

Кроссмикроконтроллерная кооперативная ОС

Основная суть проекта - абстрагировать бизнес логику от микроконтроллерной платформы. А так же создать систему разработки легковесных приложений на базе микроконтроллеров, с небольшими ресурсами, с переиспользуемыми исходными кодами.

Подготовка к работе

1. Скачать и установить [тулчейн](#)
2. Скачать и установить систему сборки [Ninja](#)
3. Скачать и установить [CMake](#)
4. Скачать и установить NuOpenOCD, ищем все [тут](#)
5. Скачать и установить OpenOCD (TODO: разобраться с Gigadevice)
6. При использовании J-Link скачать и установить [J-Link Software](#)
7. Возможно понадобится добавить в PATH директории тулчейна, Ninja, NuOpenOCD ...

Сборка проекта

```
cd <project dir>
cmake.exe -D CMAKE_BUILD_TYPE=Debug -D BOARD=<name> -S '.' -B './build' -G
Ninja
cd .\build\
cmake --build . --clean-first
```

В параметре `-D BOARD=` указывается имя платформы. Один проект может быть нацелен на множество платформ.

Пример. Прошивка микроконтроллера Nuvoton M463KG

Локально через OpenOCD:

```
$ nuopenocd.exe -s "./HDL/McuPort/ARM/Nuvoton/NUM463KG/Res" -f
"./HDL/McuPort/ARM/Nuvoton/NUM463KG/Res/tool.cfg" -f
"./HDL/McuPort/ARM/Nuvoton/NUM463KG/Res/mcu.cfg" -c "init" -c "halt" -c "flash
write_image erase ./build/bmc.hex" -c "reset run"
```

Где `tool.cfg` - конфиг NuLink,

`mcu.cfg` - конфиг контроллера.

Все можно найти в пакете SDK под конкретную серию микроконтроллера.

Удаленно через GDB + OpenOCD:

```
$ arm-none-eabi-gdb.exe
(gdb) target remote 10.20.30.40:3333
(gdb) monitor reset halt
(gdb) monitor flash erase_address 0x00000000 0x00040000
(gdb) monitor flash info 0
(gdb) monitor flash protect 0 0 7 off
(gdb) file ./build/bmc.elf
(gdb) load
(gdb) monitor reset run
```

Команды `(gdb) monitor flash info 0` и `(gdb) monitor flash protect 0 0 7 off` можно пропустить. Первая необходима чтобы посмотреть состояние защиты флеш памяти от записи, вторая снять защиту.

Аналогично для других поддерживаемых микроконтроллеров. Скрипты можно найти в директории контроллера, в `./Res`.

Установка и настройка среды VS Code

1. Скачать и установить [VS Code](#)
2. Установить расширения C/C++, CMake, Cortex-Debug

Готово.

Сборка

Для сборки проекта необходимо определить имя платформы `BOARD`. Это можно сделать через пресет `CMakePresets.json` в VisualCode:

```
{
  "name": "XXX_D",
  "displayName": "XXX Debug",
  "generator": "Ninja",
  "binaryDir": "${sourceDir}/build",
  "cacheVariables": {
    "CMAKE_BUILD_TYPE": "Debug",
    "BOARD": "XXX"
  }
},
{
  "name": "XXX_R",
  "displayName": "XXX Release",
  "generator": "Ninja",
  "binaryDir": "${sourceDir}/build",
  "cacheVariables": {
    "CMAKE_BUILD_TYPE": "Release",
    "BOARD": "XXX"
  }
},
```

Либо в файле `.\Projects\CMakeLists.txt`:

```
set(BOARD "XXX")
```

Либо любым другим удобным способом.

Прошивка

В меню **Terminal->Run Task** доступны задания по прошивке различными инструментами. (TODO: протестировать)

Отладка

В меню **Run and Debug** доступны варианты отладки различными инструментами.

Порядок действий:

Собираем проект, выбираем вариант отладки, запускаем F5.

Чтобы запитать Target через J-Link, выполнить команду в J-Link Commander

J-Link power on

Или чтобы программатор принял настройку по-умолчанию.

J-Link power on perm

(TODO: проверить все на Linux)

Портирование библиотеки на новый контроллер:

1. Скачать драйверы от производителя
 - Интерфейсы периферии положить в исходники в директорию **/Sys**, заголовки в **Sys/Inc**
 - SPL/HAL положить в директорию **/Sys/Drivers** исходники и заголовки в **Src, Inc** соответственно
2. Найти или описать linker script, переименовать **linkerscript.ld**, положить в директорию **Res**
3. Найти Svd файл, переименовать **mcu.svd**, положить в директорию **Res**
4. Для инструментов OpenOCD, также понадобятся файлы конфигурации **mcu.cfg** и **tool.cfg**, положить в **Res**. (см. примеры проектов)
5. Скопировать содержимое **/McuPort/PortTemplate** в **/McuPort/<core>/<manufacturer>/<mcu>/Port**, где **<core>** - архитектура микроконтроллера, **<manufacturer>** - производитель, **<mcu>** - модель
6. Во всех папках добавить или создать **CMakeLists.txt** по аналогии.
7. Описать все необходимые портируемые интерфейсы.

8. Для ARM унифицированы порты для ядра. Точка входа `reset_hundler` располагается в файле `McuPort/ARM/port_core_mx.c` в зависимости от выбранного ядра (m0, m3, m4 и т.п.) микроконтроллера в проекте. Порттирование следует начинать с файла `/McuPort/<core>/<manufacturer>/<mcu>/Port/port_core.c` в котором описываются обработчики для прерываний и специфическая для данного микроконтроллера часть инициализации ядра.
9. Создать тестовый проект для данного микроконтроллера.

Новый проект:

1. Создать директорию нового проекта в `/Projects`
2. Добавить `CMakeLists.txt` по аналогии с другими проектами. Определить глобальные свойства:

```
MCU - модель микроконтроллера
MCU_MANUFACTURER - производитель
CORE - архитектура
PROJECT - имя проекта
При необходимости:
TOOLCHAIN_DIR
TOOLCHAIN_PREFIX
```

Данные свойства могут подменяться через условный оператор над свойством `BOARD`, если один проект рассчитано запускать на разных платформах.

3. Создать файл графа инициализации в директории `mig_<mcu>.c`. MIG файлы должны быть уникальны для платформы-проекта. Данный файл описывает модули используемые в проекте, их зависимости и конфигурации. Это полное описание системы от ядра контроллера и вектора его прерываний до высокоуровневых драйверов внешних связанных устройств.
*можно адаптировать существующий из другого проекта.
4. Создать `./Inc/mig.h`, тут экспортируются все дескрипторы модулей задействованных в проекте

```
#ifndef MIG_H_
#define MIG_H_

extern hdl_time_counter_t mod_timer_ms; // само определение в mig_<mcu>.c
...

#endif // MIG_H_
```

5. Создать `./Inc/app.h`

```
#ifndef APP_H_
#define APP_H_
```

```
#include "hdl.h"
#include "mig.h"

#include "CodeLib.h"

void main();

#endif /* APP_H_ */
```

6. Создать файл точку входа проекта `./app.c`

```
#include "app.h"

void main() {
    // включаем необходимые модули
    hdl_enable(&mod_timer_ms.module);
    while (!hdl_init_complete()) {
        cooperative_scheduler(false);
    }
    // инициализация модулей завершена
    while (1) {
        cooperative_scheduler(false);
        // логика приложения
    }
}
```