بسم الله الرحمن الرحیم

هدیه به امام زمان عج و نائب بر حقش امام خامنه ای

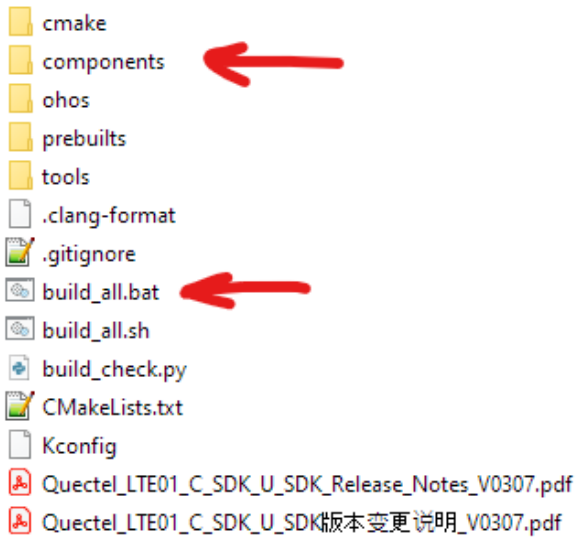
نحوه آماده سازی فایل ها و کتابخانه های پروژه شخصی به SDK QuecOpen

اطلاعات اولیه

- sdk QuecOpen از RTOS (سیستم عامل Real Time) استفاده میکنه.

- وقتی sdk QuecOpen را از حالت فشرده خارج کنید با موارد زیر روبرو میشید (به تاریخ 1405.01.21 یا به ورژن LTE01R03A07_C_SDK_U):

بطور عادی با فولدر components و فایل build_all.bat کار داریم.



- این sdk برای بیلد (Build) شدن از cmake استفاده میکنه.(بیلد یا کامپایل شدن البته صحیحش همان بیلد شدن است).

- برای بیلد شدن آموزش جدایی گذاشتم ، برای همین اینجا خیلی خلاصه توضیحش میدم:

در ویندوز برای بیلد کردن از فایل build_all.bat استفاده میکنیم:
cmd ویندوز را در حالت administrator اجرا میکنیم، به آدرس ابتدای sdk که عکسش رو در بالا گذاشتم میرید، مثلا اگر sdk در فولدری بنام prj_sdk در ابتدای ریشه درایو D باشد با دستور زیر در cmd به آدرسش میریم:

```
Cd /d D:\prj_sdk
```

و سپس دستور

```
build_all.bat new EC200UEU_AA V01
```

را مینویسیم و بعد Enter میزنیم.

اگر عملیات بیلد کردن درست پیش بره با کلمه Pass در انتهای عملیات که حدود 30 ثانیه برای sdk خام زمان لازم داره روبرو میشویم.

البته با کمک فایل اسکریپت که نحوه نوشتنش رو در همان فایل آموزش بیلد کردن گذاشتم میتوانید با دبل کلیک روی آن فایل اقدام به بیلد کردن sdk کنید.

- ما فایل ها و توابع و کتابخانه های مورد نیاز خودمون رو باید در فولدر components و در جای مناسب قرار دهیم.

- جای مناسب:

دو راه برای انتخاب جای مناسب هست: 1- دربین فایل ها و فولدر موجود 2- یک فولدر مخصوص فایل ها و فولدر های شخصی و مورد نیاز پروژه خودمان

- راه دوم مناسب پروژه های استاندارد است. و در اینجا تمرکز روی آموزش شیوه دوم است.

- پروژه ها یا سوپر لوپ هستن یا Real Time (بلادرنگ) (الان به بقیه حالات کار نداریم) در سوپر لوپ مثل اکثر برنامه های آردوینو یا میکروکنترلری یک حلقه while(1) داریم ، که در تابع main هست و حلقه اصلی برنامه است ، و ما توابع پروژه رو داخل این حلقه بی نهایت میزاریم، اما در برنامه های Real Time چندین حالت هست که ساده ترین و بصورت کلی بیان میکنم، بعد از بالا اومدن cpu ، توسط یک تابع خاص محلی هست که توابع پروژه رو اجرا میکنیم با تفاوت هایی:

در Real Time بجای اینکه توابع مورد نیاز در یک حلقه یا تابع باشد که مدام تکرار شوند، توابع مورد نیاز برای هر بخش دستگاه که معمولا (بهتره) به کمتر از 5 بخش تقسیم میشن که به هر بخش Task یا وظیفه میگن. مثلا Task کنترل پاور ، task سیستم، Task برنامه اصلی و از این قبیل. این چند وظیفه که کل پروژه رو به عهدشون هست، درتابع خاص اجرا کننده Task ها یکبار اجرا میشن. سپس تابع خاص در انتهای خودش حذف میشود. Task ها ساخته میشن و میتوان هر کدام را که خواستیم نیز حذف کنیم. بیشتر از این وارد عملکرد سیستم عامل های Real Time نمیشم.

پس ما برای اجرای شیوه دوم که استاندارد است، و با توجه به اینکه باید پروژه رو Real Time اجرا کنیم، به موارد زیر نیاز داریم:

1- فولدر پروژه های sdk QuecOpen: که فولدر components است

2- تابع خاص شروع کننده Task ها: که در آدرس زیر هست:

Components -> ql-application -> init -> ql_init.c

3- یک فولدر که تمامی فایل ها و فولدر ها و کتابخانه های مورد نیاز پروژه خودمون داخلش قرار میگیره و آدرسش باید در فولدر components باشد. هر اسمی میتواند داشته باشد و من اسمش رو گذاشتم user_code_base.

4- تعدادی فایل CmakeLists.txt که دو مورد از قبل وجود دارد و چند مورد باید بسازیم و تنظیم کنیم.

انجام مراحل آماده سازی

1- در فولدر components یک فایل CmakeLists.txt هست که باید دستور زیر رو بهش اضافه کنیم و محل مناسب آن زیر بخشی است که "ql-application " رو اضافه میکند. مطابق زیر:

```
add_subdirectory_if_exist(ql-application)

add_subdirectory_if_exist(ql-kernel)

add_subdirectory_if_exist(user_code_base)
```

<- بخشی که باید اضافه کنیم

و حالا کارمون با این فایل تمام است.

2- به آدرس زیر میرویم:

Components -> ql-application -> init

دو فایل در این فولدر هست که باهاشون کار داریم:

فایل CMakeLists.txt

فایل ql_init.c

برای تنظیم کردن CMakeLists.txt:

در بخش زیر:

```
if(NOT QL_PROJECT_MIXER AND NOT CONFIG_QUEC_PROJECT_FEATURE_VSIM_ADAPT_STD)
```

بعد از

```
set(target ${QL_APP_BUILD_VER})
```

بعد از

```
    if(CONFIG_APPIMG_LOAD_FLASH)
        add_appimg_flash_ql_example(${target} ql_init.c)
    endif()
    if(CONFIG_APPIMG_LOAD_FILE)
        add_appimg_file_ql_example(${target} ql_init.c)
    endif()
```

بعد از

```
target_link_libraries(${target} PRIVATE ql_app_nw ql_app_peripheral ql_app_osi ql_app_dev ql_app_sim ql_app_power)
```

باید اینجا دستور زیر رو اضافه کنیم:

```
target_link_libraries(${target} PRIVATE user_code_base_lib)
```
بخشی که باید اضافه کنیم <-

قبل از

```
if(QL_APP_FEATURE_USB)
    target_link_libraries(${target} PRIVATE ql_app_usb)
endif()
```

در فایل ql-init.c تغییرات زیر را میدیم:

- اضافه کردن هدر فایل توابعی که درواقع task های مورد نظرمون هستن.

- اضافه کردن توابع.

اضافه کردن هدر فایل :

بهتره که در انتهای آخرین هدر فایلی که include شده است ما هدر های خودمون رو اضافه کنیم، مثلا:

```
#include "user_code/puc_gpio.h"
```

سپس اضافه کردن توابع task خودمون رو در بخش زیر:

در تابع زیر و بعد از خط زیر(البته دو بخش میتونیم توابع رو تقسیم کنیم، اونهایی که باید قبل از فراخوانی مقدار دهی بقیه task ها باید اضافه بشن و اونهایی که باید بعد از فراخوانی مقدار دهی بقیه task ها باید اضافه بشن):

```
static void ql_init_demo_thread(void *param)
{
    QL_INIT_LOG("init demo thread enter, param 0x%x", param);
    //--------- Prj User Code: add tasks before init all
    puc_gpio_init();  <- بخشی که باید اضافه کنیم
        .
        .
        .
    //--------- Prj User Code: add tasks after init all
<- مکانی که باید اضافه کنیم ->
    ql_rtos_task_delete(NULL);  <- حتما قبل از این خط اضافه کنیم
}
```

کارمون با این دو فایل نیز تمام شد.

3- اضافه کردن فولدر اصلی پروژه

توجه: نحوه ساختاربندی کاملا سلیقه ای است و هر مدلی که دلخواه هست میتوانیم انجام دهیم اما حتما فایل های CmakeLists.txt در شاخه اصلی پروژه شخصی (ابتدا فولدر user_code_base) و CmakeLists.txt درون شاخه های پایه باید بدرستی تنظیم شوند. (البته میتوان فایل های CmakeList.txt فولدر های پایه را نیز حذف کرد اما بودنشان برای پروژه هایی که احتمال بزرگ شدن دارند بهتر است)(فولدر های پایه: به فولدر هایی که آدرسشان دقیقا بعد از فولدر اصلی پروژه است میگوییم فولدر پایه اما فولدر هایی که در درون فولدرهای پایه هستند نیازی به فایل CmakeList.txt ندارند.صرفا جهت اطلاع: در پروژه های بزرگ مثل هسته لینوکس حتی فولدر های زیرشاخه فولدر های پایه نیز گاهی فایل CmakeList.txt دارند.)

بهتره برای اینکه با بزرگ شدن پروژه به کمترین مشکل بر بخوریم با ساختار استاندارد پروژه های نیمه بزرگ پیش بریم.(اینجا پروژه بزرگ منظورسیستم هایی مثل لینوکس یا از این قبیل هست که چندین تیم روش کار میکنن، و نیمه بزرگ تقریبا برای تمامی سیستمهای امبددی که میشناسیم جوابگو هست)

فولدر پروژه اصلی باید ساختاری شبیه به این داشته باشد:

یک فولدر بنام مثلا user_code_base که در فولدر components قرارش میدیم.

این فولدر اصلی پروژه دارای 3 بخش اصلی است:

1- فایل CmakeList.txt

2- فولدر include که درونش یک فولدر بنام user_code دارد. و تمامی فایل های هدر عمومی درون این فولدر قرار میگیرند.

3- فولدر های :

app

drivers

middleware

services

system

که هر کدام از این فولدر ها میتوانند هر تعداد فایل h. و c. و فولدر و کتابخانه درون خود داشته باشند.

و بهتره که به شکل مثال زیر ساختاربندی انجام شود، البته این مثال است:

ساختار فولدر کامل

```
components/
└── user_code_base/
      ├── CMakeLists.txt
      │
      ├── include/
      │     └── user_code_base/
      │           ├── app_api.h
      │           ├── mqtt_service.h
      │           ├── http_service.h
      │           ├── ota_service.h
      │           ├── device_manager.h
      │           ├── log.h
      │           ├── event_bus.h
      │           ├── timers.h
      │           ├── utils.h
      │           └── drivers_public.h
      │
      ├── app/
      │     ├── app_main.c
      │     ├── app_tasks.c
      │     ├── app_state_machine.c
      │     └── app_config.c
      │
      ├── services/
      │     ├── mqtt/
      │     │     ├── mqtt_service.c
      │     │     └── mqtt_context.h
      │     │
      │     ├── http/
      │     │     ├── http_service.c
      │     │     └── http_context.h
      │     │
      │     ├── ota/
      │     │     ├── ota_service.c
      │     │     └── ota_internal.h
      │     │
      │     ├── device_manager/
      │     │     ├── device_manager.c
      │     │     └── device_manager_internal.h
      │     │
      │     └── connectivity/
      │           ├── net_manager.c
      │           └── sim_manager.c
      │
```

```
├── middleware/
│     ├── log/
│     │     ├── log.c
│     │     └── log_internal.h
│     │
│     ├── event_bus/
│     │     ├── event_bus.c
│     │     └── event_bus_internal.h
│     │
│     ├── timers/
│     │     ├── timers.c
│     │     └── timers_internal.h
│     │
│     └── common/
│           ├── utils.c
│           ├── ring_buffer.c
│           ├── crc.c
│           └── fifo.c
│
├── drivers/
│     ├── spi/
│     ├── i2c/
│     ├── gpio/
│     ├── uart/
│     └── sensor_x/
│
└── system/
      ├── rtos_wrapper.c
      ├── task_manager.c
      ├── sys_init.c
      ├── sys_config.c
      ├── power_manager.c
      └── bsp.c
```

چند نکته:

نکته1: چرا فولدر include دارای یک فولدر بنام پروژه یا یوزر هست؟ دلیلش این هست که وقتی میخوایم درون sdk و در درون فایل های اصلی از فایل های پروژه خودمون استفاده کنیم ، موقع اینکلود کردن زیر شاخه اسم پروژه نیز اضافه کنیم تا راحتتر تشخیص داده شود که مربوط به فایل های پروژه شخصیمان است.

نکته2: حتما نام های فولدر و فایل های بدون استفاده از نام بزرگ و بدون اسپیس باشند و بهتر است که فقط با حروف کوچک و فقط از اعداد و علامت آندرلاین "_" استفاده شود.

4- آخرین مرحله: اضافه و تنظیم کردن فایل های CmakeLists.txt به فایل و فولدر های پروژه شخصیمون.

توجه: برای درک فایل های CmakeLists.txt میتونید از هوش های مصنوعی استفاده کنید. مثل دو مورد زیر که ایرانی هستند و با آنها کل این آموزش رو یادگرفتم و سپس خلاصه نویسی کردم برای بقیه:

https://gapgpt.app/chat

https://hooshang.ai/home

فولدر حاوی کد های پروژه شخصی دارای دو دسته فایل CmakeLists.txt است:

فایل درون فولدر اصلی: فولدر user_code_base

درون فولدر های پایه

تنظیم کردن فایل CmakeLists.txt در فولدر اصلی:

```
#  components/user_code_base/CMakeLists.txt


#add_subdirectory(app)
add_subdirectory(drivers)
#add_subdirectory(middleware)
#add_subdirectory(services)
#add_subdirectory(system)

#  ایجاد لایبرری نهایی از تمام ماژول ها
add_library(user_code_base_lib  INTERFACE)

target_link_libraries(user_code_base_lib  INTERFACE
        #user_app_lib
        user_drivers_lib
        #user_middleware_lib
        #user_services_lib
        #user_system_lib
)
```

```
# include global
target_include_directories(user_code_base_lib INTERFACE
    ${CMAKE_CURRENT_SOURCE_DIR}/include
)
```

تنظیم کردن فایل CmakeLists.txt در فولدر های پایه:

مثلا برای فولدر drivesrs که دارای دو زیر شاخه gpio و i2c است.

در فولدر پایه drivers یک فایل CmakeList.txt میسازیم و به شکل زیر محتویاتش رو مینویسیم:

```
# components/user_code_base/drivers/CMakeLists.txt

#file(GLOB_RECURSE USER_DRIVERS_SRC CONFIGURE_DEPENDS "*.c" "*.h")
file(GLOB_RECURSE USER_DRIVERS_SRC CONFIGURE_DEPENDS "*.c")

add_library(user_drivers_lib STATIC ${USER_DRIVERS_SRC})

target_include_directories(user_drivers_lib PUBLIC
    ${CMAKE_CURRENT_SOURCE_DIR}/../include
    ${CMAKE_CURRENT_SOURCE_DIR}
    ${CMAKE_CURRENT_SOURCE_DIR}/gpio
    #${CMAKE_CURRENT_SOURCE_DIR}/i2c
    #${CMAKE_CURRENT_SOURCE_DIR}/spi
)

#target_link_libraries(user_drivers_lib PUBLIC ql_app_osi)
```

چند نکته:

منظور ازفولدر includeفایل های هدر عمومی چیست؟

درide هایی که برای میکروکنترلر ها برنامه نویسی میکنیم کمتر با این موارد روبرو شده ایم برای همین توضیح مختصری مدهم:

وابستگی های یک فایل c. سه مرحله طبقه بندی میشود:

عمومی public

نیمه خصوصی (امکان اشتراک با فایل هایی که درکنار فایل c. هستند)

کاملا خصوصی که فقط در خود فایل c. در دسترس هستند.

بهتره که ما همیشه این سه حالت رو رعایت کنیم تا در پروژه های بزرگ یا مشترک که ممکنه از نام های مشترکی استفاده به خطا در زمان بیلد کردن مواجه نشیم.

برای همین تمامی داده هایی که میخواهیم از فایل c. با کل پروژه یا هر فایلی از پروژه که تمایل داریم در دسترس باشد را در فولدر include قرار میدیم.

و فایل هایی که داده های آن را میخواهیم فقط در خود فایل c. یا فایل هایی که در کنارش هستند استفاده کنیم را در کنار خود فایل c. قرار میدیم و در انتهای آن نامی مثل "internal یا context" اضافه میکنیم. و داده هایی که فقط میخوام درون خود فایل c. در دسترس باشد را با کامه کلیدی static تعریف میکنیم.

مثلا:

Driver for Timers:

timers.h

در فولدر include قرارش میدیم. و مثلا تعریف توابع گلوبال درون فایل timers.c را درونش مینویسیم.

timers_internal.h

در کنار فایل timers.c قرارش میدیم و مواردی همچون فرکانس تایمر ، مقدار های اولیه متغیر های بخش timers.c که فقط برای سورس کد timers.c مورد استفاده است را قرار میدهیم تا کل sdk بهش دسترسی نداشته باشد.

timers.c

در فولدر middleware در فولدر timers قرارش میدیم و سورس کد بخش timer هارا درونش مینویسیم.

نکته: دلیل اینکه sdk به فایل های هدر درون include->user_code دسترسی دارد اما به فایل های h. در بقیه فولدر های پروژه شخصی دسترسی ندارد این هست که نحوه تنظیم کردن فایل های CmakeLists.txt در فولدر اصلی را به این شکل انجام داده ایم.

توجه: هر فولدر یا فایل سورس کد (به غیر از .h) به فولدر پروژه شخصی اضافه کردیم باید در فایل CmakeList.txt بخش مربوطه (فایل CmakeList.txt فولدر اصلی یا فولدر های پایه) تنظیمات لازم را انجام بدهیم.

توجه: در فایل های .c نحوه اینکلود کردن فایل های .h که در کنارشون هستند مثل همیشه ساده است مثلا:

```
#include "prj_gpio_init.h"
```

اما نحوه اینکلود کردن فایل های .h که در فولدر include->user_code است به شکل زیر انجام میدیم:

```
#include "user_code/prj_gpio_init.h"
```

نکته: اگر از فایل های sdk نیاز شد در پروژه شخصی استفاده کنیم، باید نام آن هدر را در CmakeList.txt مربوط به آن پروژه شخصی اضافه کنیم. مثلا فرض کنیم که از محتویات فایل ql_app_osi نیاز شده در فایل prj_gpio_init.c استفاده کنیم، بعد از اینکه بصورت ساده در فایل پروژه اینکلود میکنیم در فایل CmakeList.txt مربوط به فولدر پایه مربوط به gpio که فولدر drivers است نیز باید به شکل زیراضافه کنیم:

```
target_link_libraries(user_drivers_lib PUBLIC ql_app_osi)
```

خط بالا را در انتهای فایل اضافه میکنیم.