# Cargo-cubemx

## Formal Languages and compilers project summary

## Project topic

An .ioc to Rust compiler named cargo-cubemx.

- **CubeMX** - software provided by ST to simplify the development with STM32 MCUs
- **ioc** - one of output files of CubeMX software which describes configuration of the MCU.
- **Rust** - a compiled programming language with C-like performance and higher-level abstractions
- **Cargo** - a build tool used with Rust

## Project goals

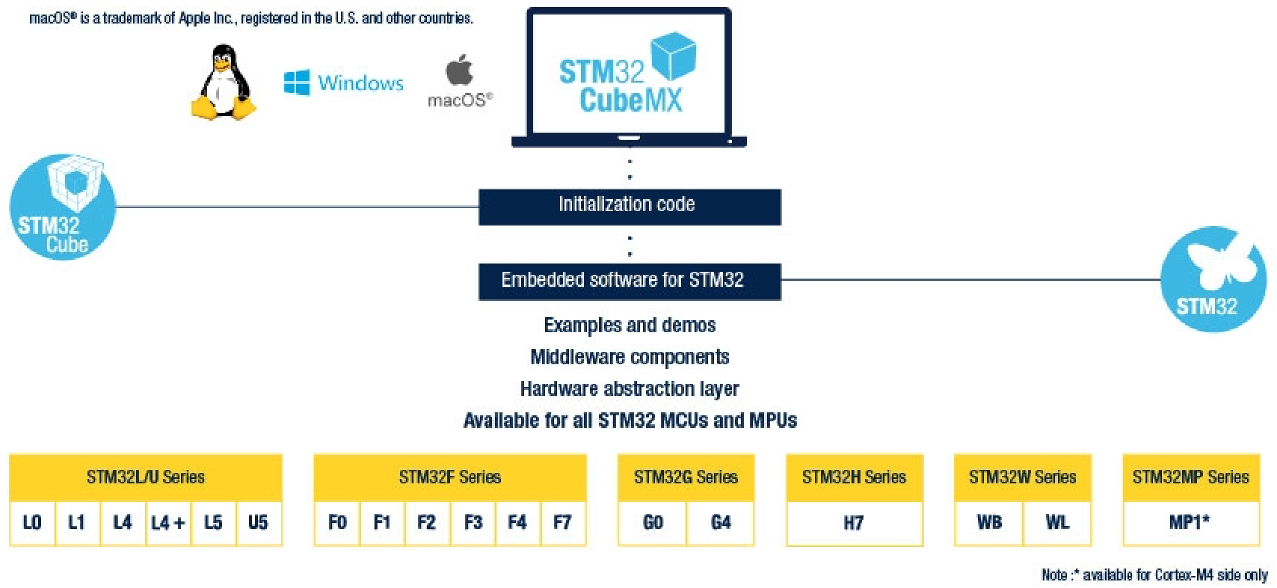
- Parse .ioc file
- Output a file which can be used as a Rust module
- Use existing HAL libraries as compilation targets
- Have the project work as a cargo extension

## Source file and output file

# Source file (.ioc)

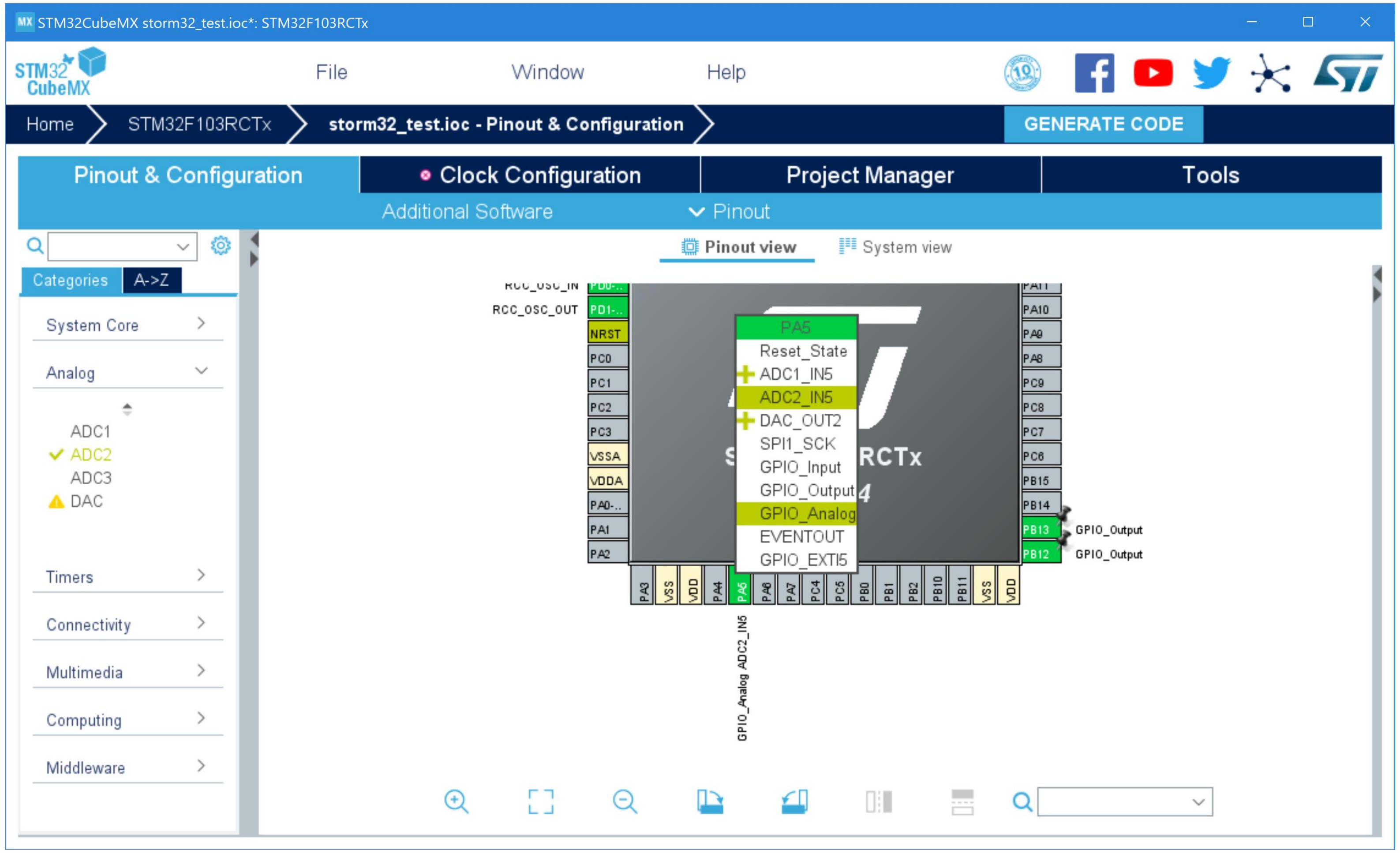
## Where does it come from?

The .ioc is an extension of a configuration file produced by STM32CubeMX software. CubeMX uses the file in further code generation. It is possible to obtain from .ioc configuration files for other STM32 tools and C code.



## Source file (.ioc)

### Where does it come from?



## Source file (.ioc)

### Contains:

MCU description

```
Mcu.Family=STM32F4
Mcu.IP0=ADC1
Mcu.IP1=CAN1
#...
Mcu.Package=LQFP100
Mcu.Pin0=PH0 - OSC_IN
Mcu.Pin1=PH1 - OSC_OUT
Mcu.Pin10=PA4
Mcu.Pin11=PA5
```

## Source file (.ioc)

### Contains:

Clocks configuration:

```
RCC.AHBFreq_Value=75000000
RCC.APB1CLKDivider=RCC_HCLK_DIV2
RCC.APB1Freq_Value=37500000
RCC.APB1TimFreq_Value=75000000
```

## Output file (Rust)

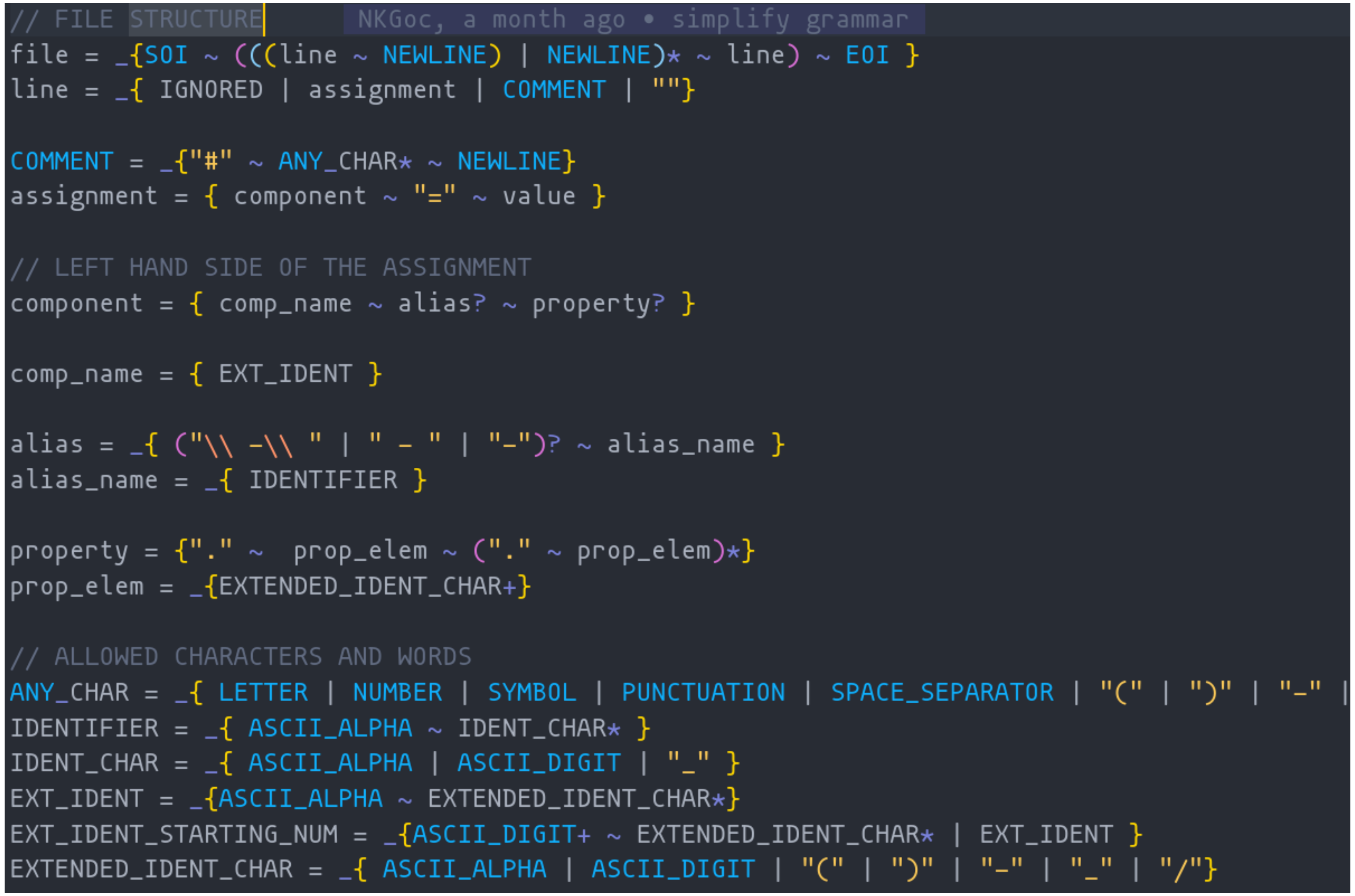
```rust
pub fn init_DMA1_Stream3(per: impl DMASet + PeriAddress, buf: impl StaticWriteBuffer) ->
 Transfer<StreamX<DMA1>, dyn DMASet<StreamX<DMA1>, PeripheralToMemory> + PeriAddress, PeripheralToMemory, 3> {
    let dp = pac::Peripherals::take().unwrap();
    let dma = dp.DMA1;
    let stream = dma::StreamsTuple::new(dma).3;
    let config = dma::config::DmaConfig::default()
    .priority(dma::config::Priority::Low)
    .memory_increment(true)
    ;
    dma::Transfer::init_peripheral_to_memory(
    stream,
    per,
    buf,
    None,
    config
    )
}
```

## Compiler data flow

From input to output:

1. Pest library using PEG is used to perform simple decmposition of the input lines and filtering of not implemented elements
2. Data is read into hashmap and then structures collecting configuration of each component (where component is a token of accepted grammar)
3. Structures are read and code templates are filled using logic dependent on target.

# PEG grammar



```
// FILE STRUCTURE        NKGoc, a month ago • simplify grammar
file = _{SOI ~ (((line ~ NEWLINE) | NEWLINE)* ~ line) ~ EOI }
line = _{ IGNORED | assignment | COMMENT | ""}

COMMENT = _{"#" ~ ANY_CHAR* ~ NEWLINE}
assignment = { component ~ "=" ~ value }

// LEFT HAND SIDE OF THE ASSIGNMENT
component = { comp_name ~ alias? ~ property? }

comp_name = { EXT_IDENT }

alias = _{ ("\\ -\\ " | " - " | "-")? ~ alias_name }
alias_name = _{ IDENTIFIER }

property = {"." ~  prop_elem ~ ("." ~ prop_elem)*}
prop_elem = _{EXTENDED_IDENT_CHAR+}

// ALLOWED CHARACTERS AND WORDS
ANY_CHAR = _{ LETTER | NUMBER | SYMBOL | PUNCTUATION | SPACE_SEPARATOR | "(" | ")" | "-" |
IDENTIFIER = _{ ASCII_ALPHA ~ IDENT_CHAR* }
IDENT_CHAR = _{ ASCII_ALPHA | ASCII_DIGIT | "_" }
EXT_IDENT = _{ASCII_ALPHA ~ EXTENDED_IDENT_CHAR*}
EXT_IDENT_STARTING_NUM = _{ASCII_DIGIT+ ~ EXTENDED_IDENT_CHAR* | EXT_IDENT }
EXTENDED_IDENT_CHAR = _{ ASCII_ALPHA | ASCII_DIGIT | "(" | ")" | "-" | "_" | "/"}
```

## Goals realization

### Parsing .ioc

Having input as following:

```
Dma.MEMTOMEM.0.Direction=DMA_MEMORY_TO_MEMORY
Dma.MEMTOMEM.0.FIFOMode=DMA_FIFOMODE_ENABLE
Dma.MEMTOMEM.0.FIFOThreshold=DMA_FIFO_THRESHOLD_FULL
Dma.MEMTOMEM.0.Instance=DMA2_Stream1
Dma.MEMTOMEM.0.MemBurst=DMA_MBURST_SINGLE
Dma.MEMTOMEM.0.MemDataAlignment=DMA_MDATAALIGN_BYTE
Dma.MEMTOMEM.0.MemInc=DMA_MINC_ENABLE
Dma.MEMTOMEM.0.Mode=DMA_NORMAL
Dma.MEMTOMEM.0.PeriphBurst=DMA_PBURST_SINGLE
Dma.MEMTOMEM.0.PeriphDataAlignment=DMA_PDATAALIGN_BYTE
Dma.MEMTOMEM.0.PeriphInc=DMA_PINC_ENABLE
Dma.MEMTOMEM.0.Priority=DMA_PRIORITY_LOW
```

# Goals realization

## Parsing .ioc

Data in the structures:

```
struct Dma {
    dma_instance: "DMA2_Stream1",
    signal: "MEMTOMEM",
    channel: "0",
    priority: DmaPriority::Low,
    direction: DmaDirection::MemoryToMemory,
    fifo_mode: true,
    mem_inc: true,
    per_inc: true,
}
```

## Goals realization

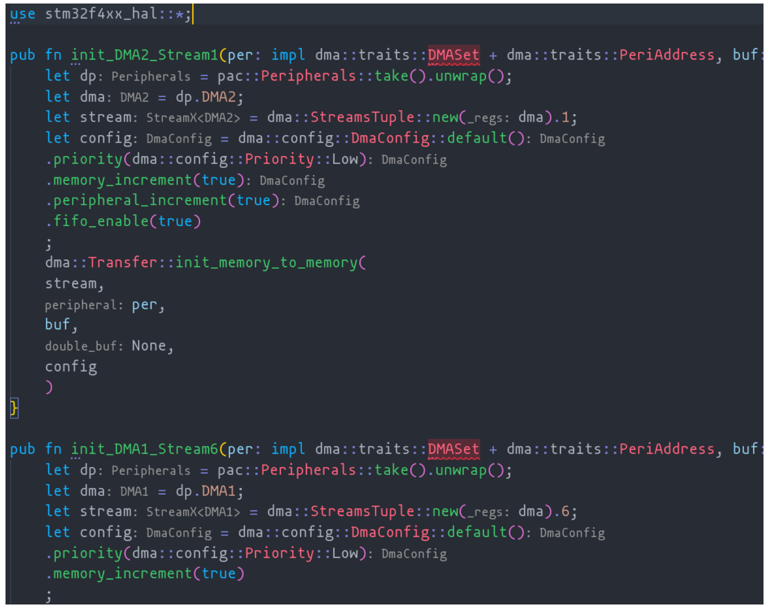
### Parsing .ioc

Currently supported are:

- USART receivers and transmitters initialization
- Direct Memory Access transfer initialization

# Goals realization

## Output as a Rust module



```rust
use stm32f4xx_hal::*;

pub fn init_DMA2_Stream1(per: impl dma::traits::DMASet + dma::traits::PeriAddress, buf
    let dp: Peripherals = pac::Peripherals::take().unwrap();
    let dma: DMA2 = dp.DMA2;
    let stream: StreamX<DMA2> = dma::StreamsTuple::new(_regs: dma).1;
    let config: DmaConfig = dma::config::DmaConfig::default(): DmaConfig
    .priority(dma::config::Priority::Low): DmaConfig
    .memory_increment(true): DmaConfig
    .peripheral_increment(true): DmaConfig
    .fifo_enable(true)
    ;
    dma::Transfer::init_memory_to_memory(
    stream,
    peripheral: per,
    buf,
    double_buf: None,
    config
    )
}

pub fn init_DMA1_Stream6(per: impl dma::traits::DMASet + dma::traits::PeriAddress, buf
    let dp: Peripherals = pac::Peripherals::take().unwrap();
    let dma: DMA1 = dp.DMA1;
    let stream: StreamX<DMA1> = dma::StreamsTuple::new(_regs: dma).6;
    let config: DmaConfig = dma::config::DmaConfig::default(): DmaConfig
    .priority(dma::config::Priority::Low): DmaConfig
    .memory_increment(true)
    ;
```

To use it your source just add:

```
mod <output_file_name>;
```

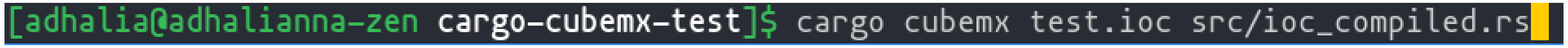
## Goals realization

### Use existing HAL libraries as compilation targets

Currently supported is only F4 family of MCUs and its HAL implementation as stm32f4xx-hal crate.

## Goals realization

### Have the project work as a cargo extension



```
[adhalia@adhalianna-zen cargo-cubemx-test]$ cargo cubemx test.ioc src/ioc_compiled.rs
```

## Difficulties met

1. Late in the project it has been noticed that CubeMX does not mention some configuration elements in its output file if their value is equal to some default.
2. Information from multiple lines must be analyzed together - language is context sensitive.
3. .ioc description of MCU uses incosistent naming conventions and has incosistent aliasing syntax.

## Thank you for your attention!