ESP32 Tips

De “onbekende” ESP32 uit het lab lijkt op de Lolin32 Lite:

ESP32: Tips to increase battery life | Savjee.be

Store data **in RTC-memory** before **deep sleep**:

However, the RTC memory is kept powered on. So, its contents are preserved during deep sleep and can be retrieved after we wake the chip up. That’s the reason, the chip stores Wi-Fi and Bluetooth connection data in RTC memory before disabling them.

So, if you want to use the data over reboot, store it into the RTC memory by defining a global variable with `RTC_DATA_ATTR` attribute. For example, `RTC_DATA_ATTR int bootCount = 0;`

In Deep sleep mode, power is shut off to the entire chip except RTC module. So, any data that is not in the RTC recovery memory is lost, and the chip will thus restart with a reset. This means program execution starts from the beginning once again.

QCC5144 (from Qualcom) for low power BLE (not ESP32)

It takes less than 5mA.

Of zoiets: **NRF52811**

Deze site lijkt een goed begin voor ESP32 low power:

ESP32: Tips to increase battery life | Savjee.be

- Using `esp_timer_get_time()` generates “wall clock” timestamps with microsecond precision, but has moderate overhead each time the timing functions are called.
- It’s also possible to use the standard Unix `gettimeofday()` and `utime()` functions, although the overhead is slightly higher.
- Otherwise, including `hal/cpu_hal.h` and calling the HAL function `cpu_hal_get_cycle_count()` will return the number of CPU cycles

executed. This function has lower overhead than the others. It is good for measuring very short execution times with high precision.

The CPU cycles are counted per-core, so only use this method from an interrupt handler, or a task that is pinned to a single core.

- If making “microbenchmarks” (i.e. benchmarking only a very small routine of code that runs in less than 1-2 milliseconds) then flash cache performance can sometimes cause big variations in timing measurements depending on the binary. This happens because binary layout can cause different patterns of cache misses in a particular sequence of execution. If the test code is larger then this effect usually averages out. Executing a small function multiple times when benchmarking can help reduce the impact of flash cache misses. Alternatively, move this code to IRAM (see Targeted Optimizations).

Built-in tracing functionality

https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/app_trace.html

Todo: overweeg AceCoroutine.

Most **ESP32** boards have **dual-core** processors, so how do you know which **core** your task is running on? Just call xPortGetCoreID() from within your

(IR-) Remote control peripheral on esp32:

Remote Control (RMT) - ESP32 - — ESP-IDF Programming Guide latest documentation (espressif.com)

yield() heeft alleen zin voor taken met dezelfde prioriteit, las ik ergens.
Maar wat blijkt uit mijn TestPreemptionOnEvent test?
Het lijkt niet te werken als een taak eventbits set, dan yield() om de ontvangende taak die er op die eventbits wacht er iets mee te laten doen.. De send-taak geeft dan zijn beurt niet af.
Een vTaskDelay lost dat op – en zorgt ook dat na de delay de beurt weer terug komt.
Maar goed, dat werkt alleen veelvouden van ms.

De low level daemon Timer_srv taak heeft slechts priority 1, maar krijgt wel de kans om op basis van priority te pre-empten, omdat hij op core 0 zit.

Probleem: als je een high-prioriteit taak hebt met alleen delays van onder de 1 ms (delayMicroseconds), dan wordt gedurende die periode niet de thread afgestaan.
Lagere-prioriteit-taken komen dan niet meer aan de beurt. Zelfs gelijke prioriteit-taken die op events

wachten komen dan niet meer aan de beurt (yield() calls ten spijt).

Dit treedt bijvoorbeeld al op als je bij een IR-protocol-implementatie stapjes van 100us wacht.

Dus hoe pak je zoiets dan aan?
Vermoedelijk door gebruik van een hardware timer.
NB: interessant: alle ESP32 hardware timers zouden beschikbaar moeten zijn: de ESP32 platform code gebruikt een andere interrupt-techniek om de preemption-tick te maken.

Interessant: het preemptive overgaan naar een higher priority task die op een event wacht en krijgt, werkt, als een vTaskDelay gebruikt wordt binnen die higher priority task of die andere.

Bizar: ongeacht de prioriteit: als je taken aanmaakt zonder vTaskDelay,
dan draait alleen de eerste taak (zelfs als die yield / taskYIELD aanroept). ?!
.. ook als een hogere prioriteit taak op de event bits wacht..

Interessant voor als je een cooperative rtos op de esp32 wil zetten:

> loop() is the only task that is guaranteed to not be ran per tasking iteration.

Kortom: yieldTASK() heb ik niet zien werken
Om een thread zijn stokje af te laten geven moet je kennelijk een van de volgende dingen doen:

- de hoge prioriteits-thread eerst aanmaken en starten, en verlaten met een vTaskDelay(1 of meer) – want die timer geeft hem zijn stokje weer terug.
- Elke taak laten blocken op iets, met: vTaskDelay(1 of meer, event, queue of mutex).

Ultra fast time measurements

Fast timer for microsecond level duration measurements - ESP32 Forum

High resolution timer

High Resolution Timer - ESP32 - — ESP-IDF Programming Guide latest documentation (espressif.com)

Je kunt de callback van die timer weer koppelen aan event notifications oid.

esp-idf/esp_timer_example_main.c at
b66cc63c413a3b012f8cce9213dcb7ca281501d2 · espressif/esp-idf · GitHub

Sneller alternatief voor queue sets:

```
/* Wait until there is something to do. */
  xQueueReceive( xNetworkEventQueue, &xReceivedEvent, portMAX_DELAY );

  /* Perform a different action for each event type. */
  switch( xReceivedEvent.eEventType )
  {
      case eNetworkDownEvent :
          prvProcessNetworkDownEvent();
          break;
```

```
case eEthernetRxEvent :
    prvProcessEthernetFrame( xReceivedEvent.pvData );
    break;

case eARPTimerEvent :
    prvAgeARPCache();
    break;

case eStackTxEvent :
    prvProcessGeneratedPacket( xReceivedEvent.pvData );
    break;
```

Pend on multiple RTOS objects (freertos.org)

Nastyness: het lijkt erop (concluderend uit mijn waitable-wrapper-gedrag) dat
zodra je een queue hebt toegevoegd aan een queueset, dat je dan niet meer op die queue individueel kunt wachten… argh..

Het blijkt ook niet toegestaan om een queue aan meerdere queuesets toe te voegen, dus dat lost het ook niet op.

Nog even wat nauwkeurigere metingen:
50 logs posten kost – als beschermd met mutex: 293 us.
Met semafoor beschermd is dat 204 us.
Zonder bescherming is dat 13us. Dat is dus 13/50 = 0.26us per log.
Dus: mutex gebruik kost (293-13)/50 = 5.6us per gebruik.
semafoor gebruik kost (204-13)/50 = 3.8us per gebruik.
Eerder testten we het voor een queue (ook 50 keer?). Dat kwam neer op 2.5us per post.

Ik vraag me nu af of hetzelfde geldt voor notifications: als je een notification aan een eventgroup hebt toegevoegd, kun je er dan niet meer apart, of in een andere eventgroup op wachten?

.. misschien dat we daar dan multiwait-comms op kunnen baseren?

Oplossing: vergeet queuesets. Gebruik eventgroups. Die kun je op alle wait-momenten hergebruiken. Je test dan alleen op andere maskers. Aan een event kun je weer een queueu koppelen.

TODO: check out: hoeveel kost het setten van een flag (eventgroup)?

NB: je zou een **snelle connectie tussen 2 tasks**: een sender en een subscriber kunnen maken met een ringbuffer oid. Zolang de data in een lijst of array / ringbuffer terecht komt, waarvan een van beide tasks de Tail alleen mag schrijven en de ander alleen de Head mag schrijven.
Je kunt dan via een event-bit / flag aan de ontvanger bekendmaken dat er nieuwe data is.
Stel dat dat setten van de event-bit voor de sender nog te traag is, dan kun je er ook voor opten de ontvanger periodiek te laten pollen of er nieuwe data is. Dat ziet hij dan wel aan de positie van de Head t.o.v. de Tail.

Gekke crash

Tijdens het runnen een crash door “corruption”. Geen plek met assert te pin-pointen.
Wat bleek: Een bepaalde klasse (test_waitables) van twee (niet-abstracte) basisklassen afleiden

leidde tot het probleem. Als een van beide als member variabele werd geinclude, was het probleem opgelost.

Uploaden lukt niet, connecting…

Wat hielp, was tijdens dat connecting proces kort de “boot” button indrukken.
Of: boot button ingedrukt houden tot connecting begint.
En natuurlijk: eerst serial monitor sluiten.

Directe IDF code Bouwen op Arduino IDE

Als je file ook ”Arduino.h” include (naast direct de andere ESP dependencies), include die dan als **eerste**. (anders werkt het niet, zo lijkt het).

Executiesnelheid

Op de IDF loopt e.e.a. nog al traag, zo lijkt het.
Op de Arduino IDE lijkt het een stuk sneller te draaien.
Helemaal als je LogVerbose terugschroeft naar LogInfo en als je in crt_Config de stacksize checks uitcommentarieert.

In de IDF lijkt dat niet helemaal te lukken geen idee wat daar mis gaat.

Bug in delayMicroseconds

Bug in delayMicroseconds() · Issue #5000 · espressif/arduino-esp32 · GitHub

Misschien toch Arduino IDE bouwen vanuit ESP IDF?

De kwaliteit van de Arduino IDE is misschien niet onbetwist, maar de ESP samples zien er vaak wat complex / moeilijk bruikbaar uit. Ik ga daarom eens kijken of ik niet de Arduino IDE kan meebouwen vanuit IDF builds. Dwz, de contents van:
D:\Users\mave\AppData\Local\Arduino15\packages\esp32\hardware\esp32\2.0.2\cores\esp32

Yep.. nog even de c / cpp files en header-dirs toevoegen, en een variants/pins_arduino.h toevoegen voor mijn devkit, en het bouwt.

Probleem nog: ik vind geen servo object in de libraries in esp32\2.02\cores\esp32

Leuke voorbeeldprojecten voor esp32:

XTronical - Electronics programming and more

Volgens Kolban beter niet gebruiken:

There are 34 distinct GPIOs available on the ESP32. They are identified as:

- `GPIO_NUM_0` – `GPIO_NUM_19`
- `GPIO_NUM_21` – `GPIO_NUM_23`
- `GPIO_NUM_25` – `GPIO_NUM_27`
- `GPIO_NUM_32` – `GPIO_NUM_39`

The ones that are omitted are 20, 24, 28, 29, 30 and 31.

Note that `GPIO_NUM_34` – `GPIO_NUM_39` are input mode only. You can **not** use these pins for signal output. Also, pins 6 (SD_CLK), 7 (SD_DATA0), 8 (SD_DATA1), 9 (SD_DATA2), 10 (SD_DATA3), 11 (SD_CMD) 16 (CS) and 17(Q) are used to interact with the SPI flash chip … you can **not** use those for other purposes.

Ook met pinnen 12, 14 en 15 gebeurt iets tijdens booten:
ESP32 Pinout Reference: Which GPIO pins should you use? | Random Nerd Tutorials

Pinnen 0,2,5,12,14,15 moeten **tijdens het booten** NC zijn.

Oplossingen:

- Ofwel een multiplexer / level converter chip ertussen voor loskoppelen van de pinnen tijdens het booten.
- Ofwel die pinnen **alleen** gebruiken als **output-only** pinnen (zoals I2C of SPIC clock-pinnen die je als master (bijna-) altijd zelf drivet, of pwm-servo outputs).

Duidelijke website over esp32 (wifi etc)

Connect to Wi-Fi networks with the ESP32 – uPesy

MicroPython

Als je wilt spelen met micropython, is deze wellicht interessant:

Wemos D32 V1.0.0 - ESP32 - CH340C - 4MB Flash - Opencircuit

TFT_eSPI library

Leuke library voor touchdisplays zoals de ILI9341.

De generic button example laat het essentiele zien: calibratie,
opslag ervan in de bijbehorende sd card, button tonen, er op kunnen klikken.
! opgepast: de standaard code bevat een out-of bounds error: de array “calData” is slechts 5 uint16s groot. Toch worden er 14 karakters naar gelezen en geschreven.
Verander dat dus naar 10, om onverwachte problemen te voorkomen.