**MICROCHIP POLARFIRE SOC**

ПРОЦЕСС РАЗРАБОТКИ

2023-09-25

Оглавление

[1 ЭТАПЫ РАЗРАБОТКИ 3](#_Toc146619136)

[2 РАБОЧЕЕ ПРОСТРАНСТВО 4](#_Toc146619137)

[2.1 Программное обеспечение 4](#_Toc146619138)

[2.2 Переменные среды пользователя 4](#_Toc146619139)

[2.3 Директория проекта 5](#_Toc146619140)

[2.4 Файловая структура проекта 5](#_Toc146619141)

[2.4.1 application 6](#_Toc146619142)

[2.4.2 board 7](#_Toc146619143)

[2.4.3 middleware 8](#_Toc146619144)

[2.4.4 platform 9](#_Toc146619145)

[3 ЗАГРУЗЧИК И СИСТЕМНЫЙ МОНИТОР 11](#_Toc146619146)

[3.1 Конфигурирование 12](#_Toc146619147)

[3.2 Сборка 14](#_Toc146619148)

[3.3 Загрузка в eNVM 14](#_Toc146619149)

[3.4 Сервис TinyCLI 15](#_Toc146619150)

[4 КОНФИГУРАТОР ПАМЯТИ И ПЕРИФЕРИИ 17](#_Toc146619151)

[4.1 Запуск 17](#_Toc146619152)

[4.2 Конфигурирование 18](#_Toc146619153)

[4.3 Сохранение 18](#_Toc146619154)

[4.4 Генерирование 18](#_Toc146619155)

[4.5 Интеграция в проект пользовательского приложения 19](#_Toc146619156)

[5 ДИЗАЙН ПЛИС (FPGA) 20](#_Toc146619157)

[5.1 Генерирование эталонного дизайна 20](#_Toc146619158)

[6 ПОЛЬЗОВАТЕЛЬСКОЕ ПРИЛОЖЕНИЕ (MSS) 22](#_Toc146619159)

[6.1 Настройка сборки 22](#_Toc146619160)

[6.2 Очистка сборки 23](#_Toc146619161)

[6.3 Сборка 23](#_Toc146619162)

[7 ОБРАЗ ПОЛЬЗОВАТЕЛЬСКОГО ПРИЛОЖЕНИЯ (PAYLOAD) 24](#_Toc146619163)

[7.1 Настройка рабочего пространства 24](#_Toc146619164)

[7.2 Конфигурирование 25](#_Toc146619165)

[7.3 Сборка 25](#_Toc146619166)

[7.4 Конвертация в hex 26](#_Toc146619167)

[8 SPI-FLASH 27](#_Toc146619168)

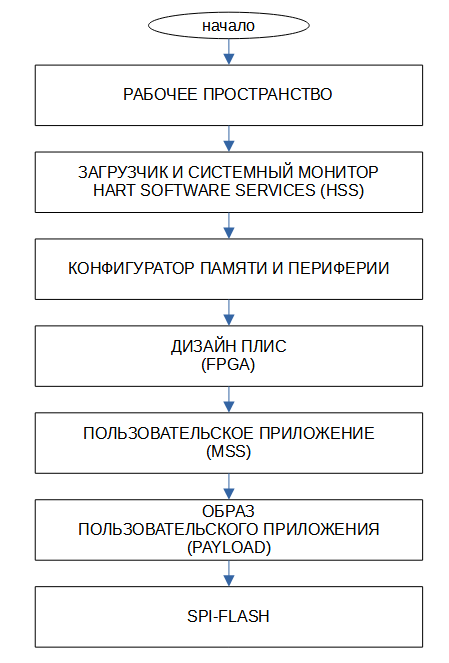
[8.1 Настройка памяти 28](#_Toc146619169)

[8.2 Добавление образа 29](#_Toc146619170)

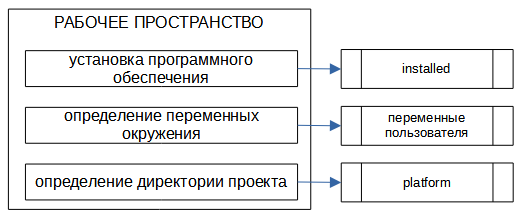
[8.3 Запись 30](#_Toc146619171)

[БАЗА ЗНАНИЙ 31](#_Toc146619172)

# ЭТАПЫ РАЗРАБОТКИ



# РАБОЧЕЕ ПРОСТРАНСТВО



## Программное обеспечение

1) Скачать и установить следующее программное обеспечение

|  |
| --- |
| Libero\_SoC\_v2022.2 |

|  |
| --- |
| SoftConsole-v2022.2-RISC-V-747 |

|  |
| --- |
| Python 3.10  + pip  + pip install windows-curses |

## Переменные среды пользователя

1) Определить переменные среды пользователя

Свойства системы / Переменные среды / Переменные среды пользователя

|  |
| --- |
| **Path** |
| D:\MPFS\SoftConsole-v2022.2-RISC-V-747\python3  *путь к Python 3 (комплект SoftConsole)*  D:\MPFS\Soft\SoftConsole-v2022.2-RISC-V-747\build\_tools\bin  *путь к инструментам сборки (комплект SoftConsole)*  D:\MPFS\SoftConsole-v2022.2-RISC-V-747\riscv-unknown-elf-gcc\bin  *путь к компилятору GCC RISC-V (комплект SoftConsole)*  D:\MPFS\Libero\_SoC\_v2022.2\Designer\bin64  *путь к Дизайнеру (комплект Libero)* |

|  |
| --- |
| **SC\_INSTALL\_DIR** |
| D:\MPFS\SoftConsole-v2022.2-RISC-V-747  *путь к SoftConsole* |

|  |
| --- |
| **JAVA\_BINARY** |
| D:\MPFS\SoftConsole-v2022.2-RISC-V-747\/eclipse/jre/bin/java.exe  *путь к Java (комплект SoftConsole)* |

|  |
| --- |
| **JAVA\_HOME** |
| D: \MPFS\SoftConsole-v2022.2-RISC-V-747\/eclipse/jre/  *путь к Java Runtime (комплект SoftConsole)* |

|  |
| --- |
| **PYTHONPATH** |
| D:\MPFS\SoftConsole-v2022.2-RISC-V-747\python3  *путь к Python (комплект SoftConsole)* |

|  |
| --- |
| **MACRO\_PYTHON\_BINARY\_EXECUTABLE** |
| python.exe  *имя исполняемого файла Python (комплект SoftConsole)* |

|  |
| --- |
| **MACRO\_PYTHON\_BINARY\_PATH** |
| D:\MPFS\SoftConsole-v2022.2-RISC-V-747\python3  *путь к Python (комплект SoftConsole)* |

|  |
| --- |
| **MACRO\_PYTHON\_BINARY\_PATH\_AND\_EXECUTABLE** |
| D:\MPFS\SoftConsole-v2022.2-RISC-V-747\python3\python.exe  *путь к Python (комплект SoftConsole)* |

## Директория проекта

1) Создать директорию рабочего пространства (например, polarfire / platform)

mkdir D:\projects\polarfire\platform

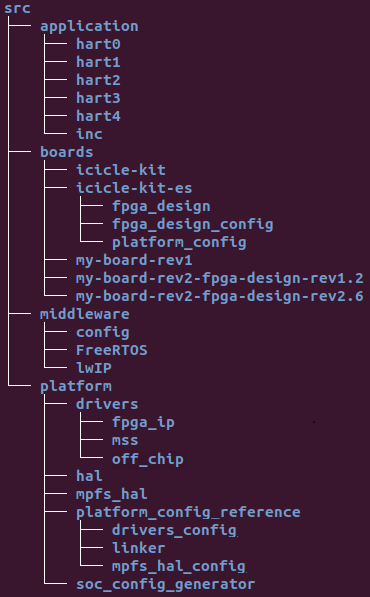
## Файловая структура проекта

1) Сформировать файловую структуру проекта

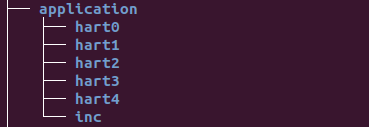
вручную

или

взять в качестве шаблона один из [демонстрационных проектов](https://github.com/polarfire-soc/polarfire-soc-bare-metal-examples)



### application

****

Нижний уровень абстракции (HAL).

Базовый код пользовательского приложения (MSS Bare Metal) конкретного ядра.

Допускается изменение содержимого.

* **hart0**

Пользовательский код для Ядра 0 (e51) [\*.c \*.h]

* **hart1**

Пользовательский код для Ядра 1 (u54\_1) [\*.c \*.h]

* **hart2**

Пользовательский код для Ядра 2 (u54\_2) [\*.c \*.h]

* **hart3**

Пользовательский код для Ядра 3 (u54\_3) [\*.c \*.h]

* **hart4**

Пользовательский код для Ядра 4 (u54\_4) [\*.c \*.h]

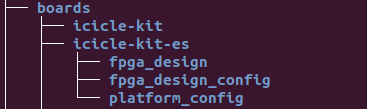
* **inc**

Общие заголовочные файлы (для всех ядер) [\*.h]

Для соблюдения принципов «Hardware Abstraction Layer», разделять код на:

* аппаратно-зависимый (работа с периферией конкретной платформы)
* аппаратно-независимый

### board



Нижний уровень абстракции (HAL).

Конфигурации, линковщики и заголовочные файлы для конкретной аппаратной платформы (например, icicle-kit-es).

Допускается изменение содержимого (кроме автогенерируемого).

* **fpga\_design**

Конфигурация FPGA и MSS

* + libero\_tcl

Копия или ссылка на TCL-скрипт Libero

*(не используется при сборке пользовательского приложения).*

* + mss\_configuration

Конфигурация (CFG), загружаемая в программу MSS-Configurator

*(не используется при сборке пользовательского приложения).*

* + design\_description

Конфигурация (XML), сгенерированная программой MSS-Configurator

*(используется при сборке пользовательского приложения на этапе «pre-build» - генерирует содержимое директории fpga\_design\_config).*

* **fpga\_design\_config**

Заголовочные файлы [\*.h] с описанием конфигурации периферии для конкретной аппаратной платформы, автоматически генерируемые при сборке пользовательского приложения на этапе «pre-build» (основа - это конфигурация из design\_description/\*.xml). Эти заголовочные файлы используются platform/mss\_hal.

* + clocks
  + ddr
  + general
  + io
  + memory\_map
  + sgmii
  + fpga\_design\_config.h

*(included in platform/mpfs\_hal/mss\_hal.h)*

* **platform\_config**

Заголовочные файлы с описанием конфигурации драйверов и HAL (например, определение рабочих аппаратных ядер и загрузчика).

* + drivers\_config

Заголовочные файлы [\*.h] с описанием конфигурации драйверов, реализованных в platform/drivers.

* + linker

Файлы линковщика [\*.ld].

* + mpfs\_hal\_config

Заголовочные файлы [\*.h] с описанием конфигурации MSS HAL.

### middleware



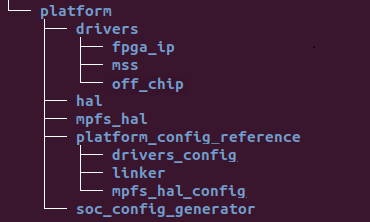
Средний уровень абстракции (HAL).

Код (обычно аппаратно-независимый):

* RTOS/ОСРВ (ядро и его компоненты),
* стек протоколов (например, TCP/IP, ModBus RTU/TCP),
* прочий пользовательский код (например, описание задач, работающих в режиме ядра ОСРВ).

Допускается изменение содержимого (кроме стороннего кода).

### platform



Нижний уровень абстракции (HAL).

Код библиотеки HAL, код для работы с аппаратной платформой (драйвера периферии), а также код для запуска пользовательской программы (startup-code).

Не допускается изменение содержимого.

* **drivers**

Код драйверов периферии

* + fpga\_ip

Периферия ядер FPGA

* + mss

Периферия MSS

* + off\_chp

Внешняя периферия

*(например, подключаемые датчики, внешняя память и т.п.)*

* **hal**

Код библиотеки HAL (универсальная часть)

*(позволяет использовать один код для различных аппаратных платформ)*

* **mpfs\_hal**

Код библиотеки HAL (специфичная для Microchip Polarfire SoC часть)

*(включая код для запуска пользовательского приложения – startup-code)*

* **platform\_config\_reference**

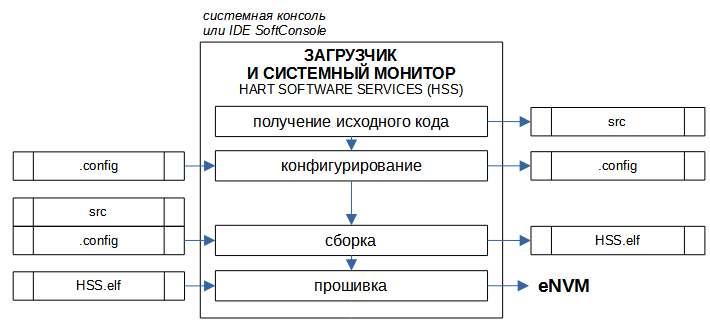
Исходная и иные версия platform\_config

*(шаблоны линковщиков и конфигураций)*

* **soc\_config\_generator**

Python-скрипт для автоматической генерации содержимого директории boards/fpga\_design\_config при сборке пользовательского приложения на этапе «pre-build».

# ЗАГРУЗЧИК И СИСТЕМНЫЙ МОНИТОР



HART SOFTWARE SERVICE (HSS)

Готовый программный продукт с [открытым исходным кодом](https://github.com/polarfire-soc/hart-software-services).

Хранится (в сжатом виде) во встроенной flash-памяти eNVM

(используется режим Boot Mode 1-Direct Boot from eNVM)

При подаче питания

* код HSS выгружается из eNVM в L2-scratchpad

Работает на ядре E51. Ядра U54 свободны для пользовательских приложений.

Включает в себя:

* 0-stage bootloader
  + запускает (стартует) пользовательское приложение, предварительно выгрузив его из SPI-FLASH|eMMC|SD в DDR|LIM
* superloop-monitor
  + получает и обрабатывает запросы от приложений пользовательских ядер U54
  + получает и обрабатывает запросы от внешних приложений через последовательный интерфейс UART0 (115200 8 N 1, если HSS был собран с поддержкой TinyCLI)
* machine-mode software interrupt trap handler
  + позволяет формировать программные прерывания для ядер U54 с передачей им полезных данных

Пользовательское приложение

* представляет собой специально оформленный образ (payload), включающий Bare Metal приложение или загрузчик ОСРВ или ОС (например, U-Boot)
* хранится во внешней SPI-FLASH|eMMC|SD
* перед запуском выгружается из хранимой памяти в DDR|LIM

## Конфигурирование

1) Перейти в директорию исходного кода

cd D:\projects\polarfire\hart-software-services

2) Сформировать файл конфигурации одним из следующих способов:

а) использовать готовый шаблон конфигурации (например, для ICICLE-KIT):

из директории шаблонов

copy boards\mpfs-icicle-kit-es\def\_config .config

из заранее подготовленного файла (см. ниже)

copy .config-icicle-kit .config

б) использовать программу-конфигуратор (псевдографический интерфейс):

со значениями по-умолчанию

make menuconfig

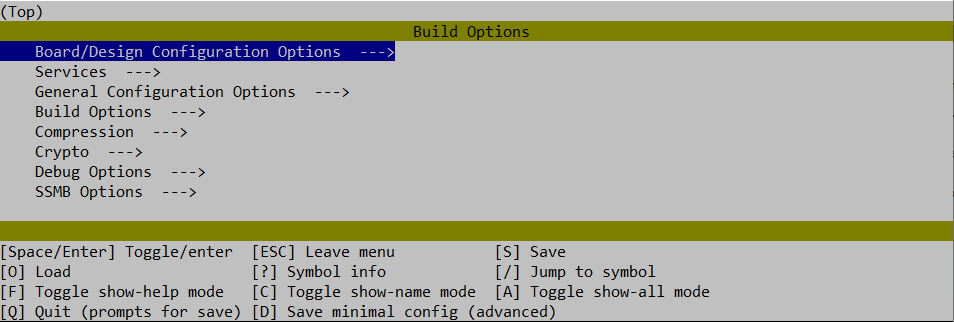
со значениями из шаблонной конфигурации

make BOARD=mpfs-icicle-kit-es menuconfig

где, BOARD – наименование системы (платы)

указание BOARD= mpfs-icicle-kit-es равносильно команде

copy boards\mpfs-icicle-kit-es\def\_config .config



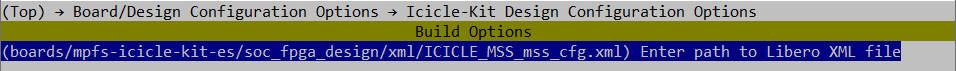
*Для работы Конфигуратора требуются:*

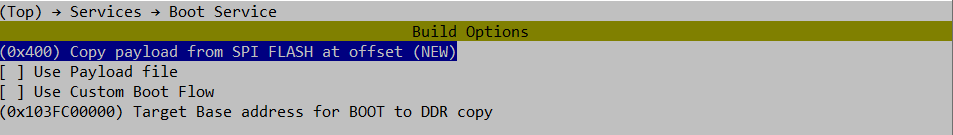
*+ python3*

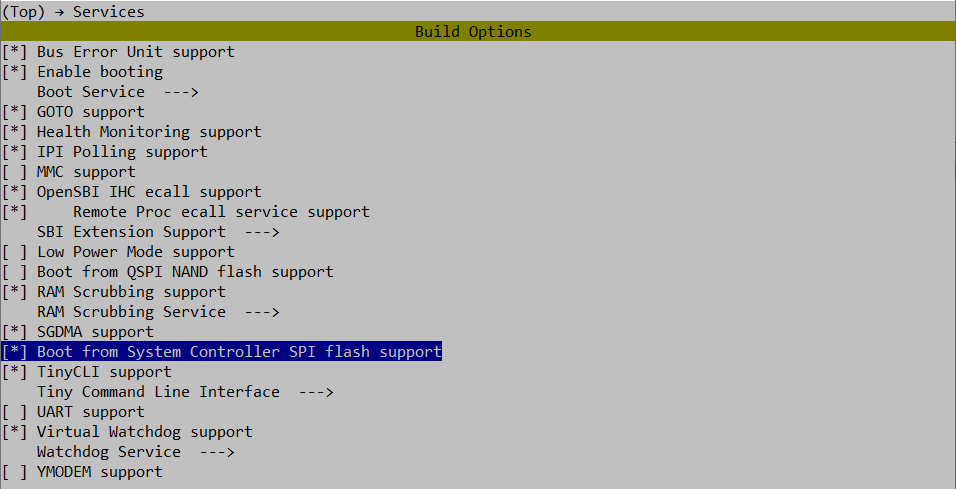
*+ pip*

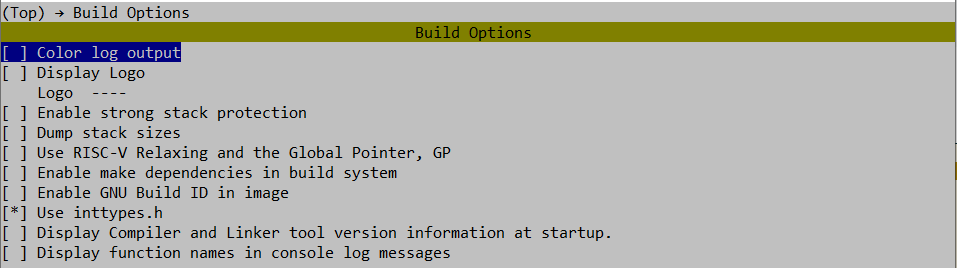
*+ pip install windows-curses*

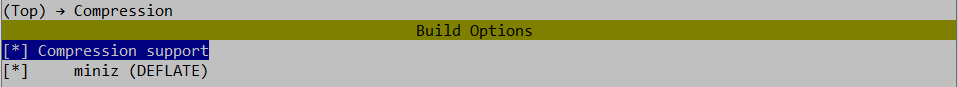
Краткий обзор .config-icicle-kit











## Сборка

1) Перейти в директорию проекта

cd D:\projects\polarfire\hart-software-services

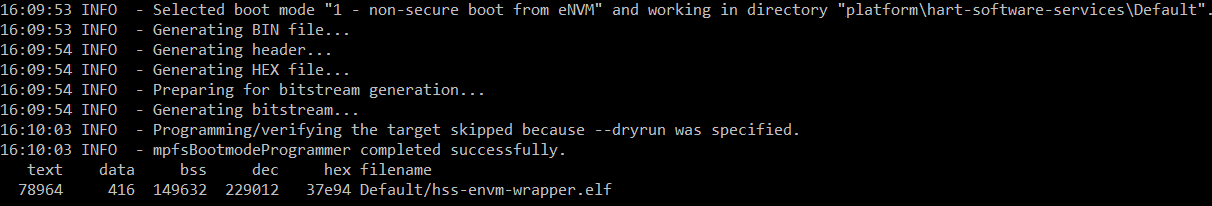
2) Очистить директорию проекта от файлов предыдущей сборки

make BOARD=mpfs-icicle-kit-es clean

3) Запустить сборку

make BOARD=mpfs-icicle-kit-es

4) Дождаться завершения процесса сборки

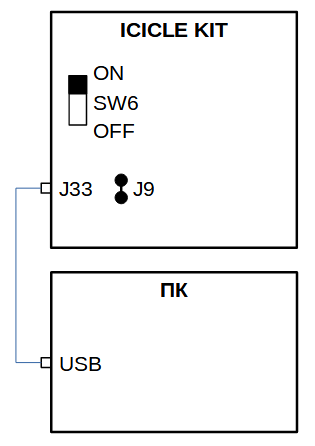


## Загрузка в eNVM

1) Включить встроенный в ICICLE KIT программатор FLASH PRO6, установив перемычку J9

2) Установить связь между программатором FLASH PRO6 и ПК посредством кабеля microUSB-USB

3) Включить ICICLE KIT, переведя переключатель SW6 в положение ON



4) Перейти в директорию проекта

cd D:\projects\polarfire\hart-software-services

5) Запустить загрузку

make BOARD=mpfs-icicle-kit-es program

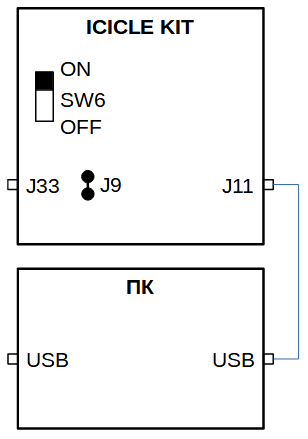
6) Дождаться завершения процесса загрузки



## Сервис TinyCLI

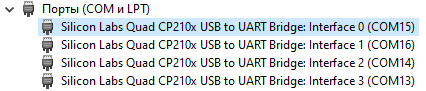
1) Установить связь между ICICLE KIT и ПК посредством кабеля microUSB-USB

2) Включить ICICLE KIT, переведя переключатель SW6 в положение ON



3) Запустить консоль (Putty, Terminal и т.п.) с настройками:

**COM** = соответствует Interface 0 (см. Диспетчер устройств)



**Speed** = 115220

**Data bits** = 8

**Stop bits** = 1

**Parity** = None

**Flow control** = None

4) В консоли дождаться подключения и далее нажать клавишу ВВОД

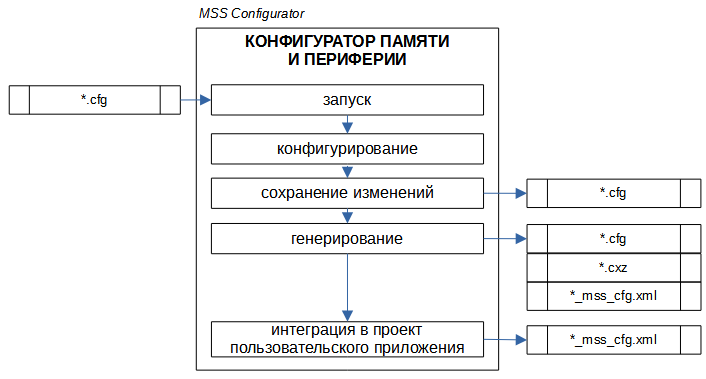
(будет выведена строка приглашения для ввода команд)



5) Команда **HELP** выводит список поддерживаемых команд



# КОНФИГУРАТОР ПАМЯТИ И ПЕРИФЕРИИ



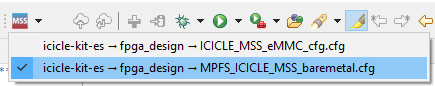
MSS Configurator (Polarfire SoC Configuration Generator)

Готовый программный продукт, интегрированный в SoftConsole и Libero (2022).

Предоставляет графический интерфейс для конфигурирования периферии и памяти Polarfire SoC.

## Запуск

Из SoftConsole:



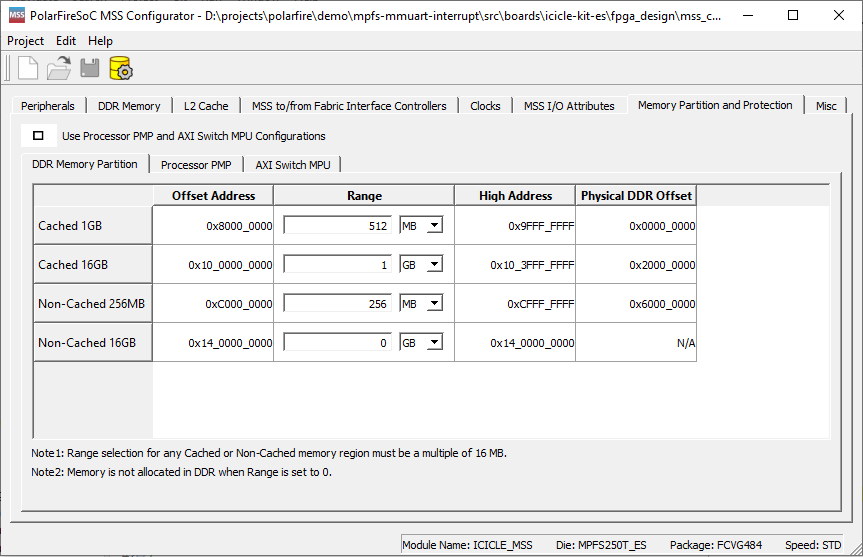
По-умолчанию, загружается конфигурация из проекта пользовательского приложения (\*.cfg), но далее можно выбрать другую конфигурацию самостоятельно (Project / Open ).

\*.cfg – исходная конфигурация:

boards/icicle-kit-es/fpga\_design/mss\_configuration/MPFS\_ICICLE\_MSS\_baremetal.cfg

## Конфигурирование

Пример окна конфигуратора:



## Сохранение

По-умолчанию, при сохранении () перезаписывается загруженная конфигурация, но можно сохранить конфигурацию под иным, заданным вручную, именем (Project / Save As ).

\*.cfg – измененная конфигурация:

boards/icicle-kit-es/fpga\_design/mss\_configuration/MPFS\_ICICLE\_MSS\_baremetal.cfg

## Генерирование

Генерирование () формирует и сохраняет конфигурацию в нескольких форматах (пакет генерации):

boards/\_generated/current/ICICLE\_MSS.cfg

boards/\_generated/current/ICICLE\_MSS.cxz

boards/\_generated/current/ICICLE\_MSS\_mss\_cfg.xml

Пакет генерации, по-умолчанию, сохраняется в директории boards/\_generated/current, но можно выбрать иное место.

Конфигурация в формате CXZ используется в проекте Дизайна ПЛИС.

## Интеграция в проект пользовательского приложения

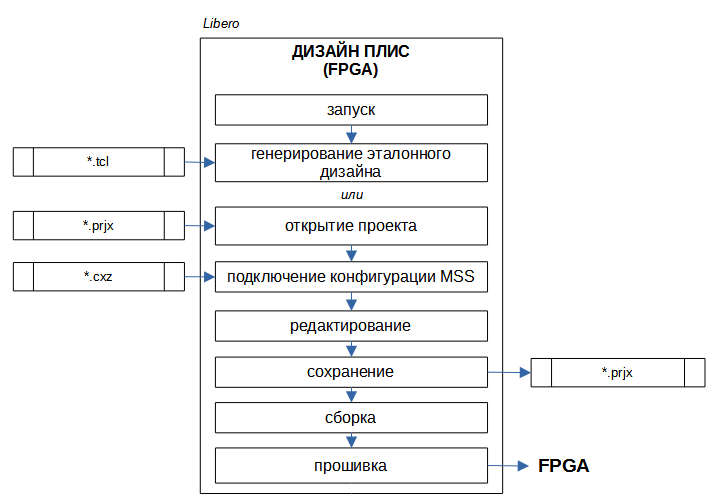
Полученный на этапе генерации XML-файл необходимо ***вручную*** скопировать в одну из директорий проекта пользовательского приложения:

copy boards/\_generated/current/ICICLE\_MSS\_mss\_cfg.xml

boards/icicle-kit-es/fpga\_design/design\_description/ICICLE\_MSS\_mss\_cfg.xml

Конфигурация в формате XML используется при сборке (этап «pre-build») пользовательского приложения.

# ДИЗАЙН ПЛИС (FPGA)



Libero (2022).

## Генерирование эталонного дизайна

1) Перейти в директорию рабочего пространства

cd D:\projects\polarfire\platform

2) Получить исходники эталонного дизайна (например, для [ICICLE-KIT](https://github.com/polarfire-soc/icicle-kit-reference-design))

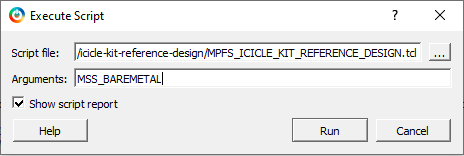
3) Запустить Libero (2022)

\* при необходимости задать настройки Proxy

Project / Preferences / Proxy

4) Выбрать скрипт генерации эталонного дизайна:

Project / Execute Script



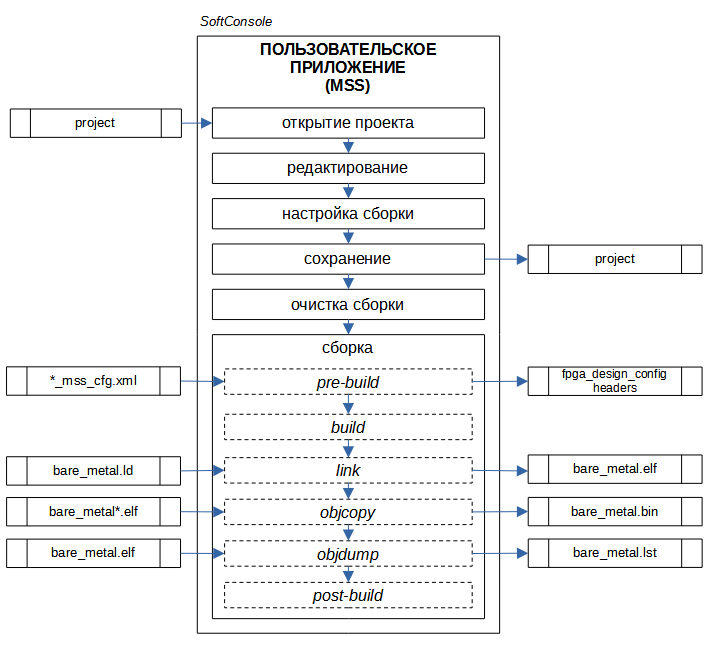
\* аргумент MSS\_BAREMETAL генерирует конфигурацию MSS с поддержкой проектов Bare Metal – без дополнительного функционала, ОЗУ DDR 1 ГБ.

5) Генерировать дизайн, нажав на кнопку Run

\* в процессе генерации дизайна будет выполнена автоматическая загрузка всех необходимых библиотек (ядра), поэтому на рабочем ПК должен быть обеспечен доступ к сети Интернет

\* дождаться завершения процесса загрузки

# ПОЛЬЗОВАТЕЛЬСКОЕ ПРИЛОЖЕНИЕ (MSS)



SoftConsole (2022).

## Настройка сборки

Project / Properties / C/C++ Build / Settings / Build Steps

Pre-build Command:

${env\_var:MACRO\_PYTHON\_BINARY\_PATH\_AND\_EXECUTABLE} ../src/platform/soc\_config\_generator/mpfs\_configuration\_generator.py ../src/boards/icicle-kit-es/fpga\_design/design\_description/ ../src/boards/icicle-kit-es

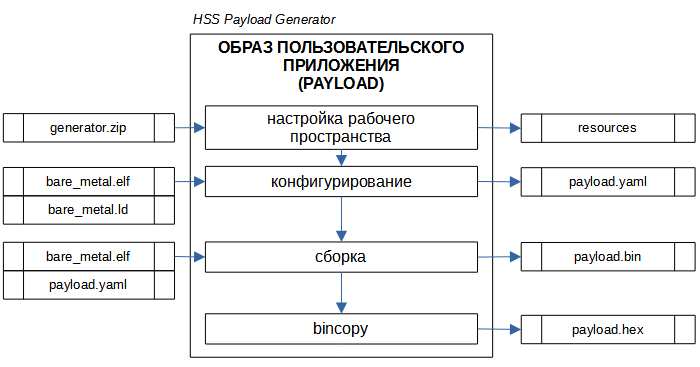
## Очистка сборки

Project / Clean…

## Сборка

Project / Build All

# ОБРАЗ ПОЛЬЗОВАТЕЛЬСКОГО ПРИЛОЖЕНИЯ (PAYLOAD)



HSS PAYLOAD GENERATOR

Готовый программный продукт с [открытым исходным кодом](https://github.com/polarfire-soc/hart-software-services/releases/download/v2023.06/hss-payload-generator.zip).

Программа без графического интерфейса (работает из консоли).

В схеме с использованием HSS:

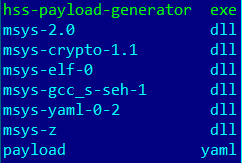
* HSS-загрузчик хранится в eNVM (встроена в Polarfire SoC)
* Пользовательское приложение хранится во внешней SPI-FLASH (SD, MMC) в виде специально оформленного образа (***Payload***)
* При включении питания (или перезагрузке):
  + сначала запускается HSS-загрузчик
  + далее загрузчик загружает пользовательское приложение из внешней памяти (например, в ОЗУ) и запускает его.

## Настройка рабочего пространства

Сборка образа пользовательского приложения выполняется в директории «resources», которая находится в директории проекта – на одном уровне с директорией «src»:



В директорию «resources» необходимо скопировать исполняемый файл самого генератора (со всеми прилагаемыми библиотеками). Здесь же будет находиться файл конфигурации (payload.yaml) и результат сборки – образ пользовательского приложения (payload.bin и payload.hex).



## Конфигурирование

Образ пользовательского приложения (payload.bin, .hex) содержит:

* строковое имя
* адреса «стартовых точек» (entry points) для каждого ядра
* описание пользовательского приложения:
  + исполняемый код (bare\_metal.elf)
  + адрес «стартовой точки» (exec address)
  + имя основного ядра и пр.

Выше перечисленная структура образа описывается конфигурационным файлом формата YAML (***payload.yaml***), где:

1) задается имя образа

2) задаются адреса «стартовых точек»

hart-entry-points: { u54\_1:'<from \*.ld>', ... }

\*.elf: {exec-addr:'<from \*.ld>'}

\* адреса берутся из файла линковщика

(bare\_metal.ld проекта пользовательского приложения)

3) исполняемый код определен как путь к исполняемому файлу (bare\_metal.elf)

## Сборка

Команда консоли:

hss-payload-generator.exe -c payload.yaml payload.bin

где,

hss-payload-generator.exe – исполняемый файл генератора образа

payload.yaml – файл конфигурации образа

payload.bin – образ пользовательского приложения в формате bin

## Конвертация в hex

Команда консоли:

bincopy.exe convert -i binary -o ihex payload.bin payload.hex

или

objcopy.exe -I binary -O ihex payload.bin payload.hex

где,

bincopy.exe – утилита конвертации

(например, из состава python3: pip install bincopy)

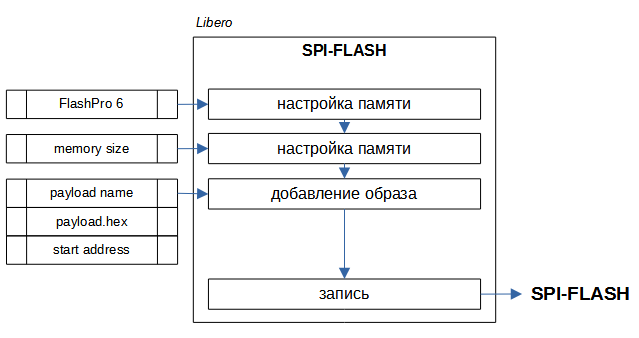
objcopy.exe – утилита конвертации

(например, riscv64-unknown-elf-objcopy.exe из состава SoftConsole 2022)

payload.bin – образ пользовательского приложения в формате bin (вход)

payload.hex – образ пользовательского приложения в формате hex (результат)

# SPI-FLASH



Libero (2022).

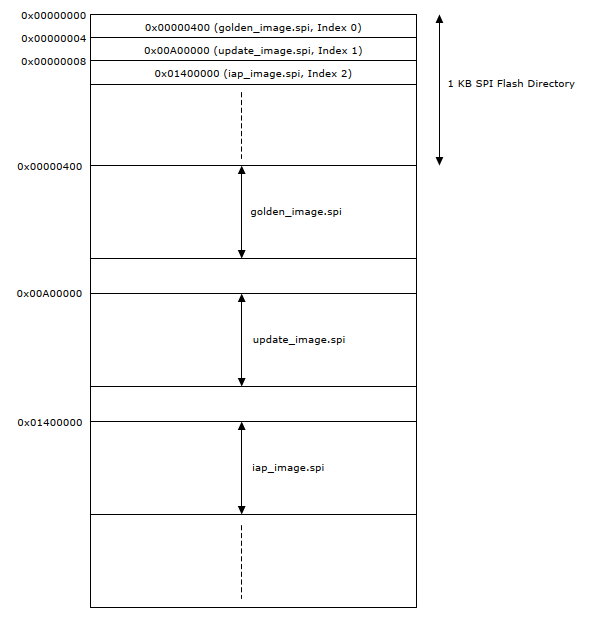
Образ пользовательского приложения хранится в SPI-Flash в отдельной директории.

Первый Килобайт файловой системы:

* таблица адресов директорий SPI-Flash

Далее располагаются директории с образами пользовательских приложений:

* Golden Image – заводской образ
* Update Image – образ обновления
* IAP Image – новый образ



## Настройка памяти

1) В конфигурации дизайна (Libero) выбрать:

Program Design / Configure Design Initialization Data and Memories

2) Выбрать вкладку:

SPI Flash

3) Указать размер используемой SPI-Flash (например, для ICICLE-KIT):

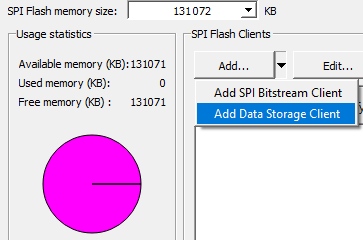
SPI Flash memory size: 131 072 KB



## Добавление образа

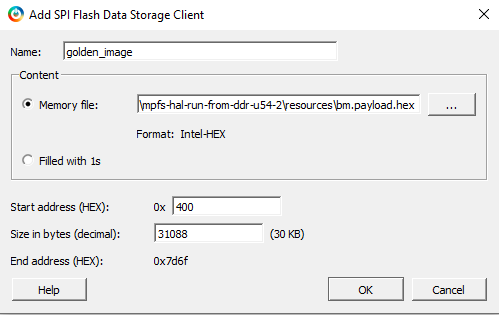
1) Добавить образ пользовательского приложения:

Add… Data Storage Client



2) Указать:

* имя директории (например, golden\_image)
* файл образа (payload.hex)
* стартовый адрес директории в SPI-Flash, куда будет записан образ



3) Применить изменения:

OK … Apply … Project/Save

## Запись

1) Записать сформированную файловую структуру в SPI-Flash:

Run PROGRAM Action/Run PROGRAM\_SPI\_IMAGE Action

\* во всех всплывающих диалоговых окнах нажать «Yes»

\* дождаться завершения процесса записи

# БАЗА ЗНАНИЙ

[Bare Metal Software Projects Structure](https://github.com/polarfire-soc/polarfire-soc-documentation/blob/master/bare-metal-embedded-software/bare-metal-software-project-structure.md)

[Hart Software Services](https://github.com/polarfire-soc/hart-software-services)

[Memory Configuration](https://github.com/polarfire-soc/polarfire-soc-documentation/blob/master/knowledge-base/mpfs-memory-configuration.md)

[Creating Bare Metal Payload For HSS](https://github.com/polarfire-soc/polarfire-soc-bare-metal-examples/tree/main/driver-examples/mss/mpfs-hal)

[Hart Software Services Payloads](https://github.com/polarfire-soc/polarfire-soc-documentation/blob/master/hss-and-u-boot/hss-payloads.md)

[Programming the SPI Flash on a PolarFire SoC Board and Booting via HSS](https://github.com/polarfire-soc/polarfire-soc-documentation/blob/master/how-to/programming-the-spi-flash-on-a-polarfire-soc-board-and-booting-via-hss.md)

[Icicle Kit Bring Up Design Bitstream Builder Readme](https://github.com/polarfire-soc/icicle-kit-minimal-bring-up-design-bitstream-builder)

[PolarFire SoC MSS Driver User Guides](https://github.com/polarfire-soc/polarfire-soc-documentation/blob/master/bare-metal-embedded-software/bare-metal-driver-user-guides/polarfire-soc-mss-driver-user-guides/README.md)