上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

数字逻辑设计

PROJ – 图像格式转换模块

（Bayer 格式转 RGB 格式）

姓名 ： 邱一航

学号 ： 520030910155

1. 电路原理与模块功能要求

彩色图像的生成需要在每个像素点上同时采集 RGB 三个通道的信号，即在图像传感器端使用 R、G、B 三种滤镜。若要同时采集三个信号则需要在同一个像素处集成三块可调节的滤镜。为了方便制造，常常在采集时采用 Bayer 格式，即每个像素点处只设置一块滤镜。

由于人眼对绿色附近的可见光最为敏感，因此在 Bayer 格式中，几乎一半的像素点采用绿色滤镜，采用红色和蓝色滤镜的像素点各占四分之一左右。Bayer 格式以 2×2 像素为一个基本单元，一个单元的四个像素点中两个采用 G 滤镜（且两个像素点为对角线关系），其余两个分别采用 R 滤镜和 B 滤镜。以下是四种 Bayer 格式的示意图，分别为 RGGB、GBRG、GRBG、BGGR。

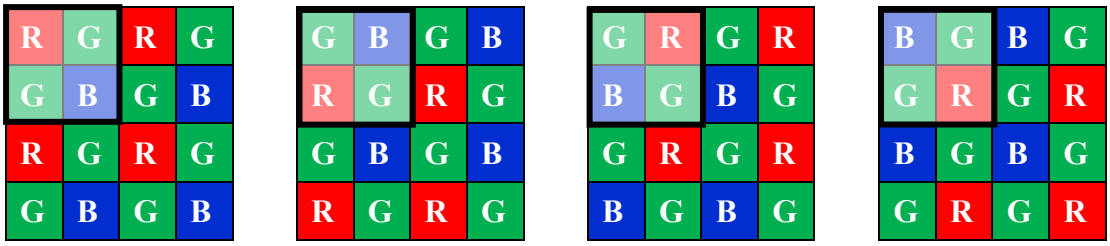


图 1　四种 Bayer 格式示意图

本项目中需要实现的模块的功能是将特定尺寸特定 Bayer 格式（以上四种之一）的图片转化为三通道的 RGB 格式。转化过程使用插值的方式，即在 2×2 的基本单元内，所有像素点的红色通道值均为 Bayer 格式中红色滤镜像素点的值，所有像素点的蓝色通道值均为 Bayer 格式中的蓝色滤镜像素点的值，所有像素点的绿色通道值均为 Bayer 格式中两个绿色滤镜像素点的平均值。

笔者分配到的图像格式为 BGGR，BGGR-Bayer 格式到 RGB 格式的转化过程如下。

R11	R12	R13	R14
R21	R22	R23	R24
R31	R32	R33	R34
R41	R42	R43	R44

G11	G12	G13	G14
G21	G22	G32	G42
G31	G32	G33	G43
G41	G42	G43	G44

B11	B12	B13	B14
B21	B22	B23	B24
B31	B32	B33	B34
B41	B42	B43	B44

图 2　BGGR-Bayer 格式到 RGB 格式的转化

以左上角的第一个 2×2 基本单元为例。BGGR-Bayer 格式下，原图中 11 保存蓝色通道值（B11），12、21 保存绿色通道值（G12、G21），22 保存红色通道值（R22）。因此，通过插值方式转化后的 RGB 格式下，左上角的第一个 2×2 基本单元的三通道值分别为：

$$\begin{cases} R_{11} = R_{22} \\ R_{12} = R_{22} \\ R_{21} = R_{22} \\ R_{22} = R_{22} \end{cases} \quad \begin{cases} G_{11} = (G_{12} + G_{21})/2 \\ G_{12} = (G_{12} + G_{21})/2 \\ G_{21} = (G_{12} + G_{21})/2 \\ G_{22} = (G_{12} + G_{21})/2 \end{cases} \quad \begin{cases} B_{11} = B_{11} \\ B_{12} = B_{11} \\ B_{21} = B_{11} \\ B_{22} = B_{11} \end{cases}$$

项目中已经给出了 SRAM 的实现和 testbench（包括对 Bayer 格式文件 img.txt 的读取），Bayer 格式图片的扫描方式与 SRAM 的输入输出如下图所示。

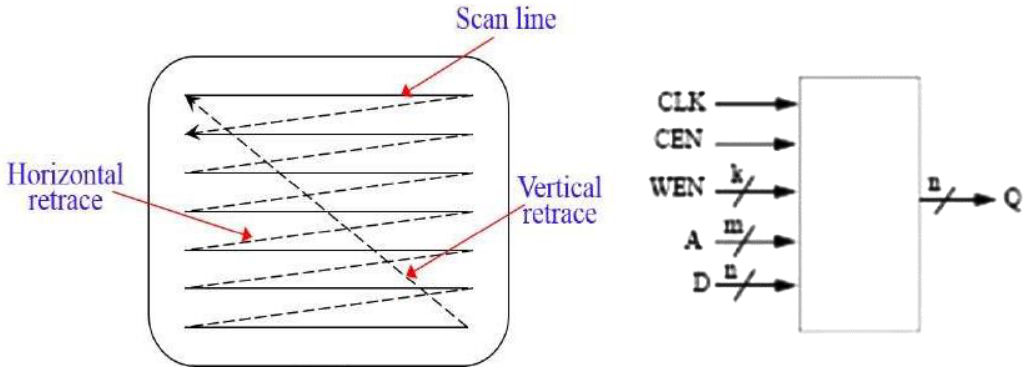


图 3　图片扫描顺序与 SRAM 输入输出

项目中已经给定了 Bayer2RGB 模块的输入输出端口定义，该模块共有三个输入、九个输出，分别如下：

端口名称	I/O	位宽	说明
clk	input	1 bit	系统时钟信号
rst_n	input	1 bit	系统复位信号（低电平有效，异步）
data_in	input	8 bits	系统输入数据，即 SRAM 的输出数据
cen	output	1 bit	SRAM 的片选信号，低电平有效
wen	output	1 bit	SRAM 的写使能信号， 为高电平表示读取数据，为低电平表示载入数据
addr	output	20 bits	SRAM 读写地址（SRAM 将在下一个时钟上升沿返回该地址的数据）
data_out	output	8 bits	需要载入 SRAM 的数据（本次项目中未使用）
start	output	1 bit	开始信号，计算用时（本次项目中禁止改变）
O_RGB_data_valid	output	1 bit	输出有效信号，为高电平时输出有效
O_RGB_data_R	output	8 bits	R 通道数值
O_RGB_data_G	output	8 bits	G 通道数值
O_RGB_data_B	output	8 bits	B 通道数值

表 1　Bayer2RGB 模块的端口定义

本项目的 testbench 已经给定，在接受到 O_RGB_valid 为高电平时，将 O_RGB_data_R、O_RGB_data_G、O_RGB_data_B 的数值输出到三个通道的对应文件中，在 O_RGB_valid 为高电平 row*column（输入图片的尺寸，row 为行数，column 为列数）个时钟周期之后，关闭文件，完成格式转换后结果的输出。

2. 设计思路

笔者实现了两种不同的版本的 Bayer2RGB 模块：第一种是模块内部只有少量寄存器、不使用寄存器数组暂存中间结果的版本，功耗较低但转换格式所需的时钟周期数相对较多，即“低功耗 Bayer2RGB”；第二种在模块内部使用寄存器数组暂存中间结果，功耗相对较高但转换格式所需的时钟周期数更少，即“快速 Bayer2RGB”。

换言之，“低功耗 Bayer2RGB”使用存储器容量很少，但完成格式转换所需的时钟周期数较多；“快速 Bayer2RGB”使用存储器容量较多，但完成格式转换所需的时钟周期数较少。

两种不同版本的 Bayer2RGB 模块的实现思路将在本章节的两个小节中具体阐述。

为方便表述，我们认为输入图片的尺寸是 row×column，行的下标为 0~(row-1)，列的下标为 0~(column-1)。

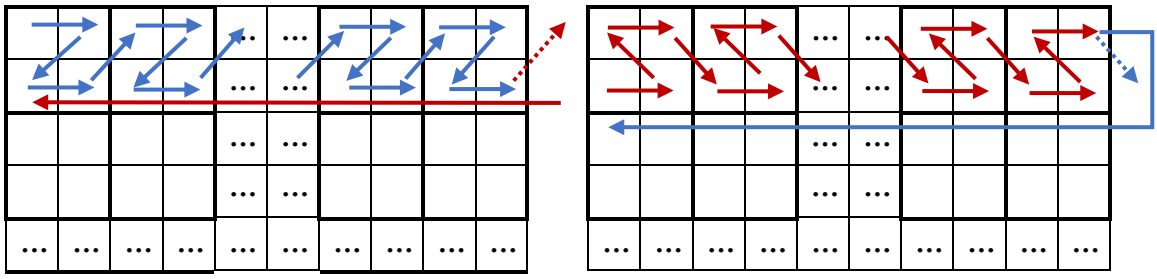
2.1 低功耗 Bayer2RGB

为了降低 Bayer2RGB 模块的功耗，该版本的模块中，我们不使用任何寄存器数组暂存中间结果，仅使用少数寄存器用于处理和暂存。

根据要求，我们需要按照图 3 左图的顺序输出整张图片中每个像素的 RGB 三通道数值。我们按照这一顺序逐一计算每个像素点的 R、G、B 三通道值。首先通过 Bayer2RGB 模块的 addr 输出端口输出计算当前像素点所需的四个像素点的地址，从 SRAM 处通过 data_in 端口获得这四个像素点的数值后，进行计算得到当前像素点的 RGB 三通道值；计算完毕后，将 O_RGB_data_valid 端置为高电平（输出有效）并完成当前像素点的 RGB 三通道输出。

注意到每个 2×2 的基本单元内，四个像素点的 R 通道完全相同、G 通道完全相同的、B 通道完全相同；因此我们实际上只需要计算偶数列（0,2,…,column-2）的像素点的 RGB 通道值，接着保持有效输出两个时钟周期，就完成了两个相邻像素点的 RGB 值输出。

同时，可以发现奇数行和偶数行的处理方式不同（计算对应的像素点时，所需要的四个像素点与该像素点的对应位置不同）。因此我们设计了以下的“扫描”顺序，该顺序能很好地完成从奇数行的“扫描”模式跳转到偶数行的“扫描”模式，而不需要做太多的细节处理。



(a) 偶数行的扫描模式　　(b) 奇数行的扫描模式

图 4　低功耗 Bayer2RGB 模块的 SRAM 扫描模式说明

（可以看出，偶数行的最后一次扫描到的像素点恰好是奇数行扫描的最开始应被扫描到的像素点，奇数行的最后一次扫描的像素点恰好是偶数行最开始应被扫描到的像素点。）

由此，对低功耗 Bayer2RGB 模块，我们设计了以下的 15 个状态和状态转移表。

注意每次从 SRAM 获取数据都需要经过两个时钟周期（第一个时钟上升沿输出对应的 addr，第二个时钟周期上升沿时 SRAM 获取到 addr 输出的地址，在第二个时钟周期内输出对应的数据到 data_in，第三个时钟上升沿时才能使用 data_in 的数值并对其进行处理）。因此，通过 data_in 输入模块的实际上是两个时钟周期前的 addr 地址对应的数值。

Reset、Hold、End 三个状态只在某一次工作开始时或结束时会使用。

此外，除了到 Reset 状态的转移是异步的以外，所有状态的转移都发生在时钟上升沿。

State	Output	Description	Input		Next State
			rst_n	clk	
Reset	无输出	**重置状态** (addr = 0，第一个 2×2 单元的左上像素地址)	**0**	**x**	**Reset**
	无输出		**1**	↑	**Hold**

Hold	无输出	**准备状态** (addr = 1，第一个 2×2 单元的右上像素地址) (count = 1, line=0)	**1**	↑	**S0000**
S0000	无输出	**正常工作状态，偶数行-0** (当前 2×2 单元的左下像素，获得当前单元左上像素即 B 通道) addr = addr+column-1 pixel_b = data_in	**1**	↑	**S0001**
S0001	无输出	**正常工作状态，偶数行-1** (当前 2×2 单元的右下像素，获得当前单元右上像素即 G 通道) addr = addr+1 pixel_g = data_in（暂存）	**1**	↑	**S0010**
S0010	无输出	**正常工作状态，偶数行-2** (下一 2×2 单元的左上像素，获得当前单元左下像素即 G 通道) addr = addr-column+1 pixel_g = (pixel_g+data_in)/2	**1**	↑	**S0011**
S0011	输出 RGB 三通道 pixel_r, pixel_g, pixel_b	**正常工作状态，偶数行-3** (下一 2×2 单元的右上像素，获得当前单元右下像素即 R 通道) addr = addr+1 pixel_r = data_in count = count+1	**1**	↑	**S01xx（偶数行-3’）** (当前行未扫描完，即 count≤column-2) **S01xx（偶数行-3”）** (当前行已经扫描完，即 count>column-2)
S01xx	输出 RGB 三通道 pixel_r, pixel_g, pixel_b	**正常工作状态，偶数行-3’** (同一 2×2 单元内输出完全一致) count = count+1	**1**	↑	**S0000**
		正常工作状态，偶数行-3” (同一 2×2 单元内输出完全一致) count = 1, line=line+1	**1**	↑	**S1000**
S1000	无输出	**正常工作状态，奇数行-0** (当前 2×2 单元的左上像素，获得当前单元左下像素即 G 通道) addr = addr-column-1 pixel_g = data_in（暂存）	**1**	↑	**S1001**

S1001	无输出	**正常工作状态，奇数行-1** (当前 2×2 单元的右上像素，获得当前单元右下像素即 R 通道) addr = addr+1 pixel_r = data_in	**1**	↑	**S1010**
S1010	无输出	**正常工作状态，奇数行-2** (下一 2×2 单元的左下像素，获得当前单元左上像素即 B 通道) addr = addr+column+1 pixel_b = data_in	**1**	↑	**S1011**
S1011	输出 RGB 三通道 pixel_r, pixel_g, pixel_b	**正常工作状态，奇数行-3** (下一 2×2 单元的右下像素，获得当前单元右上像素即 G 通道) addr = addr+1 pixel_g = pixel_g+data_in count = count+1	**1**	↑	**S11xx（奇数行-3'）** (当前行未扫描完，即 count≤column-2)
					S11xx（奇数行-3"） (当前行已经扫描完，即 count>column-2)
S11xx	输出 RGB 三通道 pixel_r, pixel_g, pixel_b	**正常工作状态，奇数行-3'** (同一 2×2 单元内输出完全一致) count = count+1	**1**	↑	**S1000**
		正常工作状态，奇数行-3" (同一 2×2 单元内输出完全一致) count = 1, line = line+1	**1**	↑	**S0000** (当前行并非最后一行，即 line≤row-1)
			1	↑	**End** (当前行是最后一行，即 line==row)
End	无输出	**工作完成状态** 转换工作已经完成	**1**	↑	**End**
任意状态	见对应状态	见对应状态	**0**	**x**	**Reset**

表 2　低功耗 Bayer2RGB 模块的状态与状态转移表

注意在奇数行、偶数行模式切换时，虽然 addr 提前 data_in 两个时钟周期，但由图 4 可见，根据前一行模式提前扫描的后两个像素点正好是切换模式后应该扫描的两个像素点。因此不需要对 addr 提前进行特别处理。

此外，设计 Hold 状态的主要原因是 data_in 输入的完成相对 addr 的输出落后两个时钟周期，即使在 Reset 状态中预先让 addr 开始工作，要进入到正常的工作状态依然需要再单独让 addr 变化一个时钟周期（同时其余部分不工作），才能使 addr 正常地提前两个时钟周期。

特别地，**S01xx** 和 **S11xx** 实际上分别是两个不同的状态（**S01xx** 是 **S01xx（偶数行-3'）**和 **S01xx（偶数行-3"）**两个状态合并后的状态，**S11xx** 是 **S11xx（奇数行-3'）**和 **S11xx（奇数行-3"）**两个状态合并后的状态）。因此，低功耗 Bayer2RGB 模块实际上总共有 15 个状态。

但由于这两个状态的工作情况只相差两个赋值语句（count 的赋值以及 line = line+1），因此在实现时笔者将两个状态合并为一个带有 if 语句的状态来实现，在下面的状态转换图中也合并用同一个状态来表示。

低功耗 Bayer2RGB 模块的状态转换图如下：

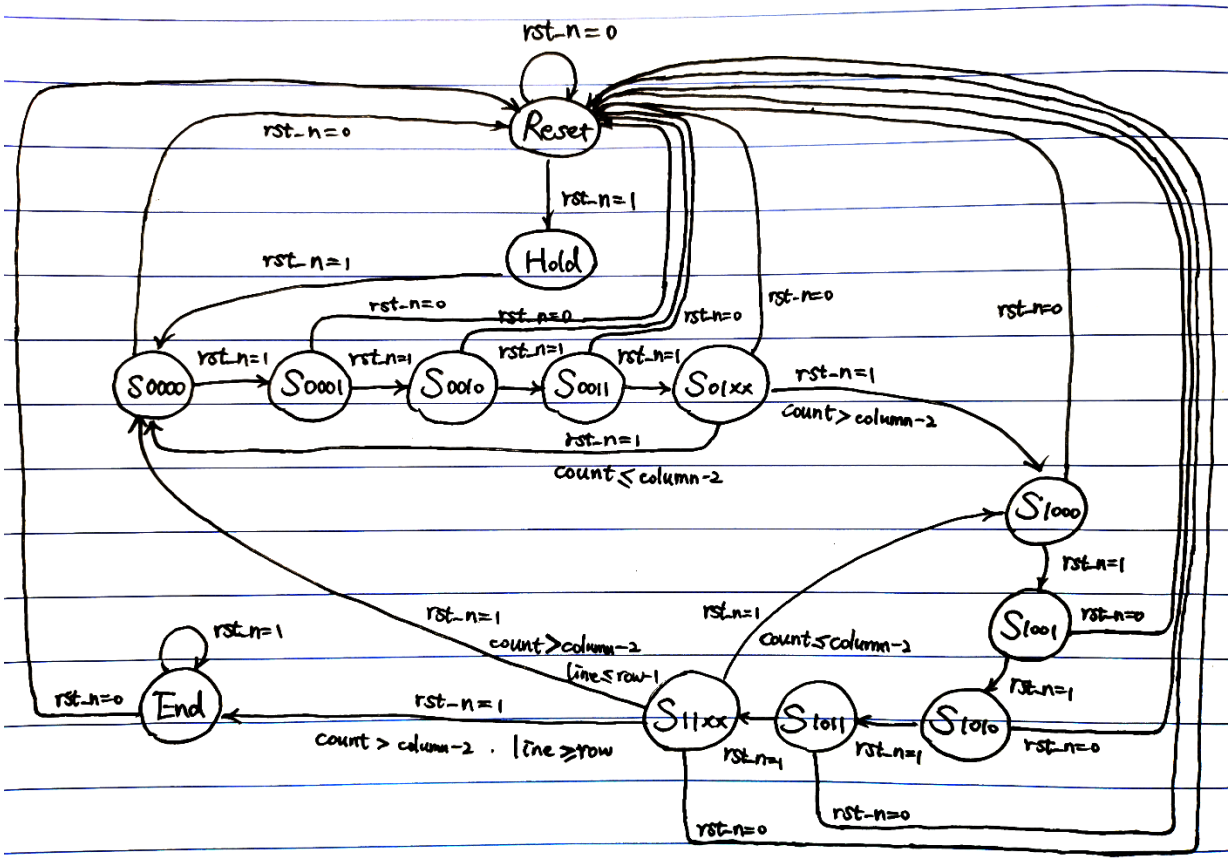


图 5　低功耗 Bayer2RGB 模块的状态转换图（**其中 rst_n=0 均为异步转换**）

为了减少加法器的数量，笔者对 addr 的改变进行了一些优化。可以发现偶数行-1、3 和奇数行-1、3 工作状态中，addr 的操作都是累加，而偶数行-0、2 和偶数行-0、2 对 addr 的操作也有类似之处。优化后 addr 的改变如下：

State	addr
S0000、S1010	**addr = addr+(column-1)+State[3]<<1**
S0010、S1000	**addr = addr-(column-1)-State[3]<<1**
S0001、S0011、S1001、S1011	**addr = addr+1**

表 3　低功耗 Bayer2RGB 模块的加法器使用优化

此外，我们发现经常会使用到 column-1 和 column-2 两个数值。为了减少加法器的使用，我们将两者作为 `parameter` 进行定义（在程序中分别为 `column` 和 `precolumn`）。

考虑到右移实际上就是除以 2，因此实际实现中 pixel_g 为 9 位存储器，我们选择高 8 位作为 G 通道的输出。

该版本的模块的实现详见 `Bayer2RGB_fewer_regs.v`。其中，不同的状态由 `mask`、`pattern`、`fetch_count`（2 位）和 `hold` 变量来表示。

请注意程序中 `rows`、`column` 为行数-1 和列数-1。

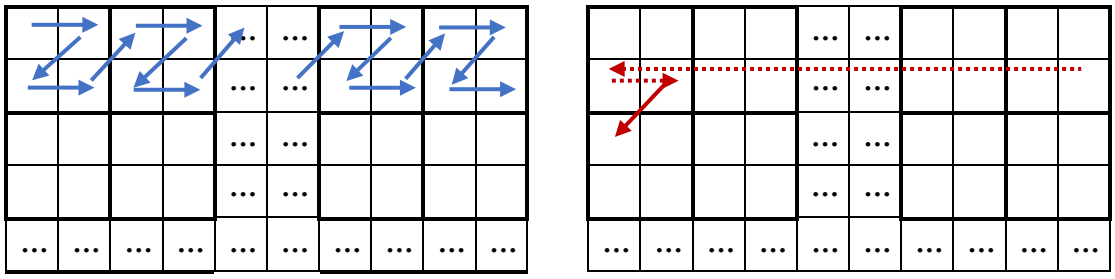
2.2 快速 Bayer2RGB

注意到对每个 2×2 基本单元的四个像素点的四个像素点的 R 通道完全相同、G 通道完全相同的、B 通道完全相同。因此，处理完偶数行像素的 R、G、B 三通道值的同时，我们已经完成了下一行像素 R、G、B 三通道值的计算。因此，只要在低功耗 Bayer2RGB 模块的基础上，使用三个寄存器分别存储当前行的 R、G、B 通道值，就可以避免重复计算，直接输出奇数行像素点的 R、G、B 三通道值。

下面我们用 r_[]、g_[]、b_[]来表示三个用于存储当前行像素点 R、G、B 三个通道值的寄存器数组。

由于同一个 2×2 基本单元中四个像素点的 RGB 三通道完全相同，偶数列与右侧的奇数列的 R、G、B 通道是完全一致的，所以实际上只需要三个 column/2 大小的数组就足够分别存储某一行全部像素点的 R、G、B 通道值。对下一行第 count 列的像素点，其 R、G、B 通道的值为 r_[count/2]、g_[count/2]和 b_[count/2]。

在这一版本中，我们只需要在偶数行的时候对 Bayer 格式的原图进行“扫描”，因此只需要采用 **2.1** 中的偶数行扫描模式。



(a) 扫描模式　　(b) 下一行扫描开始前

图 6　快速 Bayer2RGB 模块的 SRAM 扫描模式说明

其中，实线为会被使用到的 SRAM 读取，虚线为在准备下一次的行扫描时的“副产物”（由于 addr 提前 data_in 两个时钟周期，因此不可避免地会产生这样的“未使用”的 addr 变化），根据虚线变化的 addr 读取的 SRAM 数据并不会被我们使用。

注意在处理奇数行（根据寄存器暂存的结果直接输出）的最后需要提前改变 addr。这是因为 addr 总是需要提前 data_in 的使用两个时钟周期。

对低功耗 Bayer2RGB 模块的状态及状态转移表稍作修改（删去原本的奇数行处理，修改奇数行处理的几个状态），即可得到快速 Bayer2RGB 模块的状态及状态转移表。

类似低功耗 Bayer2RGB 模块，由于 addr 相对 data_in 的使用总是需要提前两个时钟周期，因此在快速 Bayer2RGB 模块中也需要引入 Hold 状态来确保 addr 提前两个时钟周期完成模块的输出。

下表中变量说明：

- line, count: 当前已经输出到 line 行 count 列的像素点的 RGB 通道值。其中，line 的范围为 0≤line≤row-1, 0≤count≤column-1。
- pixel_r, pixel_g, pixel_b: 当前正在处理的像素点的 R、G、B 通道值。
- r_[], g_[], b_[]: 暂存当前行每一个像素点所在的 2×2 基本单元的 R、G、B 通道值。因此实际上 r[count>>1]、g_[count>>1]、b_[count>>1]存储的是第 line 行（和第(line+1)行）第 count 列像素点的 R、G、B 通道值。

快速 Bayer2RGB 模块的状态（共计 12 个状态）及状态转移表如下。

State	Output	Description	Input		Next State
			rst_n	clk	
Reset	无输出	**重置状态** (addr = 0，第一个 2×2 单元的左上像素地址)	**0**	**x**	**Reset**
	无输出		**1**	↑	**Hold**
Hold	无输出	**准备状态** (addr = 1，第一个 2×2 单元的右上像素地址) (count = 1, line=0)	**1**	↑	**S0000**
S0000	无输出	**正常工作状态，偶数行-0** (当前 2×2 单元的左下像素，获得当前单元左上像素即 B 通道) addr = addr+column-1 pixel_b = data_in 保存至 b_[count>>1]	**1**	↑	**S0001**
S0001	无输出	**正常工作状态，偶数行-1** (当前 2×2 单元的右下像素，获得当前单元右上像素即 G 通道) addr = addr+1 pixel_g = data_in（暂存）	**1**	↑	**S0010**
S0010	无输出	**正常工作状态，偶数行-2** (下一 2×2 单元的左上像素，获得当前单元左下像素即 G 通道) addr = addr-column+1 pixel_g = (pixel_g+data_in)/2 保存至 g_[count>>1]	**1**	↑	**S0011**
S0011	输出 RGB 三通道 pixel_r, pixel_g, pixel_b	**正常工作状态，偶数行-3** (下一 2×2 单元的右上像素，获得当前单元右下像素即 R 通道) addr = addr+1 pixel_r = data_in count = count+1 保存至 b_[count>>1]	**1**	↑	**S01x0** (当前行未扫描完，即 count≤column-2) **S01x1** (当前行已经扫描完，即 count>column-2)
S01x0	输出 RGB 三通道 pixel_r, pixel_g, pixel_b	**正常工作状态，偶数行-3'** count = count+1	**1**	↑	**S0000**
S01x1		**正常工作状态，偶数行-3"** count = 0, line=line+1	**1**	↑	**S1x00**

S1x00	输出 RGB 三通道 r_[count<<1], g_[count<<1], b_[count<<1]	**正常工作状态，奇数行** count = count+1	**1**	↑	**S1x00** (当前行未输出完,即 count＜column-2)
					S1x01 (应准备下一行扫描, 即 count≥column-2)
S1x01		**正常工作状态，准备偶数行** count = count+1 addr = addr+column-1	**1**	↑	**S1x10**
S1x10		**正常工作状态，准备偶数行** count = 1 addr = addr+1 line = line+1	**1**	↑	**S0000** (当前行并非最后一行，即 line≤row-1)
			1	↑	**End** (当前行是最后一行，即 line==row)
End	无输出	**工作完成状态** 转换工作已经完成	**1**	↑	**End**
任意状态	见对应状态	见对应状态	**0**	**x**	**Reset**

表 4　快速 Bayer2RGB 模块的状态与状态转移表

其中，**S01x0** 和 **S01x1** 状态的工作非常相近，因此实际实现时合并为一个带有 if 语句的状态（**S01xx**）。**S1x00**、**S1x01**、**S1x10** 三个状态的工作也非常相近，因此实际实现时合并为一个含有多个 if 语句的状态（**S1xxx**）。

快速 Bayer2RGB 模块的状态转换图如下。

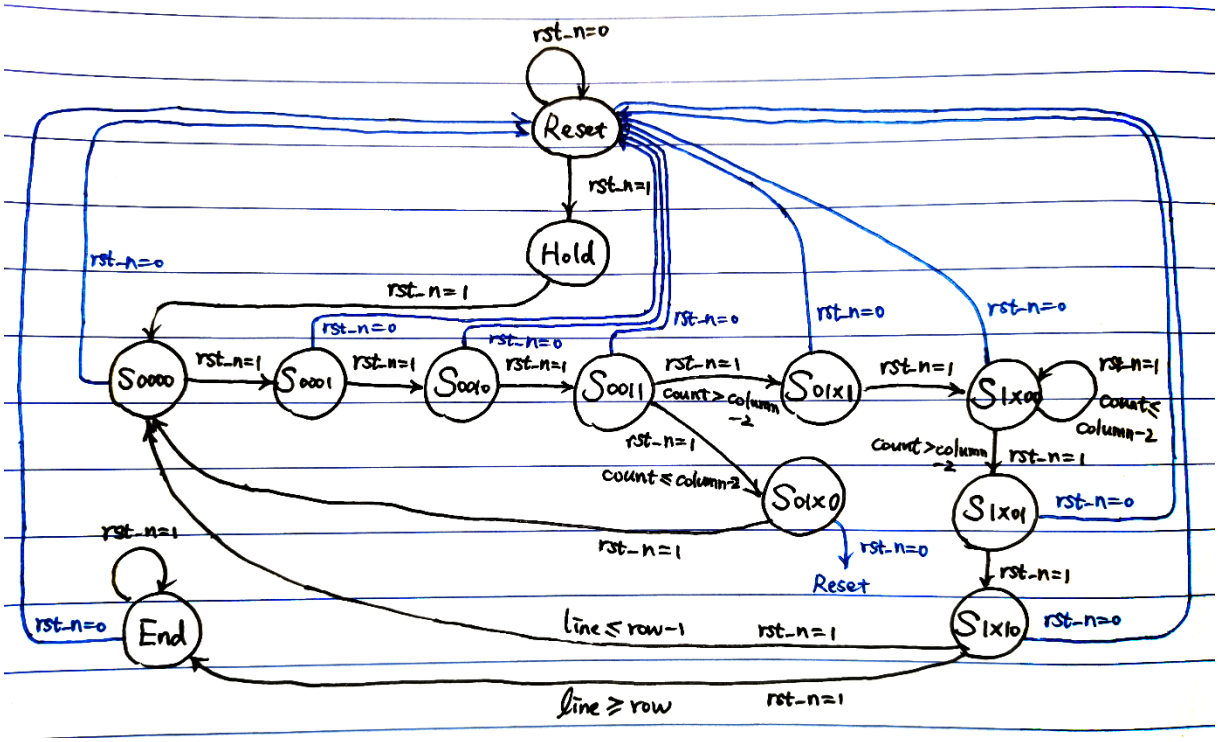


图 7　快速 Bayer2RGB 模块的状态转换图（其中蓝线均为异步转换）

为了进一步优化，减少加法器的数量，笔者在实现快速 Bayer2RGB 模块时，将 addr 的改变根据不同状态进行了状态融合。

State	addr
S0000	**addr = addr+(column-1)**
S0010	**addr = addr-(column-1)**
S0001、S0011	**addr = addr+1**

表 5　快速 Bayer2RGB 模块的加法器使用优化

此外，我们发现经常会使用到 column-1 和 column-2 两个数值。为了减少加法器的使用，我们将两者作为 `parameter` 进行定义（在程序中分别为 `column` 和 `precolumn`）。

快速 Bayer2GB 模块的实现详见 `Bayer2RGB_fewer_clks.v`。默认的 `Bayer2RGB.v` 也是使用了快速 Bayer2RGB 版本的 Bayer2RGB 模块。

在实际实现中，考虑到右移一位就是除以 2，因此 pixel_g 实际上是两个 G 通道数值的相加（一个 9 位的二进制数），其高 8 位作为当前像素的 G 通道，存储时也只保存 pixel_g 的高八位。这样处理能够变相地实现除以 2。

程序中，不同的状态由 `mask`、`pattern`、`fetch_count`（2 位）和 `hold` 变量来表示。

请注意程序中 `rows`、`column` 为行数-1 和列数-1。

2.3 细节处理

Bayer2RGB 模块的复位信号是异步信号，因此在复位信号的下降沿（从有效到复位），需要将输出信号置为无效（O_RGB_data_valid=0），同时将模块内部的计数器 `count`、`mask`、`pattern`、`fetch_count`、`line` 全部清零。

由于两种版本的 Bayer2RGB 模块的实现都不可避免地涉及到 SRAM 的读取，因此需要考虑模块输出（作为 SRAM 输入的几个输出端口，即 cen、wen、data_out、addr）的合法性。为了防止模块的输出 addr 越过合法范围（0~row*column-1），笔者在模块中特别增加对 addr 合法性的判断。如果 addr 越界，则将 cen 置为 1（无效），阻止对 SRAM 的非法访问。

同时，考虑到功耗问题，在不需要 SRAM 工作时，cen 应置为 1 以减少 SRAM 的能耗。

总结而言，对 cen 的值，应作以下处理：

1） 系统在复位状态下时，绝对不需要访问 SRAM。因此，在 rst_n（复位信号）的下降沿处将 cen 置为 1。

2） 系统完成工作之后时，也绝对不需要访问 SRAM。因此在 **End** 状态时，cen 置为 1。

3） 当 addr 越过合法范围时，为了避免非法访问 SRAM，应将 cen 置为 1。

4） 对**低功耗 Bayer2RGB 模块**：

可以发现 addr 从状态 **S0011** 之后到 **S0000** 或 **S1000** 之前、从 **S1011** 之后到 **S1000** 或 **S0000** 之前都不再变化，因此对这两个过程，分别至少有一个时钟周期不需要 SRAM 工作（即使工作也是做重复冗余的任务）。显然在 addr 没有变化的状态 **S01xx** 和 **S11xx** 处，SRAM 并不需要工作。

因此，在状态 **S0011** 处和 **S1011** 处可以将 cen 置为 1，在状态 **S01xx** 处和 **S11xx** 处再将 cen 重新置为 0。

5） 对**快速 Bayer2RGB 模块**：

a) 发现 addr 从状态 **S0011** 之后到状态 **S0000** 或 **S1x00** 之前不再变化。因此在状态 **S01x0** 和 **S01x1** 处 SRAM 不需要工作。因此，在状态 **S0011** 处将 cen 置为 1，在 **S01x0** 处重新将 cen 置为 0。

b) 当模块在输出奇数行像素点的 R、G、B 通道值时，SRAM 也不需要工作。因此，在状态 **S01x1** 处 cen 置为 1，直到状态 **S1x01** 处 cen 重新置为 0（addr 改变，需要重新从 SRAM 处读取数据）。

附件中，两个版本的 Bayer2RGB 模块都已经做了上述优化。

3. 结果分析

使用 ModelSim 2020.3 进行仿真模拟。

笔者分配到的图像为 **id=31** 的图片，图片尺寸为 296×246 pixels。

为了判断模块的输出是否正确，笔者使用 python 写了标准程序 **evaluate.py**（附在附件中）。该程序读取 **img.txt** 后产生 **std_R.txt、std_G.txt、std_B.txt** 三个标准输出文件。

3.1 低功耗 Bayer2RGB

在测试平台上仿真 2000μs。仿真波形保存在附件中 wave/wave_fewer_regs.do 和 wave/Bayer2RGB_fewer_regs.wlf 中。得到如下输出：

```
VSIM 4 > run
# the image Bayer TO RGB is done!!!
#
# the cost time is :      182041
```

将模块的输出 Bayer2RGB_R.txt、Bayer2RGB_G.txt 和 Bayer2RGB_B.txt 与标准输出 std_R.txt、std_G.txt、std_B.txt 对比，结果如下。

```
D:\Textbooks\2021-2022-2\Digital Logic Design\Project\1_pics\31>fc Bayer2RGB_B.txt std_B.txt
正在比较文件 Bayer2RGB_B.txt 和 STD_B.TXT
FC: 找不到差异


D:\Textbooks\2021-2022-2\Digital Logic Design\Project\1_pics\31>fc Bayer2RGB_R.txt std_R.txt
正在比较文件 Bayer2RGB_R.txt 和 STD_R.TXT
FC: 找不到差异


D:\Textbooks\2021-2022-2\Digital Logic Design\Project\1_pics\31>fc Bayer2RGB_G.txt std_G.txt
正在比较文件 Bayer2RGB_G.txt 和 STD_G.TXT
FC: 找不到差异
```

图 8　低功耗 Bayer2RGB 模块的输出评判

因此，低功耗 Bayer2RGB 的输出是正确的。转换后的图像可视化如下：

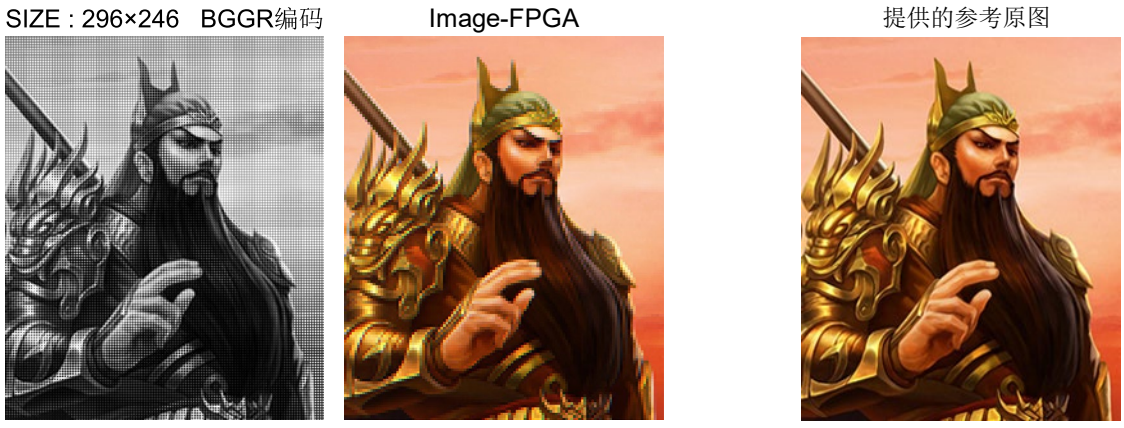


图 9　低功耗 Bayer2RGB 模块格式转换前后的图像及与原图的对比

3.2 快速 Bayer2RGB

在测试平台上仿真 1500μs。仿真波形保存在附件中 wave/wave_fewer_clks.do 和 wave/Bayer2RGB_fewer_clks.wlf 中。得到如下输出：

```
VSIM 9 > run
# the image Bayer TO RGB is done!!!
#
# the cost time is :      127429
```

将模块的输出 Bayer2RGB_R.txt、Bayer2RGB_G.txt 和 Bayer2RGB_B.txt 与标准输出 std_R.txt、std_G.txt、std_B.txt 对比，结果如下。

```
D:\Textbooks\2021-2022-2\Digital Logic Design\Project\1_pics\31>fc Bayer2RGB_B.txt std_B.txt
正在比较文件 Bayer2RGB_B.txt 和 STD_B.TXT
FC: 找不到差异


D:\Textbooks\2021-2022-2\Digital Logic Design\Project\1_pics\31>fc Bayer2RGB_R.txt std_R.txt
正在比较文件 Bayer2RGB_R.txt 和 STD_R.TXT
FC: 找不到差异


D:\Textbooks\2021-2022-2\Digital Logic Design\Project\1_pics\31>fc Bayer2RGB_G.txt std_G.txt
正在比较文件 Bayer2RGB_G.txt 和 STD_G.TXT
FC: 找不到差异
```

图 10　快速 Bayer2RGB 模块的输出评判

因此，快速 Bayer2RGB 的输出是正确的。转换后的图像可视化如下：

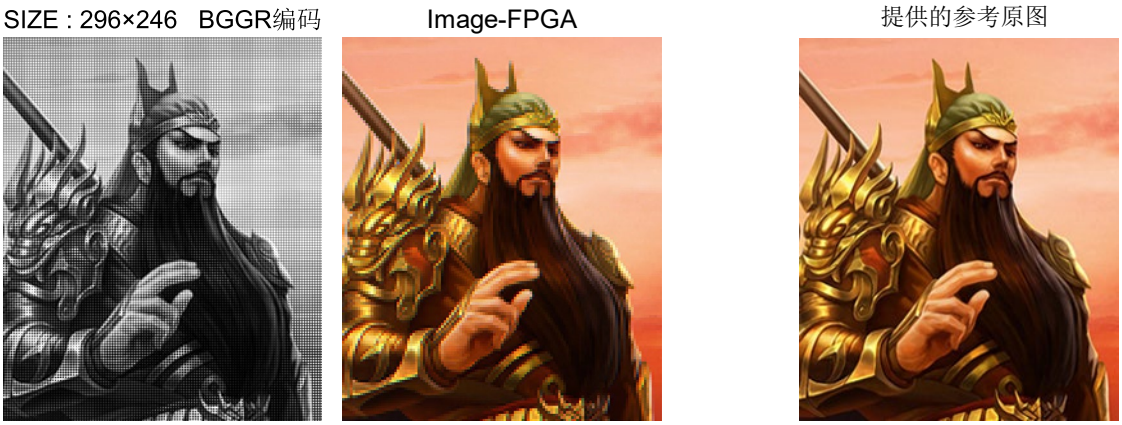


图 11　快速 Bayer2RGB 模块格式转换前后的图像及与原图的对比

可以看出转换为 RGB 格式后图像的色彩与提供的参考原图一致。由此可见，快速 Bayer2RGB 的输出结果是正确的。

4. 评估

4.1 功能正确性

功能的正确性评判已在**【3.结果分析】**中给出。无论是低功耗 Bayer2RGB 模块还是快速 Bayer2RGB 模块，输出结果均与笔者用 python 写的标准程序生成的标准输出完全一致。

由两个版本模块输出的可视化可见，转换后的图像与提供的参考原图颜色一致。

因此，低功耗 Bayer2RGB 和快速 Bayer2RGB 模块实现的功能符合要求。

4.2 性能评估

从完成转换所需的时钟周期数、存储器使用量、运算资源三个方面对低功耗 Bayer2RGB 模块和快速 Bayer2RGB 的性能进行评估，性能评估对比如下。（以下性能均基于尺寸为 296×246 pixels 大小的图片输入）

模块	完成转换所需的时钟周期数	模块内存储器使用量	运算资源（加法器个数）
低功耗 Bayer2RGB	**182041**	**95 bits**（包括输出缓存）	**17**
快速 Bayer2RGB	**127429**	**3047 bits**（包括输出缓存）	**17**

表 6　低功耗 Bayer2RGB 模块、快速 Bayer2RGB 模块的性能评估
（输入图片固定为 296×246 pixels 情况下）

两个模块中均在 parameter 定义块部分预留了 row 和 column 的定义，可以根据输入尺寸的变化对应地修改 row 和 column、precolumn 的值来完成不同尺寸 BGGR-Bayer 格式图片到 RGB 格式的转换。对 row 和 column 未知的情况，我们估算了两种模块的性能（所需时钟周期数、存储器使用量、运算资源）如下。

模块	完成转换所需的时钟周期数	模块内存储器使用量	运算资源（加法器个数）
低功耗 Bayer2RGB	$2.5\,R \times C + 1$	$75 + O(\log N)$ bits（包括输出缓存）	**17**
快速 Bayer2RGB	$1.75\,R \times C + 1$	$75 + O(\log N) + 12C$ bits（包括输出缓存）	**17**

表 7　低功耗 Bayer2RGB 模块、快速 Bayer2RGB 模块的性能评估
（输入图片固定为$R \times C$ pixels 情况下，其中$N = RC$）

5. 可能的优化方案

1）当前的低功耗 Bayer2RGB 模块和快速 Bayer2RGB 模块都不支持不同尺寸图片的格式转换，只能通过修改 parameter 定义部分来完成在不同尺寸图片上的格式转换。后续可以给模块增加两个输入端，通过输入图片尺寸（长和宽）的方式，实现只用一个模块就能完成不同尺寸的图片格式转换。

2）虽然已经对工作类似的状态进行了一定的合并，但当前的低功耗 Bayer2RGB 和快速 Bayer2RGB 模块内仍有较多工作类似的语句未进行优化。（如低功耗 Bayer2RGB 模块的奇数行处理和偶数行处理时的 count、line 累加，以及快速 Bayer2RGB 模块中状态 **S01xx** 和状态 **S1x01**、**S1x10** 有部分相同的工作，但尚未进行状态合并等），加法器的数量上仍存在优化空间。

3）快速 Bayer2RGB 模块在时间上也存在优化空间。在根据前一行存储的 RGB 三通道值输出奇数行的像素时，实际上也可以同时对下一行（偶数行）的像素进行计算和处理。这种优化后，理论上时钟周期数能减小到$(1.25\,R \times C + 2C + 1)$个时钟周期（对输入图片尺寸为$R \times C$ pixels 的情况）。