kotyara12.ru

О ESP32 и не только

Удаленный перехват backtrace на ESP32

IoT и микроконтроллеры, ESP-IDF & FreeRTOS

Метки: esp32 FreeRTOS PlatformIO Telegram Умный дом

Иногда уже после сборки и отладки очередного устройства, а затем и установки его “на место”, вдруг начинают проявляться самопроизвольные перезагрузки из-за паники. Это означает, что в прошивке где-то присутствует ранее незамеченная проблема. Возникает задача: определить место возникновения сбоя. Зная файл исходного кода и номер строки, можно с большой степенью вероятности предположить причину проблемы и устранить её. При локальной отладке в случае сбоя паники в COM-порт обычно печатается backtrace, который можно “расшифровать” и выяснить место точку возникновения проблемы. Как это сделать, я писал в предыдущей статье.

```
Core 1 register dump:
PC      : 0x400e14ed  PS      : 0x00060030  A0      : 0x800d0805  A1      :
A2      : 0x00000000  A3      : 0x00000001  A4      : 0x00000001  A5      :
A6      : 0x00000000  A7      : 0x00000001  A8      : 0x00000000  A9      :
A10     : 0x00000000  A11     : 0x3ffb2bac  A12     : 0x40082d1c  A13     :
A14     : 0x3ffb7078  A15     : 0x00000000  SAR     : 0x00000014  EXCCAUSE:
EXCVADDR: 0x00000000  LBEG    : 0x4000c46c  LEND    : 0x4000c477  LCOUNT  :

Backtrace: 0x400e14ed:0x3ffb5030 0x400d0802:0x3ffb5050
```

Осталось получить этот самый backtrace после сбоя. Если доступ к устройству имеется, можно подключиться к USB / COM-порту и проследить вывод логов. Но и при доступе к устройству так же могут быть некоторые сложности – ошибка может возникать раз в неделю или реже, в этом случае “мониторить” устройство через COM-порт становится проблематично. В PlatformIO имеется возможность сохранять отладочный вывод в COM-порт в файл, можно использовать этот способ.

Гораздо хуже, если физического доступа к устройству нет – например, если оно физически расположено в другом населенном пункте или в труднодоступном месте. В таком случае отладка возможна, только если устройство самостоятельно отправит какие-либо данные для отладки в telegram, на mqtt-брокер или куда-либо ещё. Проблема в том,

что на текущий момент ни **framework-espidf**, ни **framework-arduinoespressif32** не предусматривают сохранение backtrace куда-либо, кроме COM-порта. Придется “колхозить” перехват и сохранение backtrace самостоятельно.

Массив для сохранения backtrace

Под сохранение backtrace достаточно выделить сравнительно небольшой массив uint32_t чисел. Размер массива зависит от того, сколько адресов нужно там хранить. Обычно достаточно 10-16 ячеек, но в некоторых особо тяжелых случаях нужно и больше (например если перезагрузка возникает при ошибке выделения памяти, когда куча исчерпана – в этом случае в начале backtrace будут вызовы системных функций выделения памяти).

Я определил этот массив так:

```
#if CONFIG_RESTART_DEBUG_INFO

typedef struct {
  size_t heap_total;
  size_t heap_free;
  size_t heap_free_min;
  time_t heap_min_time;
  #if CONFIG_RESTART_DEBUG_STACK_DEPTH > 0
  uint32_t backtrace[CONFIG_RESTART_DEBUG_STACK_DEPTH];
  #endif // CONFIG_RESTART_DEBUG_STACK_DEPTH
} re_restart_debug_t;

#endif // CONFIG_RESTART_DEBUG_INFO
```

С помощью макроса **CONFIG_RESTART_DEBUG_INFO** можно включать или отключать сохранение отладочной информации, с помощью другого макроса **CONFIG_RESTART_DEBUG_STACK_DEPTH** можно устанавливать размер буфера под сохранение backtrace. Кроме того, сохраняются данные о размере свободной кучи.

```
// EN: Preserve debugging information across device software restarts
// RU: Сохранять отладочную информацию при программном перезапуске уст
#define CONFIG_RESTART_DEBUG_INFO 1
// EN: Allow heap information to be saved periodically, with subsequer
// RU: Разрешить периодическое сохранение информации о куче / памяти с
#define CONFIG_RESTART_DEBUG_HEAP_SIZE_SCHEDULE 1
// EN: Depth to save the processor stack on restart. 0 - do not save
// RU: Глубина сохранения стека процессора при перезапуске. 0 - не сох
```

```
9.  #define CONFIG_RESTART_DEBUG_STACK_DEPTH 28
```

Примечание: здесь и далее все управляющие макросы определены в файле project_config.h, который есть в любом моем проекте ESP-IDF. Пример файла можно посмотреть по ссылке: github.com/kotyara12/dzen_autowatering/blob/master/include/project_config.h. Но вы можете задать их любым другим способом, например в CMakeLists.txt

Возникает вопрос в том, где сохранить эти данные при перезагрузке.

Первая мысль была сохранить необходимые данные во flash памяти, например в NVS-разделе. Но, увы, это не всегда прокатывает, так как не удается выполнить запись. Поэтому пришлось найти другое решение: пометил переменную, хранящую данные для отладки модификатором **__NOINIT_ATTR**. Это заставит компилятор поместить данный блок данных в область, которая не очищается при перезагрузке. Конечно, при выключении и включении устройства данные будут утеряны в любом случае, но в этом случае они не требуются.

```
1.  #if CONFIG_RESTART_DEBUG_INFO
2.
3.  __NOINIT_ATTR static re_restart_debug_t _debug_info;
4.
5.  #endif // CONFIG_RESTART_DEBUG_INFO
```

Заполнение данных для backtrace

Как я уже упомянул выше, в ESP-IDF не предусмотрен вывод backtrace куда-либо ещё, кроме com-порта. Что ж, придется сделать это самостоятельно. Для этого я воспользовался функцией **esp_backtrace_print** из файла **\.platformio\packages\framework-espidf\components\xtensa\debug_helpers.c**, и написал похожий вариант:

```
1.  #if CONFIG_RESTART_DEBUG_INFO && (CONFIG_RESTART_DEBUG_STACK_DEPTH > 0
2.  #include "esp_types.h"
3.  #include "esp_attr.h"
4.  #include "esp_err.h"
5.  #include "esp_debug_helpers.h"
```

```
#include "soc/soc_memory_layout.h"
#include "soc/cpu.h"
#endif // CONFIG_RESTART_STACK_DEPTH

#if CONFIG_RESTART_DEBUG_INFO

void IRAM_ATTR debugHeapUpdate()
{
  _debug_info.heap_total = heap_caps_get_total_size(MALLOC_CAP_DEFAULT
  _debug_info.heap_free = heap_caps_get_free_size(MALLOC_CAP_DEFAULT);
  size_t _new_free_min = heap_caps_get_minimum_free_size(MALLOC_CAP_DE
  if ((_debug_info.heap_free_min == 0) || (_new_free_min < _debug_info
    _debug_info.heap_free_min = _new_free_min;
    _debug_info.heap_min_time = time(nullptr);
  };
}

#if CONFIG_RESTART_DEBUG_STACK_DEPTH > 0

void IRAM_ATTR debugBacktraceUpdate()
{
  esp_backtrace_frame_t stk_frame;
  esp_backtrace_get_start(&(stk_frame.pc), &(stk_frame.sp), &(stk_fram
  _debug_info.backtrace[0] = esp_cpu_process_stack_pc(stk_frame.pc);

  bool corrupted = (esp_stack_ptr_is_sane(stk_frame.sp) &&
                    esp_ptr_executable((void*)esp_cpu_process_stack_pc
                    false : true;

  #if CONFIG_RESTART_DEBUG_STACK_DEPTH > 1
    uint8_t i = CONFIG_RESTART_DEBUG_STACK_DEPTH;
    while (i-- > 0 && stk_frame.next_pc != 0 && !corrupted) {
      if (!esp_backtrace_get_next_frame(&stk_frame)) {
        corrupted = true;
      };
      _debug_info.backtrace[CONFIG_RESTART_DEBUG_STACK_DEPTH - i] = es
    };
  #endif // CONFIG_RESTART_DEBUG_STACK_DEPTH > 1
}

#endif // CONFIG_RESTART_DEBUG_STACK_DEPTH

void IRAM_ATTR debugUpdate()
{
  debugHeapUpdate();
  #if CONFIG_RESTART_DEBUG_STACK_DEPTH > 0
  debugBacktraceUpdate();
  #endif // CONFIG_RESTART_DEBUG_STACK_DEPTH
}

#endif // CONFIG_RESTART_DEBUG_INFO
```

Для сохранения отладочной информации теперь достаточно вызывать **debugUpdate()**. При сохранении данных stk_frame.sp не сохраняется, так как это особо не влияет на

расшифровку адресов. Осталось обеспечить вызов этой функции перед перезагрузкой и в случае паники.

Сохранение backtrace перед перезагрузкой устройства

Реализовать сохранение данных при “штатной” перезагрузке устройства командой **esp_restart** очень просто. Для этого достаточно зарегистрировать обработчик “штатной” перезагрузки. Всего можно использовать от 1 до 5 обработчиков, и это число, увы, изменить нельзя.

```
#if CONFIG_RESTART_DEBUG_INFO

esp_register_shutdown_handler(debugUpdate);

#endif // CONFIG_RESTART_DEBUG_INFO
```

Это обеспечит сохранение отладочной информации в зарезервированной области памяти при перезагрузке из-за обновления OTA или в других “нормальных” ситуациях.

Но, увы, при “панике” из-за ошибке метод не сработает. Решить эту задачу для ESP-IDF можно, создав wrapper для системной функции **esp_panic_handler** с помощью опции -Wl. Для этого создаем обертку для **esp_panic_handler**:

```
// This function will be considered the esp_panic_handler to call in c
void __wrap_esp_panic_handler(void* info)
{
  #if CONFIG_RESTART_DEBUG_INFO && (CONFIG_RESTART_DEBUG_STACK_DEPTH >
    debugBacktraceUpdate();
  #endif // CONFIG_RESTART_DEBUG_STACK_DEPTH

  // Call the original panic handler function to finish processing thi
  __real_esp_panic_handler(info);
}
```

Добавление обертки в ESP-IDF

Для того, чтобы заставить наш обработчик паники работать, необходимо в файл **CMakeLists.txt** добавить строку `idf_build_set_property(LINK_OPTIONS "-Wl,--wrap=esp_panic_handler" APPEND)`:

```
cmake_minimum_required(VERSION 3.16.0)
include($ENV{IDF_PATH}/tools/cmake/project.cmake)
idf_build_set_property(LINK_OPTIONS "-Wl,--wrap=esp_panic_handler" APP
project(village_garage)
```

Это заставит компилятор вызвать нашу функцию **esp_panic_handler** вместо “штатной”, а затем уже наша функция вызовет встроенную. Вуаля!

Добавление обертки в Arduino

Решение для фреймворка Arduino32 подсказал один из читателей, но работает оно только в PlatformIO. Достаточно добавить в platformio.ini строчку:

```
[env:esp32dev]
platform = espressif32
board = esp32dev
framework = arduino
build_flags = -Wl,--wrap=esp_panic_handler
```

Отправка backtrace после перезагрузки устройства

Осталось только отправить эти данные после перезагрузки куда-нибудь на сервер. Для этого необходимо считать из сохраненной ранее области и из системных функций:

```
re_restart_debug_t debugGet()
{
  re_restart_debug_t ret;
  memset(&ret, 0, sizeof(re_restart_debug_t));
  esp_reset_reason_t esp_reason = esp_reset_reason();
  if ((esp_reason != ESP_RST_UNKNOWN) && (esp_reason != ESP_RST_POWERO
    ret = _debug_info;
    if (_debug_info.heap_total > heap_caps_get_total_size(MALLOC_CAP_D
      memset(&ret, 0, sizeof(re_restart_debug_t));
    };
  };
  memset(&_debug_info, 0, sizeof(re_restart_debug_t));
  return ret;
}
```

Теперь можно отправить эти данные разработчику. Например можно это можно сделать, опубликовав эти данные через mqtt-брокер с флагом retained. А можно отправить в telegram, добавив backtrace к информации о запуске устройства, вот мои шаблоны сообщений при “нормальном” и “аварийном” перезапуске:

```
#define CONFIG_MESSAGE_TG_VERSION_DEF "🔸 <b>Устройство запущено</b>\r

#define CONFIG_MESSAGE_TG_VERSION_TRACE "🔸 <b>Устройство запущено</b>
```

Заливаем прошивку в устройство. Теперь при “программной перезагрузке” или сбое получим примерно такое сообщение:

```
🔸 Устройство запущено

Версия прошивки:     20220320.039
Причина перезапуска: EXCEPTION / PANIC
CPU0:  Software reset CPU
CPU1:  Software reset CPU
HEAP:  272800 : 104648 (38.4%) : 57644 (21.1%) 21.03.2022 07:30:00
TRACE: 0x400d39bf 0x40089792 0x40089855 0x40082d16 0x400014fa 0x4014fb51 0x
```

Теперь можно скопировать эти данные в attr2line и расшифровать место возникновения ошибки. Задача решена.

🔹 Полный архив статей вы найдете здесь

14 комментариев для “Удаленный перехват backtrace на ESP32”

МИР

23.04.2024 в 11:17

Здорово! А откуда берется APP_VERSION?

Ответить

KOTYARA12

23.04.2024 в 19:43

#define APP_VERSION “20240417.001” в project_config.h

Ответить

МИР

24.04.2024 в 06:23

Вручную чтоли или генерируется автоматом?

Ответить

KOTYARA12

24.04.2024 в 19:18

Чаще вручную, но при массовой компиляции проектов автоматом

Ответить

FILIP

10.05.2024 в 12:49

very useful guide, I tried to use your idea in my project. I created a person file composed of the code presented on the blog. However, when trying to panic (divide by 0) I still get ESP-IDF’s built-in panic handling. Do you have any idea what else is wrong? Of course, I added idf_build_set_property(LINK_OPTIONS “-Wl,–wrap=esp_panic_handler” APPEND) to CmakeLists

Ответить

KOTYARA12

10.05.2024 в 13:12

The built-in panic handler is not disabled. The custom handler acts only as an additional tool.

Additionally, you must ensure that the data remains in memory between reboots. Perhaps you forgot to change the buffer for storing data as __NOINIT_ATTR static???

Ответить

FILIP

10.05.2024 в 16:33

.C file:
#include “backtrace_stats.h”

#if CONFIG_RESTART_DEBUG_INFO && (CONFIG_RESTART_DEBUG_STACK_DEPTH > 0)
#include “esp_types.h”
#include “esp_attr.h”
#include “esp_err.h”
#include “esp_debug_helpers.h”
#include “soc/soc_memory_layout.h”
//#include “soc/cpu.h”
#endif // CONFIG_RESTART_STACK_DEPTH

#include “esp_system.h”
#include “esp_heap_caps.h”
#include “esp_cpu_utils.h”

#if CONFIG_RESTART_DEBUG_INFO
__NOINIT_ATTR static re_restart_debug_t _debug_info;
#endif // CONFIG_RESTART_DEBUG_INFO

#if CONFIG_RESTART_DEBUG_INFO

void IRAM_ATTR debugHeapUpdate()
{
_debug_info.heap_total = heap_caps_get_total_size(MALLOC_CAP_DEFAULT);
_debug_info.heap_free = heap_caps_get_free_size(MALLOC_CAP_DEFAULT);
size_t _new_free_min = heap_caps_get_minimum_free_size(MALLOC_CAP_DEFAULT);
if ((_debug_info.heap_free_min == 0) || (_new_free_min 0

```
void IRAM_ATTR debugBacktraceUpdate()
{
esp_backtrace_frame_t stk_frame;
esp_backtrace_get_start(&(stk_frame.pc), &(stk_frame.sp), &(stk_frame.next_pc));
_debug_info.backtrace[0] = esp_cpu_process_stack_pc(stk_frame.pc);

bool corrupted = (esp_stack_ptr_is_sane(stk_frame.sp) &&
esp_ptr_executable((void*)esp_cpu_process_stack_pc(stk_frame.pc))) ?
false : true;

#if CONFIG_RESTART_DEBUG_STACK_DEPTH > 1
uint8_t i = CONFIG_RESTART_DEBUG_STACK_DEPTH;
while (i– > 0 && stk_frame.next_pc != 0 && !corrupted) {
if (!esp_backtrace_get_next_frame(&stk_frame)) {
corrupted = true;
};
_debug_info.backtrace[CONFIG_RESTART_DEBUG_STACK_DEPTH – i] =
esp_cpu_process_stack_pc(stk_frame.pc);
};
#endif // CONFIG_RESTART_DEBUG_STACK_DEPTH > 1
}

#endif // CONFIG_RESTART_DEBUG_STACK_DEPTH

void IRAM_ATTR debugUpdate()
{
debugHeapUpdate();
#if CONFIG_RESTART_DEBUG_STACK_DEPTH > 0
debugBacktraceUpdate();
#endif // CONFIG_RESTART_DEBUG_STACK_DEPTH
}

#endif // CONFIG_RESTART_DEBUG_INFO

/* Declare the real panic handler function. We'll be able to call it after executing our custom
code */
void __real_esp_panic_handler(void*);

// /* This function will be considered the esp_panic_handler to call in case a panic occurs */
void __wrap_esp_panic_handler (void* info) {
// /* Custom code, count the number of panics or simply print a message */
printf("FROM OWN HANDLER\n");
```

```
/* Call the original panic handler function to finish processing this error (creating a core dump for example…) */
__real_esp_panic_handler(info);
}
```

CMakeLists:

```
cmake_minimum_required(VERSION 3.5)

set(V_MAJOR 0)
set(V_MINOR 11)
set(V_REVISION 3)
add_definitions(-DCODE_NAME="Added own panic handler")
add_definitions(-DCODE_PREFIX="feat")
add_definitions(-DCODE_DATE="10/05/2024")

add_definitions(-DSW_VERSION_MAJOR=${V_MAJOR})
add_definitions(-DSW_VERSION_MINOR=${V_MINOR})
add_definitions(-DSW_VERSION_REVISION=${V_REVISION})

set(PROJECT_VER "${V_MAJOR}.${V_MINOR}.${V_REVISION}")

include($ENV{IDF_PATH}/tools/cmake/project.cmake)
project(my_project)

idf_build_set_property(LINK_OPTIONS "-Wl,–wrap=esp_panic_handler" APPEND)
```

That's look my code. If I thinking right now on my terminal I should find test from printf in function __wrap_esp_panic_handler, but this doesn't happend

Ответить

KOTYARA12

10.05.2024 в 18:46

Where are these macros defined???

```
// EN: Preserve debugging information across device software restarts
// RU: Сохранять отладочную информацию при программном перезапуске устройства
#define CONFIG_RESTART_DEBUG_INFO 1
// EN: Allow heap information to be saved periodically, with subsequent output on restart
// RU: Разрешить периодическое сохранение информации о куче / памяти с последующем
```

выводом при перезапуске
#define CONFIG_RESTART_DEBUG_HEAP_SIZE_SCHEDULE 1
// EN: Depth to save the processor stack on restart. 0 – do not save
// RU: Глубина сохранения стека процессора при перезапуске. 0 – не сохранять
#define CONFIG_RESTART_DEBUG_STACK_DEPTH 28

Ответить

ANTOINE
07.07.2024 в 16:51

Note for future people that will use this tutorial using C++: you need to declare __real_esp_panic_handler and __wrap_esp_panic_handler as extern “C”. i.e.
extern “C” void __real_esp_panic_handler(void* info);

extern “C” void __wrap_esp_panic_handler(void* info) {…}

Ответить

PETER
16.08.2024 в 15:25

Does your method work in Arduino framework also?

Ответить

KOTYARA12
16.08.2024 в 21:13

As far as I know – yes. But you need to use PlatformIO, not Arduino IDE. How to do it in Arduino IDE – I don’t know.

Ответить

PETER
23.08.2024 в 11:01

I could catch the exception using your code. But unfortunately there is no heap data, only backtrace: (I stored it in a json file on my on site SD card):

{"Year":24,"Month":8,"Day":20,"Hour":8,"Minute":4,"Second":50,"Boot reason":4,"heap_total":0,"heap_free":0,"heap_free_min":0,"heap_min_time":0,"backtrace0":"0x400D68AF","backtrace1":"0x400F7237","backtrace2":"0x40083675","backtrace3":"0x4008513C","backtrace4":"0x400DEBF8","backtrace5":"0x400DF51F","backtrace6":"0x400DF537","backtrace7":"0x400DF555","backtrace8":null}

And this is the decoded version (I am searching a bug in AsyncWebServer because it crashes randomly):

```
d:\ZMonitorV4\SW\ZMonitor\zmonitor\Panic>xtensa-esp32-elf-addr2line.exe -pfiaC -e firmware.elf 0x400D68AF 0x400F7237 0x40083675 0x4008513C 0x400DEBF8 0x400DF51F 0x400DF537 0x400DF555
0x400d68af: __wrap_esp_panic_handler at C:\B+N\ZMonitorV4\SW\Munka\zmonitor/src/ZMonitor.cpp:1567
0x400f7237: panic_handler at /home/runner/work/esp32-arduino-1ib-builder/esp32-arduino-lib-builder/esp-idf/components/esp_system/port/panic_handler.c:188
0x40083675: xt_unhandled_exception at /home/runner /work/esp32-arduino-lib-builder /esp32-arduino-1ib-builder/esp-idf/components/esp_system/port/panic_handler.c:219
0x4008513c: _xt_user_exc at /home/runner/work/esp32-arduino-1ib-builder/esp32-arduino-lib-builder/esp-idf/components/freertos/port/xtensa/xtensa_vectors.S:703
0x400debf8: AsyncWebSocket: :_cleanBuffers() at C:\B+N\ZMonitorV4\SW\Munka\zmonitor/.pio/libdeps/zmon/ESP Async WebServer/src/AsyncWebSocket.cpp:1229 (discriminator 2)
0x400df51f: AsyncWebSocket: :textAll (AsyncWebSocketMessageBuffer*) at C:\B+N\ZMonitorV4\SW\Munka\zmonitor/.pio/libdeps/zmon/ESP Async WebServer/src/AsynchlebSocket.cpp:962
0x400df537: AsyncWebSocket: :textAll(char const*, unsigned int) at C:\B+N\ZMonitorV4\sW\Munka\zmonitor/.pio/libdeps/zmon/ESP Async WebServer/src/AsyncebSocket.cpp:968
0x400df555: AsyncWebSocket: :textAll (String const&) at C:\B+N\ZMonitorV4\SW\Munka\zmonitor/.pio/libdeps/zmon/ESP Async WebServer/src/AsyncWebSocket.cpp:1109
```

Could you help me to get heap information also?

Ответить

PÉTER

28.08.2024 в 16:40

Hi,
Everything is ok now.

It is a good solution!

Thanks!

Ответить

СЕРГЕЙ

06.10.2024 в 19:43

Добрый день. Добавьте пример пожалуйста, я не могу заставить это работать.

Ответить

Добавить комментарий

Ваш адрес email не будет опубликован. Обязательные поля помечены *

Комментарий *

Имя *

Email *

Сайт

Сохранить моё имя, email и адрес сайта в этом браузере для последующих моих комментариев.

Решите Капчу*

Новые статьи

Разделы

Комментарии

kotyara12 к записи Как и на чём программировать ESP32 и ESP8266

Павел к записи Как и на чём программировать ESP32 и ESP8266

362 к записи Плата Wireless-Tag WT32-ETH01

Вадим к записи Временные интервалы и таймеры на ESP32

Василий к записи ESP32 MQTT Client API

Виталий к записи Kincony KC868-A16 – контроллер для домашней автоматизации

kotyara12 к записи Монитор качества воздуха с ИК-датчиком CO2

Метки

3D модели (1) @fl_monitor_bot (2) Arduino (24) C/C++ (2) Datasheets (2) Delphi (12) ESP-IDF (58) esp32 (89) ESP8266 (24) Espressif IDE (1) Ethernet (1) Falcon Eye (3) FreeRTOS (13) I2C (5) Keenetic (2) Mosquitto (1) MQTT (12) MS SQL (2) Open Monitoring (4) OTA updates (4) PlatformIO (12) PWM (1) rs485 (10) RX433 (1) SoftLoto (3) Telegram (6) ThingSpeak (3) VSCode (6) Автополив (4) Базы данных (10) Для студентов (10) Медиабиблиотека (1) Народный мониторинг (3) ОПС (3) Обзоры (14) Парсеры (3) Переводы (2) Прошивка k12 (5) CO2 (2) Сенсоры (12) Теплица (5) Умный дом (51) Утилиты (2) Утилиты для Excel (2) Электроника (29)

Контакты

- **Канал в Telegram** - новости и анонсы статей
- **Чат в Telegram** - здесь вы можете задать свои вопросы
- **Электронная почта** - написать письмо автору

Счетчики

Авторизация

Вход | Регистрация

Поддержать сайт

Если Вам понравились статьи, Вы можете поддержать автора добровольным пожертвованием на любую приемлемую для вас сумму.

Подарить

Примечание: перевод по кнопке, увы, работает только для личных (не зарплатных) карт - таковы правила ЮМоней

Популярное

ESP32: чипы, модули, платы…
18.11.2022

Как и на чём программировать ESP32 и ESP8266
10.12.2023

Переползаем с Arduino IDE на VSCode + PlatformIO
21.04.2021

Временные интервалы и таймеры на ESP32
15.11.2022

ESP32 SoC :: datasheet на русском (перевод)
02.04.2025

Датчики температуры и влажности для Arduino
31.12.2020

Работа с шиной RS485 и протоколом Modbus RTU на ESP32
16.04.2024

Последние записи

Расширитель GPIO R4IOI16 для шины RS485
09.05.2025

ESP32 MQTT Client API
01.05.2025

Что такое MQTT ~~и с чем его едят~~?
28.04.2025

Монитор качества воздуха с ИК-датчиком CO2
26.04.2025

Плата Wireless-Tag WT32-ETH01
15.04.2025

ESP32 LEDC PWM-контроллер
11.04.2025

ESP32 SoC :: datasheet на русском (перевод)
02.04.2025

Последние комментарии

kotyara12 к записи Как и на чём программировать ESP32 и ESP8266

Павел к записи Как и на чём программировать ESP32 и ESP8266

362 к записи Плата Wireless-Tag WT32-ETH01

Вадим к записи Временные интервалы и таймеры на ESP32

Василий к записи ESP32 MQTT Client API

Виталий к записи Kincony KC868-A16 – контроллер для домашней автоматизации

kotyara12 к записи Монитор качества воздуха с ИК-датчиком CO2

Наум Слабый к записи Монитор качества воздуха с ИК-датчиком CO2

Последние оценки

Голосовать **5** из **anonymous** на **Как и на чём программировать ESP32 и ESP8266**

Голосовать **5** из **anonymous** на **Контроллер ЙоТик32 v2.0B: обзор и функциональные возможности**

Голосовать **5** из **anonymous** на **ESP32: чипы, модули, платы…**

Голосовать **5** из **anonymous** на **Расширитель GPIO R4IOI16 для шины RS485**

Голосовать **5** из **anonymous** на **Как и на чём программировать ESP32 и ESP8266**