How to interpret RAW Bayer data from the PiCam HQ

J. Fabrizio

2020-01-06

Abstract

The PiCams (V1, V2 and now HQ) are interesting cameras as they are not expensive and provide RAW data from the sensor. Recently, a new one has been released, the PiCam HQ, but the official documentation does not provide any information on the internal format. This document, explains how to extract/interpret RAW images with this new PiCam HQ.

1 Introduction

The PiCams (V1, V2 and now HQ) are interesting cameras, usable in a research context and not expensive. It is possible to get RAW data from these cameras but it lacks some documentation to interpret data for the last PiCam HQ. This lack will certainly be filled very soon, but until this update, we propose to share my work to help other people to make it work[1]. I expose in this document the few things you need to interpret the data.

Before starting, you can read *The official raspberry pi camera guide* [7] which provide a lot of precious information. Details about PiCams can be found here [1]. This PiCam HQ is based on the Sony IMX 477 CMOS sensor. You should have a look to its specification.

Configuration and acquisition; The official documentation [2, 3] is well done except that the document states that the minimum *GPU* memory size is 128Mo. It was true for previous PiCams but not enough for the PiCam HQ. We were obliged to set it to 256Mo to make it work properly. Not really a problem (note: It is the `gpu_mem` option defined in `/boot/config.txt`).

To get the RAW data, we use the python library picamera [6] (we did not try with V4L.). The procedure is explained in [4]. This procedure works properly to acquire data from the PiCam HQ however, this documentation is not up to date as it does not mention the PiCam HQ. This is this temporary lack of documentation we will try to fill in this technical document.

This document is organized as follow: in section 2, we try to understand how the data are stored[2] and it is the main section of this document. In section 3, we recall the minimal scheme you need to get a picture from the RAW data. In the last section, section 4, we conclude.

2 Encoding

To understand how the data are stored, let's have a look at the encoding used by the previous PiCam. According to [4]:

- The RAW data are located at the end of the file.
- The header of the RAW data starts with the string `BRCM`.
- The first $32,768$ bytes of this part is the header data, then comes the Bayer data.
- Bayer data is always full resolution.
- Bayer data occupies the last $6,404,096$ bytes of the output file for the V1 module, or the last $10,270,208$ bytes for the V2 module. Bayer data consists of 10-bit values

[1] My level in English is rather low, and especially, I have no time to write this report. I write it very fast, without proofreading it. There must be plenty of mistakes - please forgive me for that. It is better than nothing.

[2] Be careful, my interpretation is maybe wrong; you use my conclusions at your own risk!

- The 10-bit values are organized as 4 8-bit values, followed by the low-order 2-bits of the 4 values packed into a fifth byte.
- Bayer data is organized in a BGGR pattern
- For the V1 module, the data consists of 1952 rows of 3264 bytes of data. The last 8 rows of data are unused (they only exist because the maximum resolution of 1944 rows is rounded up to the nearest 16).
- For the V2 module, the data consists of 2480 rows of 4128 bytes of data. There's actually 2464 rows of data, but the sensor's raw size is 2466 rows, rounded up to the nearest multiple of 16: 2480.
- The last few bytes of each row are unused.

To be able to interpret data from the PiCam HQ, we have to know all these parameters for the PiCam HQ, i.e., we have to find:

- The position and the size of the data in the file/the position and the size of the header,
- The size in byte of a line (the line stride) and how many pixels are store in this line/the number of lines,
- The correct variant of Bayer format,
- The encoding of the pixel values.

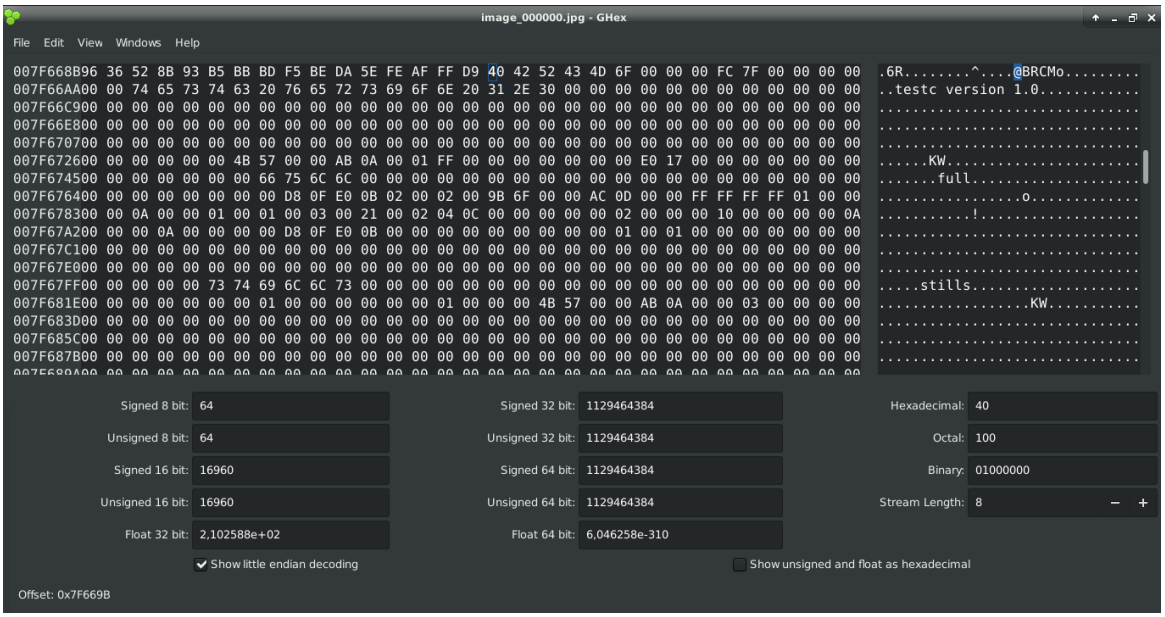
So many things to discover... To do so, `ghex` [5] is our friend. Let's go!

The position and the size of the header/the data in the file

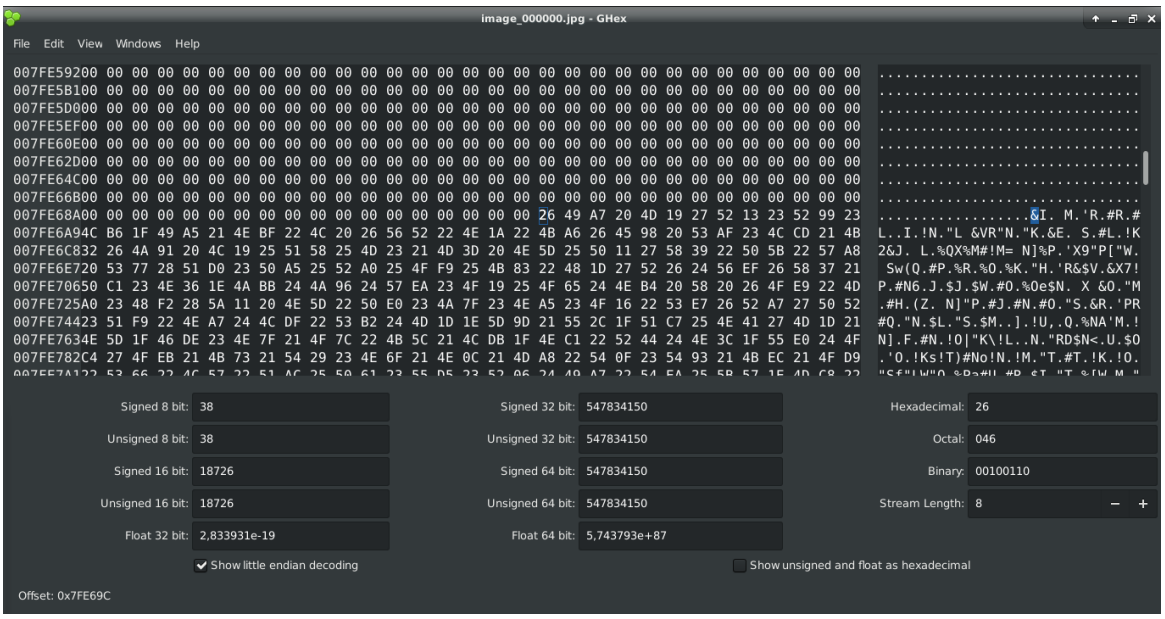
We bet, the format is compatible with the previous modules. We summary this format in the following table:

Module	Total length of the raw part (Bytes)	length of the header (Bytes)	length of the image (Bytes)
V1	6404096	32768	6371328
V2	10270208	32768	10237440
HQ	?	?	?

We have to take a picture with the PiCam HQ and we will see its content. The length of the file of the picture we took is 27060380 bytes. We open it with `ghex` and look for the string `BRCM`.



The offset of this string is at offset 8349340, this means that the complete RAW part is $27060380 - 8349340 = 18711040$ bytes long. We bet that the header has the same length compared to the previous modules: 32768 bytes. If we are right, this means that the image begins at offset 8382108 ($0x7FE69C$). This seems to be correct - data seem to start at this offset:



Then the data is 18711040 bytes long (i.e. occupies the last 18711040 bytes of the file), the header is 32768 bytes long and then the RAW image is $18711040 - 32768 = 18678272$ bytes long. We summary these results in the following table:

Module	Total length of the raw part (Bytes)	length of the header (Bytes)	length of the image (Bytes)
V1	6404096	32768	6371328
V2	10270208	32768	10237440
HQ	18711040	32768	18678272

The size in byte of a line (the line stride) and how many pixels are stored in this line/the number of lines

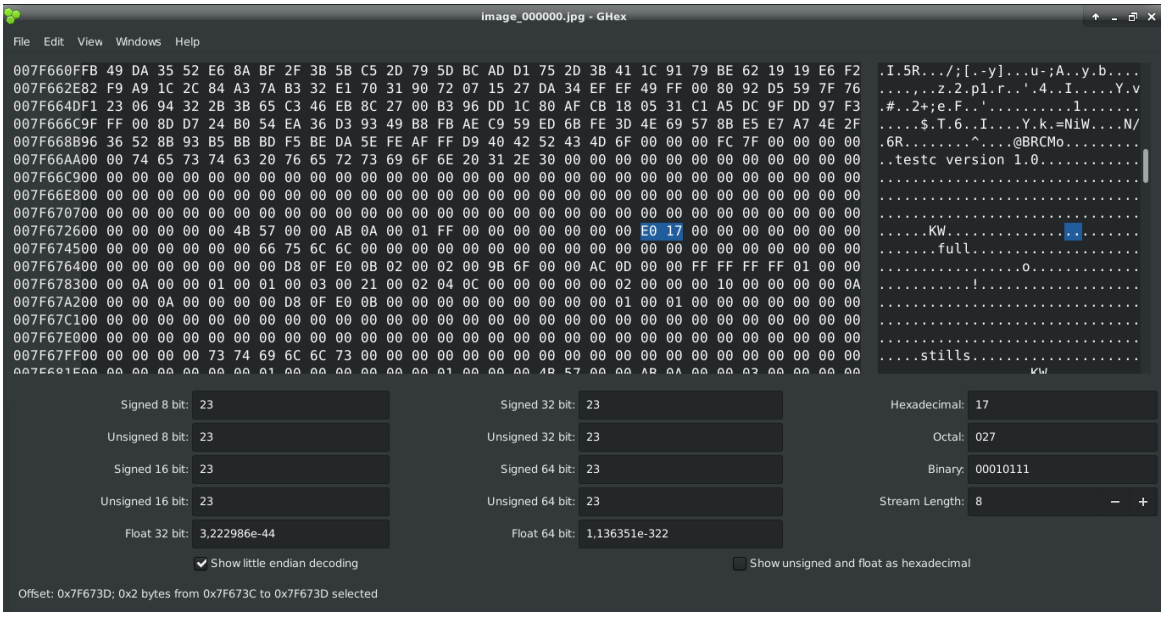
According to [1] the sensor resolution of the V1 module is 2592x1944, the resolution of the v2 module is 3280x2464 and the PiCam HQ the resolution is 4056x3040.

According to [4]: *For the V1 module, the data consists of* 1952 *rows of* 3264 *bytes of data. The last 8 rows of data are unused (they only exist because the maximum resolution of 1944 rows is rounded up to the nearest* 16*). For the V2 module, the data consists of* 2480 *rows of* 4128 *bytes of data. There's actually* 2464 *rows of data, but the sensor's raw size is* 2466 *rows, rounded up to the nearest multiple of 16:* 2480*. Likewise, the last few bytes of each row are unused.* We have to guess the image dimension.

Module	number pixels of a line	size in byte of a line	used part of the line
V1	2592	3264	$2592 * 10/8 = 3240$
V2	3280	4128	$3280 * 10/8 = 4100$
HQ	4056	?	?

We have to deduce the size of one line in byte. The pitfall is that the PiCam HQ encodes values on 12bits instead of previous modules that encode values on 10bits. This means that a line of 4056 pixels will consume $4056 * 12/8 = 6084$ bytes. The length of a line in byte in the file must certainly be a multiple of 16. It then can be 6096, 6112... This value can be deduced by finding the number of the lines in the file and dividing the total size by the number of lines or by simply studying the header.

If we look for in a picture taken by the V1 module the value 3264 (0x0CC0) and in a picture taken by the V2 module the value 4128 (0x1020), we find that these values are at a constant offset (160 - 0xA0) from the beginning of the header. If we *peek* the value at the same offset in a image taken by a PiCam HQ, we find the value 6112 (0x17E0). This validates our though.



Then we can fill in our table:

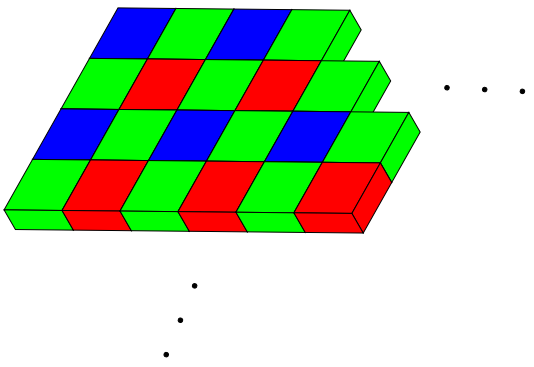
Module	number of pixels of a line	size in byte of a line	used part of the line
V1	2592	3264	$2592 * 10/8 = 3240$
V2	3280	4128	$3280 * 10/8 = 4100$
HQ	4056	6112	$4056 * 12/8 = 6084$

The correct variant of Bayer format/The encoding of the pixel values

It is said in [4] that for V1 and V2 modules, Bayer data is organized in a BGGR pattern but later, it is said that data are organized as follow:

```
# GBGBGBGBGBGBGB
# RGRGRGRGRGRGRG
# GBGBGBGBGBGBGB
# RGRGRGRGRGRGRG
```

In my point of view, the data start with a blue pixel followed by a red pixel and next line, the line starts with a green pixel, followed by a red one (and so on) like in the following picture:



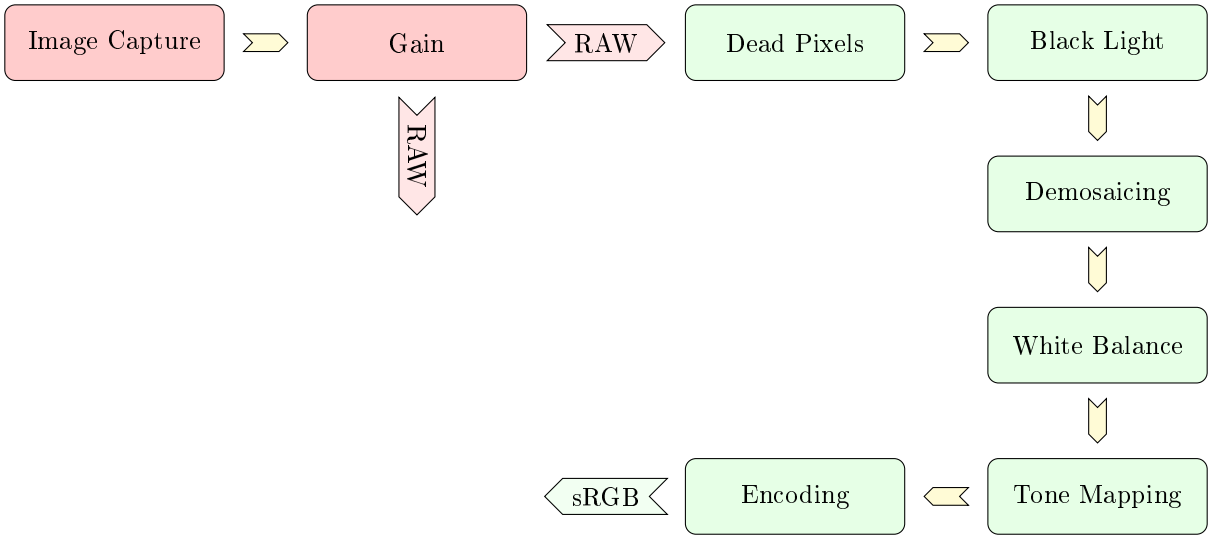
This is what we have observed in previous modules and what we think for this PiCam HQ.

One point important to notice is that previous modules were 10bits modules. According to [4], for the previous modules: *the 10-bit values are organized as 4 8-bit values, followed by the low-order 2-bits of the 4 values packed into a fifth byte.*. As we have 12bits values, we suspect that 2 consecutive 12bits values (two successive pixels) are organized as 2 8-bit values, followed by the low-order 4-bits of the 2 values packed into a third byte. After an examination with `ghex`, it seems to be correct.

Notice that this part of our report must be validated. We have taken blue pictures, red pictures... to try to validate but we do not have enough time to provide an absolute/rigorous validation. We may be wrong.

3 From RAW format to a beautiful picture

Once we have decoded the RAW data, the minimal work you have to do to get a acceptable image is summarized in the following picture:

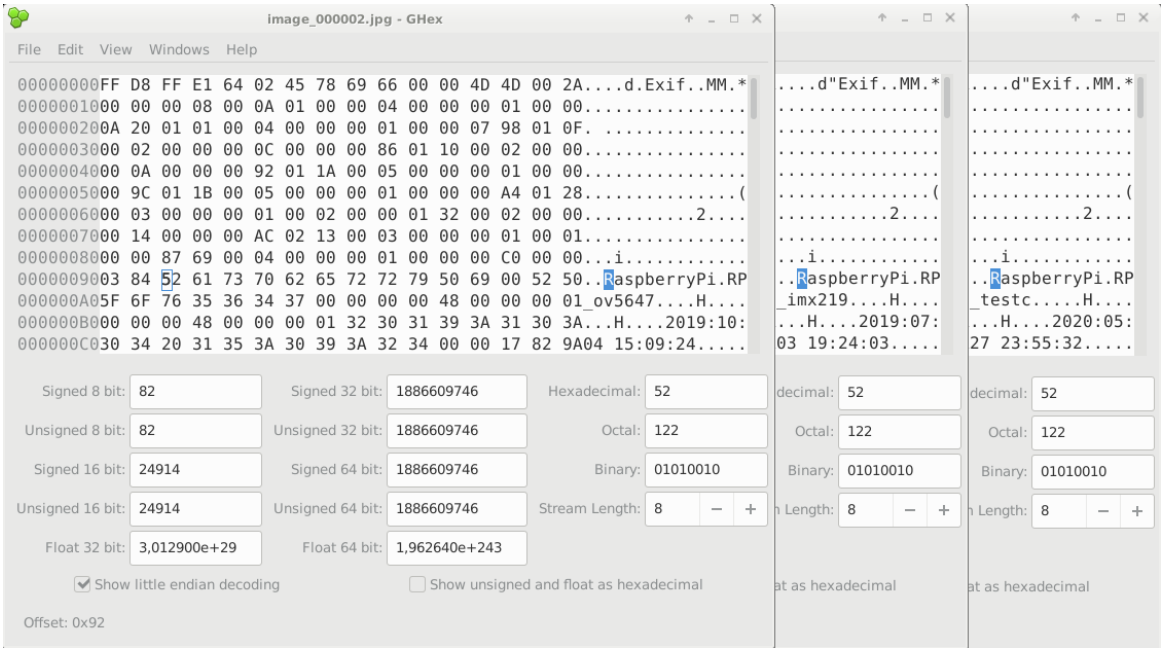


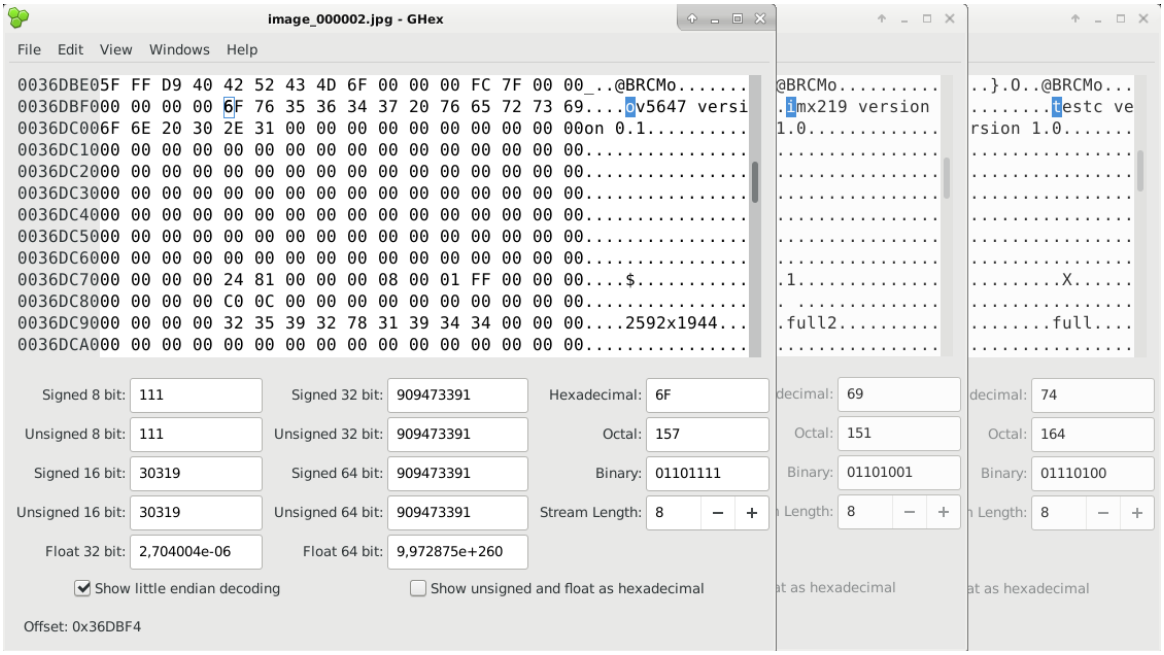
The green boxes have to be implemented but it is out of the scope of this document.

How to know which PiCam module has been used?

The problem if you want to generate an image using RAW bayer data from a PiCam is that you have to detect which PiCam has been used to take the picture.

As we can see in the following pictures, it is simple to detect which PiCam has been used by just using the header of the file or the header of the RAW part. Unfortunately, my new PiCam HQ module has not the value we would have expected in headers...





4 Conclusion

We provide in this document all missing parameters we need to interpret RAW bayer data from PiCam HQ. However some parameters need to be validated and it still misses some important points. For example, specification of the IMX219 sensor (the sensor used in the PiCam V2) has a region dedicated to optical black. It would be great if the PiCam HQ has such a region and if we could fetch values from this area.

References

[1] https://www.raspberrypi.org/documentation/hardware/camera/.

[2] https://www.raspberrypi.org/documentation/configuration/camera.md.

[3] https://www.raspberrypi.org/documentation/raspbian/applications/camera.md.

[4] https://picamera.readthedocs.io/en/release-1.13/recipes2.html#raw-bayer-data-captures.

[5] ghex. https://wiki.gnome.org/Apps/Ghex.

[6] picamera python library. https://picamera.readthedocs.io.

[7] Dan Aldred, Wesley Archer, Jody Carter, PJ Evans, Richard Hayler, James Singleton, and Rob Zwetsloot. *The official raspberry pi camera guide*. Raspberry Pi Trading Ltd, 2020.