制过来，然后用petalinux创建工程，导入xsa文件。

- 通过图像化界面配置修改系统镜像的加载位置为sd卡。

- 编译fsbl：`petalinux-build -c bootloader`

- 编译uboot：`petalinux-build -c u-boot`

- 生成BOOT.bin：`petalinux-package --boot --fsbl --u-boot --dtb no --force`，这里用dtb no使dtb不被打包进入BOOT.bin。

执行完上述步骤将会生成包含fsbl.elf和u-boot.elf的BOOT.bin文件，之后再修改linux驱动也不需要改动这个文件。

- 生成boot.scr脚本：`petalinux-build`编译整个工程，之后会生成boot.scr。

- 修改boot.scr，修改系统镜像的启动命令，添加.bit的启动命令，让uboot将.bit文件作为单独的文件加载进去。

### 编译内核

- 这里不用petalinux生成的linux镜像，选择自己下载一份xilinx的linux源码(正点原子的资料给了)，编译前先添加交叉编译工具链环境变量，即`source /opt/petalinux/2020.2/environment-setup-cortexa9t2hf-neon-xilinx-linux-gnueabi`，选用的系统版本是linux-xlnx-2020.2版本，内核版本是5.4.0。

- 之后是添加设备树文件，之前在编译uboot的时候生成了几个dts和dtsi文件，这是petalinux根据xsa文件产生的。

|文件名|内容|

|:--|:--|

|pcw.dtsi|vivado里使能的ps外设|

|zynq-7000.dtsi|zynq7系列的通用基础ps外设|

|pl.dtsi|PL端的外设，如果PL端有跟PS端通过AXI连接的模块，就会生成这个文件|

|system-conf.dtsi|包含一些系统配置，包括内存映射、中断控制器配置等系统相关外设置信息|

|system-user.dtsi|用户自定义或追加的设备树内容放在这里|

|system-top.dts|顶层设备树文件，包含了除system-conf.dtsi之外的其他几个dtsi文件|

- 将上述文件以及petalinux生成的system-user.dtsi复制到内核源码，然后修改该文件，修改后的内容如下所示。

```

///include/ "system-conf.dtsi"

#include <dt-bindings/gpio/gpio.h>

#include <dt-bindings/input/input.h>

#include <dt-bindings/media/xilinx-vip.h>

#include <dt-bindings/phy/phy.h>

/ {

model = "Alientek mini Zynq Development Board";

compatible = "xlnx,zynq-zc702", "xlnx,zynq-7000";

chosen {

bootargs = "console=ttyPS0,115200 earlycon root=/dev/mmcblk0p2 rw rootwait";

stdout-path = "serial0:115200n8";

};

};

&uart0 {

u-boot,dm-pre-reloc;

status = "okay";

};

&sdhci0 {

u-boot,dm-pre-reloc;

status = "okay";

};

&gem0 {

local-mac-address = [00 0a 35 00 8b 87];

phy-handle = <&ethernet\_phy>;

ethernet\_phy: ethernet-phy@7 { /\* yt8521 \*/

reg = <0x7>;

device\_type = "ethernet-phy";

};

};

```

- 修改内核源码的dts目录下Makefile文件，添加system-top.dtb编译目标。

- `make -j8`编译内核源码，生成zImage和dtb文件。

### 编译根文件系统

- 回到petalinux工程目录，执行`petalinux-config -c rootfs`和`petalinux-build -c rootfs`配置和编译根文件系统，这里可以保持默认配置。

- 编译后在image/linux得到rootfs.tar.gz

### 制作sd启动卡

将上面得到的BOOT.bin，boot.scr，zImage，system.dtb和vivado生成的PL的system.bit文件替换到sd卡的boot目录。根文件系统一般后续都不需要改动。

如需格式化sd卡，参考\*\*正点原子启明星的嵌入式linux开发指南\*\*第6.2.10节。

### 修改镜像

- 后续如果在做驱动开发的时候，在不需要修改vivado工程，单独修改设备树的情况下，只需要在linux源码里的system-user.dtsi文件修改，然后make dtbs重新编译设备树即可，不需要重新编译其他东西，也不需要重新编译内核。然后替换新的dtb文件到sd卡即可。

- 如果修改了vivado工程，这里分为两种情况：

1. 修改了arm内部的配置，或者修改了与arm核连接的axi接口地址，时钟等影响arm与外设通信有关的东西：在这种情况下，需要重新导出xsa文件，然后重新在petalinux工程里导入xsa文件。然后重新编译bootloader和uboot，生成新的BOOT.bin，因为修改了这些东西，设备树信息会跟着修改，设备树文件是编译uboot的时候生成的，而它又是根据xsa文件来生成设备树文件的，所以需要导入新的xsa。编译完后替换新生成的设备树文件到内核源码目录，然后重新编译dtb。最后将dtb，BOOT.bin和修改后的vivado工程生成的bit文件替换到sd卡中。

2. 只修改与arm核无关的东西：这种情况下，与arm核有关的设备树信息没有变化，所以不需要重新导入xsa。此时可能是修改了PL中的模块，如果需要修改对应模块的设备树信息，直接在内核源码的system-user.dtsi中修改，然后重新编译dtb即可。最后将dtb和修改后的vivado工程生成的bit文件替换到sd卡中。

参考链接：

- 《正点原子启明星ZYNQ之嵌入式linux开发指南》第20章：搭建驱动开发使用的ZYNQ镜像

- [Linux Reserved Memory 预留内存](https://blog.csdn.net/zhenglie110/article/details/101671786)

## 设备驱动

## 注意：这是早期版本的驱动实现，最终版本的驱动不是用预留内存的方案实现的。！！！！！

CIS Controller硬件的行为可以通过端口寄存器控制，而CIS扫描的结果数据会通过控制器直接写入到DDR的指定地址中，所以驱动的任务需要能够让用户读写axi-lite的控制接口相关寄存器，同时要能让用户读取为CIS准备的预留内存空间。

### 实现方案：

\*\*需求分析\*\*：

- 第一个需求通过编写正常的字符型设备驱动就可以实现。

- 第二个需求首先需要修改设备树，添加预留内存节点，然后在驱动程序中通过`memremap`或者`ioremap`映射预留内存到内核空间，再然后通过`write/read`文件操作集接口来支持用户对预留内存的读写，但是这样的方法在读写时需要先进入内核空间，然后在内核空间进行操作，每次都需要将数据在用户空间和内核空间之间进行复制，这种方式效率低，不适合大数据量的操作。更好的方案是在驱动中实现mmap，将设备地址（这里的预留内存就相当于外部设备）映射到用户的虚拟地址空间。

\*\*具体实现\*\*：

\*\*修改设备树\*\*：

在内核源码目录的`arch/arm/boot/dts/system-user.dtsi`文件中添加预留内存节点：

```

/{

......

reserved-memory {

#address-cells = <1>;

#size-cells = <1>;

ranges;

reserved:buffer@0x38000000 {

no-map;

reg = <0x38000000 0x2400000>;

};

};

.......

}

```

然后重新编译设备树：

```

make dtbs

```

将生成的新的dtb文件替换到sd卡。

\*\*重载mmap\*\*：

- \*\*虚拟地址空间\*\*：操作系统每个进程都有4G的虚拟空间，其中3G为用户空间，另外1G是内核空间，但每个进程的用户空间是独立的，内核空间是所有进程共享的，所以运行在内核空间的驱动程序是面向所有进程的。

- \*\*内存映射\*\*：而设备也有自己的硬件地址空间，用户程序无法直接访问，需要通过内存映射，将设备的物理地址映射到进程的虚拟地址空间，就能让进程像操作内存一样操作设备。常见的映射函数有`ioremap()以及它的变体memremap()`等，但这些函数是将设备地址映射到内核空间，只能在内核态里操作，当需要操作的数据量较大时，需要频繁的在内核态和用户态之间进行切换和数据交换，执行效率低。

- \*\*mmap\*\*：用来将一个文件或其他对象映射到进程虚拟地址空间的系统调用，在映射文件时，是将文件磁盘某段地址与进程某段虚拟地址进行映射，此时进程就可以像操作内存一样操作文件，系统会自动将写入的内容同步到磁盘文件中，这样就完成了对文件的操作而不用调用read，write等系统调用。当mmap的fd参数为-1时，表示进行的是匿名映射，而如果传入的是一个文件的话，则mmap会帮我们创建一块指定大小的虚拟空间后，调用fd文件的文件操作集中指定的.mmap函数（即调用file->f\_op->mmap()）。

- \*\*驱动程序实现mmap\*\*：用户程序调用的mmap函数最终会调用驱动程序的mmap()，所以只需要在驱动程序重载mmap，将设备物理地址与用户进程的虚拟地址建立映射，就可以在用户进程中直接通过读写mmap返回的虚拟地址来操作实际的设备内存。

由于驱动程序是编译成.ko文件的，没有编进内核，所以不需要重新编译内核，直接在系统启动之后再加载进去即可。

```

insmod scanner.ko

mknod /dev/scanner c 200 0

chmod 666 /dev/scanner

```

参考链接：

- [Linux Reserved Memory 预留内存](https://blog.csdn.net/zhenglie110/article/details/101671786)

- [韦东山：Linux驱动程序基石之mmap](https://cloud.tencent.com/developer/article/1708985)

- [Linux驱动mmap内存映射详解及例子实现](https://blog.csdn.net/weixin\_42096901/article/details/103227930)

- [从内核世界透视 mmap 内存映射的本质（原理篇）](https://www.cnblogs.com/binlovetech/p/17712761.html)

- [linux内核内存管理：Memory (re)mapping](https://www.cnblogs.com/wanglouxiaozi/p/15036469.html)

- 《正点原子启明星ZYNQ之嵌入式linux开发指南》第21章：字符驱动设备开发

### 网卡MAC地址的问题

- 最后同时使用多块板子时发现，按照正点原子教程做出来的系统的网卡MAC地址都是一样的，且不是设备树中定义的那个，因为被uboot中的设置覆盖了。因此，如果多个板子同时工作，网络会出现冲突，即使为每块板子设置不同的IP。解决方法是需要为每块板子设置唯一的MAC地址，可以修改uboot，在系统启动的时候进入uboot，然后使用以下命令修改:

```

// 查看是否设置了MAC

printenv ethaddr

// 修改变量，并保存

setenv ethaddr 00:0a:35:00:1e:60

saveenv

```