https://doi.org/10.2352/EI.2024.36.7.ISS-292
© 2024, Society for Imaging Science and Technology

Magic Lantern as a Platform for Digital Photography Research

Paul Eberhart, Henry Dietz; University of Kentucky; Lexington, Kentucky

Abstract

The Magic Lantern project describes itself as "Magic Lantern is a free software add-on that runs from the SD/CF card and adds a host of new features to Canon EOS cameras that weren't included from the factory by Canon." In doing so, they have provided APIs, documentation, and the means to run code on many Canon EOS interchangeable lens cameras, and also useful well-documented interchange formats for data extracted via that access. The current work describes how these facilities can be applied by researchers to develop new imaging techniques on a professional/prosumer camera platform.

Specifically, this work covers an attempt to use the Magic Lantern development tools to manipulate a cameras' Embedded Direct Memory Access engine (EDMAC) to perform on-the-fly frame diffing, and/or to the use of the project's MLV format for raw sensor data streams to extract data from the relatively large, high-performance sensor of a prosumer camera as input for alternative processing pipelines.

Introduction

Modern digital cameras contain sophisticated computer systems, comprising a CPU, a sensor with typically many ADC channels for readout, a relatively large RAM, one or more storage devices, various special hardware function units such as encoders for specific image formats, and the bussing and DMA (Direct Memory Access) devices to move data between the parts.

These cameras, therefore, function as cameras by virtue of the software loaded into them. In the case of Canon devices, the native software is built on a Canon developed RTOS (**R**eal **T**ime **O**perating **S**ystem referred to as "DryOS" [1] with various computer platform features and camera-specific functionality implemented on top. DryOS appears to implement the $\mu ITRON$ RTOS kernel specification, and also exposes some POSIX and DOS-like interfaces - for example, when configured to boot from an external storage device, it attempts to launch a file named `autoexec.bin`, a fact which is very important when attempting to load you own code on a camera.

DryOS has been used across the Canon line since around 2007; the older related CHDK (Canon Hack Development Kit) [2] project targeting Canon's fixed-lens cameras is also based around reverse engineering of Canon's interfaces, though CHDK generally limits itself to calling Canon functions, while ML focuses on higher-end interchangeable-lens cameras, and directly manipulates hardware configuration registers. In fact, though they are independent projects with entirely separate code bases, Magic Lantern originally derives from the reverse engineered documentation of the CHDK project, and they continue to share information. Magic Lantern operates as a program that runs along side the vendor software; the only modification made to the original system is setting the `BOOTDISK` flag in the onboard Flash, so the camera will attempt to load code from `autoexec.bin` in the root of the CF or SD card.

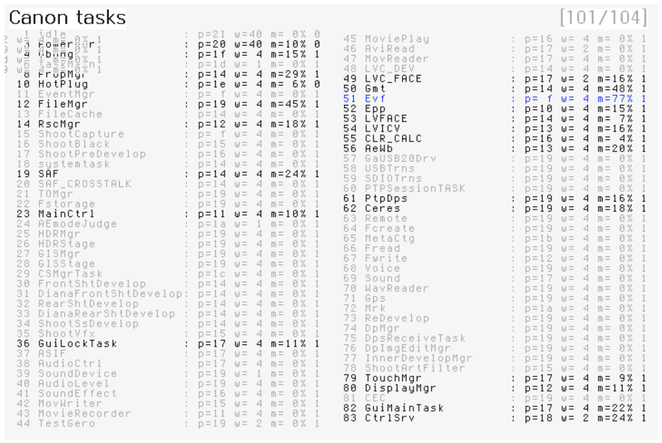
The Magic Lantern project [3] is a community developed, Open Source (GPL Licensed) project which has developed a development framework and extensive collection of software which is loaded into the camera by the aforementioned process to load user code from attached storage. The Magic Lantern community is chiefly focused on adding video recording features and capabilities to available cameras. Specifically, the project was initiated by Trammell Hudson in 2009 to add extended video features to the 5D Mark II, and management of the project transitioned to A1ex in late 2010. As the understanding of the camera internals, set of supported cameras, and community expanded, it has also extended camera capabilities for general shooting like focus peaking, zebra highlights, and live histograms, added useful shooting modes like intervalometer, motion detect, and automatic exposure bracketing, and a wide variety of other features. Magic Lantern also provides a set of sophisticated developer tools for inspecting camera behavior, and several methods to run custom code on the camera including a scripting interface in the Lua programming language, and a module system to load custom compiled code.

As academics, our interests in Magic Lantern can take several forms. The first source of academic interest in Magic Lantern is quite similar to the main ML community interest; the possibility of modifying a (relatively) capable commercial camera to behave in new ways to construct research prototypes. Because of the exposed internal memory structures and full programming environment, the functional units of the camera can be freely reconfigured up to the limits of the reverse-engineered understanding and/or capabilities of the hardware.

A second interest for research applications is the sophisticated scripting interface exposed by Magic Lantern. The aforementioned Lua scripting and loadable compiled module interfaces allow researchers to both arrange programmatic capture to use the camera as an apparatus for experiments, and also programmatically operate the camera to study its properties. Simple applications of custom code may include programmed timelapses, motion detection, exposure stacking for focus or dynamic range, etc. but as it is a complete programming environment, the possible functionality is limited only by the resources present in the device. The authors of this paper and a number of their collaborators have previously made extensive use this functionality on smaller CHDK-enabled cameras to support a number of research projects; in [4] it was used to produce programmatically controlled series of exposures to study camera behavior, in [5] it was used to coax cameras to export a novel image encoding, and in [6] it was used to perform photoplethysmography in-camera.

Another other major academic interest for MagicLantern is the insight it provides into the internal design of a (relatively) modern camera system. Camera manufacturers do not generally intentionally provide low-level access to the camera or its' embedded computer system, or even document its features, over concerns about licensing and competition. The reverse engineering efforts required to create community modifications like Magic Lantern expose that information. Close knowledge of the workings of widely available camera systems allow researchers and photographers alike to reason about both the resulting images and the potential capabilities of a camera in ways that a black-box treatment does not, even if they are not aiming to expand or alter the functionality.

The ML software also allows an enormous degree of live instrumentation into the running camera. For example, a built-in feature allows one to view the memory map and utilization, the process tables of both ML and Canon's native software, as in Fig. 1, and a wide variety of other logging and monitoring, several of which are applied later in this work.



```
Canon tasks                                                                         [101/104]
 1 idle                  : p=21 w=40 m= 0% 0      45 MoviePlay            : p=16 w= 4 m= 0% 1
 3 PowerMgr              : p=20 w=40 m=10% 0      46 AviRead              : p=17 w= 2 m= 0% 1
 4 DbgMgr                : p=1f w= 4 m=15% 1      47 MovReader            : p=17 w= 4 m= 0% 1
 5 TaskMain              : p=1d w= 1 m= 0% 1      48 LVC_DEV              : p=14 w= 4 m= 0% 1
 8 PropMgr               : p=14 w= 4 m=29% 1      49 LVC_FACE             : p=17 w= 2 m=16% 1
10 HotPlug               : p=1e w= 4 m= 6% 0      50 Gmt                  : p=14 w= 4 m=48% 1
11 EventMgr              : p= f w= 4 m= 0% 1      51 Evf                  : p= f w= 4 m=77% 1
12 FileMgr               : p=19 w= 4 m=45% 1      52 Epp                  : p=10 w= 4 m=15% 1
13 FileCache             : p=14 w= 4 m= 0% 1      53 LVFACE               : p=14 w= 4 m= 7% 1
14 RscMgr                : p=12 w= 4 m=18% 1      54 LVICV                : p=13 w= 4 m=16% 1
15 ShootCapture          : p= f w= 4 m= 0% 1      55 CLR_CALC             : p=16 w= 4 m= 4% 1
16 ShootBlack            : p=15 w= 4 m= 0% 1      56 AeWb                 : p=13 w= 4 m=20% 1
17 ShootPreDevelop       : p=16 w= 4 m= 0% 1      57 GaUSB20Drv           : p=19 w= 4 m= 0% 1
18 systemtask            : p=14 w= 4 m= 0% 1      58 USBTrns              : p=19 w= 4 m= 0% 1
19 SAF                   : p=14 w= 4 m=24% 1      59 SDIOTrns             : p=19 w= 4 m= 0% 1
20 SAF_CROSSTALK         : p=14 w= 4 m= 0% 1      60 PTPSessionTASK       : p=19 w= 4 m= 0% 1
21 TOMgr                 : p=19 w= 4 m= 0% 1      61 PtpDps               : p=19 w= 4 m=16% 1
22 Fstorage              : p=19 w= 4 m= 0% 1      62 Ceres                : p=19 w= 4 m=18% 1
23 MainCtrl              : p=11 w= 4 m=10% 1      63 Remote               : p=19 w= 4 m= 0% 1
24 AEmodeJudge           : p=1a w= 1 m= 0% 1      64 Fcreate              : p=19 w= 4 m= 0% 1
25 HDRMgr                : p=19 w= 4 m= 0% 1      65 MetaCtg              : p=1b w= 4 m= 0% 1
26 HDRStage              : p=19 w= 4 m= 0% 1      66 Fread                : p=19 w= 4 m= 0% 1
27 GISMgr                : p=19 w= 4 m= 0% 1      67 Fwrite               : p=12 w= 4 m= 0% 1
28 GISStage              : p=19 w= 4 m= 0% 1      68 Voice                : p=19 w= 4 m= 0% 1
29 CSMgrTask             : p=1c w= 4 m= 0% 1      69 Sound                : p=17 w= 4 m= 0% 1
30 FrontShtDevelop       : p=14 w= 4 m= 0% 1      70 WavReader            : p=19 w= 4 m= 0% 1
31 DianaFrontShtDevelop  : p=14 w= 4 m= 0% 1      71 Gps                  : p=19 w= 4 m= 0% 1
32 RearShtDevelop        : p=14 w= 4 m= 0% 1      72 Mrk                  : p=1a w= 4 m= 0% 1
33 DianaRearShtDevelop   : p=14 w= 4 m= 0% 1      73 ReDevelop            : p=19 w= 4 m= 0% 1
34 ShootSsDevelop        : p=14 w= 4 m= 0% 1      74 DpMgr                : p=19 w= 4 m= 0% 1
35 ShootVfx              : p=15 w= 4 m= 0% 1      75 DpsReceiveTask       : p=19 w= 4 m= 0% 1
36 GuiLockTask           : p=17 w= 4 m=11% 1      76 DpImgEditMgr         : p=19 w= 4 m= 0% 1
37 ASIF                  : p=17 w= 4 m= 0% 1      77 InnerDevelopMgr      : p=19 w= 4 m= 0% 1
38 AudioCtrl             : p=17 w= 4 m= 0% 1      78 ShootArtFilter       : p=15 w= 4 m= 0% 1
39 SoundDevice           : p=19 w= 1 m= 0% 1      79 TouchMgr             : p=17 w= 4 m= 9% 1
40 AudioLevel            : p=19 w= 4 m= 0% 1      80 DisplayMgr           : p=12 w= 4 m=11% 1
41 SoundEffect           : p=16 w= 4 m= 0% 1      81 CEC                  : p=19 w= 4 m= 0% 1
42 MovWriter             : p=15 w= 4 m= 0% 1      82 GuiMainTask          : p=17 w= 4 m=22% 1
43 MovieRecorder         : p=11 w= 4 m= 0% 1      83 CtrlSrv              : p=18 w= 2 m=24% 1
44 TestGero              : p=19 w= 2 m= 0% 1
```

Figure 1. Process table of Canon native processes

The authors of this paper are not affiliated with Canon nor are we the developers of Magic Lantern. We are long-time users who have been trying to adopt/adapt the Canon + Magic Lantern target to support various computational photography research projects.

In the specific case of the EOS M used as an example in this paper, the embedded computer is a Canon DIGIC5 chipset. A rough diagram of the relevant architecture is shown in Fig. 2. On the computer front, two processor cores are ARM5TE 32-bit processors, and total RAM is approximately 256MB. The "Image Preprocess" and "JPCORE" blocks represent memory-mapped fixed-function hardware for RAW processing and JPEG en/de coding (names derived from firmware strings), the SDCON/SD Card blocks represent the interface to the SD card, and the EDMAC is a sophisticated DMA engine which will be discussed in its own section below. Some devices - such as the onboard FlashROM that contains the built-in software, and the various IO devices for reading buttons, blinking LEDs, and interfacing lens mechanics are not included in this diagram, though they are also memory-mapped devices. Strings internal to the firmware refer to those IO devices as the "MPU" and "LPU" for the medium and low speed devices, respectively.

Another relevant detail of the low-level memory behavior of Canon cameras is that the native RAW pixel format for 14- bit Canon cameras is packed in a somewhat convoluted way, shown in figure 3. This means efficient processing of the native format involves working on 224-bit (least common multiple of 14 and 32) blocks, necessitating some interesting bit-twiddling and/or

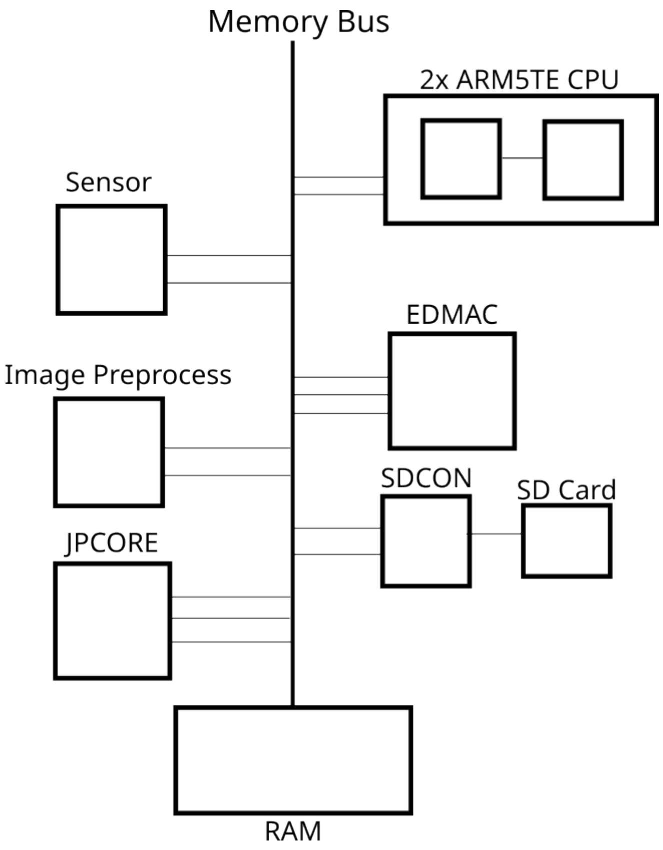
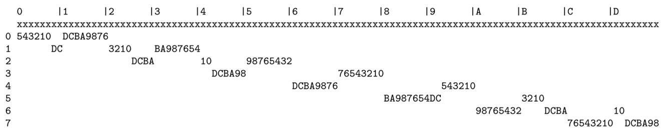


Figure 2. Rough layout of DIGIC 5 SoC

specific hardware support, some of which is accomplished by the aforementioned Image Preprocessing hardware, but much of which is managed by the EDMAC discussed in the following section.



```
  0      |1      |2      |3      |4      |5      |6      |7      |8      |9      |A      |B      |C      |D
  xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0 543210  DCBA9876
1       DC        3210    BA987654
2                     DCBA        10      98765432
3                                   DCBA98                76543210
4                                                 DCBA9876                  543210
5                                                                 BA987654DC              3210
6                                                                                 98765432    DCBA        10
7                                                                                                 76543210  DCBA98
```

Figure 3. The packed layout of Canon 14-bit RAW encoding

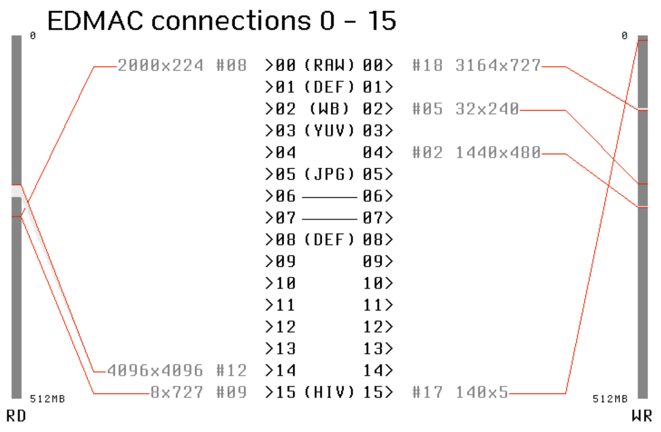
EDMAC

One of the major accomplishments of the ML project has been the reverse engineering of the EDMAC "Engine DMA Controller", where "DMA" in turn means "Direct Memory Access", device present in Canon Digic SoCs. This functionality was not available for the first several years of the project, but even a partial understanding of it has enabled a great deal of the functionality now supported by ML.

The initial reverse engineering work was performed by a1ex in late 2016 [7]. Shortly after the initial behavior and strings were documented, it was back-matched to a patent, [8] initially filed by Canon in 2003. The EDMAC name and much of the related terminology in the reverse-engineered documentation is derived directly from strings in the firmware, and later more terms were matched to the patent, so unlike many systems whose public understanding is based on reverse engineering, the terminology more-or-less lines up between Canon published documentation and public reverse engineering.

The EDMAC is a sophisticated point-to-point data transfer engine, which can be programmed to move data not only between memory regions, but also to and from various hardware devices. For example, on many camera models EDMAC channel 0 is connected to the raw read-out of the sensor [9], while channel 3 is connected to a hardware JPEG encoder/decoder.

The ML project now contains a loadable module which contains a great deal of live and logging instrumentation for the running EDMAC; it can provide a live view of current EDMAC activity as in Figure 4, logging at regular intervals to allow study of EDMAC activity during specific operations, and automatically identify unused EDMAC channels which can potentially be repurposed.



```
EDMAC connections 0 - 15
0                                                              0
        2000x224 #08  >00 (RAW) 00>  #18 3164x727
                      >01 (DEF) 01>
                      >02  (WB) 02>  #05 32x240
                      >03 (YUV) 03>
                      >04       04>  #02 1440x480
                      >05 (JPG) 05>
                      >06 ----- 06>
                      >07 ----- 07>
                      >08 (DEF) 08>
                      >09       09>
                      >10       10>
                      >11       11>
                      >12       12>
                      >13       13>
      4096x4096 #12   >14       14>
          8x727 #09   >15 (HIV) 15>  #17 140x5
512MB                                                      512MB
RD                                                            WR
```

Figure 4. *First screen of EDMAC live monitoring*

The specification of the memory regions to read or write from in the EDMAC is also quite sophisticated; it supports x and y block size, stride, count, and offset arguments, special sizes for last blocks in sequences, and it supports these specifications for both the source and the destination region, allowing it to transfer and transform memory regions of complicated shape and size with minimal CPU involvement. A variety of useful camera behaviors - like cropped sensor readout - are performed using this mechanism. The EDMAC DMA transfer behavior can be described in C as shown in Fig. 5, taken directly from the QEMU-based development tools built by the ML project.

Visually, this produces access patterns like those in Fig. 6 illustrating an access pattern with $xn = 3$, $yn = 2$, $xb \neq xa$, $yb \neq ya$, and differing positive values for $off1a$ and $off1b$ which affect the stride in the X direction. Negative offsets are also legal; it would cause the tiles to overlap rather than skip.

MLV

Another relevant accomplishment of the Magic Lantern project has been the creation of the MLV (Magic Lantern Video) format and associated tooling. The MLV format is, essentially, a container for sequences of raw frame data read directly from the sensor, as well as standardized ways of writing out supporting metadata, and optionally synchronized audio blocks. The format is specified as a roughly 200-line LGPL licensed header

```
for (int jn = 0; jn <= yn; jn++)
{
    int y    = (jn < yn) ? ya   : yb;
    int off2 = (jn < yn) ? off2a : off2b;
    for (int in = 0; in <= xn; in++)
    {
        int x     = (in < xn) ? xa    : xb;
        int off1  = (in < xn) ? off1a : off1b;
        int off23 = (in < xn) ? off2  : off3;
        for (int j = 0; j <= y; j++)
        {
            int off = (j < y) ? off1 : off23;
            cpu_physical_memory_write(dst, src
                , x);
            src += x;
            dst += x + off;
        }
}
```

Figure 5. *EDMAC DMA Transfer Behavior*

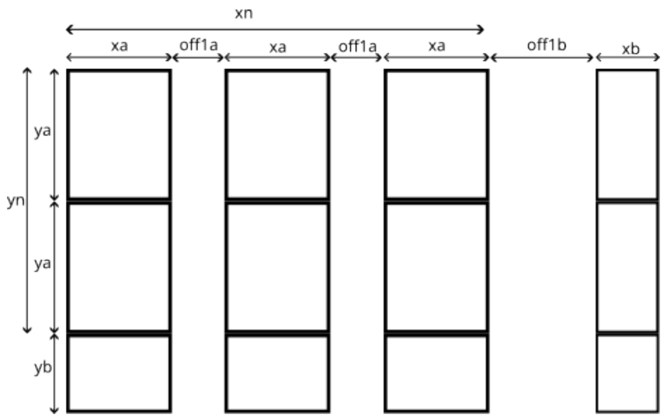


Figure 6. *EDMAC Memory Layout, with Offsets*

`mlv_structure.h`.

The initial “RAWv1.0” design was chiefly experimental. The more widely used specification, implemented by the `MLV_Rec` and `MLV_Lite` modules and all the major post-processing tooling is considered internally to be “RAWv2.0.” Later developments have not altered the on-disc format, but have focused on optimizing the implementation, maximizing utilization of the camera hardware to extract as much data as possible within the available sensor, memory, and (critically) storage bandwidth. Most camera-side development has been focused on the `MLV_Lite` module since it was forked by David Milligan around 2016 [10].

On the EOS M used for testing, the SD bandwidth has an empirical limit of around 60MB/s, which means - best case - full 14-bit RAW MLVs can be captured for longer than the handful of seconds it takes to fill the camera’s internal buffers at a resolution of roughly 1728x692 at 30FPS, and then only if one finds an SD card that happens to sustain the maximum observed speed. This resolution is not *terribly* impressive by modern standards, but an 14-bit RAW video from a ultra-compact body released in 2012 is quite an accomplishment. Unfortunately, the SD standards are not terribly uniformly implemented, so finding a card that will negotiate to the proper mode involves either locating an exact match to a card verified to do so by another user, or extensive trial-and-error.

The chief optimization of the on-camera MLV tooling is that it uses the EDMAC (above) to perform all the data motion; the sensor read-out is DMA on the EDMAC. The SD card write out is DMA on the EDMAC. The cropping is DMA on the EDMAC. The MLV capture tooling also works around various platform limitations, such as the 4GB file size limit on the FAT32 format volumes used on the SD cards in (most) cameras with integrated support for multi-file encoding.

An ecosystem of processing and conversion tools have formed around the MLV specification. Examples include MLVApp [11], a MLV processing tool initiated by ilia3101 (Ilia Sibiryakov), a sophisticated, open-source, community-built tool for processing the resulting MLV files. MLVApp supports the native formats produced by the various ML supported camera models, and allows a user to post-process and manipulate the RAW video in various desirable ways; sophisticated Demosaicing, highly parameterized exposure analysis and manipulation for toning and look, various forms of RAW correction for dead pixels and noise reduction, and conversion to a wide variety of conventional output video formats. It is distributed under a GPL3 license, and is built chiefly in C++ using the Qt toolkit. Many of its features are the result of community members adding specific functionality they desired, and/or hooking code from other open source developments, such as the librtprocess [12] tools derived from the open source RAW still processing software RawTherapee.

Another approach to accessing MLV data is MLVFS [13], also initiated by David Milligan of `MLV_Lite`. It mounts a MLV file as a virtual file system using the FUSE (File System in UserSpace) facility on UNIX-like systems, which allows the individual frames of the MLV stream to be accessed as though they area directory of DNG format RAWs, and consumed by any software which can operate on DNGs.

Limitations

The current list of camera models supported by Magic Lantern are the Canon 5D Mark III, 5D Mark II , 6D , 7D, 60D, 60Da , 50D, 700D / Rebel T5i , 650D / Rebel T4i , 600D / Rebel T3i , 550D / Rebel T2i, 500D / Rebel T1i, 1100D / Rebel T3 and EOS M, as enumerated on the current builds page at `https://builds.magiclantern.fm/`, with a few ports in progress. The newest of these cameras came out in 2012-2013, as platforms after the DIGIC5+ generation have not yet been adequately reverse engineered to build a working port.

Even those cameras which are supported will have features which are not fully exploitable. As an example of attempting to make use of a not-fully-understood feature, the original application that lead to this work was an interest in using the subtraction channel apparently available the EDMAC to perform on-the-fly frame diffing, for use in ongoing research projects. It can be experimentally verified, using the extensive introspection tools included in ML, that the EDMAC has some sort of subtraction mechanism. This mechanism is exposed in the camera UI for "Long Exposure Noise Reduction."

Actually making use of the subtraction mechanism presents two issues: the setup for two-reader one-writer EDMAC operations - like subtract - is not publicly documented, and the location of the subtraction engine in a particular camera is not stubbed into the ML code.

The second problem is relatively straightforward; a camera with ML active can log EDMAC activity at user-controlled intervals, and on the EOS M a comparison of the logs from a series of otherwise identical exposures, with "Long Exposure Noise Reduction" active and without reveals that EDMAC channel 20 is activated only when the subtraction mechanism is active.

The first problem, however, proved to be beyond reasonable effort for the experiment at hand. Though two-reader-one-writer EDMAC functions are visible in the logs, the existing ML code calling EDMAC functions primarily uses it to implement a fast `memcpy()`, or other one-reader-one-writer functions. When an EDMAC operation is configured, there are calls to `StartEDmac(ChanN, 0);` to configure a channel for writing and `StartEDmac(chanN, 2);` to configure a channel for reading. This leads to the natural conclusion that calling `StartEDmac(ChanN, 1);` might plumb a second reader, but making calls of that form doesn't appear to do anything other than hang the camera.

Conclusions

As compelling as the low-level access afforded by the EDMAC is, and despite Magic Lantern's successes at reverse engineering Canon's camera architecture, it is still likely a bad idea to use the direct hardware access afforded by ML as a development target for research.

The supported cameras are aging - on the order of a decade at this point - the understood functionality is spotty. The processor and memory resources available on-camera are fairly feeble, so any applications that aren't simply remixes of existing functionality will be unreasonably awkward or impossible.

That said, scripting behaviors which are essentially remixes of existing functionality are likely to work well. Scripted shooting to permute camera parameters to study camera or lens behavior are very straightforward to set up, as are event-triggered shooting or programmed exposure sequences to study the subject of the photograph, all of which are compelling applications not exposed in most commercial cameras.

Another potential win for researchers is the use of the MLV format to export RAW data for later processing when doing work involving the low-level pixel-format, such as algorithms for demosaicing.

Overall, using ML for research can be a very frustrating experience. The cameras it supports are now old, but they were relatively high-end consumer cameras when they were released, and the ML software infrastructure cleanly exposes more of their inner workings than can be accessed using any other software environment on any comparable or newer cameras from any major brand. If newer cameras were supported, and especially if the project were to be given a bit of help directly by Canon, the level of access provided by ML would be compelling for a very wide range of research projects – and very appealing as a platform for third-party software developers.

Unfortunately, no manufacturer of high-end consumer cameras has yet come to appreciate the potential value of providing an open API for running applications in the camera. Perhaps the failure of Digita [14], which failed largely because it was simply too early, continues to discourage manufacturers from making an open API for camera apps? We look forward to the day that camera apps for high-end cameras become as easy to write as smartphone apps.

Bibliography

[1] C. Inc. "Canon technology — dryos (archived)." (), [Online]. Available: https://web.archive.org/web/20080116050120/http://www.canon.com/technology/canon_tech/explanation/dryos.html (visited on 11/20/2008).

[2] *Canon hack development kit*. [Online]. Available: http://chdk.wikia.com/wiki/CHDK.

[3] *The magic lantern project*. [Online]. Available: https://www.magiclantern.fm/.

[4] H. G. Dietz and P. S. Eberhart, "Iso-less?" In *Electronic Imaging 2014*, vol. 9404, 2015, pp. 94040L-94040L–14. DOI: 10.1117/12.2080168. [Online]. Available: http://dx.doi.org/10.1117/12.2080168.

[5] K. Long, H. Dietz, and C. Demaree, "A canon hack development kit implementation of time domain continuous imaging," *Electronic Imaging*, vol. 2017, no. 15, pp. 66–72, 2017, ISSN: 2470-1173. DOI: doi:10.2352/ISSN.2470-1173.2017.15.DPMI-075. [Online]. Available: http://www.ingentaconnect.com/content/ist/ei/2017/00002017/00000015/art00011.

[6] H. Dietz, C. Parrish, and K. D. Donohue, "Self-contained, passive, non-contact, photoplethysmography: Real-time extraction of heart rates from live view within a canon powershot," *Electronic Imaging*, vol. 31, no. 13, pp. 146-1–146-1, 2019. DOI: 10.2352/ISSN.2470-1173.2019.13.COIMG-146. [Online]. Available: https://library.imaging.org/ei/articles/31/13/art00012.

[7] a1ex et.al. "Magic lantern forum: Topic: Edmac internals." (), [Online]. Available: https://www.magiclantern.fm/forum/index.php?topic=18315.msg245609 (visited on 12/20/2023).

[8] K. Ushida, Y. Naoi, Y. Katahira, and K. Morishita, "Image processing apparatus and image processing method," US7817297B2, Dec. 2010. [Online]. Available: https://patents.google.com/patent/US7817297.

[9] The Magic Lantern Community. "Register magic lantern firmware wiki: Register map." (), [Online]. Available: https://magiclantern.fandom.com/wiki/Register_Map (visited on 12/20/2023).

[10] David Milligan, et.al. "Magic lantern forum: Topic: Mlv lite." (), [Online]. Available: https://www.magiclantern.fm/forum/index.php?topic=16650.0 (visited on 12/20/2023).

[11] I. Sibiryakov. "Mlv-app: All in one mlv processing app." (), [Online]. Available: https://github.com/ilia3101/MLV-App (visited on 12/20/2023).

[12] CarVac. "Librtprocess: A project to make rawtherapee's processing algorithms more readily available." (), [Online]. Available: https://github.com/CarVac/librtprocess (visited on 12/20/2023).

[13] D. Milligan. "Mlvfs: Mlv (magic lantern raw video format) "converter" that uses fuse to allow "mounting" of mlv files as filesystems." (), [Online]. Available: https://bitbucket.org/dmilligan/mlvfs/src/master/ (visited on 12/20/2023).

[14] F. Technology. "Digitadev for digita application developers (archived)." (), [Online]. Available: https://web.archive.org/web/20010814080824/http://www.digitadev.com:80/home.emm (visited on 08/14/2001).