How to integrate my (proprietary) code in Zephyr

Iuliana Prodan

April 2024

Content

Zephyr Project structure

```
zephyrproject/
├── .west/
│    └── config
├── bootloader/
├── modules/
├── optional/
├── tools/
└── zephyr/
```

Examples of modules from zephyrproject-rtos:

- sdk-ng / crosstool-ng
- docker-image*
- cmsis-dsp / cmsis-nn
- OpenAMP / libmetal
- littlefs

Zephyr structure

```
zephyrproject/
├── .west/
│     └── config
├── bootloader/
├── modules/
├── optional/
├── tools/
└── zephyr/
```

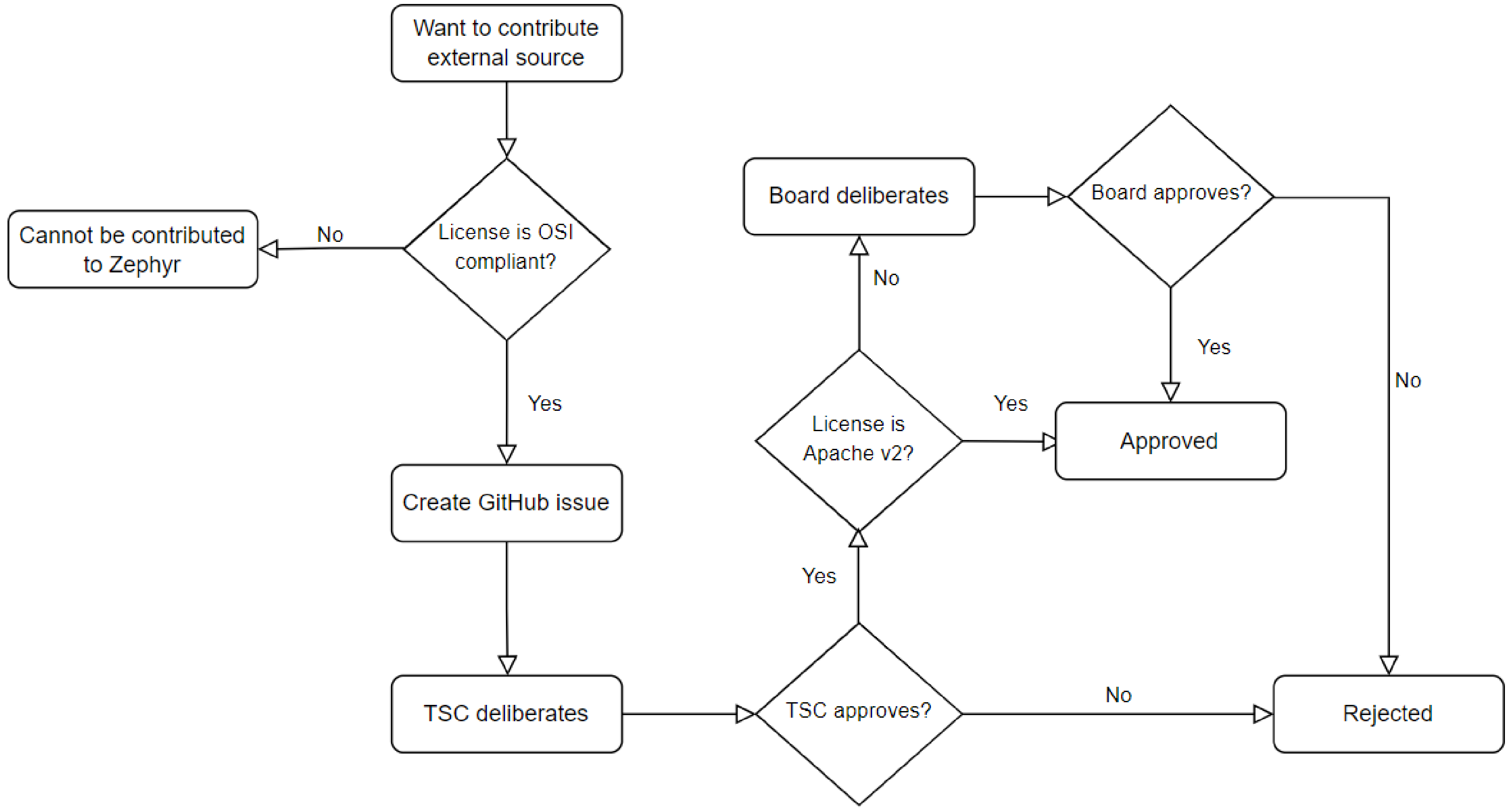
```
zephyr/
├── arch
├── boards
├── cmake
├── drivers
├── dts
├── kernel
├── lib
├── samples
│   ├── application_development
│   ├── hello_world
│   └── ...
├── soc
├── submanifests
│   ├── optional.yaml
│   └── ...
├── tests
├── west.yml
└── ...
```

Licensing

- Zephyr uses the Apache 2.0 license
  - permissive open-source license that allows you to freely use, modify, distribute and sell your own products
- Imported or reused components that use other licensing - GPLv2 License
- When submitting a patch with `Signed-off-by` one agrees to Developer Certificate of Origin (DCO)
- Each Zephyr source code is mandatory to have a one-line `SPDX-License-Identifier` comment

Licensing

- Importing code into the Zephyr OS from other projects that use a license other than the Apache 2.0 license needs to be approved by the Zephyr governing board



https://github.com/zephyrproject-rtos/zephyr/blob/main/doc/contribute/external.rst

Application development - Prerequisites

- Arch support is mandatory in Zephyr
  - board, SoC, device tree support are expected to be in Zephyr
  - structure for out-of-tree board, SoC development needs to be like boards and SoCs maintained in the Zephyr tree

```
west build -b <board name> --
-DBOARD_ROOT=<path to boards>
-DSOC_ROOT=<path to soc>
-DDTS_ROOT=<path to dts root>
```

Board structure:

```
boards/<VENDOR>/plank
├── board.yml
├── board.cmake
├── CMakeLists.txt
├── doc
│   ├── plank.png
│   └── index.rst
├── Kconfig.plank
├── Kconfig.defconfig
├── plank_defconfig
├── plank_<qualifiers>_defconfig
├── plank.dts
├── plank_<qualifiers>.dts
└── plank.yaml
```

https://docs.zephyrproject.org/latest/develop/application/index.html#custom-board-devicetree-and-soc-definitions

Application development - Prerequisites

- Arch support is mandatory in Zephyr
- Zephyr's build system is based on [Cmake]:
  - application centric
- Modes of integration for external code:
  - Integration in the main Zephyr repository
  - Integration as a module
    - main manifest file (west.yaml)
      - vendor HAL
      - libraries (source code)
      - tools
    - optional modules (submanifests/optional.yml)
    - external modules

Zephyr application structure:

```
<app>
   ├── CMakeLists.txt
   ├── app.overlay
   ├── prj.conf
   ├── sample.yml
   ├── README
   └── src
        └── main.c
```

Creating an application

- Where do we add the application?
  - Application types

Repository	Workspace	Freestanding
Zephyr repository	west workspace where Zephyr is installed	Out-of-tree /Other locations

zephyrproject/
├── .west/
│ └── config
└── zephyr/
  ├── arch
  ├── boards
  ├── cmake
  ├── drivers
  ├── samples
  │ └─ **app/**
  ├── tests
  └── …
	zephyrproject/
	├── .west/
	│ └── config
	├── bootloader/
	├── modules/
	├── optional/
	├── tools/
	├── zephyr/
	└── applications
	  └─ **app/**
		home/
		├── …
		├── zephyrproject/
		│ ├── .west/
		│ │ └── config
		│ ├── bootloader/
		│ ├── modules/
		│ ├── optional/
		│ ├── tools/
		│ └── zephyr/
		└── **app/**

Creating an application

- Number crunching sample
- Where do we add the application?
  - In and out of Zephyr tree
- How to integrate proprietary code?
  - NatureDSP library, from Cadence, from an out-of-tree location
- How to use a Zephyr module?
  - CMSIS-DSP

Creating an application

```
zephyr/samples/
└── application_development
    ├── ...
    └── number_crunching
        ├── CMakeLists.txt
        ├── include
        │     ├─ input.h
        │     └─ math_ops.h
        ├── prj.conf
        ├── README.rst
        ├── sample.yaml
        └── src
              ├─ cmsis_dsp_wrapper.c
              ├─ main.c
              ├─ math_ops.c
              └─ nature_dsp_wrapper.c
```

Creating an application – Solution 1

```
zephyr/samples/
└── application_development
    ├── ...
    └── number_crunching
        ├── CMakeLists.txt
        ├── include
        │   ├ input.h
        │   └ math_ops.h
        ├── prj.conf
        ├── README.rst
        ├── sample.yaml
        └── src
            ├ cmsis_dsp_wrapper.c
            ├ main.c
            ├ math_ops.c
            └ nature_dsp_wrapper.c
```

```cmake
# SPDX-License-Identifier: Apache-2.0

cmake_minimum_required(VERSION 3.20.0)
find_package(Zephyr REQUIRED HINTS $ENV{ZEPHYR_BASE})
project(proprietary_lib)

# defines targets and sources
target_sources(app PRIVATE
    src/main.c
    src/math_ops.c
)
zephyr_include_directories(include)

if(DEFINED ENV{LIB_LOCATION})
    message(STATUS "LIB_LOCATION environment variable defined")

    # contains a "proprietary" library we will link to
    # this should set the INCLUDE_DIR, LIB_DIR and LIB_NAME variables
    add_subdirectory($ENV{LIB_LOCATION} ${CMAKE_CURRENT_BINARY_DIR}/proprietary)

    # this is an example for NatureDSP backend
    target_sources(app PRIVATE
        src/nature_dsp_wrapper.c
    )

    if(INCLUDE_DIR)
        zephyr_include_directories($ENV{LIB_LOCATION}/${INCLUDE_DIR})
    endif()

    if(LIB_DIR AND LIB_NAME)
        zephyr_link_libraries($ENV{LIB_LOCATION}/${LIB_DIR}/${LIB_NAME})
    endif()
else()
    message(STATUS "LIB_LOCATION environment variable NOT defined")
    # this is an example for CMSIS-DSP backend
    target_sources(app PRIVATE
        src/cmsis_dsp_wrapper.c
    )
endif()
```

Creating an application – Solution 1

```
zephyr/samples/
└── application_development
    ├── ...
    └── number_crunching
        ├── CMakeLists.txt
        ├── include
        │   ├─ input.h
        │   └─ math_ops.h
        ├── prj.conf
        ├── README.rst
        ├── sample.yaml
        └── src
            ├─ cmsis_dsp_wrapper.c
            ├─ main.c
            ├─ math_ops.c
            └─ nature_dsp_wrapper.c
```

```cmake
# SPDX-License-Identifier: Apache-2.0

cmake_minimum_required(VERSION 3.20.0)
find_package(Zephyr REQUIRED HINTS $ENV{ZEPHYR_BASE})
project(proprietary_lib)

# defines targets and sources
target_sources(app PRIVATE
    src/main.c
    src/math_ops.c
)
zephyr_include_directories(include)

if(DEFINED ENV{LIB_LOCATION})
    message(STATUS "LIB_LOCATION environment variable defined")

    # contains a "proprietary" library we will link to
    # this should set the INCLUDE_DIR, LIB_DIR and LIB_NAME variables
    add_subdirectory($ENV{LIB_LOCATION} ${CMAKE_CURRENT_BINARY_DIR}/proprietary)

    # this is an example for NatureDSP backend
    target_sources(app PRIVATE
        src/nature_dsp_wrapper.c
    )

    if(INCLUDE_DIR)
        zephyr_include_directories($ENV{LIB_LOCATION}/${INCLUDE_DIR})
    endif()

    if(LIB_DIR AND LIB_NAME)
        zephyr_link_libraries($ENV{LIB_LOCATION}/${LIB_DIR}/${LIB_NAME})
    endif()
else()
    message(STATUS "LIB_LOCATION environment variable NOT defined")
    # this is an example for CMSIS-DSP backend
    target_sources(app PRIVATE
        src/cmsis_dsp_wrapper.c
    )
endif()
```

Creating an application – Solution 1

```
zephyr/samples/
└── application_development
    ├── ...
    └── number_crunching
        ├── CMakeLists.txt
        ├── include
        │   ├ input.h
        │   └ math_ops.h
        ├── prj.conf
        ├── README.rst
        ├── sample.yaml
        └── src
            ├ cmsis_dsp_wrapper.c
            ├ main.c
            ├ math_ops.c
            └ nature_dsp_wrapper.c
```

```
# SPDX-License-Identifier: Apache-2.0

cmake_minimum_required(VERSION 3.20.0)
find_package(Zephyr REQUIRED HINTS $ENV{ZEPHYR_BASE})
project(proprietary_lib)

# defines targets and sources
target_sources(app PRIVATE
    src/main.c
    src/math_ops.c
)
zephyr_include_directories(include)

if(DEFINED ENV{LIB_LOCATION})
    message(STATUS "LIB_LOCATION environment variable defined")

    # contains a "proprietary" library we will link to
    # this should set the INCLUDE_DIR, LIB_DIR and LIB_NAME variables
    add_subdirectory($ENV{LIB_LOCATION} ${CMAKE_CURRENT_BINARY_DIR}/proprietary)

    # this is an example for NatureDSP backend
    target_sources(app PRIVATE
        src/nature_dsp_wrapper.c
    )

    if(INCLUDE_DIR)
        zephyr_include_directories($ENV{LIB_LOCATION}/${INCLUDE_DIR})
    endif()

    if(LIB_DIR AND LIB_NAME)
        zephyr_link_libraries($ENV{LIB_LOCATION}/${LIB_DIR}/${LIB_NAME})
    endif()
else()
    message(STATUS "LIB_LOCATION environment variable NOT defined")
    # this is an example for CMSIS-DSP backend
    target_sources(app PRIVATE
        src/cmsis_dsp_wrapper.c
    )
endif()
```

Creating an application – Solution 1

```
home/external_lib_location/
  ├── CMakeLists.txt
  ├── include
  │      ├── NatureDSP_Signal_audio.h
  │      ├── NatureDSP_Signal_complex.h
  │      ├── NatureDSP_Signal_fft.h
  │      ├── NatureDSP_Signal_fir.h
  │      ├── NatureDSP_Signal_fit.h
  │      ├── NatureDSP_Signal.h
  │      ├── NatureDSP_Signal_id.h
  │      ├── NatureDSP_Signal_iir.h
  │      ├── NatureDSP_Signal_img.h
  │      ├── NatureDSP_Signal_math.h
  │      ├── NatureDSP_Signal_matinv.h
  │      ├── NatureDSP_Signal_matop.h
  │      ├── NatureDSP_Signal_vector.h
  │      └── NatureDSP_types.h
  └── lib
        └── NatureDSPLib.a
```

Creating an application – Solution 1

```
home/external_lib_location/
 ├── CMakeLists.txt
 ├── include
 │      ├── NatureDSP_Signal_audio.h
 │      ├── NatureDSP_Signal_complex.h
 │      ├── NatureDSP_Signal_fft.h
 │      ├── NatureDSP_Signal_fir.h
 │      ├── NatureDSP_Signal_fit.h
 │      ├── NatureDSP_Signal.h
 │      ├── NatureDSP_Signal_id.h
 │      ├── NatureDSP_Signal_iir.h
 │      ├── NatureDSP_Signal_img.h
 │      ├── NatureDSP_Signal_math.h
 │      ├── NatureDSP_Signal_matinv.h
 │      ├── NatureDSP_Signal_matop.h
 │      ├── NatureDSP_Signal_vector.h
 │      └── NatureDSP_types.h
 └── lib
     └── NatureDSPLib.a
```

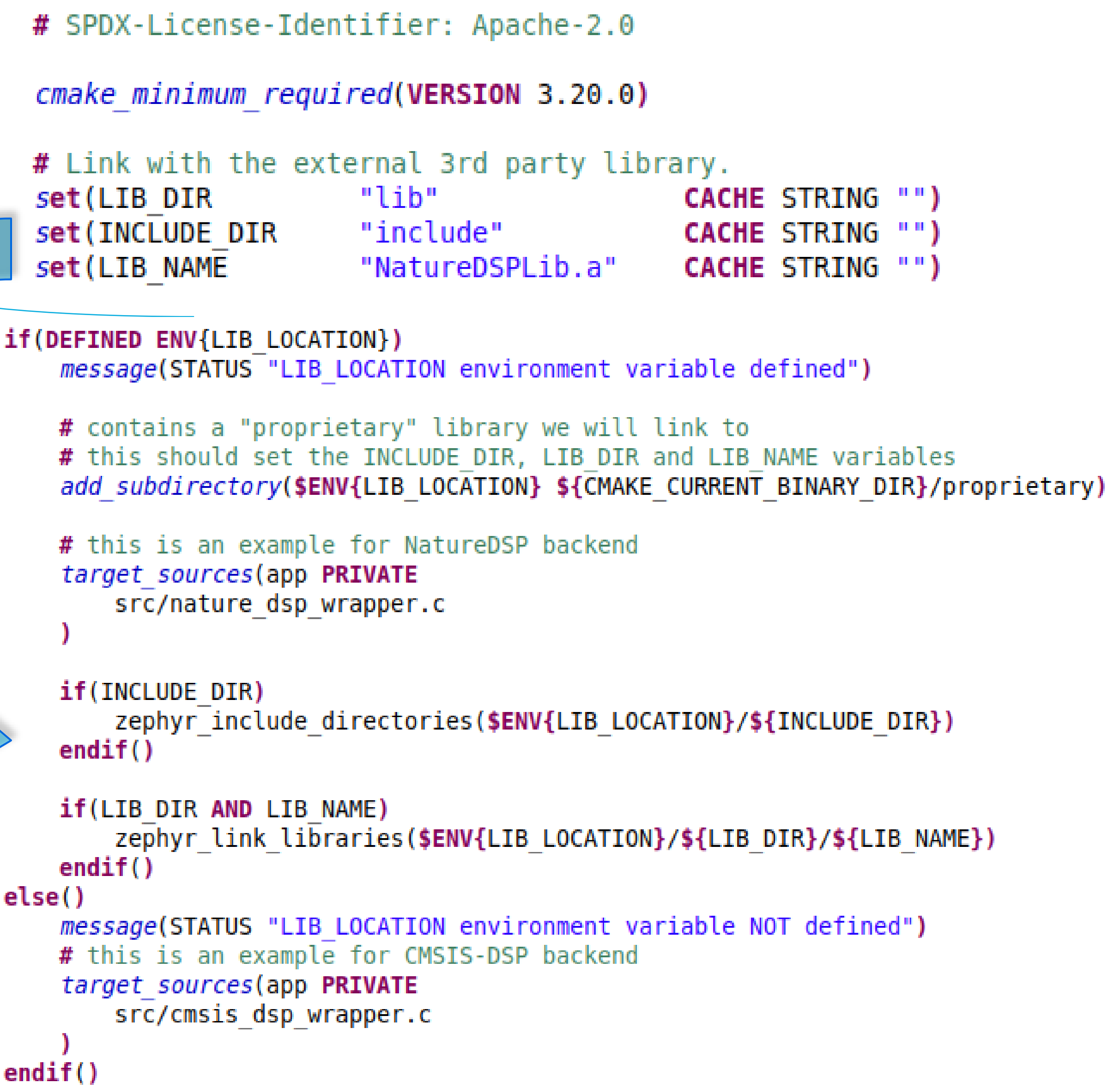
```cmake
# SPDX-License-Identifier: Apache-2.0

cmake_minimum_required(VERSION 3.20.0)

# Link with the external 3rd party library.
set(LIB_DIR          "lib"              CACHE STRING "")
set(INCLUDE_DIR      "include"          CACHE STRING "")
set(LIB_NAME         "NatureDSPLib.a"   CACHE STRING "")
```

Creating an application – Solution 1

```
home/external_lib_location/
├── CMakeLists.txt
├── include
│      ├── NatureDSP_Signal_audio.h
│      ├── NatureDSP_Signal_complex.h
│      ├── NatureDSP_Signal_fft.h
│      ├── NatureDSP_Signal_fir.h
│      ├── NatureDSP_Signal_fit.h
│      ├── NatureDSP_Signal.h
│      ├── NatureDSP_Signal_id.h
│      ├── NatureDSP_Signal_iir.h
│      ├── NatureDSP_Signal_img.h
│      ├── NatureDSP_Signal_math.h
│      ├── NatureDSP_Signal_matinv.h
│      ├── NatureDSP_Signal_matop.h
│      ├── NatureDSP_Signal_vector.h
│      └── NatureDSP_types.h
└── lib
    └── NatureDSPLib.a
```



```
# SPDX-License-Identifier: Apache-2.0

cmake_minimum_required(VERSION 3.20.0)

# Link with the external 3rd party library.
set(LIB_DIR         "lib"              CACHE STRING "")
set(INCLUDE_DIR     "include"          CACHE STRING "")
set(LIB_NAME        "NatureDSPLib.a"   CACHE STRING "")
```

```
if(DEFINED ENV{LIB_LOCATION})
    message(STATUS "LIB_LOCATION environment variable defined")

    # contains a "proprietary" library we will link to
    # this should set the INCLUDE_DIR, LIB_DIR and LIB_NAME variables
    add_subdirectory($ENV{LIB_LOCATION} ${CMAKE_CURRENT_BINARY_DIR}/proprietary)

    # this is an example for NatureDSP backend
    target_sources(app PRIVATE
        src/nature_dsp_wrapper.c
    )

    if(INCLUDE_DIR)
        zephyr_include_directories($ENV{LIB_LOCATION}/${INCLUDE_DIR})
    endif()

    if(LIB_DIR AND LIB_NAME)
        zephyr_link_libraries($ENV{LIB_LOCATION}/${LIB_DIR}/${LIB_NAME})
    endif()
else()
    message(STATUS "LIB_LOCATION environment variable NOT defined")
    # this is an example for CMSIS-DSP backend
    target_sources(app PRIVATE
        src/cmsis_dsp_wrapper.c
    )
endif()
```

Creating an application – Solution 1

```
home/external_lib_location/
├── CMakeLists.txt
├── include
│       ├── NatureDSP_Signal_audio.h
│       ├── NatureDSP_Signal_complex.h
│       ├── NatureDSP_Signal_fft.h
│       ├── NatureDSP_Signal_fir.h
│       ├── NatureDSP_Signal_fit.h
│       ├── NatureDSP_Signal.h
│       ├── NatureDSP_Signal_id.h
│       ├── NatureDSP_Signal_iir.h
│       ├── NatureDSP_Signal_img.h
│       ├── NatureDSP_Signal_math.h
│       ├── NatureDSP_Signal_matinv.h
│       ├── NatureDSP_Signal_matop.h
│       ├── NatureDSP_Signal_vector.h
│       └── NatureDSP_types.h
└── lib
     └── NatureDSPLib.a
```

```
# SPDX-License-Identifier: Apache-2.0

cmake_minimum_required(VERSION 3.20.0)

# Link with the external 3rd party library.
set(LIB_DIR         "lib"               CACHE STRING "")
set(INCLUDE_DIR     "include"           CACHE STRING "")
set(LIB_NAME        "NatureDSPLib.a"    CACHE STRING "")

zephyr_include_directories(${CMAKE_CURRENT_SOURCE_DIR}/${INCLUDE_DIR})
zephyr_link_libraries(${CMAKE_CURRENT_SOURCE_DIR}/${LIB_DIR}/${LIB_NAME})
```

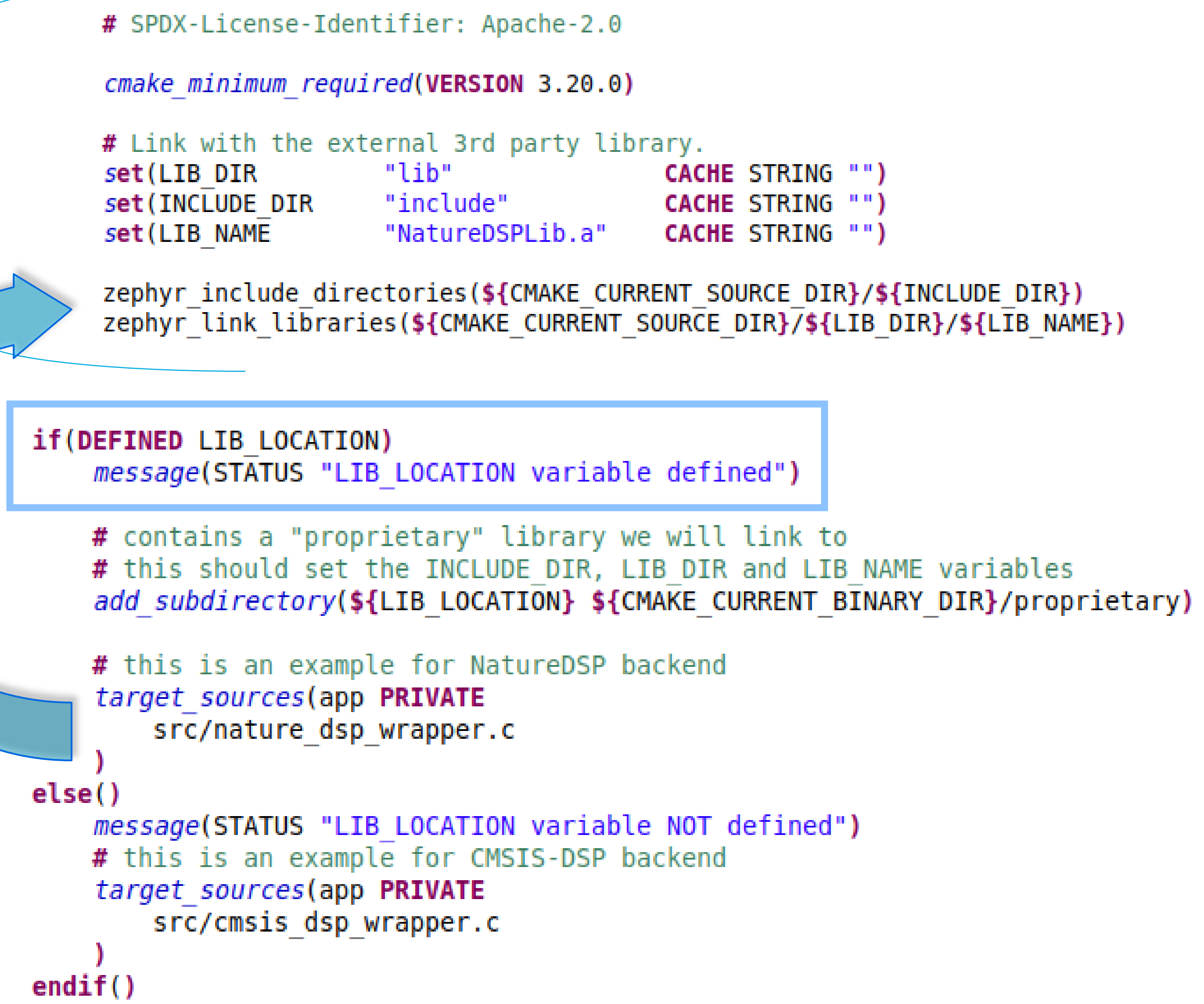
```
if(DEFINED ENV{LIB_LOCATION})
    message(STATUS "LIB_LOCATION environment variable defined")

    # contains a "proprietary" library we will link to
    # this should set the INCLUDE_DIR, LIB_DIR and LIB_NAME variables
    add_subdirectory($ENV{LIB_LOCATION} ${CMAKE_CURRENT_BINARY_DIR}/proprietary)

    # this is an example for NatureDSP backend
    target_sources(app PRIVATE
        src/nature_dsp_wrapper.c
    )
else()
    message(STATUS "LIB_LOCATION environment variable NOT defined")
    # this is an example for CMSIS-DSP backend
    target_sources(app PRIVATE
        src/cmsis_dsp_wrapper.c
    )
endif()
```

Creating an application – Solution 1

```
home/external_lib_location/
 ├── CMakeLists.txt
 ├── include
 │      ├── NatureDSP_Signal_audio.h
 │      ├── NatureDSP_Signal_complex.h
 │      ├── NatureDSP_Signal_fft.h
 │      ├── NatureDSP_Signal_fir.h
 │      ├── NatureDSP_Signal_fit.h
 │      ├── NatureDSP_Signal.h
 │      ├── NatureDSP_Signal_id.h
 │      ├── NatureDSP_Signal_iir.h
 │      ├── NatureDSP_Signal_img.h
 │      ├── NatureDSP_Signal_math.h
 │      ├── NatureDSP_Signal_matinv.h
 │      ├── NatureDSP_Signal_matop.h
 │      ├── NatureDSP_Signal_vector.h
 │      └── NatureDSP_types.h
 └── lib
     └── NatureDSPLib.a
```



```
# SPDX-License-Identifier: Apache-2.0

cmake_minimum_required(VERSION 3.20.0)

# Link with the external 3rd party library.
set(LIB_DIR         "lib"             CACHE STRING "")
set(INCLUDE_DIR     "include"         CACHE STRING "")
set(LIB_NAME        "NatureDSPLib.a"  CACHE STRING "")

zephyr_include_directories(${CMAKE_CURRENT_SOURCE_DIR}/${INCLUDE_DIR})
zephyr_link_libraries(${CMAKE_CURRENT_SOURCE_DIR}/${LIB_DIR}/${LIB_NAME})
```

```
if(DEFINED LIB_LOCATION)
    message(STATUS "LIB_LOCATION variable defined")

    # contains a "proprietary" library we will link to
    # this should set the INCLUDE_DIR, LIB_DIR and LIB_NAME variables
    add_subdirectory(${LIB_LOCATION} ${CMAKE_CURRENT_BINARY_DIR}/proprietary)

    # this is an example for NatureDSP backend
    target_sources(app PRIVATE
        src/nature_dsp_wrapper.c
    )
else()
    message(STATUS "LIB_LOCATION variable NOT defined")
    # this is an example for CMSIS-DSP backend
    target_sources(app PRIVATE
        src/cmsis_dsp_wrapper.c
    )
endif()
```

Building the application

For NatureDSP, set LIB_LOCATION as environment or simple variable:

```
/home/zephyrproject/zephyr$ export LIB_LOCATION=/home/external_lib_location
/home/zephyrproject/zephyr$
west build -p always -b imx8mp_evk//adsp samples/application_development/number_crunching/

/home/zephyrproject/zephyr$
west build -p always -b imx8mp_evk//adsp samples/application_development/number_crunching/
-DLIB_LOCATION=/home/external_lib_location
```

For CMSIS-DSP:

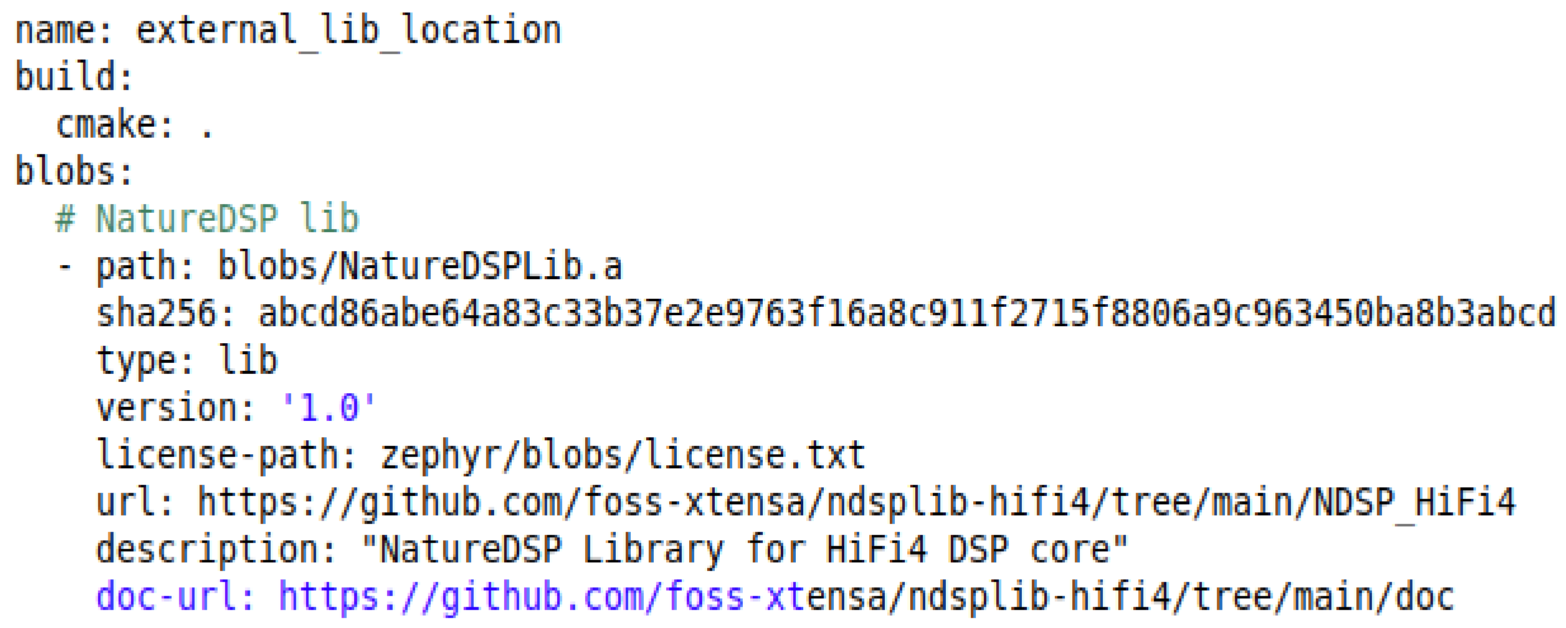
```
/home/zephyrproject/zephyr$ unset LIB_LOCATION
/home/zephyrproject/zephyr$
west build -p always -b imx8mp_evk//adsp samples/application_development/number_crunching/
```

Creating an application – Solution 2 (Zephyr way)

➢ Build the application by specifying -DZEPHYR_EXTRA_MODULES

```
/home/zephyrproject/zephyr$ west build -p always -b imx8mp_evk//adsp
samples/application_development/number_crunching/ -DZEPHYR_EXTRA_MODULES=/home/external_lib_location
```

```
/home/external_lib_location
 ├── blobs
 │    ├── license.txt
 │    └── NatureDSPLib.a
 ├── CMakeLists.txt
 ├── include
 │    ├── NatureDSP_Signal.h
 │    ├── …
 │    └── NatureDSP_types.h
 └── zephyr
      └── module.yml
```



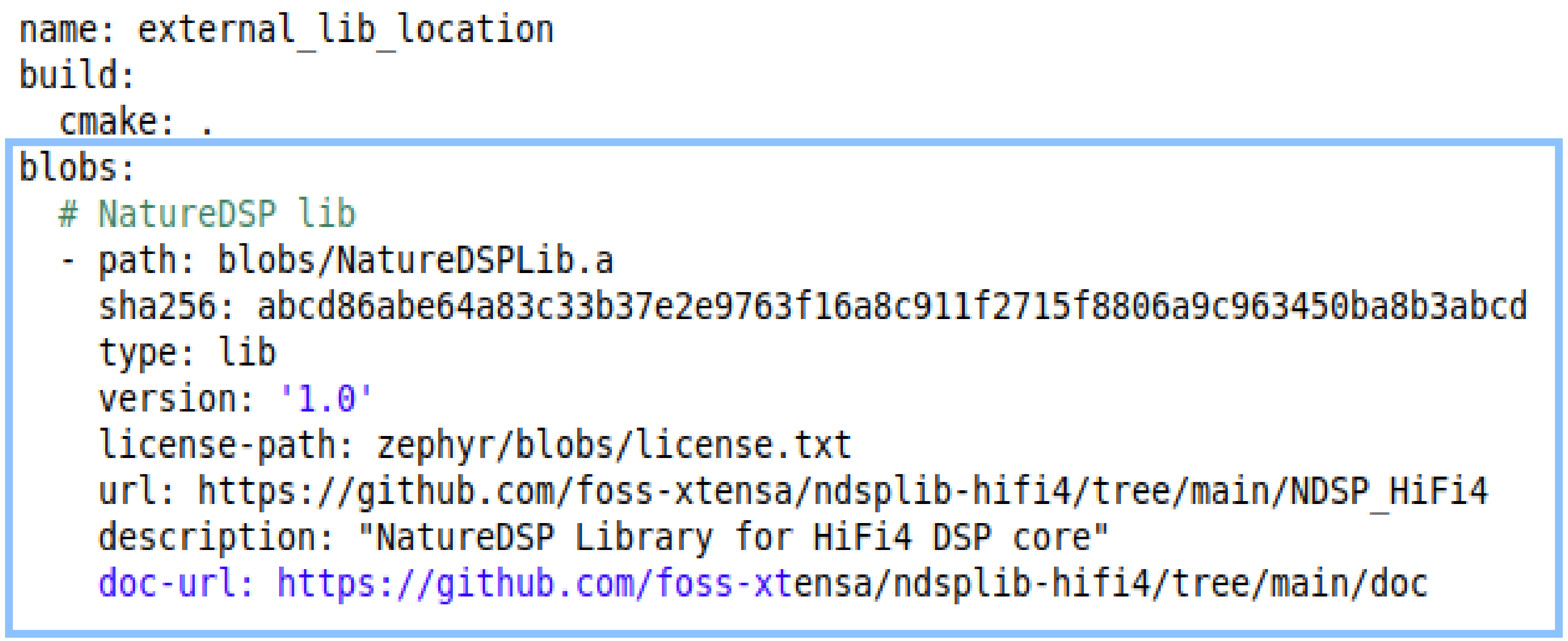
```
name: external_lib_location
build:
  cmake: .
blobs:
  # NatureDSP lib
  - path: blobs/NatureDSPLib.a
    sha256: abcd86abe64a83c33b37e2e9763f16a8c911f2715f8806a9c963450ba8b3abcd
    type: lib
    version: '1.0'
    license-path: zephyr/blobs/license.txt
    url: https://github.com/foss-xtensa/ndsplib-hifi4/tree/main/NDSP_HiFi4
    description: "NatureDSP Library for HiFi4 DSP core"
    doc-url: https://github.com/foss-xtensa/ndsplib-hifi4/tree/main/doc
```

Creating an application – Solution 2

- ➢ Build the application by specifying -DZEPHYR_EXTRA_MODULES
- ➢ Use binary blobs to link against the library

```
/home/zephyrproject/zephyr$ west build -p always -b imx8mp_evk//adsp
samples/application_development/number_crunching/ -DZEPHYR_EXTRA_MODULES=/home/external_lib_location
```

```
/home/external_lib_location
├── blobs
│   ├── license.txt
│   └── NatureDSPLib.a
├── CMakeLists.txt
├── include
│   ├── NatureDSP_Signal.h
│   ├── …
│   └── NatureDSP_types.h
└── zephyr
    └── module.yml
```



```
name: external_lib_location
build:
  cmake: .
blobs:
  # NatureDSP lib
  - path: blobs/NatureDSPLib.a
    sha256: abcd86abe64a83c33b37e2e9763f16a8c911f2715f8806a9c963450ba8b3abcd
    type: lib
    version: '1.0'
    license-path: zephyr/blobs/license.txt
    url: https://github.com/foss-xtensa/ndsplib-hifi4/tree/main/NDSP_HiFi4
    description: "NatureDSP Library for HiFi4 DSP core"
    doc-url: https://github.com/foss-xtensa/ndsplib-hifi4/tree/main/doc
```

Creating an application – Solution 2

- Build the application by specifying -DZEPHYR_EXTRA_MODULES
- Use binary blobs to link against the library

```
/home/zephyrproject/zephyr$ west build -p always -b imx8mp_evk//adsp
samples/application_development/number_crunching/ -DZEPHYR_EXTRA_MODULES=/home/external_lib_location
```

```
/home/external_lib_location
├── blobs
│   ├── license.txt
│   └── NatureDSPLib.a
├── CMakeLists.txt
├── include
│   ├── NatureDSP_Signal.h
│   ├── ...
│   └── NatureDSP_types.h
└── zephyr
    └── module.yml
```

```
# SPDX-License-Identifier: Apache-2.0

cmake_minimum_required(VERSION 3.20.0)

zephyr_include_directories(include)

# Link with the external 3rd party library.
set(LIB_NAME        "NatureDSPLib.a"    CACHE STRING "")

zephyr_link_libraries(${CMAKE_CURRENT_SOURCE_DIR}/blobs/${LIB_NAME})
```

Creating an application – Solution 2

```
zephyr/samples/
└── application_development
    ├── ...
    └── number_crunching
        ├── CMakeLists.txt
        ├── include
        │    ├─ input.h
        │    └─ math_ops.h
        ├── prj.conf
        ├── README.rst
        ├── sample.yaml
        └── src
             ├─ cmsis_dsp_wrapper.c
             ├─ main.c
             ├─ math_ops.c
             └─ nature_dsp_wrapper.c
```

```
# SPDX-License-Identifier: Apache-2.0

cmake_minimum_required(VERSION 3.20.0)

find_package(Zephyr REQUIRED HINTS $ENV{ZEPHYR_BASE})
project(proprietary_lib)

# defines targets and sources
target_sources(app PRIVATE
    src/main.c
    src/math_ops.c
)
zephyr_include_directories(include)

if(DEFINED ZEPHYR_EXTRA_MODULES)
    message(STATUS "We have a ZEPHYR_EXTRA_MODULES defined")
    target_sources(app PRIVATE
        src/nature_dsp_wrapper.c
    )
else()
    message(STATUS "ZEPHYR_EXTRA_MODULES NOT defined")
    target_sources(app PRIVATE
        src/cmsis_dsp_wrapper.c
    )
endif()
```

Out-of-tree application

- Same application is moved out of Zephyr workspace
- Everything applies as before, except build command

```
/home/
├── number_crunching
│   ├── CMakeLists.txt
│   ├── include
│   │   ├── input.h
│   │   └── math_ops.h
│   ├── prj.conf
│   ├── README.rst
│   ├── sample.yaml
│   └── src
│       ├── cmsis_dsp_wrapper.c
│       ├── main.c
│       ├── math_ops.c
│       └── nature_dsp_wrapper.c
└── zephyrproject
    ├── bootloader
    ├── modules
    ├── optional
    ├── tools
    └── zephyr
```

Building the out-of-tree application

From Zephyr workspace:

```
/home/zephyrproject/zephyr$
west build -p always -b imx8mp_evk//adsp
../../number_crunching/
```

```
/home/
├── number_crunching
│   ├── CMakeLists.txt
│   ├── include
│   │   ├── input.h
│   │   └── math_ops.h
│   ├── prj.conf
│   ├── README.rst
│   ├── sample.yaml
│   └── src
│       ├── cmsis_dsp_wrapper.c
│       ├── main.c
│       ├── math_ops.c
│       └── nature_dsp_wrapper.c
└── zephyrproject
    ├── bootloader
    ├── modules
    ├── optional
    ├── tools
    └── zephyr
        └── build
```

Building the out-of-tree application

From outside Zephyr workspace:

```
# set ZEPHYR_BASE
/home/number_crunching$
export ZEPHYR_BASE=/home/zephyrproject/zephyr
# generate build.ninja files
/home/number_crunching$
cmake -Bbuild -GNinja -DBOARD=imx8mp_evk//adsp
# build the application
/home/number_crunching$ ninja -Cbuild
```

```
/home/
 ├── number_crunching
 │    ├── CMakeLists.txt
 │    ├── include
 │    │    ├── input.h
 │    │    └── math_ops.h
 │    ├── prj.conf
 │    ├── README.rst
 │    ├── sample.yaml
 │    └── src
 │         ├── cmsis_dsp_wrapper.c
 │         ├── main.c
 │         ├── math_ops.c
 │         └── nature_dsp_wrapper.c
 └── zephyrproject
      └── zephyr
```

Building the out-of-tree application

From outside Zephyr workspace:

```
# set ZEPHYR_BASE
/home/number_crunching$
export ZEPHYR_BASE=/home/zephyrproject/zephyr
# generate build.ninja files
/home/number_crunching$
cmake -Bbuild -GNinja -DBOARD=imx8mp_evk//adsp
# build the application
/home/number_crunching$ ninja -Cbuild
```

```
/home/
├── number_crunching
│   ├── build
│   ├── CMakeLists.txt
│   ├── include
│   │   ├── input.h
│   │   └── math_ops.h
│   ├── prj.conf
│   ├── README.rst
│   ├── sample.yaml
│   └── src
│       ├── cmsis_dsp_wrapper.c
│       ├── main.c
│       ├── math_ops.c
│       └── nature_dsp_wrapper.c
└── zephyrproject
    └── zephyr
```

Building the out-of-tree application

From outside Zephyr workspace:

```
# set ZEPHYR_BASE
/home/number_crunching$
export ZEPHYR_BASE=/home/zephyrproject/zephyr
# generate build.ninja files
/home/number_crunching$
cmake -Bbuild -GNinja -DBOARD=imx8mp_evk//adsp
# build the application
/home/number_crunching$ ninja -Cbuild
```

```
/home/
├── number_crunching
│    ├── build
│    │    └── zephyr
│    │         └── zephyr.elf
│    ├── CMakeLists.txt
│    ├── include
│    │    ├── input.h
│    │    └── math_ops.h
│    ├── prj.conf
│    ├── README.rst
│    ├── sample.yaml
│    └── src
│         ├── cmsis_dsp_wrapper.c
│         ├── main.c
│         ├── math_ops.c
│         └── nature_dsp_wrapper.c
└── zephyrproject
     └── zephyr
```

Application custom workspace

- Application west manifest

```
/home/
└── number_crunching
    ├── CMakeLists.txt
    ├── include
    │   ├── input.h
    │   └── math_ops.h
    ├── prj.conf
    ├── README.rst
    ├── sample.yaml
    ├── src
    │   ├── cmsis_dsp_wrapper.c
    │   ├── main.c
    │   ├── math_ops.c
    │   └── nature_dsp_wrapper.c
    └── west.yml
```

Application custom workspace

- Application west manifest
  - Select only needed dependencies

```yaml
manifest:
  self:
    west-commands: scripts/west-commands.yml

  remotes:
    - name: zephyrproject-rtos
      url-base: https://github.com/zephyrproject-rtos

  projects:
    - name: zephyr
      remote: zephyrproject-rtos
      revision: main
      import:
        # By using name-allowlist we can clone only the modules that are
        # strictly needed by the application.
        name-allowlist:
          - cmsis-dsp      # required by the application
          - hal_xtensa     # required by the imx8mp_evk//adsp board (Xtensa arch core)
          - hal_nxp        # required by the imx8mp_evk board (NXP board)
```

```
/home/
└── number_crunching
    ├── CMakeLists.txt
    ├── include
    │   ├── input.h
    │   └── math_ops.h
    ├── prj.conf
    ├── README.rst
    ├── sample.yaml
    ├── src
    │   ├── cmsis_dsp_wrapper.c
    │   ├── main.c
    │   ├── math_ops.c
    │   └── nature_dsp_wrapper.c
    └── west.yml
```

Application custom workspace

- Application west manifest
  - Select only needed dependencies

```
# initialize workspace
/home/workspace$ west init -l number_crunching/
# update Zephyr modules
/home/workspace$ west update

# build the application
/home/workspace$ cd zephyr
/home/workspace/zephyr$
west build -p always -b imx8mp_evk//adsp
../number_crunching/
-- west build: generating a build system
Loading Zephyr default modules (Zephyr base).
-- Application: /home/workspace/number_crunching
...
```

```
/home/
├── number_crunching
├── .west
├── zephyr
└── modules
    ├── hal
    │   ├── nxp
    │   └── xtensa
    └── lib
        └── cmsis-dsp
```

Out-of-tree device driver

- Simplest case
  - (Known) base address of memory mapped registers
  - Interrupt service routine
    - IRQ_CONNECT()
    - irq_enable()

```
*** Booting Zephyr OS build v3.6.0-2484-g33ffc2af3cbc ***

Interrupt driver example!


 >IRQ is triggered!
```

```c
#include <zephyr/kernel.h>
#include <zephyr/irq.h>
#include <stdio.h>

#define DSP_IRQ 19
/* Channel n Interrupt Set Register */
uint32_t *CHn_SET4 = (uint32_t *)0x30a80028;
/* Channel n Interrupt Mask Register */
uint32_t *CHn_MASK4 = (uint32_t *)0x30a80014;
/* Channel n Master Interrupt Disable Register */
uint32_t *CHn_MINTDIS = (uint32_t *)0x30a80040;

void function_isr(void)
{
        *CHn_SET4 = 0x0;
        printk("\r\n >IRQ is triggered!\r\n\n");
}

int main(void)
{
        *CHn_MINTDIS = 0x0;
        *CHn_MASK4 = 0x0;
        *CHn_SET4 = 0x0;

        printk("\r\nInterrupt driver example!\r\n\n");

        /* Initialize the interrupt handler */
        IRQ_CONNECT(DSP_IRQ, 0, function_isr, 0, 0);

        /* Enable the interrupt from DSP_IRQ source */
        irq_enable(DSP_IRQ);

        /* Enable the interrupts */
        *CHn_MINTDIS = 0x0;
        *CHn_MASK4 = 0x1;
        *CHn_SET4 = 0x1;

        return 0;
}
```

Out-of-tree device driver

- Simplest case
  - (Known) base address of memory mapped registers
  - Interrupt service routine
    - IRQ_CONNECT()
    - irq_enable()

- Can be enhanced to be as Zephyr device drivers:
  - use `dts/bindings` for base address, IRQ number
  - interface with interrupt management subsystem
  - integrate with the build infrastructure

```c
#include <zephyr/kernel.h>
#include <zephyr/irq.h>
#include <stdio.h>

#define DSP_IRQ 19
/* Channel n Interrupt Set Register */
uint32_t *CHn_SET4 = (uint32_t *)0x30a80028;
/* Channel n Interrupt Mask Register */
uint32_t *CHn_MASK4 = (uint32_t *)0x30a80014;
/* Channel n Master Interrupt Disable Register */
uint32_t *CHn_MINTDIS = (uint32_t *)0x30a80040;

void function_isr(void)
{
        *CHn_SET4 = 0x0;
        printk("\r\n >IRQ is triggered!\r\n\n");
}

int main(void)
{
        *CHn_MINTDIS = 0x0;
        *CHn_MASK4 = 0x0;
        *CHn_SET4 = 0x0;

        printk("\r\nInterrupt driver example!\r\n\n");

        /* Initialize the interrupt handler */
        IRQ_CONNECT(DSP_IRQ, 0, function_isr, 0, 0);

        /* Enable the interrupt from DSP_IRQ source */
        irq_enable(DSP_IRQ);

        /* Enable the interrupts */
        *CHn_MINTDIS = 0x0;
        *CHn_MASK4 = 0x1;
        *CHn_SET4 = 0x1;

        return 0;
}
```

Toolchains

- Zephyr SDK contains toolchains for each of Zephyr’s supported architectures:
  - ARC (32-bit and 64-bit; ARCv1, ARCv2, ARCv3)
  - ARM (32-bit and 64-bit; ARMv6, ARMv7, ARMv8; A/R/M Profiles)
  - MIPS (32-bit and 64-bit)
  - Nios II
  - RISC-V (32-bit and 64-bit; RV32I, RV32E, RV64I)
  - x86 (32-bit and 64-bit)
  - Xtensa
- Zephyr SDK usage is highly recommended
- Required under certain conditions (e.g., running tests in QEMU for some architectures)

Custom toolchains

- Set environment variables:
  - `ZEPHYR_TOOLCHAIN_VARIANT`
    - toolchain name
  - `TOOLCHAIN_ROOT`
    - the path to the directory containing toolchain’s CMake configuration file

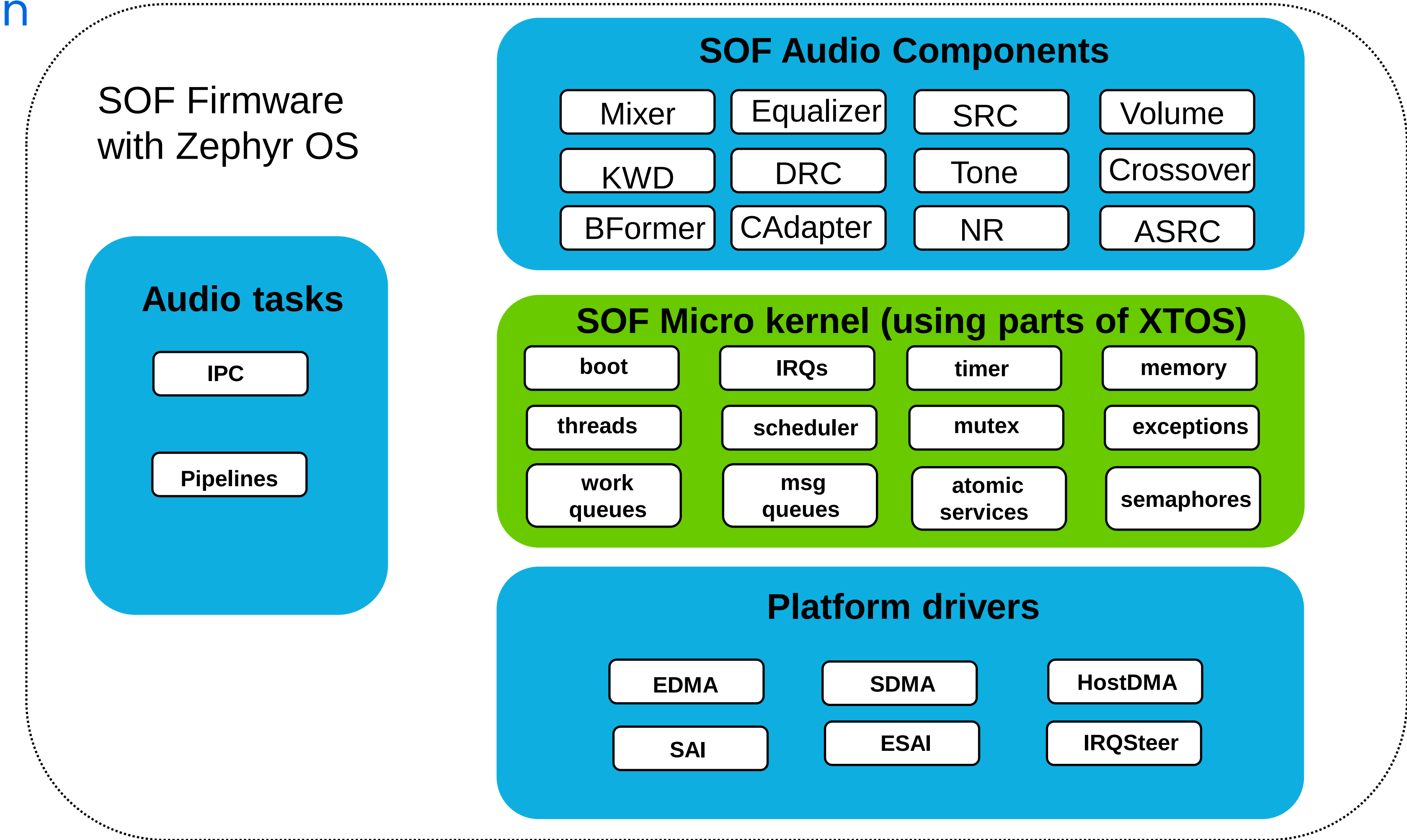
- Or pass them directly when building the application:

```
$ west build ... -- -DZEPHYR_TOOLCHAIN_VARIANT=... -DTOOLCHAIN_ROOT=...

$ cmake -DZEPHYR_TOOLCHAIN_VARIANT=... -DTOOLCHAIN_ROOT=...
```

Case study: Sound Open Firmware Zephyr integration

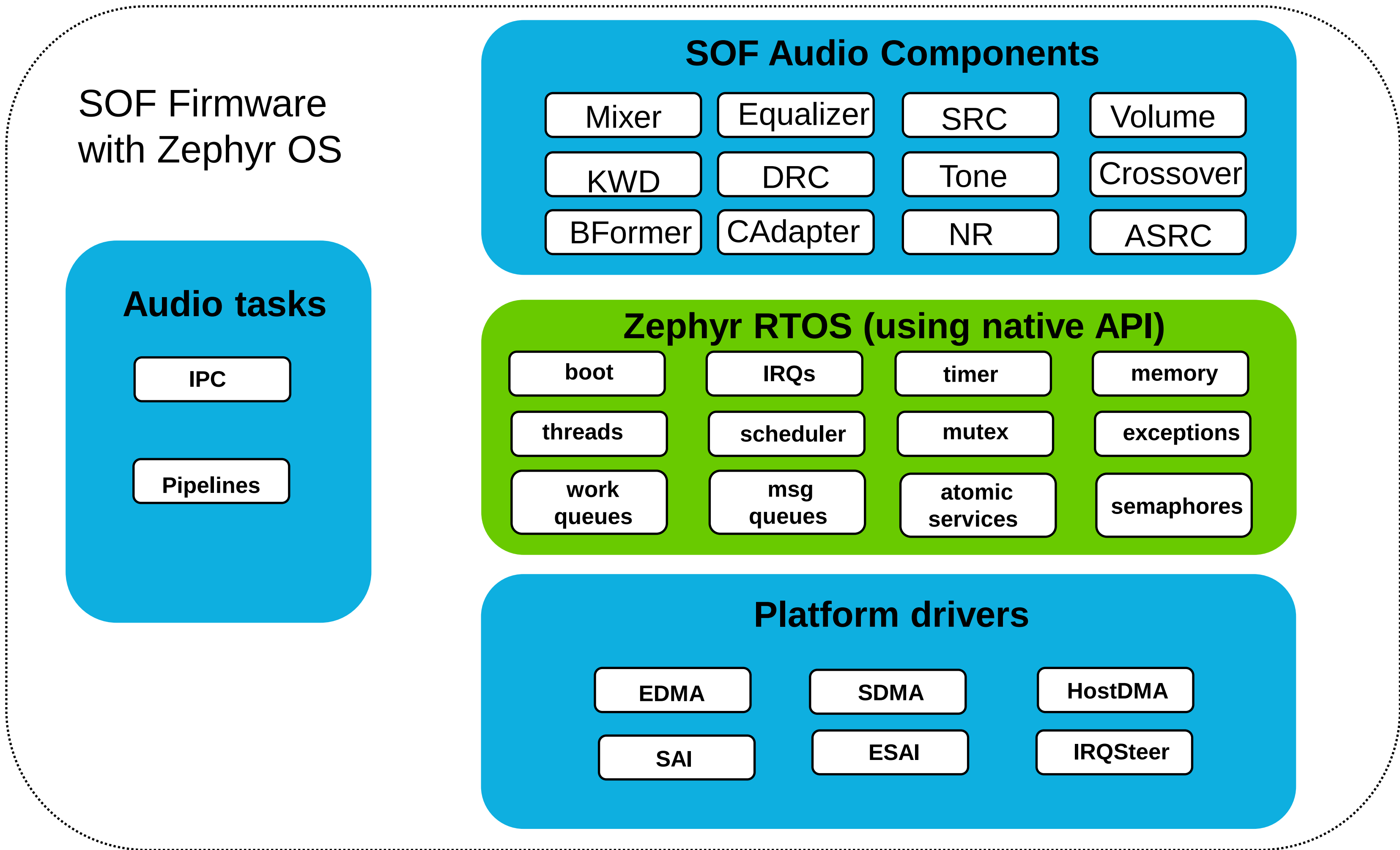
- Open-source audio DSP firmware and SDK
- BSD-3-Clause license firmware and BSD/GPL licensed drivers
- Stage 0: no Zephyr integration



https://github.com/thesofproject/sof

Case study: Sound Open Firmware Zephyr integration

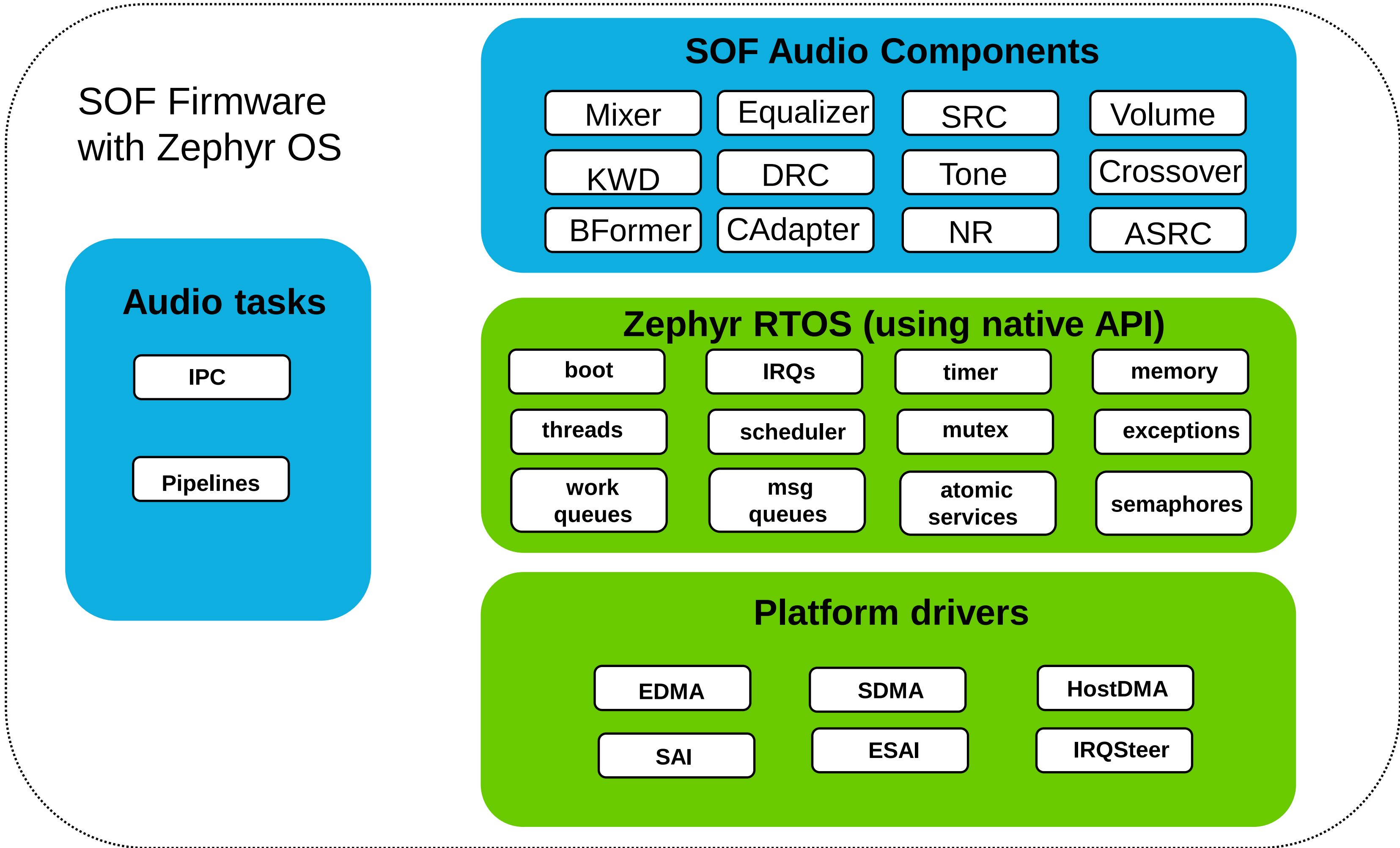
- Added board support for HiFi4 DSP Core in Zephyr (Xtensa arch present)
- Stage 1: out-of-tree application, no Zephyr drivers, Kernel API only



https://github.com/thesofproject/sof

Case study: Sound Open Firmware Zephyr integration

- Port Platform drivers from SOF to Zephyr – ongoing
- Stage 2: optional Zephyr module, Kernel and drivers API



https://github.com/thesofproject/sof

Conclusions

- Zephyr building framework is very powerful
- There are multiple ways to do the same thing
  - Except arch support
- There are extensive documentation and examples
- Start simple with an existing application and Zephyr SDK
- Pay attention to license

Thank you!

Questions?

Get in touch

Iuliana Prodan
iuliana.prodan@nxp.com

nxp.com

nxp.com