Improvements to v12345vtm's timelapseESP32-AP.ino

Jim van Zee - 21 July 2019

New Features:

- **Use the FlashLED and SD-card together, without modifying the PCB**
- **Add PWM dimming (0-1024) for the FlashLED**
- **Add 'Flash Mode' to turn the LED on just when taking a picture**
- **Add sliders and buttons to web page for new LED control features**
- **Add a way to set the RTC 'by hand' if no NTP server is available**
- **Add a way to change the Time Zone, at both compile and run-time**
- **Use a Timer Alarm for Time Lapse timing instead of RTC comparison**
- **Add 'ls', 'cd', 'rm' commands for managing file storage on the SD card**
- **Add 'tl' and 'led' commands for fine-tuning Time Lapse & LED settings**
- **Add recovery from failed WiFi connection with run-time SSID/PW entry**
- **And other commands for debugging and recovery purposes - see below**

These mods were developed by Jim van Zee <jim.van.zee(at)gmail(dot)com>

0. Background

Ivan Grokhotkov of Espressif created a demo to show off the abilities of the ESP32 camera interface. It was ported to the Arduino environment with a clear target of 'surveillance camera' usage, an idea enhanced by including 'face recognition' code. All the ESP32 pundits immediately created show-and-tell videos about how to run this demo - only a few took it as a starting point for other applications. **v12345vtm** was one of those, and his contribution to the development of the ESP32-CAM, though seemingly less noticed than the endless repeats of "this amazing new product with face recognition", are hugely important.

First he brought the web interface part of the code into the main program file, making it extremely easy to modify both the server and client code at the same time, and then he added support for the SD card, which the original demo did not have, and along with that he added a very useful (even for surveillance purposes) Time Lapse feature, implemented with another of his improvements, the addition of a webs-ocket server. He also came up a way to solve a very serious design flaw in the ESP32-CAM, namely the dual use of GPIO4 for the 'Flash LED' as well as the SD-card interface. Since no one else had actually added SD support, this major mistake had gone unnoticed. It was at this point that I discovered v12345vtm's work, and began making a few improvements of my own.

1. Improvements to the 'Flash LED' control

As v12345vtm pointed out in his YouTube video, the Flash LED on the ESP32-CAM board is unfortunately wired to the same GPIO pin used by the SDMMC interface for the SD card. This is a fixed-in-silicon feature of the SDMMC hardware (but NOT the simpler 'SD-IO' interface), so the only way to truly decouple these two conflicting uses is to make a hardware mod to the PCB, as v12345vtm has nicely shown here: https://youtu.be/1ojUXSgMBVY. This mod involves running a wire from GPIO33, where it connects to the cathode of the small red LED1 on the bottom of the board, to the base of transistor Q1, which drives the Flash LED. The resistor R1, which connects GPIO4 to the base of Q1, must also be removed.

This is a true solution, but few owners of the ESP32-CAM (and there are 10s, if not 100s of thousands of them) have the skill or the tools necessary to make this mod, so I explored how to fix this problem, in so far as possible, using just software. Indeed a fix **_is_** possible -- in the sense that you can control the LED without disrupting use of the SD card. But the reverse is not possible: when you read or write to the SD card, the Flash LED will, indeed, flash! So that's the drawback to the software fix, but it's a fairly-acceptable drawback in most cases, and it only takes 4 lines of code!

Once I had the LED working on my unmodified boards, I immediately wanted to improve it! Two things sprang to mind: first, the LED is **so bright** (it draws ~200mA from the 3.3V supply!) that in many cases the image is almost completely 'washed out'. So the first improvement was to try to add PWM control in place of simple on/off operation. This proved possible, although there were some initial complications. Due to the way images are captured by the OV2640, one needs to have a high PWM frequency, since the LED is actually either on or off, and pixels that are exposed when it's off just see ambient light! Fortunately it seems to work as expected, i.e. you can adjust the exposure by changing the PWM setting and the auto-exposure system seems to work, although I have not done any careful testing.

The second item on my LED wish-list was to add a control mode in which the LED only turns on when you take a picture, i.e. "flash mode", just like your mobile phone. This turned out to be a bit tricky because when the program 'takes a picture', it actually gets it from a buffer of previously captured pictures: the camera in fact runs continuously, so the picture you see in your browser, or the one you save to the SD card, is actually an image taken sometime earlier - up to 2 'frame times' before. With a little testing I determined that images 800x600 and smaller take ~80msec/frame, while higher resolution images take 160 msec/frame. So for Flash Mode to work properly we have to wait for 3 or more frames to pass before we have one that was taken when the LED was actually on. Obviously this extends the 'image-capture' time, which has implications if what you are taking a picture of is moving. However Flash Mode is more-or-less essential for time-lapse operation that needs LED lighting, since otherwise the LED could be on continuously for hours and hours, generating a large amount of heat and light (at full power the LED dissipates ~0.7W of heat - as much or more than the ESP32 itself with the radio in use).

Of course PWM dimming reduces the average power, so in some cases it may make sense to leave the LED on. One reason might be that the auto-exposure system takes time to stabilize. I added an extra frame-time to somewhat account for this, but you may find that more tweaking is needed for your particular application. I have an experimental situation that is susceptible to 'photo-bleaching', so I need to keep the LED off as much as possible. I found that I got the best results if I turned AE off and then adjusted the LED brightness & camera gain manually.

To bring these new LED control features to the user, I modified the 'home page' of the web interface by adding two 'sliders': the first one has three positions: "off", "flash" and "on", corresponding to the three LED options. I tried using various combinations of buttons for this, but the slider was by far the best way to do it (it's like a 3-position slide switch). The second slider changes the PWM setting 'exponentially': it has 10 steps, each being a factor-of-two brighter (or dimmer) than the previous one. Since the eye responds in a non-linear way, this slider gives one the expected experience of 'brighter' and 'dimmer'. Turn the LED 'on' and give it a try!

The slider however is obviously a very crude way to adjust the PWM setting, so I also added a **C**ommand **L**ine **I**nterface (**CLI**) that allows you, via the serial port, to specify any PWM value you like. For example: '**led 398**' sets the PWM value to 398. The range is 0-1024 (or 0-100% if you prefer: '**led 39%**'). The '**led**' command has 3 variations:: '**led0**', '**led1**' and '**ledf**' which correspond to turning the LED off ('0'), on ('1') and putting it in "flash mode" ('f'). The PWM value is optional, so once that is set (either with a **led** command or from the browser) you can just use '**led0/1/f**' to turn the LED off/on or put it in Flash Mode.

The "off/flash/on" slider replaced the LED ON, OFF buttons, so I had room for a new button, which I hopefully(!) labeled "**QR decode**". This is a feature that I would dearly love to have for the purpose of reading custom-coded sample labels. A little web-searching turned up at least 3 people who have implement this on the ESP32-CAM, but the code is either private or part of an

IDF project. Adding QR decoding is on my todo list, but I won't complain if someone helps me out! I imagine clicking the button to (a) put the camera in grey-scale mode and (b) start sending images to the browser. When I have a good image I can click the button again (or perhaps the program will click it for me, just like my mobile does when it succeeds), and the contents of the code will be sent to the serial port and maybe even (wouldn't this be nice?) displayed as a caption below the image.

2. RTC Improvements

v12345vtm added RTC support, which is needed for Time Lapse (TL) operations, and more generally for labeling images taken at any time (just the way all cameras and phones work). In order to set the clock (this being an ESP32 board) he followed the "SimpleTime" example and connected to a NTP server, thus getting the correct time to the nearest millisecond. That's great if you have an internet connection - but what if you're "in the field" with no internet available? In that case v12345vtm simply reverted to sequential numbering.

An unappreciated feature of "Simple Time" is that once the RTC is set up, **it runs by itself**! In fact the sketch makes a point of this by turning WiFi off after the clock is set. So if we could just set the clock **by hand** we wouldn't need internet access at all, not for **anything,** because the ESP32-CAM can operate as an Access Point, providing a WiFi connection that we can use to see pictures and run the program far from civilization, yet still have our pictures properly date/time-stamped! How could we do this? How can we manually set up the RTC?

I can not tell you how many hours I spent trying to figure this out! In the end it was simple enough, but I never found an example program. How irritating! I'm sure lots of people know how to do it, but they have managed to keep it a secret. Anyway, if you don't have access to the internet (i.e. you get a timeout when you try to connect using your ssid/pw), the program now gives you the option of setting the clock by typing in the the time via the serial port. There are many ways to do this - I decided to mimics the way the images are labeled: YYMMDD-HHMMSS, i.e. "190715-132639" for "15 July 2019, 13:26:39".

When prompted you just type in the current date+time in this format (the backspace key works if you need it!), press Enter and the clock is set! Obviously some people will worry about getting the "seconds" right. if you are one of them, you can type in a "future time", then wait until you see it on your mobile, and then hit 'Enter'. Another way is to just make a rough guess and correct it later using the CLI commands '**ct+**' or '**ct-**' to adjust the '**c**urrent **t**ime' by '+' or '-' so many secs (append '**m**' for **m**ins) . This solution to the RTC problem eliminates the need for "picture numbering", although I improved that feature a little bit too (see **fnc** & **fnr** commands).

Time Zones: one of the first things I noticed about the original version was that it defaulted to "Belgium Time". This was not surprising given that v12345vtm lives in Belgium, so I added a small set of 'defines' to the code to make it easy to compile for your time zone (mine is Pacific Time: '-8', compared to '+1' for Belgium) The TZ value is just the # of hours between your part of the world and Greenwich UK: '+' if you see the sun first (East) and '-' if you see it later (West).

Sometimes, however, you might take your camera on a holiday and find yourself in another TZ - what to do? Ans: You can use the '**ctz**' command to **C**hange the **T**ime **Z**one: '**ctz +1**' if you find yourself in Brussels, and '**ctz -8**' if you wake up in Seattle. To support users in India, TZs can be entered in half-hours: use '**ctz 5.5**' in you live in Bombay. You can type '**ct**' anytime you want to know the Current Time, which will also show the Time Zone. This is a fun way to see what time it is in another part of the world!

3. Time Lapse improvements

An invisible RTC-related issue in the original version was how the Time Lapse timing was implemented. It was, I must say, “rather creative”, using a string comparison in the loop() to watch the clock. I replaced this with a “**TimerAlarm**”, which runs in the background with no attention needed until the time period is up - much cleaner, with almost no code. I tested it on a 15-hour TL run and the pictures were time-stamped correctly every time. However this only confirmed that the TimerAlarm and the RTC were in sync. According to the datasheet the ESP-32’s internal oscillator is only accurate to ca. 5% unless you add an external 32KHz crystal. Not so great!

For my work the time intervals in v12345vtm’s version were not very useful, so I changed the “export” web page to have a wider set of choices, from every 5 secs (useful for testing!) up to 2 hours. The truth is, however, that almost every user will have their own needs. To address that situation I added a ‘**tl**’ command to the CLI. Using ‘**tl**’ you can set any time interval you need, from 1 sec (‘**tl 1**’) to 1 year (‘**tl 525960m**’ - note ‘**m**’ for **m**inutes). The resolution is 1 sec, and **yes**, the ESP32-CAM **can** take a picture every second (even in Flash Mode)! A picture every sec isn’t a movie, but it might be useful for ‘filming’ ants building a nest, or a 3D printer at work(?).

I also added a **CLI-only** feature that allows you to **pause** a TL sequence without changing the time interval, i.e. if you need to adjust some aspect of what you’re photographing, but don’t want to mess with the overall timing (you just want a ‘missing data point or two’), then you can type ‘**tlp**’ to **p**ause TL operation, and ‘**tlr**’ to **r**esume it, starting at the next interval. Use ‘**tlb**’ to **b**egin a new sequence, and ‘**tle**’ to **e**nd it. All TL operations can thus be done via the serial port - no WiFi needed at all. At any time you can type ‘**tl**’ to find out the **state** (running/paused/etc), the **number** of pictures taken, and how many **seconds remain** until the next picture.

As for the web-interface (the /export page), once you start a TL operation the screen ‘goes dead’: there are no updates because the web page itself would have to generate the request, and it doesn’t have it’s own internal timer for doing so. To solve this problem I added a ‘dead key’ just to the left of the time-selection menu labeled “**T.L.Update@**”. This odd descriptor serves to identify the adjacent menu item as well as giving a hint that if you click on this button it will update the page, telling you how many pictures have been taken, the name (aka the time-stamp) of the last one, and the time remaining until the next picture is due to be taken.

One thing that’s still missing, however, is a display of the last picture itself. To get that we would need to read the file from the SD card, which is not hard to do I think, but I don’t have all the skills to do it myself. In that same vein, someone asked v12345vtm if he had thought of adding a ‘browser’ so you could look at the pictures on the SD card. Good Idea! Again, it might be easy, or it might be hard - I don’t know. You’d probably want to display and navigate through the SD’s directory, which seems to me like it might be on the ‘hard’ side…

4. SD card management

TL operation has the potential to generate lots of images, and we certainly don’t want them to all be in the root directory! To provide some rudimentary file management I added the ‘**ls**’ and ‘**cd**’ commands to the CLI. ‘**ls’** (or its variant ‘**lsd**’) lists all the files (or just the **d**irectories) in the **c**urrent **d**irectory, i.e. the one you’ve ‘**cd**’ to. ‘**cd**’ always starts at the top level, although you can specify any number of sub-directories. Critically, if the directory doesn’t exist, **cd** will ask if you want to create it (the answer is ‘Y+Enter’, not just ‘Y’, or just ‘Enter’ to say ‘No’).

My suggestion for naming directories is to use the relevant calendar date - in the same way that file names are done: e.g. ‘190715’ for a directory holding images taken on 15 July 2019. I find this a super-easy way to locate pictures, but you can use any scheme you like. If properly managed, the root directory will consist of only directories (or ‘folders’ as they are sometimes called), with no ‘loose’ image files. The SD card I’ve been using, however, has lots of pictures I took before the **cd** command came along, which is the reason I added the ‘**lsd**’ variation of ‘**ls**’.

Note that **ls** does **not** alphabetize the files, nor put the directories at the top - it just dumps what it finds in the file system as it is structured on the SD card. This is a ‘wouldn’t it be nice’ issue, along with some other common directory-listing features, that might not be too hard to implement **if** one knew a good library to help with the effort. In addition to ‘**ls**’ and ‘**cd**’ I added a ‘**rm**’ command to ‘**re**move’ (i.e. ‘delete’) a file. There’s no ‘Are You Sure’ question, it just does it! The corresponding ‘**rmd**’ command is a ‘TBD’ at the moment (feel free!).

Could file management be added to the web interface? Probably - it’s just a variation on the theme of being able to display previously-taken pictures. I don’t have the skills to do this, but perhaps someone out there does (but please, helpful people, let’s get QR decoding done first!). The good news is that without the ‘face-recognition’ feature that v12345vtm so kindly removed, there’s lots of room for code. At the moment the program is under 1MB, and so fits comfortably in a 2MB partition -- it could even have OTA-updating if you wanted to add it!

One final note about SD cards: the ‘specs’ for the ESP32-CAM say something about a ‘4GB card’. **Ignore this!** The file system used by the program is “FAT32”, so basically any SD card up to 32GB will work (I’ve been using a 32GB card for testing). Such cards come pre-formatted in FAT32, whereas larger cards are usually formatted in ExFAT, which **won’t work** with the ESP32-CAM software. However FAT32 itself is good for terabytes of data - the only limitation is that a file can’t be larger than 4GB (that’s probably where the misleading number came from). Microsoft won’t let you format cards larger than 32GB using FAT32, but if you want to use your old 256GB card , see http://www.ridgecrop.demon.co.uk/index.htm?guiformat.htm.

5. Summary & General Observations

The ESP32-CAM board, like all ESP32 products, uses a lot of power: with the WiFi in full-blast (streaming) mode I see over 400 mA from my power supply. Turn the LED on 100% and its up to 600mA (or more). This kind of power can not be obtained from a USB-2 port - you either need USB-C or an external power supply. See Section 6 for thoughts on such things.

Turning WiFi off helps a lot - in fact, I have to **sadly repo**rt that there seems to be a major conflict between the AP service and the SDMMC interface. It is almost inevitable that a ‘**ls**’ command will fail immediately after power-up, or that a TL run will come to a screeching halt at some point with an ‘0x107’ error from the bowels of FreeRTOS, saying that a SDMMC operation ‘timed out’. With the WiFi AP service turned off this never happens.

Which brings up a few CLI commands that I’ve added to help with such situations. First we have ‘**reset sd**’ and ‘**reset esp**’, which do the obvious. I rarely do a full reset, but in contrast I use ‘**reset sd**’ all the time: it gets the SD card working again after the WiFi system does something nasty to it. ‘**reset esp**’ is what you type when things have really gone bonkers - it mimics the reset switch on the board (which no one can reach, of course, because it’s on the bottom!) A little-known fact is that if ‘setup()’ fails due to a hardware problem, the code simply drops into ‘loop()’, which is where the CLI commands live. So if you get a ‘Camera Failed to Initialize’ or the SDMMC interface sends you a complaint, just type ‘**reset esp**’ to try again - it usually works!

There are also a few WiFi commands: ‘**wfe**’ turns the WiFi system off, allowing you to access the SD card without any problems. You can **cd**, **ls** and take TL pictures without any fears. When you need WiFi you scan use ‘**wfs**’ to restart ‘Station Mode’, which uses your ssid/pw to connect to a handy LAN. Station Mode doesn’t seem to mess with the SD card, and it uses less power, so if you have a good LAN system, do ‘**wfe**’, then ‘**wfs**’ to create a stable working environment.

The corresponding ‘**wfa**’ command starts the Access Point, which is desirable for field work when you are not in range of a reliable LAN. For me this only works for initial setup purposes, i.e.

I can ‘get still’ and ‘start stream’ to align the camera and set the exposure, but after doing that if I try any SD operation I almost immediately get the 0x107 timeout error. So I do ‘**wfe**’+‘**reset sd**’ and then I’m able to get back to work using the ‘**tl**’ commands. Solving the WiFi-SD problem is definitely ‘unfinished business’, and I’m hoping that people like v12345vtm, with much more experience than I, will be able to figure this out.

What follows is a table listing all the CLI commands in this version. In general all commands will give you their ‘bottom line status’ if you use them without an argument, i.e. ‘**led**’ will print the state (Off/Flash/On) & the PWM setting, ‘**wf**’ will show the current state of the WiFi system, ‘**tl**’ shows how a Time Lapse operation is doing, and ‘**cd**’ tells you what directory you’re in. Want to know if it’s time for lunch? Just type ‘**ct**’ to get the Current Time, and so forth.

It’s easy to add new commands, as I have done for various debugging purposes - you’ll find everything you need in the code just before the setup() function. I’ve been lazy about forcing case, so ‘**LS**’ doesn’t work, but you can fix that IF YOU LIKE TO TYPE IN ALL CAPS.

In the table an optional arg is shown in []. Note that ‘**led**’ can use ‘%’ to specify the PWM value (‘**led1 46%**’) while ‘**tl**’ and ‘**ct+,-**’ can use ‘**m**’ to specify time in **m**inutes (‘**tl 5m**’ = ‘**tl 300**’). The time interval can only be changed when TL operation is fully stopped (not paused); if not, the time value is ignored. Here’s another hint: you can copy & paste filenames from a ‘**ls**’ listing to use in ‘**cd**’ or ‘**rm**’ commands (especially handy for ‘**rm**’).

ls	List all files in CurDir	tl [time]	Check status [set time]
lsd	List only Directories	tlb “	Begin TL operation
		tlp	Pause TL operation
cd	Show Current Directory	tlr	Resume TL operation
cd dir	Change/Make directory	tle	End TL operation
led [pwm]	Check status [set pwm]	gs	Get Still (saved in CurDir)
led1 “	Turn on [w/given pwm]		
ledf “	Set Flash Mode “	rm filenm	Remove (delete) file in CurDir
Led0	Turn off	rmd <dir>	Remove a directory (TBD)
ct	Check Time, Current Time	wf	Check status of WiFi system
ctz	Change Time Zone	wfe	End (stop) all WiFi operations
sct	Set time from serial port	wfs	Start WiFi in StaionMode (ssid)
gct	Get time from NTP server	wfa	Start WiFi Access Point
ct+, ct- [xx]	Incr, decr CT by secs[**m**ins]	wf scan	Scan for available networks
reset sd	Restore SD card operation	fnc [A-Z]	Set/examine leading file char*
reset esp	Same as pushing Reset Sw	fnr [###]	Set/examine next file number*

* The ‘**fnc**’ and ‘**fnr**’ commands are basically obsolete now that we have a way to set the RTC ‘by hand’ from the serial port. They provide a way to control the ‘alternate file-naming system’,

which simply labels each image with a sequential number. This idea was not fully implemented in v12345vtm's version (the number wasn't incremented, for example), so I added a few embellishments. File numbers are only used if the RTC has not been initialized - either via a NTP server ('**gct**'), or by typing in the date+time ('**sct**'). If neither of these things have happen then files are named: ESPCAM-A00000.jpg, ESPCAM-A00001.jpg, etc, using a 'leading char ('A') plus a 5-digit number. '**fnc**' lets you to change the leading char, while '**fnr**' lets you change the number.

The latter is particularly important, since every time the ESP32 is reset the number starts from '0' and the char is set to 'A'. This means that new files will have the the same numbers as earlier files, and will thus overwrite them. To prevent this you should first use '**ls**' to see what the last number was, then use '**fnr**' to set the number to the next higher value - or you can leave gaps if you wish, creating different series by changing the '1000's digit, or you can identify a series of files by giving them their own leading character. It's not a bad system, but it's clearly inferior to the date-time-stamp method available when the RTC is working.

6. Power considerations

A thought or two on power. As mentioned several times above, the ESP32-CAM uses a lot of mAs , especially with the LED turned on full blast (which happens, you may remember, whenever you access the SD card, for instance just to list a directory). It's way more mAs than a poor old USB-2 port was ever designed to deliver (500mA being the max). Of course USB-C solves this problem, but most of us still have older PCs and I can warn you that I have personally damaged at least one of my ports while working on this project. (Good PC's have re-settable PTC fuses on the USB power pins, but there is such a thing as 'too many times for too long').

So a separate power supply is pretty much a necessity for this board, and, because the LED as well as the ESP32 itself all run from the 3.3V rail, that's where you want to put the power. However it's 'best practice' to power the 5V domain as well (and 5V supplies are everywhere), so what I've done is add an external 3.3V regulator (a LM1117 is good: 4/€ from China), which I simply connect between the 5V and 3.3V pins. This parallels the on-board 3.3V regulator and dissipates some of the heat away from the PCB (500mA x (5-3.3)V = 0.85W - mind your fingers!). It's also necessary to put a nice big capacitor on the 3.3V side - I grabbed a 220µF electrolytic from my parts drawer and it made a remarkable difference in how theESP32-CAM behaved.

7. Other Ideas

I've been working on a version that uses 'light sleep' in between TL pictures. One can't use 'deep sleep' because that clears all the data in RAM (things like the SD card & WiFi parameters). I was pleased that it worked almost 'out of the box', but the problem is that the only wakeup source it recognizes is the sleep timer, and I need it to recognize a UART interrupt as well so I can regain control. This is a specific hardware feature of 'light sleep', but it doesn't work, so I need assistance ('light sleep' isn't well documented, hence it can be hard to make progress).

I also have a working version that allows you to put **comments** in a **log** file to record e.g. the experimental setup and/or other relevant details that happen during a 'run'. That code would be easy to add to this version, but I left it out because it wasn't essential. That version also uses the 'Preferences' library to keep track of your ssid/pw, which are easily changed at runtime with (you guessed it) a CLI command! It's nice to have a single source file that works in a variety of setups, and can be easily connected to whatever LAN is available. This version has a little of that functionality, but the ssid/pw (which **can** be entered at runtime) aren't saved for future use.

Jim van Zee 21 July 2019