Sveprisutno računarstvo

1. Uvod

- Potpoglavlje 1
- Potpoglavlje 2
- …

Creative Commons

Power Saving Mode

- As gas is important for trucks, and cars to move, so is electric power for electronic devices.
- Embedded devices usually use battery (limited energy)
- The limited nature of battery power means the rate of power consumption of these devices should be reasonable to encourage their adoption and use
- Implementation of low power considerations in the design of embedded systems

Power Saving Mode

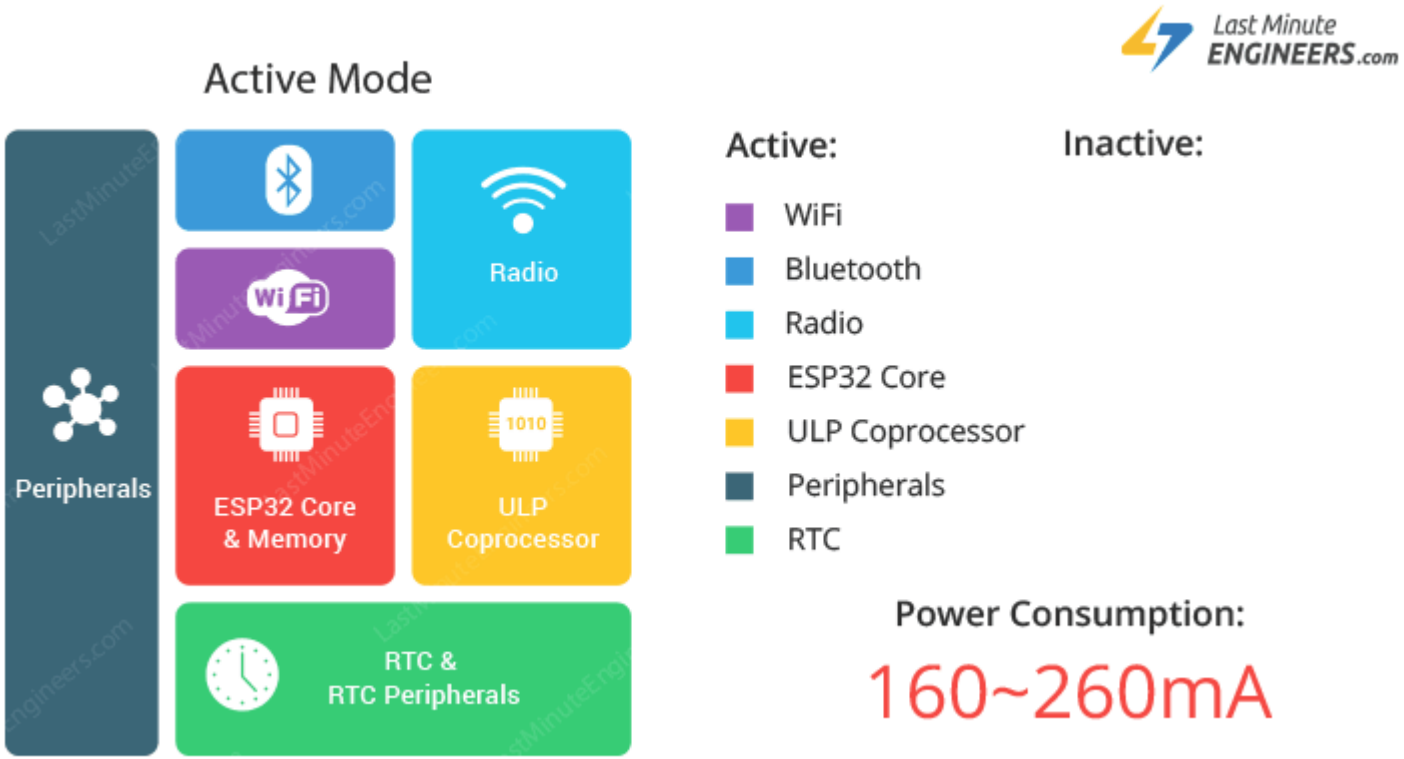
- Power Saving Techniques for Microcontrollers
  - Sleep Modes
    - The sleep modes (generally referred to as low power modes) are arguably the most popular technique for reducing the power consumption in microcontrollers. They generally involve disabling certain circuitry or clocks that drive certain peripherals of the microcontrollers.
    - Depending on the architecture and manufacturer, microcontrollers usually have different kinds of sleep modes, with each mode possessing the ability to disable more internal circuitry or peripheral compared to the other. Sleep modes usually range from deep sleep or off, to idle and doze modes.
  - Dynamic Modification of Processor Frequency
    - oldest technique and a little more complicated than the sleep modes.
    - It involves the firmware dynamically driving the processor clock, alternating between high and low frequency
  - Interrupt Handler Firmware Structure
    - most extreme techniques for power management in microcontrollers
    - available microcontrollers like the ARM cortex-M
      - have a sleep-on-exit bit in the SCR register, this bit provides the microcontroller with the ability to sleep after running an interrupt routine.

ESP32 Power Modes

- ESP32 can be a relatively power-hungry device
- When your IoT project is powered by a plug in the wall, you tend not to care too much about power consumption. But if you are going to power your project by batteries, every mA counts.
- ESP32 Sleep mode is a power-saving state that ESP32 can enter when not in use. The ESP32's state is maintained in RAM. When ESP32 enters sleep mode, power is cut to any unneeded digital peripherals, while RAM receives just enough power to enable it to retain its data.
- ESP32 offers 5 power modes:
  - Active Mode
  - Modem Sleep Mode
  - Light Sleep Mode
  - Deep Sleep Mode
  - Hibernation

ESP32 Active Mode

- In this mode all the features of the chip are active.
- „Normal mode”



ESP32 Active Mode

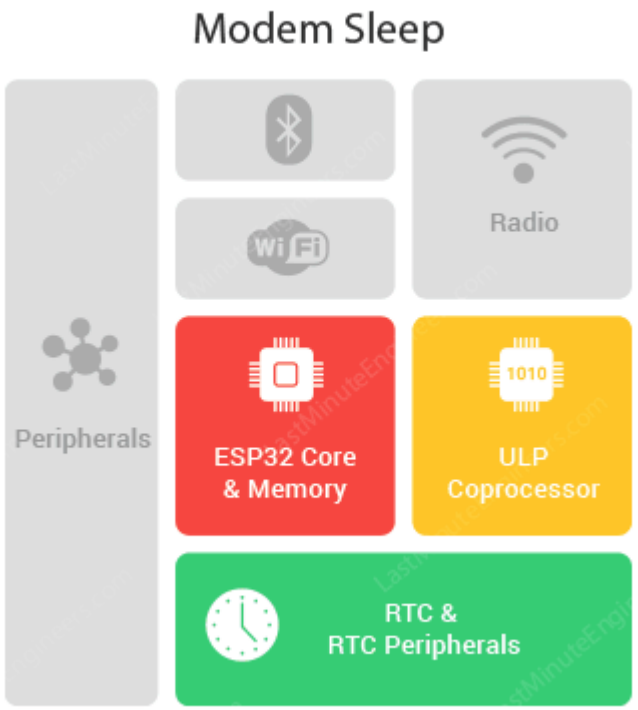
- Power consumption during Active power mode, with RF working is as follows:

Mode	Power Consumption
Wi-Fi Tx packet 13dBm~21dBm	160~260mA
Wi-Fi/BT Tx packet 0dBm	120mA
Wi-Fi/BT Rx and listening	80~90mA

- sometimes high power spikes appear (biggest was 800mA).
- The higher current peaks are short (wifi Tx) and can be averaged out by a capacitor
- ESP32 datasheet – 3.3V 500 mA

ESP32 Modem Sleep

- WiFi, Bluetooth and radio are disabled
- The CPU is operational and the clock is configurable.
- In this mode the chip consumes around 3mA at slow speed and 20mA at high speed.



Last Minute ENGINEERS.com

Active:

- ESP32 Core
- ULP Coprocessor
- RTC

Inactive:

- WiFi
- Bluetooth
- Radio
- Peripherals

Power Consumption:

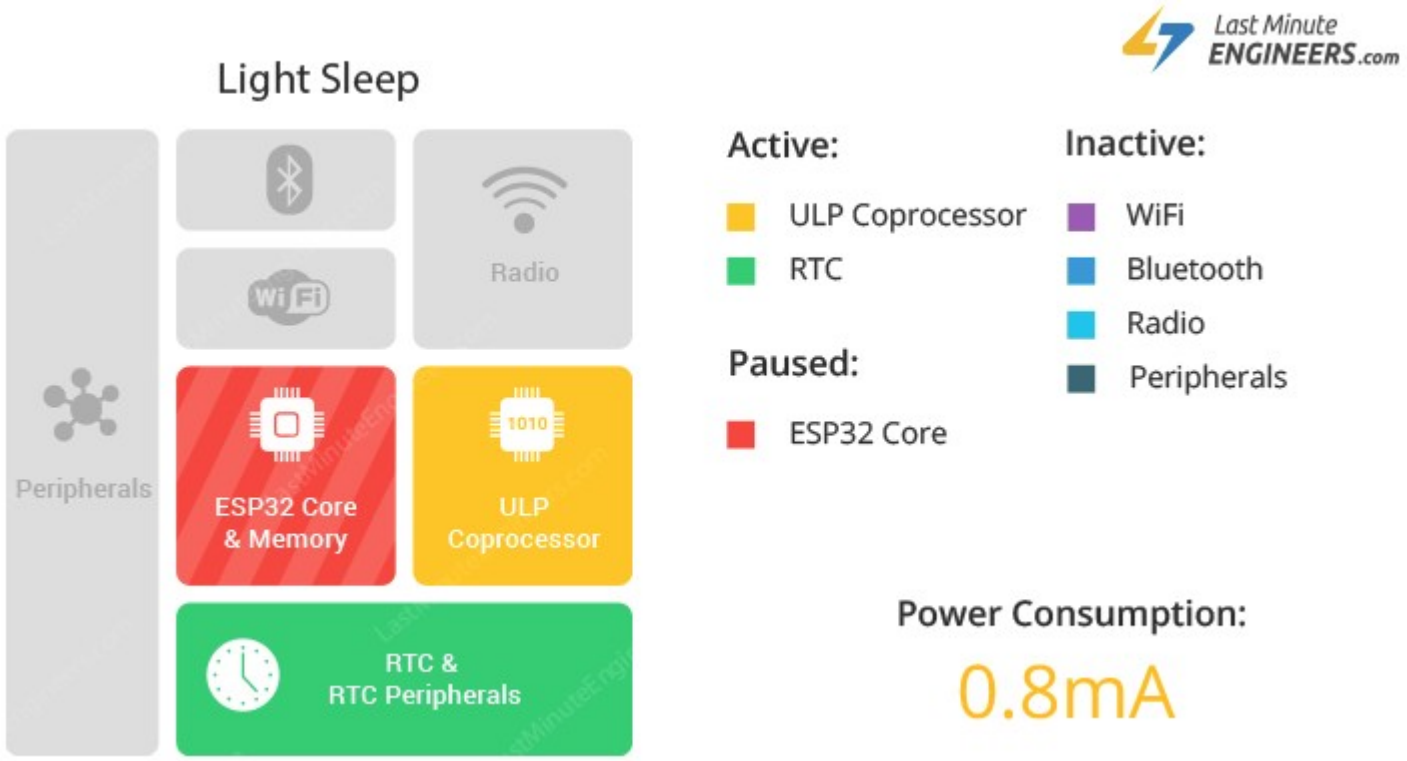
3~20mA

ESP32 Modem Sleep

- To keep WiFi/Bluetooth connections alive, the CPU, Wi-Fi, Bluetooth, and radio are woken up at predefined intervals. It is known as Association sleep pattern.
- During this sleep pattern, the power mode switches between the active mode and Modem sleep mode.
- ESP32 can enter modem sleep mode only when it connects to the router in station mode. ESP32 stays connected to the router through the DTIM (Delivery Traffic Indication Message) beacon mechanism.
- In order to save power, ESP32 disables the Wi-Fi module between two DTIM Beacon intervals and wakes up automatically before the next Beacon arrival.
- The sleep time is decided by the DTIM Beacon interval time of the router which is usually 100ms to 1000ms.

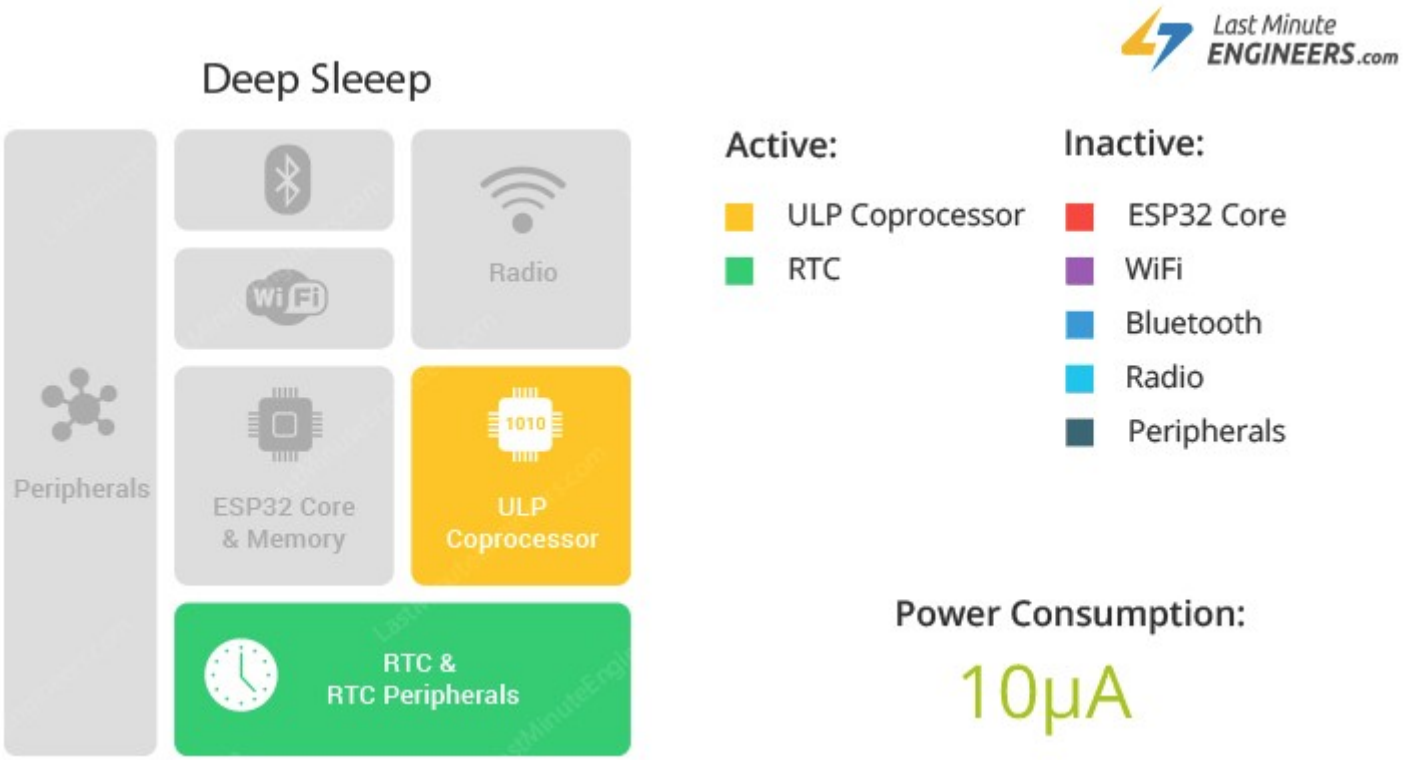
ESP32 Light Sleep

- similar to modem sleep
- during light sleep mode, digital peripherals, most of the RAM and CPU are **clock-gated**
- During light sleep mode, the CPU is paused by powering off its clock pulses.



ESP32 Deep Sleep

- the CPU, most of the RAM and all the digital peripherals are powered off
- powered on are: RTC controller, RTC peripherals (including ULP co-processor), and RTC memories

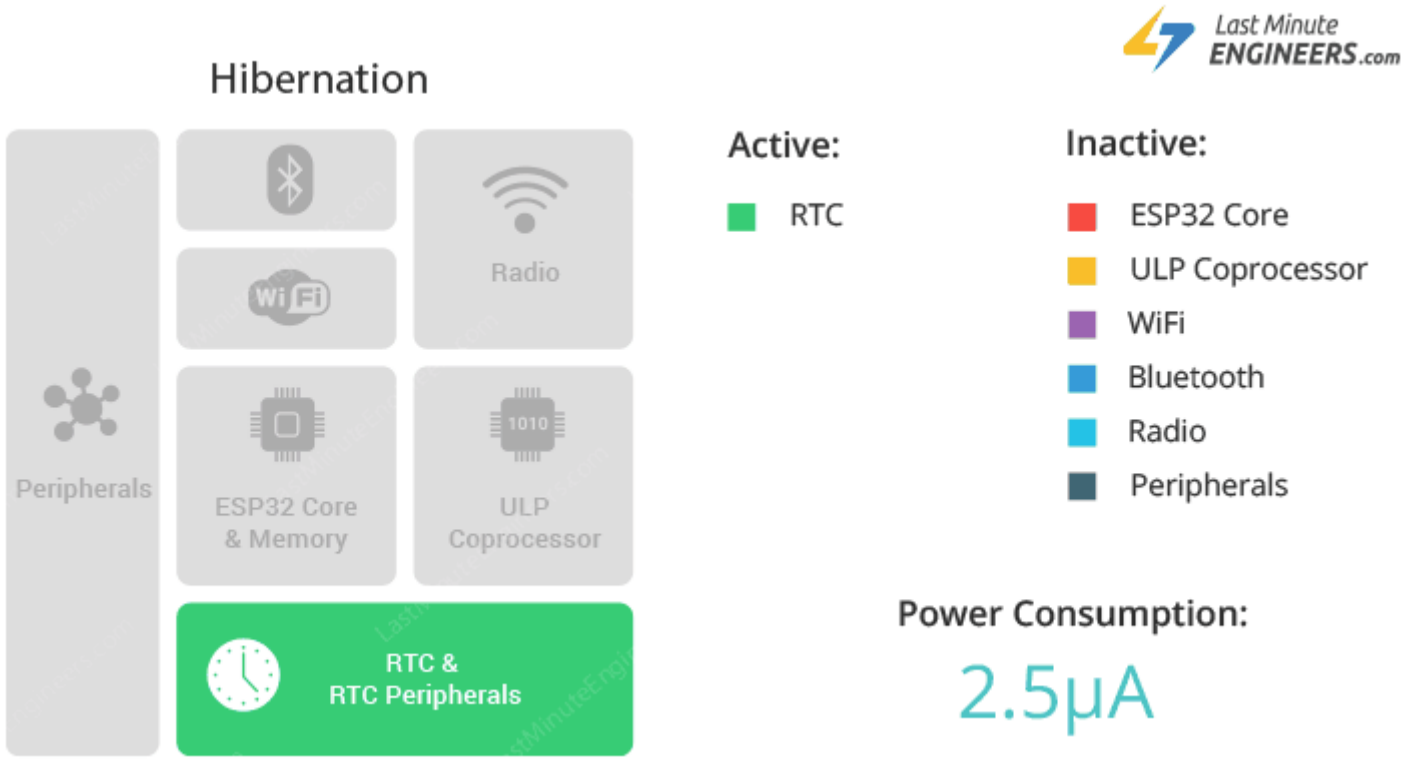


ESP32 Deep Sleep

- During deep sleep mode, the main CPU is powered down.
- Along with the CPU, the main memory of the chip is also disabled. So, everything stored in that memory is wiped out and cannot be accessed.
- 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.
- 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. This means program execution starts from the beginning once again.

ESP32 Hibernation mode

- in hibernation mode the chip disables internal 8MHz oscillator and ULP-coprocessor as well. The RTC recovery memory is also powered down, meaning there's no way we can preserve any data during hibernation mode.
- everything else is shut off except only one RTC timer on the slow clock and some RTC GPIOs are active.



ESP32 Wakeup Sources

- Timer
- Touch pad
- External wakeup (ext0)
- External wakeup (ext1)
- ULP coprocessor wakeup
- GPIO wakeup (light sleep only)
- UART wakeup (light sleep only)

ESP32 Wakeup Sources

- Timer
  - RTC controller has a built in timer which can be used to wake up the chip after a predefined amount of time. Time is specified at microsecond precision.
  - For details on RTC clock options, see ESP32 Technical Reference Manual > ULP Coprocessor [PDF].
  - This wakeup mode doesn’t require RTC peripherals or RTC memories to be powered on during sleep.

ESP32 Wakeup Sources

- Touch pad
  - RTC IO module contains logic to trigger wakeup when a touch sensor interrupt occurs. You need to configure the touch pad interrupt before the chip starts deep sleep.
  - esp_sleep_enable_touchpad_wakeup() function can be used to enable this wakeup source.

ESP32 Wakeup Sources

- External wakeup (ext0)
  - RTC IO module contains logic to trigger wakeup when one of RTC GPIOs is set to a predefined logic level.
  - RTC IO is part of RTC peripherals power domain, so RTC peripherals will be kept powered on during deep sleep if this wakeup source is requested.
  - Because RTC IO module is enabled in this mode, internal pullup or pulldown resistors can also be used. They need to be configured by the application using rtc_gpio_pullup_en() and rtc_gpio_pulldown_en() functions, before calling esp_deep_sleep_start().

ESP32 Wakeup Sources

- External wakeup (ext1)
  - This wake up source allows you to use multiple RTC GPIOs. You can use two different logic functions:
    - Wake up the ESP32 if any of the pins you've selected are high;
    - Wake up the ESP32 if all the pins you've selected are low.
  - This wake up source is implemented by the RTC controller. So, RTC peripherals and RTC memories can be powered off in this mode.

ESP32 Wakeup Sources

- ULP coprocessor wakeup
  - ULP coprocessor can run while the chip is in sleep mode, and may be used to poll sensors, monitor ADC or touch sensor values, and wake up the chip when a specific event is detected. ULP coprocessor is part of RTC peripherals power domain, and it runs the program stored in RTC slow memory. RTC slow memory will be powered on during sleep if this wakeup mode is requested. RTC peripherals will be automatically powered on before ULP coprocessor starts running the program; once the program stops running, RTC peripherals are automatically powered down again.

ESP32 Wakeup Sources

- GPIO wakeup (light sleep only)
  - In addition to EXT0 and EXT1 wakeup sources described above, one more method of wakeup from external inputs is available in light sleep mode. With this wakeup source, each pin can be individually configured to trigger wakeup on high or low level using gpio_wakeup_enable() function. Unlike EXT0 and EXT1 wakeup sources, which can only be used with RTC IOs, this wakeup source can be used with any IO

- UART wakeup (light sleep only)
  - When ESP32 receives UART input from external devices, it is often required to wake up the chip when input data is available. UART peripheral contains a feature which allows waking up the chip from light sleep when a certain number of positive edges on RX pin are seen. This number of positive edges can be set using uart_set_wakeup_threshold() function. Note that the character which triggers wakeup (and any characters before it) will not be received by the UART after wakeup. This means that the external device typically needs to send an extra character to the ESP32 to trigger wakeup, before sending the data.

ESP32 Power consumption

	Reference [mA]	Light-Sleep [mA]	Deep-Sleep [mA]	Hibernation [mA]
ESP32 – DevKitC	51	10	9	9
Ai-Thinker NodeMCU-32S	55	15.05	6.18	6.18
Adafruit HUZZAH32	47	8.43	6.81	6.80
Sparkfun ESP32 Thing	41	5.67	4.43	4.43
FireBeetle ESP32	39	1.94	0.011	0.008
WiPy 3.0	192	-	0.015	-

https://diyi0t.com/reduce-the-esp32-power-consumption/

ESP32 Dynamic Modification of Processor Frequency

- Power management algorithm included in ESP-IDF can adjust the advanced peripheral bus (APB) frequency, CPU frequency, and put the chip into light sleep mode to run an application at smallest possible power consumption, given the requirements of application components.
- Application components can express their requirements by creating and acquiring power management locks

ESP32 Dynamic Modification of Processor Frequency

- For example:
  - Driver for a peripheral clocked from APB can request the APB frequency to be set to 80 MHz while the peripheral is used.
  - RTOS can request the CPU to run at the highest configured frequency while there are tasks ready to run.
  - A peripheral driver may need interrupts to be enabled, which means it will have to request disabling light sleep.
- Since requesting higher APB or CPU frequencies or disabling light sleep causes higher current consumption, please keep the usage of power management locks by components to a minimum.

ESP32 Processor Frequency

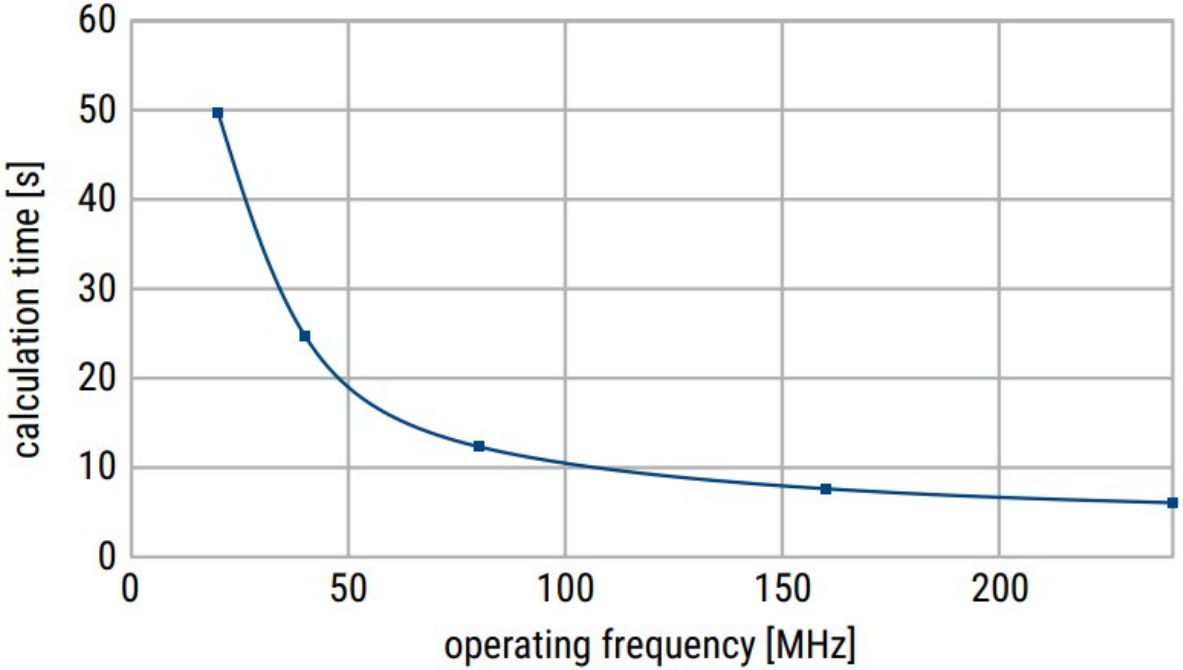
- proper selection of the device's operating parameters allows to extend its operating time without having to replace the battery or recharge it.
- In order to select the optimal operating parameters, the measurements of the algorithm execution time and the current consumption of the device were carried.
- As a calculation algorithm, the calculation of the π number was used in accordance with formula:

$$\pi = 4 \cdot \sum_{n=1}^{\infty} \frac{(-1)^{n-1}}{2n-1} = 4 \cdot \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \ldots\right)$$

n = 125000

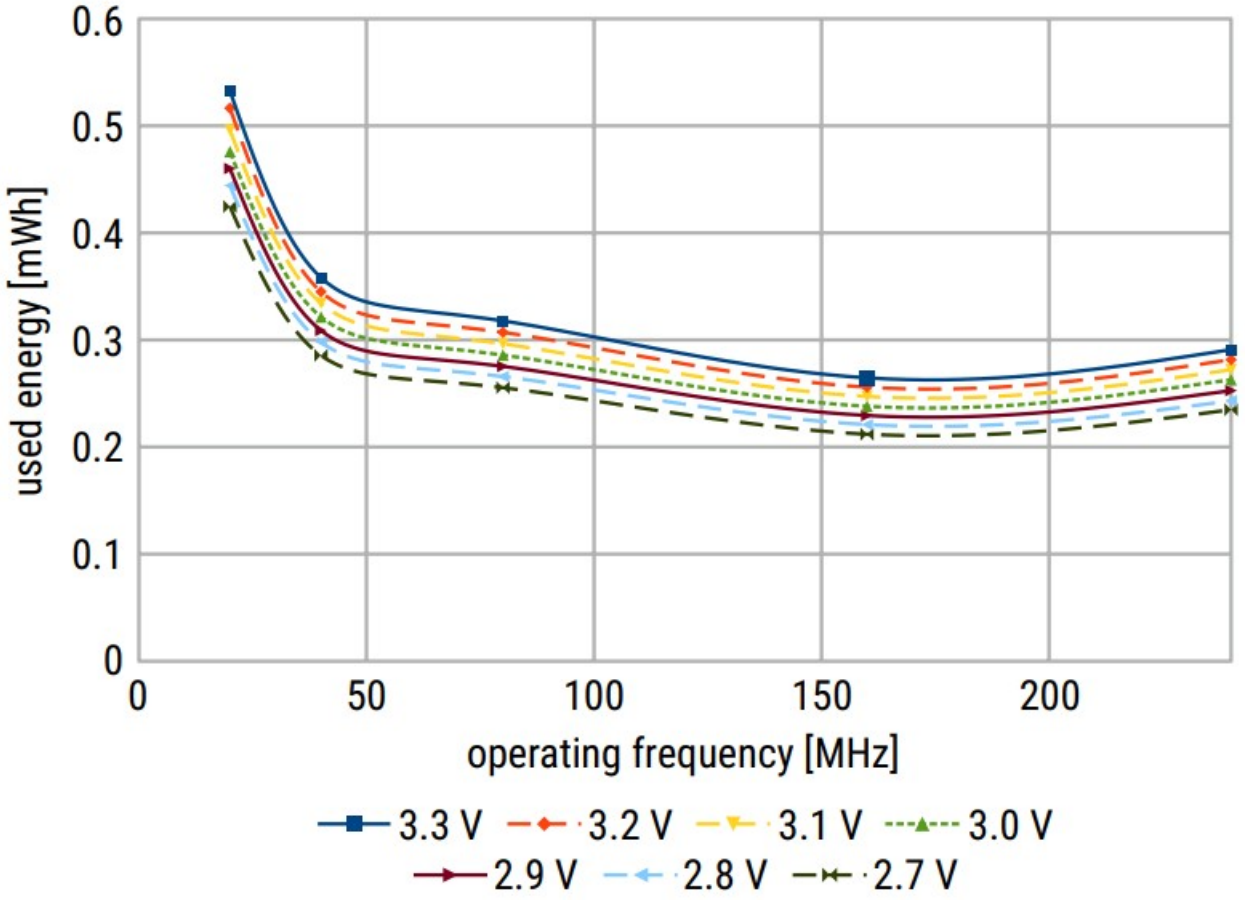
ESP32 Processor Frequency

- dependence of the calculation time on the processor clock frequency



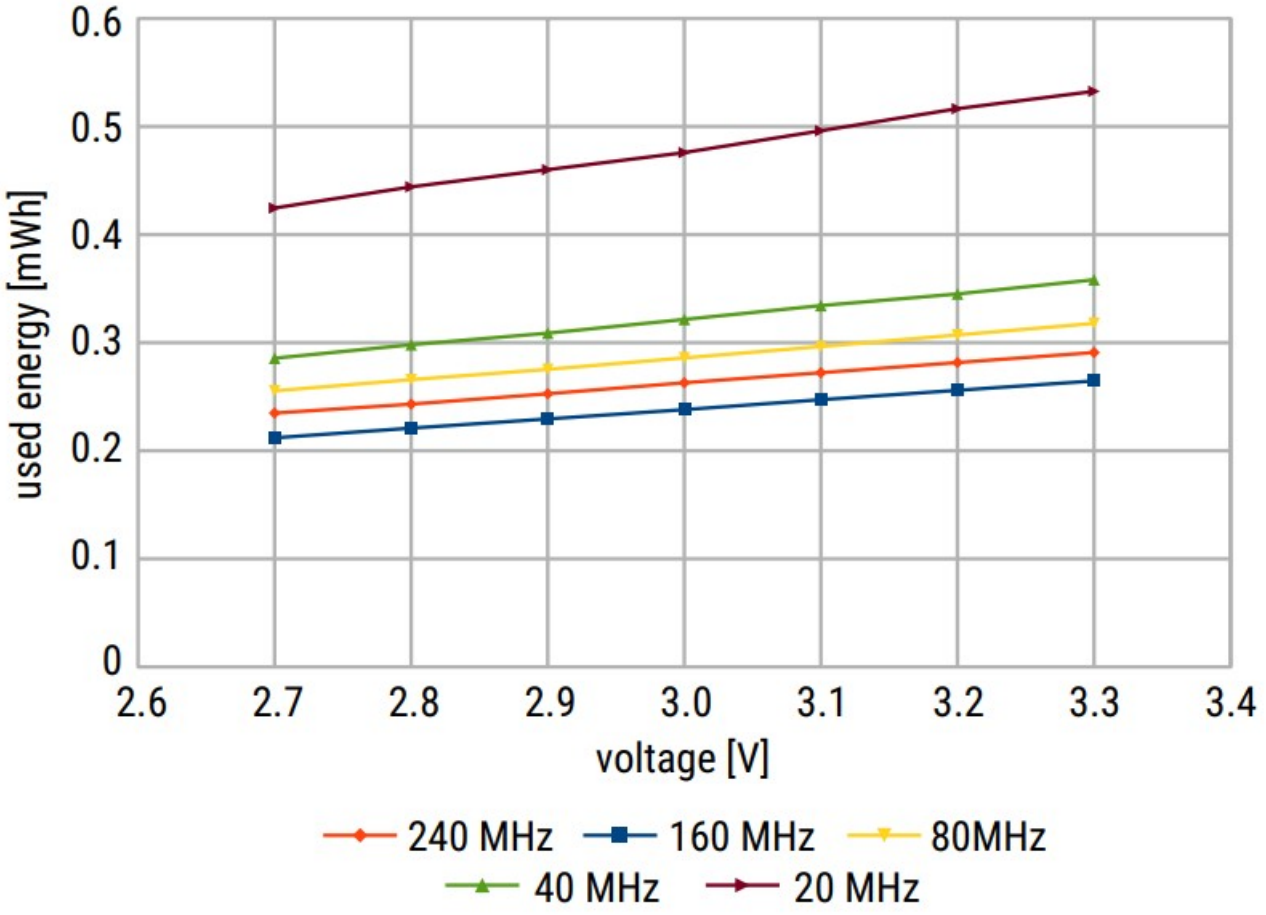
ESP32 Processor Frequency

- used energy on supply voltage on the processor clock frequency



ESP32 Processor Frequency

- energy consumption as a function of supply voltage



ESP32 Processor Frequency

ESP-WROOM-32 module supply current consumption:

CPU frequency	ESP-WROOM-32 supply current
240 MHz	37.7 mA
160 MHz	27.8 mA
80 MHz	21.6 mA
40 MHz	11.9 mA
10 MHz	**8.0 mA**

(The ESP-WROOM-32 module used for these tests is an older version with ESP32 gen 0.)

Algoritmi i njihova izvedba u rač. sustavima

- Što je to algoritam:
  - **algoritam** je konačni niz precizno definiranih, računalno izvedljivih uputa, tipično za rješavanje klase problema ili za izvršavanje računa. Algoritmi su uvijek nedvosmisleni i koriste se kao specifikacije za obavljanje izračuna, obrade podataka, automatiziranog zaključivanja i drugih zadataka.

- Kako izvesti algoritam
  - Jednostavno uzmemo standard i raspišemo ga u nekom višem programskom jeziku

- Stvar će naravno raditi
  - Brzina je upitna!!!

Algoritmi

- Sljedeći korak: provjeriti usko grlo algoritma:
  - Pogledajmo kod JPEG-a
    - DCT/IDCT
    - kvantizacija
    - RGB2YUV i YUV2RGB
    - GetBits (vrlo jednostavna funkcija koja se jako puno puta poziva)

- Prvi način :
  - Standardni pristup:

$$X_{k_1,k_2} = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x_{n_1,n_2} \cos\left[\frac{\pi}{N_1}\left(n_1+\frac{1}{2}\right)k_1\right] \cos\left[\frac{\pi}{N_2}\left(n_2+\frac{1}{2}\right)k_2\right].$$

- Drugi način:
  - DCT/IDCT: da li postoji neki drugi pristup od standardnog
  - Pojednostavljeni algoritam putem FFT ili neki drugi pristup. AAN Algoritam
  - Treba nam pomoć matematike.
  - Problem znanja i primjene

AAN 1D IDCT

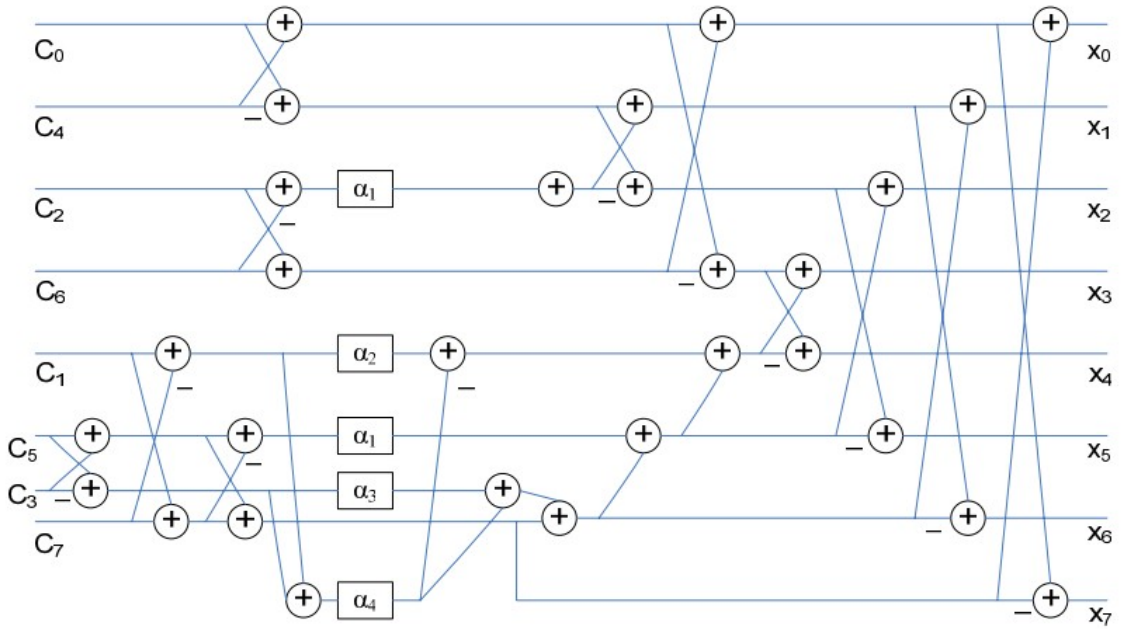


Figure 1. AAN 1-D Eight-point IDCT

RGB-YUV YUV-RGB

- **RGB to YUV Conversion**

$$Y = (0.257 * R) + (0.504 * G) + (0.098 * B) + 16$$
$$V = (0.439 * R) - (0.368 * G) - (0.071 * B) + 128$$
$$U = -(0.148 * R) - (0.291 * G) + (0.439 * B) + 128$$

- **YUV to RGB Conversion**

$$B = 1.164(Y - 16) + 2.018(U - 128)$$
$$G = 1.164(Y - 16) - 0.813(V - 128) - 0.391(U - 128)$$
$$R = 1.164(Y - 16) + 1.596(V - 128)$$

RGB-YUV YUV-RGB

- RGB to YUV cjelobrojni

```
y = (66 * r + 129 * g + 25 * b + 128) >> 8 + 16;
u = (-38 * r - 74 * g + 112 * b + 128) >> 8 + 128;
v = (112 * r - 94 * g - 18 * b + 128) >> 8 + 128;
```

- RGB to YUV cjelobrojni pojednostavljeno

```
y = (66 * r + 4* 32 * g + 25 * b + 128) >> 8 + 16;
u = (-38 * r - 74 * g + 112 * b + 128) >> 8 + 128;
v = (3 * 38 * r – 3 * 32 * g - 18 * b + 128) >> 8 +
128;
```

GetBits

- Problem dohvata bitova iz memorije
- Nije zahtjevan potprogram ali se često poziva i zato može izazvati probleme.
- Nije efikasno dohvaćati bit po bit na 32-bitnom procesoru
- Problem protočne arhitekture
- Problem pričuvne memorije

- Kako računalo računa sinus funkciju?

Aproksimacija funkcija

- Aproksimacija funkcija polinomom n-tog reda...

$$\frac{1}{x} = 1 - (x-1) + (x-1)^2 - (x-1)^3 + (x-1)^4 - .. + (-1)^n (x-1)^n$$

$$\frac{1}{1+x} = 1 - x + x^2 - x^3 + x^4 - x^5 + .. + (-1)^n x^n$$

$$\ln(x) = (x-1) - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} - \frac{(x-1)^4}{4} + .. + \frac{(-1)^{n-1}(x-1)^n}{n}$$

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + .. + \frac{x^n}{n!}$$

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - .. + \frac{(-1)^n x^{2n}}{(2n)!}$$

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - .. + \frac{(-1)^n x^{2n+1}}{(2n+1)!}$$

k_sin.c

```
* Developed at SunSoft, a Sun Microsystems, Inc. business.
 * Permission to use, copy, modify, and distribute this
 * software is freely granted, provided that this notice
 * is preserved.
 * ====================================================
 */

/* __kernel_sin( x, y, iy)
 * kernel sin function on [-pi/4, pi/4], pi/4 ~ 0.7854
 * Input x is assumed to be bounded by ~pi/4 in magnitude.
 * Input y is the tail of x.
 * Input iy indicates whether y is 0. (if iy=0, y assume to be 0).
 *
 * Algorithm
 *  1. Since sin(-x) = -sin(x), we need only to consider positive x.
 *  2. if x < 2^-27 (hx<0x3e400000 0), return x with inexact if x!=0.
 *  3. sin(x) is approximated by a polynomial of degree 13 on
 *     [0,pi/4]
 *                  3            13
 *      sin(x) ~ x + S1*x + ... + S6*x
 *     where
 *
 *  |sin(x)         2     4     6     8     10     12  |     -58
 *  |----- - (1+S1*x +S2*x +S3*x +S4*x +S5*x  +S6*x   )| <= 2
 *  |  x               |
 *
 *  4. sin(x+y) = sin(x) + sin'(x')*y
 *      ~ sin(x) + (1-x*x/2)*y
 *     For better accuracy, let
 *       3      2      2      2      2
 *  r = x *(S2+x *(S3+x *(S4+x *(S5+x *S6))))
 *     then                    3    2
 *  sin(x) = x + (S1*x + (x *(r-y/2)+y))
 */
```

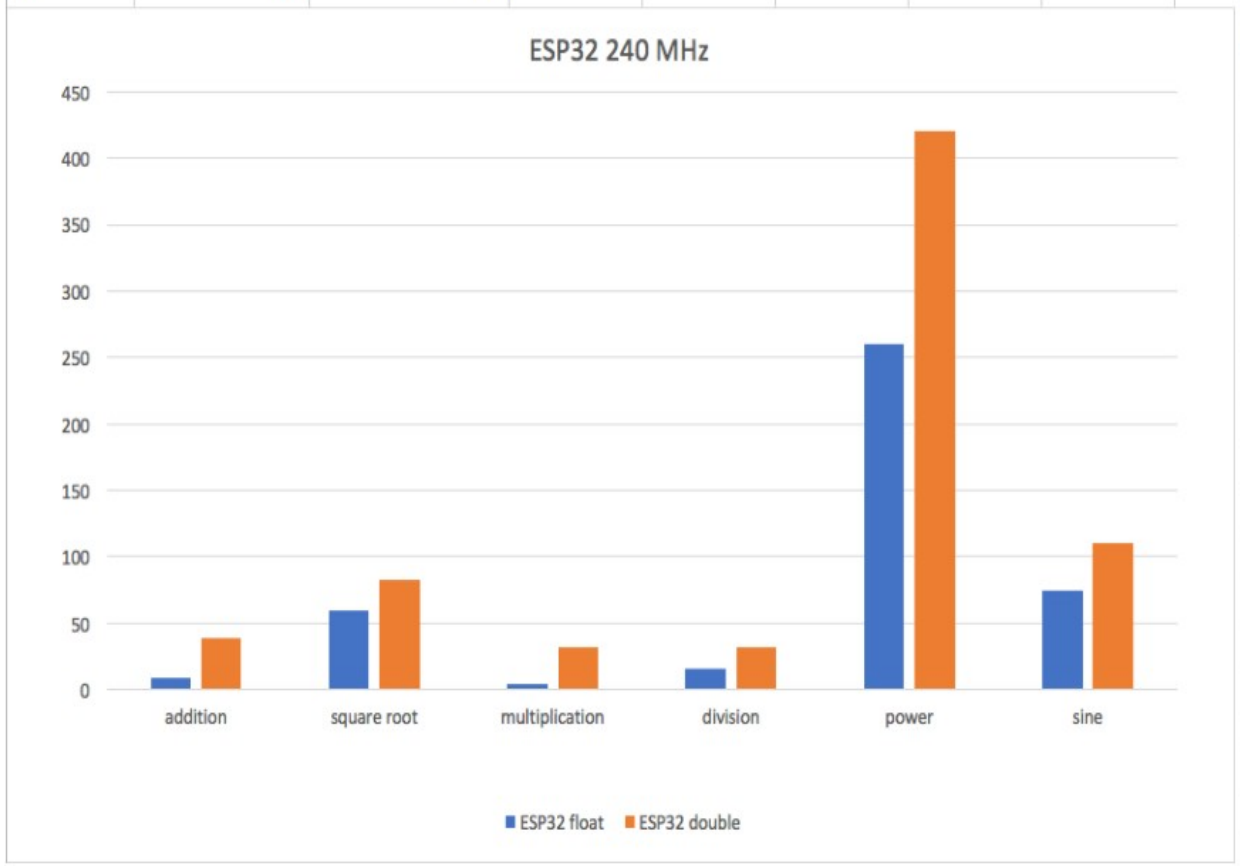
```
#include "fdlibm.h"

#ifdef __STDC__
static const double
#else
static double
#endif
half =  5.00000000000000000000e-01, /* 0x3FE00000, 0x00000000 */
S1  = -1.66666666666666324348e-01, /* 0xBFC55555, 0x55555549 */
S2  =  8.33333333332248946124e-03, /* 0x3F811111, 0x1110F8A6 */
S3  = -1.98412698298579493134e-04, /* 0xBF2A01A0, 0x19C161D5 */
S4  =  2.75573137070700676789e-06, /* 0x3EC71DE3, 0x57B1FE7D */
S5  = -2.50507602534068634195e-08, /* 0xBE5AE5E6, 0x8A2B9CEB */
S6  =  1.58969099521155010221e-10; /* 0x3DE5D93A, 0x5ACFD57C */

#ifdef __STDC__
    double __kernel_sin(double x, double y, int iy)
#else
    double __kernel_sin(x, y, iy)
    double x,y; int iy;       /* iy=0 if y is zero */
#endif
{
    double z,r,v;
    int ix;
    ix = __HI(x)&0x7fffffff;  /* high word of x */
    if(ix<0x3e400000)         /* |x| < 2**-27 */
       {if((int)x==0) return x;}      /* generate inexact */
    z    =  x*x;
    v    =  z*x;
    r    =  S2+z*(S3+z*(S4+z*(S5+z*S6)));
    if(iy==0) return x+v*(S1+z*r);
    else      return x-((z*(half*y-v*r)-y)-v*S1);
}
```

Floating point to fix point

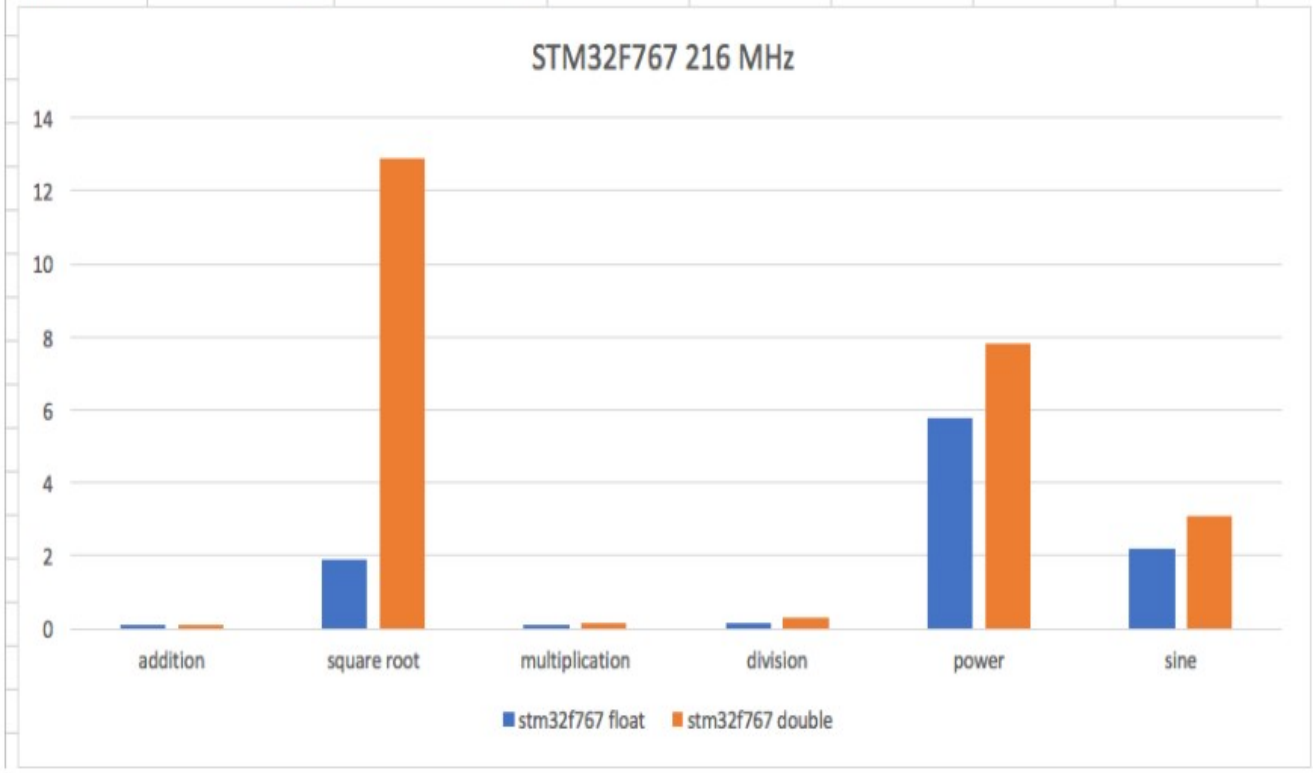
- Prevođenje algoritama iz realne u cjelobrojnu domenu (fix-point)
- Uvelike smanjuje potrošnju i povećava brzinu rada.
- Ideja je pomnožiti varijablu sa potencijom broja 2, vidjeli smo na primjeru YUV-RGB

ESP32 floating point

	ESP32 float	ESP32 double
addition	8.7	39
square root	60	83
multiplicatio	4.1	32
division	16	32
power	260	420
sine	75	110



	stm32f767 float	stm32f767 double
addition	0.12	0.1
square root	1.9	12.9
multiplicatio	0.13	0.18
division	0.17	0.29
power	5.8	7.8
sine	2.2	3.1

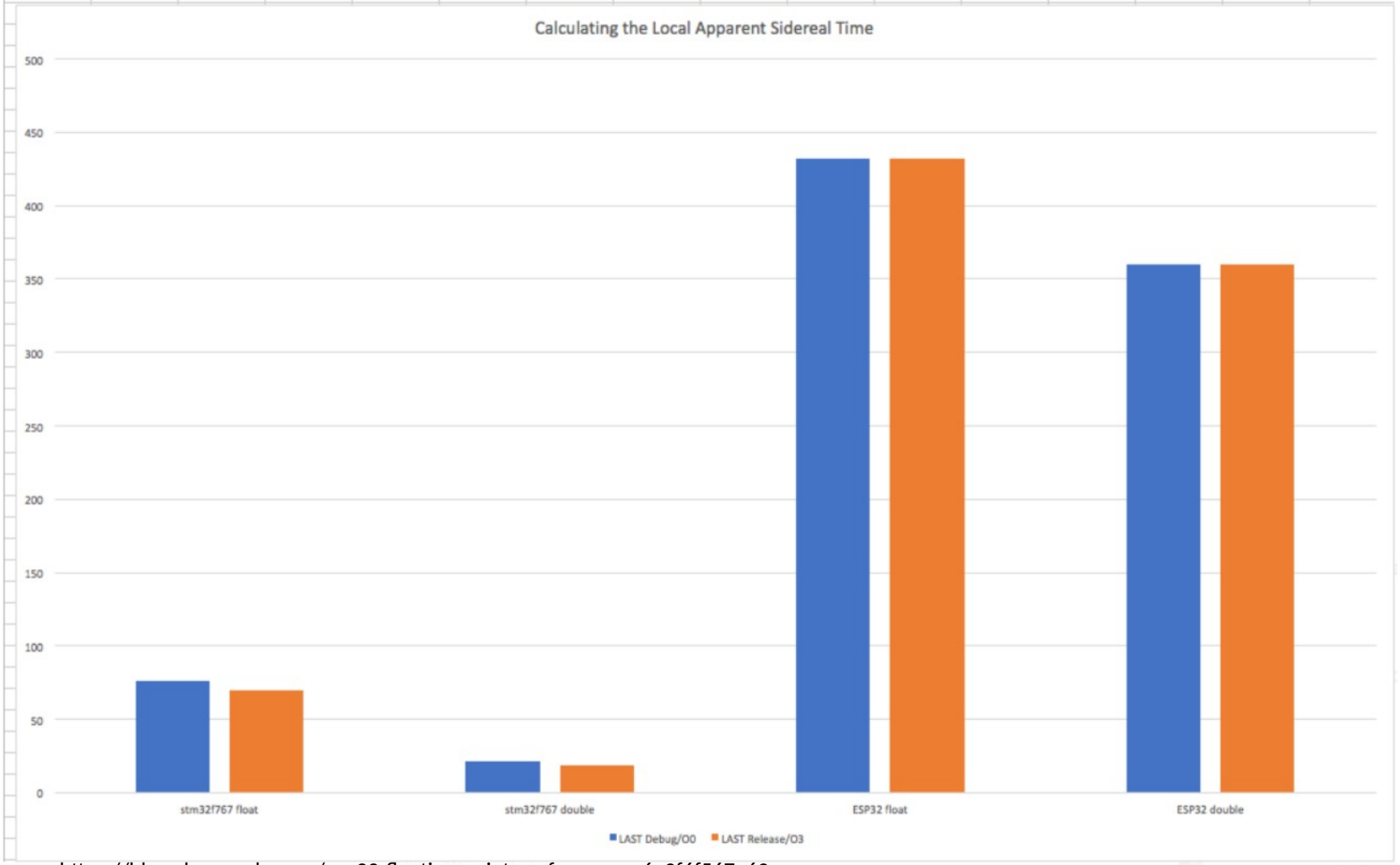


Source: https://blog.classycode.com/esp32-floating-point-performance-6e9f6f567a69

ESP32 floating point

- performance for calculating the local apparent sidereal time according to the formulas in the Astronomical Almanach

	stm32f767 fl	stm32f767 d	ESP32 float	ESP32 double
LAST Debug/	76	21.6	432	360
LAST Release	70	18.4	432	360



Source: https://blog.classycode.com/esp32-floating-point-performance-6e9f6f567a69

Arhitektura procesora

- Proučiti arhitekturu procesora
  - Napisati pretvorbu u strojnom jeziku
  - Možda postoje posebne naredbe u strojnom jeziku koje mogu ubrzati algoritam
    - HITACHI SH4 (množenje matrice 4x4 s vektorom) (‘97)
  - SIMD, ARM NEON, 3D naredbe
  - Voditi računa gdje držite podatke
    - Registar
    - Pričuvna memorija
    - SRAM
    - FLASH