# 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<sup>1</sup>. 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<sup>2</sup> 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

 $<sup>^{1}</sup>$ 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.

 $<sup>^{2}</sup>$ 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.

÷	image_000000.jpg - GHex	+ _ 8 ×
File Edit View Windows Help		
File         Edit         View         Windows         Help           00776668190         36         52         88         93         85         88         D0         56         57         73         74         63         20         76         65         72         73         74         63         20         76         65         72         73         74         63         20         76         65         72         73         74         63         20         76         65         72         73         00	FE AF FF D9       40       42       52       43       4D       6F       00       00       FC       7F       00	1
007F685C00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00	
Signed 8 bit: 64	Signed 32 bit: 1129464384 Hexadecimal: 40	
Unsigned 8 bit: 64	Unsigned 32 bit: 1129464384 Octai: 100	
Signed 16 bit: 16960	Signed 64 bit: 1129464384 Binary: 01000000	
Unsigned 16 bit: 16960	Unsigned 64 bit: 1129464384 Stream Length: 8	- +
Float 32 bit: 2,102588e+02	Float 64 bit: 6,046258e-310	
<ul> <li>Show little endian decoding</li> </ul>	Show unsigned and float as hexadecimal	
Offset: 0x7F669B		

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:

e image_000000.jpg - GHex + _									
File Edit View Windows Help									
007FE59206         00 00         00	$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	00 00 00							
007FE782C4 27 4F EB 21 4B 73 21 54 29 23 4E	6F 21 4E 0C 21 4D A8 22 54 0F 23 54 93 21 4B EC	21 4F D9 .'0.!Ks!T)#No!N.!M."T.#T.!K.!O. ハ							
Signed 8 bit: 38	Signed 32 bit: 547834150	Hexadecimal: 26							
Unsigned 8 bit: 38	Unsigned 32 bit: 547834150	Octal: 046							
Signed 16 bit: 18726	Signed 64 bit: 547834150	Binary: 00100110							
Unsigned 16 bit: 18726	Unsigned 64 bit: 547834150	Stream Length: 8 - +							
Float 32 bit: 2,833931e-19	Float 64 bit: 5,743793e+87								
Show little endian decoding	Show u	nsigned and float as hexadecimal							
Offset: 0x7FE69C									

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 P1CAM HQ, we find the value 6112 (0x17E0). This validates our though.

<u>image_000000.jpg - GHex</u> + ر	г×						
File Edit View Windows Help							
00776600FFB       49       DA 35       52       E6       AA       TA       B3       32       E1       TO       31       90       72       07       15       27       DA       34       EF       F9       F9       60       80       92       D5       59       75       76      ,,,,,,,							
6676691660 06 60 60 60 60 61 60 60 61 60 60 60 60 60 60 60 60 70 57 60 60 70 67 60 60 63 60 60 60 60 70 VW							
Signed 8 bit: 23 Signed 32 bit: 23 Hexadecimal: 17							
Unsigned 8 bit:         23         Unsigned 32 bit:         23         Octal:         027							
Signed 16 bit:         23         Signed 64 bit:         23         Binary:         00010111							
Unsigned 16 bit: 23 Unsigned 64 bit: 23 Stream Length: 8 –							
Float 32 bit: 3,222986e-44 Float 64 bit: 1,136351e-322							
Show little endian decoding Show unsigned and float as hexadecimal							
Offset: 0x7F673D; 0x2 bytes from 0x7F673C to 0x7F673D selected							

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...

<b>`</b>			image	_000002.	jpg	- GHex					Ŷ	- 0	×		$\uparrow$	- 0	×			- 0	$\times$
File Edit View	Windows	Help																			
00000000FF 000000206A 000000206A 0000005060 0000005060 0000007060 0000008060 0000008060 000000405F 0000008060 0000008060 0000008060	D8         FF         E1           00         00         08           20         01         01           02         00         00           0A         00         00           0A         00         01           03         00         00           14         00         00           00         87         69           84         52         61           6F         76         35           00         00         48           34         20         31	64 0 00 0 00 0 00 0 00 0 00 4 00 0 73 7 36 3 00 0 35 3	2       45         A       01         04       00         02       01         05       00         01       00         02       01         03       00         04       00         05       00         04       00         05       00         04       00         05       00         06       02         07       62         08       30	78         69           00         00           00         00           1A         00           02         00           13         00           00         00           13         00           04         00           05         72           00         01           32         34	660 04 01 860 05 01 00 03 00 30 30 32	$ \begin{bmatrix} 5 & 00 & 00 \\ 00 & 00 \\ 00 & 00 \\ 00 & 00 \\ 00 & 00 \\ 00 & 00 \\ 00 & 00 \\ 00 & 00 \\ 00 & 00 \\ 00 & 00 \\ 00 & 00 \\ 00 & 48 \\ 00 & 31 \\ 39 \\ 2 & 34 & 00 \\ \end{bmatrix} $	4D 00 07 00 00 00 00 00 00 69 00 3A 00	4D 01 98 02 01 A4 02 01 C0 00 00 31 17	00 01 00 01 00 00 00 52 00 30 82	2Ad. 00 0F 00 28 00 00 01 00 50 8asp 01_ov564 3AH 9A04 15:	Exif.	. MM .		d" 	Exif 	MM	*	d" 	Exif.	. MM. *	
Signed 8 bit:	82		Sig	gned 32 b	it:	18866097	46		) He	exadecimal:	52			decimal:	52			decimal:	52		
Unsigned 8 bit:	82		Unsig	gned 32 b	it:	18866097	46			Octal:	122			Octal:	122			Octal:	122		
Signed 16 bit:	24914		Sig	gned 64 b	it:	18866097	46			Binary:	01010	010		Binary:	010100	10		Binary:	01010	010	
Unsigned 16 bit:	24914		Unsig	ned 64 b	it:	18866097	46		Stre	eam Length:	8	-	+	h Length:	8	-	+	h Length:	8	-	+
Float 32 bit:	3,012900e+	-29	F	loat 64 b	it:	1,962640e	+243	3													
Show	little endian	decod	ing			Show	/ unsi	gneo	d and	float as hex	adecima	al		at as hex	adecimal			at as hexa	idecima	il	
Offset: 0x92																					

<b>%</b>		image_000002.jpg - 0	GHex		+ - • ×	↑ _ □ >	< ) + _ = × )
File Edit View	Windows Help						
0036DBE05F 0036DBF000 0036DC006F 0036DC2000 0036DC3000 0036DC3000 0036DC5000 0036DC5000 0036DC5000 0036DC8000 0036DC9000 0036DC9000	F         D9         40         42         5           00         00         00         06         7           6E         20         30         2E         3           00         00         00         00         00           00         00         00         00         00           00         00         00         00         00           00         00         00         00         00           00         00         00         00         00           00         00         00         00         00           00         00         00         00         00           00         00         00         00         00           00         00         00         00         00           00         00         00         00         00           00         00         00         00         00           00         00         00         00         00           00         00         00         00         00	2         43         4D         6F         00           66         35         36         34         37           1         00         00         00         00           00         00         00         00         00           00         00         00         00         00           00         00         00         00         00           00         00         00         00         00           00         00         00         00         00           00         00         00         00         00           00         00         00         00         00           00         00         00         00         00           00         00         00         00         00           00         00         00         00         00           00         00         00         00         00           00         00         00         00         00           00         00         00         00         00           00         00         00         00         00	000         000         FC         FC         FC           200         76         65         72         7           00         00         60         60         00           000         00         00         00         00           000         00         00         00         00           000         00         00         00         00           000         00         00         00         00           000         00         00         00         00           000         00         00         00         00           000         00         00         00         00           000         00         00         00         00           000         00         00         00         00           000         00         00         00         00           000         00         00         00         00           000         00         00         00         00           000         00         00         00         00           000         00         00         00         00 <td>00 00@BR( 3 690v1 00 000n 0.1 00 00 00 00.</td> <td>CMo 5647 versi</td> <td><pre>@BRCMo</pre></td> <td>}.0@BRCM0  rsion 1.0</td>	00 00@BR( 3 690v1 00 000n 0.1 00 00 00 00.	CMo 5647 versi	<pre>@BRCMo</pre>	}.0@BRCM0  rsion 1.0
Signed 8 bit:	111	Signed 32 bit: 90	09473391	Hexadecimal:	6F	decimal: 69	decimal: 74
Unsigned 8 bit:	111	Unsigned 32 bit: 90	09473391	Octal:	157	Octal: 151	Octal: 164
Signed 16 bit:	30319	Signed 64 bit: 90	09473391	Binary:	01101111	Binary: 01101001	Binary: 01110100
Unsigned 16 bit:	30319	Unsigned 64 bit: 90	09473391	Stream Length:	8 – +	Length: 8 – +	h Length: 8 - +
Float 32 bit:	2,704004e-06	Float 64 bit: 9,	,972875e+260				
Show	little endian decod	ing	Show unsigned a	and float as hexa	decimal	at as hexadecimal	at as hexadecimal
Offset: 0x36DBF	4						

## 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.