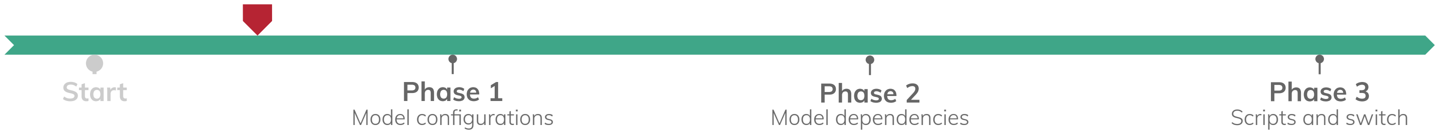
RIOT Summit 2020 - Breakout sessions

Kconfig for RIOT

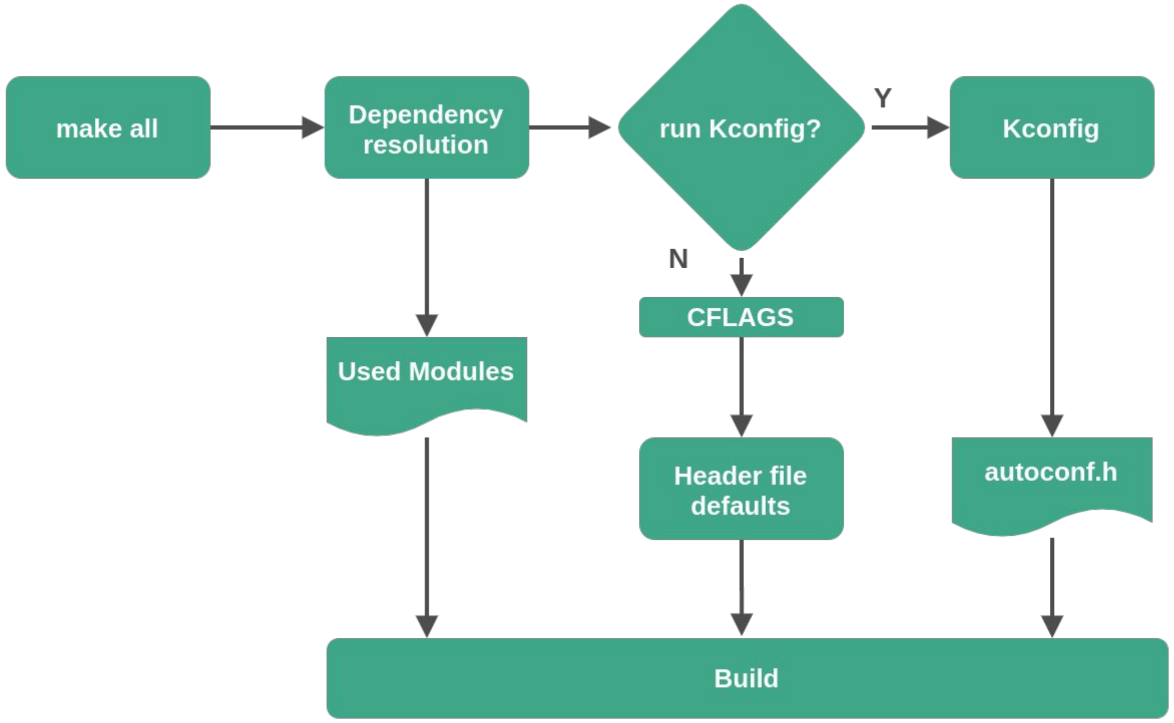
Configuration Task Force

A short recap

What we aim for



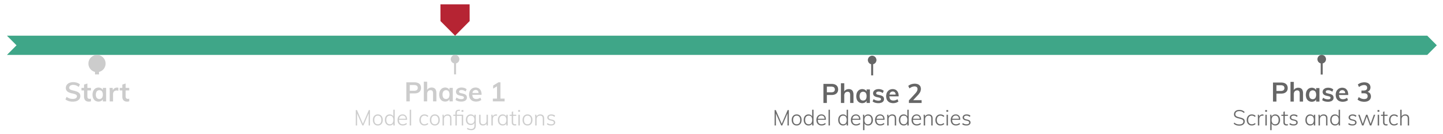
Current state



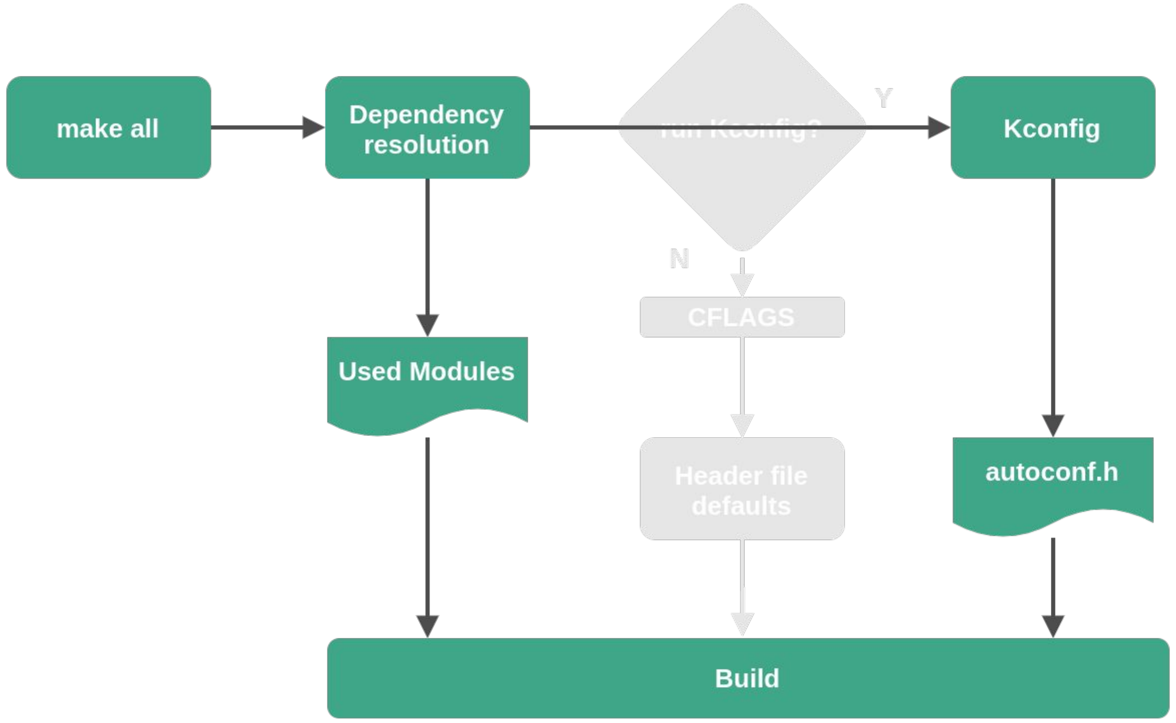
Kconfig migration

- **Phase 1**
  - Identification and documentation of **compile-time configuration parameters**
  - Modelling of those parameters as Kconfig symbols
    - Configuration via Kconfig is **optional** and can be activated
    - Tracking of modules can be found in issue #12888
  - ~60 drivers, networking modules and packages
  - Ongoing work on boards and CPUs
    - Clock configuration for nucleo-based boards
    - Clock and WiFi configuration for ESP

This phase is still in progress. **Contributions are welcome!**



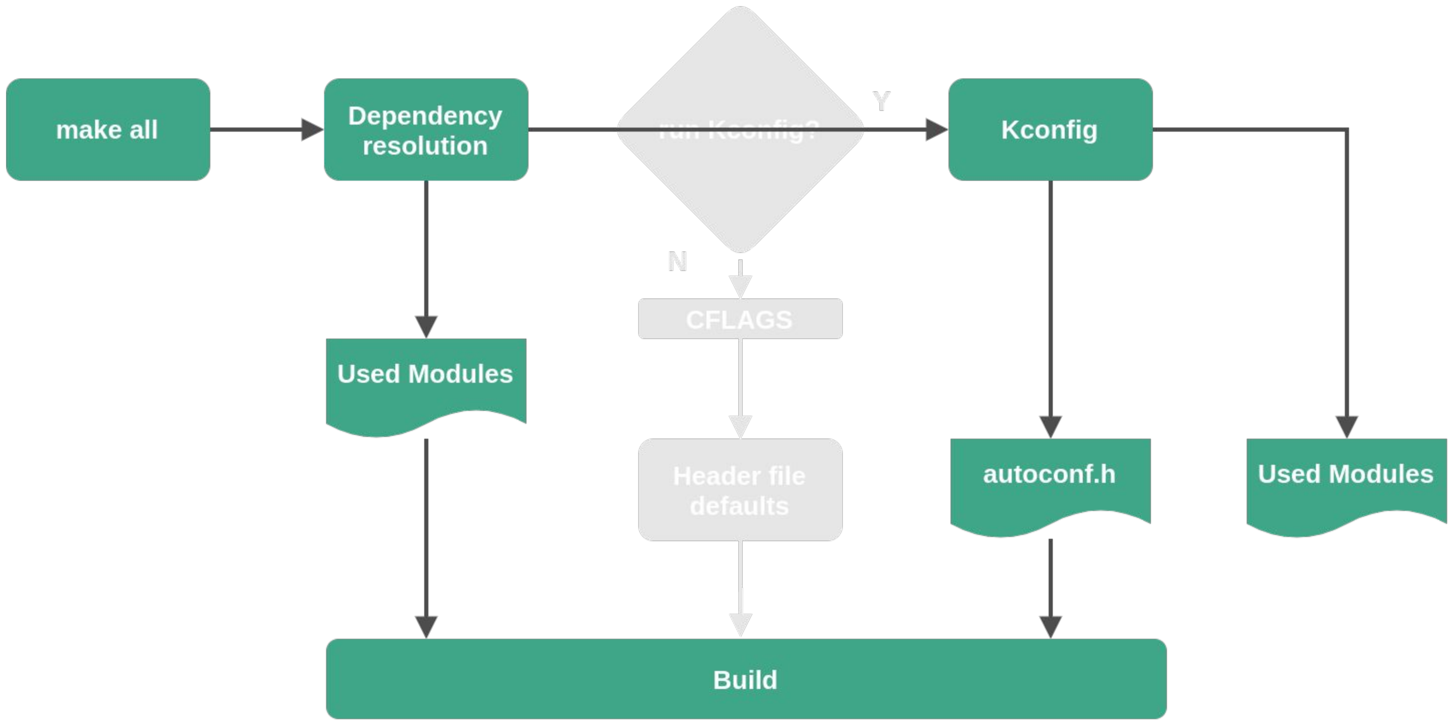
After completion of Phase 1



Kconfig migration

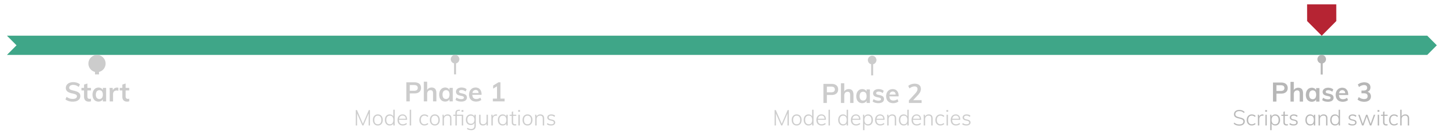
- **Phase 2**
  - First milestone: done! 🎉
    - Model **features** as Kconfig symbols
    - Model **CPUs** (model, line, family, arch) and **boards** as Kconfig symbols
    - A **test** has been added to keep sync with Makefile
  - Second milestone:
    - Model **modules** as Kconfig symbols
    - Add **default configurations** (`.config` files) for boards, CPUs and applications
    - **Test** to check binaries resulting binaries

After completion of Phase 2

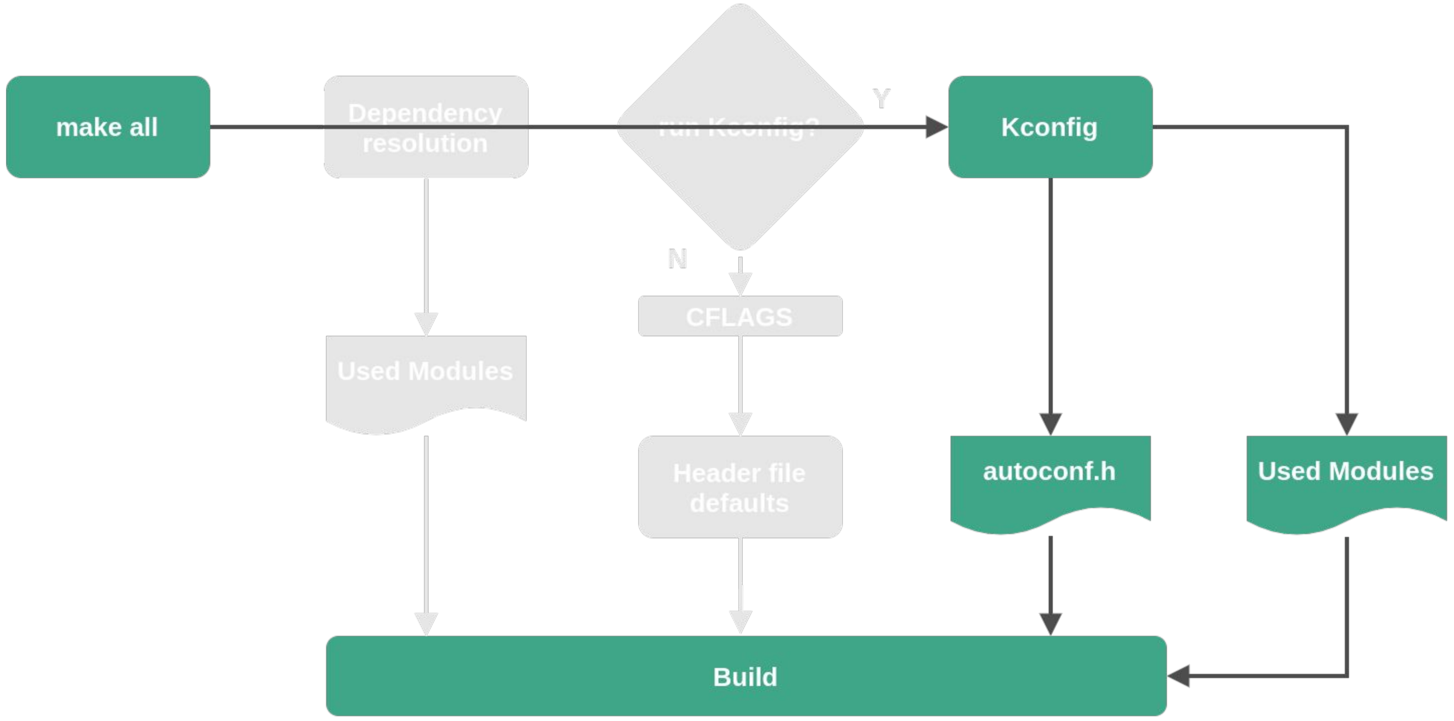


Kconfig migration

- **Phase 3**
  - `Make` targets (e.g. the ones used CI and testing)
    - boards-supported
    - features-missing
    - ...
  - Switch to Kconfig as default



After completion of Phase 3



Some advanced features

Incremental compilation

- PR #14654 introduced incremental compilation when configuration parameters are modified via Kconfig.
- Allows to reduce build time when iterating over different configurations.
- Same approach as Linux, using the **fixdep.c** script
    - Configuration macros are searched in the .c file
    - .d files are modified so the object file depends on a dummy header file
    - Kconfig generates one dummy file per configuration parameter. The file is touched when the configuration changes.

Parameterized tests

- In issue #14669 it was suggested to build applications in the CI using different environments.
- Could be done by providing multiple .config files to:
  - Apply different groups of configurations
  - Enable/Disable modules
- Leveraging incremental compilation, object files can be shared between every build.
- Potentially some test application could be merged => reducing CI build time

Features provided by modules and packages

- Features are being modelled as Kconfig symbols, just as modules.
- In Kconfig features are provided and checked at the same moment modules are selected.
- Make it easier to model dependencies and conditions in the build system.

Advanced configurations

- By using ranges and choices, there is fine control over the possible values the user can assign to a configuration parameter:
  - Multiplier and divider parameters during clock configurations

- By evaluating the features provided by the hardware and other modules the defaults can be adapted as the user changes configurations:
  - When hardware acceleration is available for cryptographic operations, use that peripheral.
  - If a driver which provides hardware acceleration is selected, use that implementation over software one.

Modelling in Kconfig

Features

- Boolean non-visible symbols.
- Selected by providers:
    - `CPU_MODEL`, `CPU_ARCH`, etc.
    - Boards
    - Modules and packages
- Selection may be conditional

```
config HAS_PERIPH_UART_MODECFG
   bool
   help
       Indicates that the UART peripheral allows mode configuration.

config CPU_COMMON_SAM0
   bool
   # [...]
   select HAS_PERIPH_UART_MODECFG
```

Modules and Packages

- Boolean symbols, most times visible (i.e. have a prompt).
- May or may not have dependencies on:
  - Hardware (e.g. `CPU_MODEL, CPU_FAM`)
  - Hardware features (e.g. `HAS_PERIPH_HWRNG`)
  - Other modules or conditions
- Defaults may apply conditionally

```
config MODULE_PERIPH_ADC
   bool "ADC peripheral driver"
   depends on HAS_PERIPH_ADC
   select MODULE_PERIPH_COMMON

config MODULE_PERIPH_INIT_ADC
   bool "Auto initialize ADC peripheral"
   default y
   depends on MODULE_PERIPH_INIT
   depends on MODULE_PERIPH_ADC
```

Configuration parameters

- Most of times associated to a module, CPU, board or package
- Could also be provided by the application
- Multiple types: bool, int, string, hex
- Multiple defaults using conditionals
- Adding conditions to the prompts configurability can be controlled
- Adding dependencies configurability and generation of the values can be controlled

```
config CLOCK_PLL_M
   int "M: Division factor 'M' for the main PLL input clock" if USE_CLOCK_PLL
   default 6 if !BOARD_HAS_HSE
   default 5
   range 1 8
config CLOCK_PLL_N
   int "Main PLL multiplication factor 'N' for VCO" if USE_CLOCK_PLL
   default 20
   range 8 86
```

APIs with multiple implementations

- Frontend / Backend.
- Choices with multiple options:
    - Defaults can depend on features or other symbols.
    - Choices can be extended from other files.
    - Configuration parameters:
        - That apply to all implementations.
        - That are available only for one implementation.
- One symbol for the API module, and one symbol for the implementer.

APIs with multiple implementations

```
menuconfig CRYPTO_AES
   bool "AES"
   select MOD_CRYPTO

choice CRYPTO_AES_IMPLEMENTATION
   bool "AES implementation"
   depends on CRYPTO_AES
   default MOD_PERIPH_CRYPTO_AES

config MOD_PERIPH_CRYPTO_AES
   bool "Hardware accelerated"
   depends on HAS_PERIPH_CRYPTO_AES

config MOD_CRYPTO_AES
   bool "Software"

endchoice
```

```
config PKG_CRYPTOAUTHLIB
   bool "Cryptoauth Library"

choice CRYPTO_AES_IMPLEMENTATION

menuconfig CRYPTOAUHLIB_AES
   bool "Cryptoauth Library"
   depends on PKG_CRYPTOAUTHLIB

# configurations for this implementation
config CRYPTOAUHLIB_AES_BUFFER
   bool "Some buffer"
   depends on CRYPTOAUHLIB_AES

endchoice
```

Peripheral driver configurations

- Peripheral driver symbols, feature symbols and generic configurations are shared.
- Some platforms present extra configurations.
- By using a convention we can display the configurations in the correct place

```
menuconfig KCONFIG_USEMODULE_PERIPH_TIMER
   bool "Configure timer peripheral driver"
   depends on USEMODULE_PERIPH_TIMER
   help
       Configure Timer peripheral using Kconfig.

# Include CPU specific configurations
if KCONFIG_USEMODULE_PERIPH_TIMER
osource "$(RIOTCPU)/$(CPU)/periph/Kconfig.timer"
endif
```

```
# cpu/efm32/periph/Kconfig.timer
config EFM32_XTIMER_USE_LETIMER
   bool "Xtimer uses letimer"
   depends on CPU_COMMON_EFM32
   Depends on USEMODULE_XTIMER
   help
       Xtimer will use EFM32 Low Energy Timer as
       its low level timer.
```

Features conflicting

- Currently express that two features can’t be used at the same time
  - Used to express mutual exclusion between two peripheral drivers
- In Kconfig mutual exclusion is modelled using choices
  - We need to know the choices and options beforehand
  - Depend on the platform (board, CPU, etc.)
- The conflicting condition can be defined by setting an ERROR symbol
  - The ERROR symbol could be a string which is set when a given condition is true

```
config ERROR_CONFLICT
   string

config ERROR_CONFLICT
   default "Can't select RTT and RTC drivers at the same time"
   depends on CPU_COMMON_SAM0
   depends on MODULE_PERIPH_RTT && MODULE_PERIPH_RTC
```