```
# cmake_minimum_require must be the first command of the file
cmake_minimum_required(VERSION 3.5.0)

project(Eigen3)

# guard against in-source builds

if(${CMAKE_SOURCE_DIR} STREQUAL ${CMAKE_BINARY_DIR})
  message(FATAL_ERROR "In-source builds not allowed. Please make a new directory
(called a build directory) and run CMake from there. You may need to remove
CMakeCache.txt. ")
endif()


# Alias Eigen_*_DIR to Eigen3_*_DIR:

set(Eigen_SOURCE_DIR ${Eigen3_SOURCE_DIR})
set(Eigen_BINARY_DIR ${Eigen3_BINARY_DIR})

# guard against bad build-type strings

if (NOT CMAKE_BUILD_TYPE)
  set(CMAKE_BUILD_TYPE "Release")
endif()


#############################################################################
# retrieve version information                                               #
#############################################################################

# automatically parse the version number
file(READ "${PROJECT_SOURCE_DIR}/Eigen/src/Core/util/Macros.h"
_eigen_version_header)
string(REGEX MATCH "define[ \t]+EIGEN_WORLD_VERSION[ \t]+([0-9]+)"
_eigen_world_version_match "${_eigen_version_header}")
set(EIGEN_WORLD_VERSION "${CMAKE_MATCH_1}")
string(REGEX MATCH "define[ \t]+EIGEN_MAJOR_VERSION[ \t]+([0-9]+)"
_eigen_major_version_match "${_eigen_version_header}")
set(EIGEN_MAJOR_VERSION "${CMAKE_MATCH_1}")
string(REGEX MATCH "define[ \t]+EIGEN_MINOR_VERSION[ \t]+([0-9]+)"
_eigen_minor_version_match "${_eigen_version_header}")
set(EIGEN_MINOR_VERSION "${CMAKE_MATCH_1}")
set(EIGEN_VERSION_NUMBER ${EIGEN_WORLD_VERSION}.${EIGEN_MAJOR_VERSION}.$
{EIGEN_MINOR_VERSION})

# if we are not in a git clone
if(IS_DIRECTORY ${CMAKE_SOURCE_DIR}/.git)
  # if the git program is absent or this will leave the EIGEN_GIT_REVNUM string
empty,
  # but won't stop CMake.
  execute_process(COMMAND git ls-remote --refs -q ${CMAKE_SOURCE_DIR} HEAD
OUTPUT_VARIABLE EIGEN_GIT_OUTPUT)
endif()

# extract the git rev number from the git output...
if(EIGEN_GIT_OUTPUT)
string(REGEX MATCH "^([0-9;a-f]+).*" EIGEN_GIT_CHANGESET_MATCH "$
{EIGEN_GIT_OUTPUT}")
set(EIGEN_GIT_REVNUM "${CMAKE_MATCH_1}")
```

```
endif()
#...and show it next to the version number
if(EIGEN_GIT_REVNUM)
  set(EIGEN_VERSION "${EIGEN_VERSION_NUMBER} (git rev ${EIGEN_GIT_REVNUM})")
else()
  set(EIGEN_VERSION "${EIGEN_VERSION_NUMBER}")
endif()

include(CheckCXXCompilerFlag)
include(GNUInstallDirs)
include(CMakeDependentOption)

set(CMAKE_MODULE_PATH ${PROJECT_SOURCE_DIR}/cmake)


option(EIGEN_TEST_CXX11 "Enable testing with C++11 and C++11 features (e.g. Tensor
module)." OFF)


macro(ei_add_cxx_compiler_flag FLAG)
  string(REGEX REPLACE "-" "" SFLAG1 ${FLAG})
  string(REGEX REPLACE "\\+" "p" SFLAG ${SFLAG1})
  check_cxx_compiler_flag(${FLAG} COMPILER_SUPPORT_${SFLAG})
  if(COMPILER_SUPPORT_${SFLAG})
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${FLAG}")
  endif()
endmacro()

check_cxx_compiler_flag("-std=c++11" EIGEN_COMPILER_SUPPORT_CPP11)

if(EIGEN_TEST_CXX11)
  set(CMAKE_CXX_STANDARD 11)
  set(CMAKE_CXX_EXTENSIONS OFF)
  if(EIGEN_COMPILER_SUPPORT_CPP11)
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11")
  endif()
else()
  #set(CMAKE_CXX_STANDARD 03)
  #set(CMAKE_CXX_EXTENSIONS OFF)
  ei_add_cxx_compiler_flag("-std=c++03")
endif()

# Determine if we should build shared libraries on this platform.
get_cmake_property(EIGEN_BUILD_SHARED_LIBS TARGET_SUPPORTS_SHARED_LIBS)

#############################################################################
# find how to link to the standard libraries                                #
#############################################################################

find_package(StandardMathLibrary)


set(EIGEN_TEST_CUSTOM_LINKER_FLAGS  "" CACHE STRING "Additional linker flags when
linking unit tests.")
set(EIGEN_TEST_CUSTOM_CXX_FLAGS     "" CACHE STRING "Additional compiler flags when
compiling unit tests.")

set(EIGEN_STANDARD_LIBRARIES_TO_LINK_TO "")
```

```
if(NOT STANDARD_MATH_LIBRARY_FOUND)

  message(FATAL_ERROR
    "Can't link to the standard math library. Please report to the Eigen
developers, telling them about your platform.")

else()

  if(EIGEN_STANDARD_LIBRARIES_TO_LINK_TO)
    set(EIGEN_STANDARD_LIBRARIES_TO_LINK_TO "${EIGEN_STANDARD_LIBRARIES_TO_LINK_TO}
${STANDARD_MATH_LIBRARY}")
  else()
    set(EIGEN_STANDARD_LIBRARIES_TO_LINK_TO "${STANDARD_MATH_LIBRARY}")
  endif()

endif()

if(EIGEN_STANDARD_LIBRARIES_TO_LINK_TO)
  message(STATUS "Standard libraries to link to explicitly: $
{EIGEN_STANDARD_LIBRARIES_TO_LINK_TO}")
else()
  message(STATUS "Standard libraries to link to explicitly: none")
endif()

option(EIGEN_BUILD_BTL "Build benchmark suite" OFF)

# Disable pkgconfig only for native Windows builds
if(NOT WIN32 OR NOT CMAKE_HOST_SYSTEM_NAME MATCHES Windows)
  option(EIGEN_BUILD_PKGCONFIG "Build pkg-config .pc file for Eigen" ON)
endif()

set(CMAKE_INCLUDE_CURRENT_DIR OFF)

option(EIGEN_SPLIT_LARGE_TESTS "Split large tests into smaller executables" ON)

option(EIGEN_DEFAULT_TO_ROW_MAJOR "Use row-major as default matrix storage order"
OFF)
if(EIGEN_DEFAULT_TO_ROW_MAJOR)
  add_definitions("-DEIGEN_DEFAULT_TO_ROW_MAJOR")
endif()

set(EIGEN_TEST_MAX_SIZE "320" CACHE STRING "Maximal matrix/vector size, default is
320")

if(NOT MSVC)
  # We assume that other compilers are partly compatible with GNUCC

  # clang outputs some warnings for unknown flags that are not caught by
check_cxx_compiler_flag
  # adding -Werror turns such warnings into errors
  check_cxx_compiler_flag("-Werror" COMPILER_SUPPORT_WERROR)
  if(COMPILER_SUPPORT_WERROR)
    set(CMAKE_REQUIRED_FLAGS "-Werror")
  endif()
  ei_add_cxx_compiler_flag("-pedantic")
  ei_add_cxx_compiler_flag("-Wall")
  ei_add_cxx_compiler_flag("-Wextra")
  #ei_add_cxx_compiler_flag("-Weverything")              # clang
  ei_add_cxx_compiler_flag("-Wundef")
  ei_add_cxx_compiler_flag("-Wcast-align")
  ei_add_cxx_compiler_flag("-Wchar-subscripts")
  ei_add_cxx_compiler_flag("-Wnon-virtual-dtor")
  ei_add_cxx_compiler_flag("-Wunused-local-typedefs")
  ei_add_cxx_compiler_flag("-Wpointer-arith")
  ei_add_cxx_compiler_flag("-Wwrite-strings")
  ei_add_cxx_compiler_flag("-Wformat-security")
  ei_add_cxx_compiler_flag("-Wshorten-64-to-32")
  ei_add_cxx_compiler_flag("-Wlogical-op")
  ei_add_cxx_compiler_flag("-Wenum-conversion")
  ei_add_cxx_compiler_flag("-Wc++11-extensions")
  ei_add_cxx_compiler_flag("-Wdouble-promotion")
#  ei_add_cxx_compiler_flag("-Wconversion")

  ei_add_cxx_compiler_flag("-Wshadow")

  ei_add_cxx_compiler_flag("-Wno-psabi")
  ei_add_cxx_compiler_flag("-Wno-variadic-macros")
  ei_add_cxx_compiler_flag("-Wno-long-long")

  ei_add_cxx_compiler_flag("-fno-check-new")
  ei_add_cxx_compiler_flag("-fno-common")
  ei_add_cxx_compiler_flag("-fstrict-aliasing")
  ei_add_cxx_compiler_flag("-wd981")                    # disable ICC's "operands
are evaluated in unspecified order" remark
  ei_add_cxx_compiler_flag("-wd2304")                   # disable ICC's "warning
#2304: non-explicit constructor with single argument may cause implicit type
conversion" produced by -Wnon-virtual-dtor


  # The -ansi flag must be added last, otherwise it is also used as a linker flag
by check_cxx_compiler_flag making it fails
  # Moreover we should not set both -strict-ansi and -ansi
  check_cxx_compiler_flag("-strict-ansi" COMPILER_SUPPORT_STRICTANSI)
  ei_add_cxx_compiler_flag("-Qunused-arguments")        # disable clang warning:
argument unused during compilation: '-ansi'

  if(COMPILER_SUPPORT_STRICTANSI)
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -strict-ansi")
  else()
    ei_add_cxx_compiler_flag("-ansi")
  endif()

  if(ANDROID_NDK)
    ei_add_cxx_compiler_flag("-pie")
    ei_add_cxx_compiler_flag("-fPIE")
  endif()

  set(CMAKE_REQUIRED_FLAGS "")

  option(EIGEN_TEST_SSE2 "Enable/Disable SSE2 in tests/examples" OFF)
  if(EIGEN_TEST_SSE2)
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -msse2")
    message(STATUS "Enabling SSE2 in tests/examples")
  endif()

  option(EIGEN_TEST_SSE3 "Enable/Disable SSE3 in tests/examples" OFF)
  if(EIGEN_TEST_SSE3)
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -msse3")
    message(STATUS "Enabling SSE3 in tests/examples")
  endif()

  option(EIGEN_TEST_SSSE3 "Enable/Disable SSSE3 in tests/examples" OFF)
  if(EIGEN_TEST_SSSE3)
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -mssse3")
    message(STATUS "Enabling SSSE3 in tests/examples")
  endif()

  option(EIGEN_TEST_SSE4_1 "Enable/Disable SSE4.1 in tests/examples" OFF)
  if(EIGEN_TEST_SSE4_1)
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -msse4.1")
    message(STATUS "Enabling SSE4.1 in tests/examples")
  endif()

  option(EIGEN_TEST_SSE4_2 "Enable/Disable SSE4.2 in tests/examples" OFF)
  if(EIGEN_TEST_SSE4_2)
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -msse4.2")
    message(STATUS "Enabling SSE4.2 in tests/examples")
  endif()

  option(EIGEN_TEST_AVX "Enable/Disable AVX in tests/examples" OFF)
  if(EIGEN_TEST_AVX)
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -mavx")
    message(STATUS "Enabling AVX in tests/examples")
  endif()

  option(EIGEN_TEST_FMA "Enable/Disable FMA in tests/examples" OFF)
  if(EIGEN_TEST_FMA AND NOT EIGEN_TEST_NEON)
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -mfma")
    message(STATUS "Enabling FMA in tests/examples")
  endif()

  option(EIGEN_TEST_AVX2 "Enable/Disable AVX2 in tests/examples" OFF)
  if(EIGEN_TEST_AVX2)
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -mavx2 -mfma")
    message(STATUS "Enabling AVX2 in tests/examples")
  endif()

  option(EIGEN_TEST_AVX512 "Enable/Disable AVX512 in tests/examples" OFF)
  if(EIGEN_TEST_AVX512)
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -mavx512f -mfma")
    if (NOT "${CMAKE_CXX_COMPILER_ID}" STREQUAL "Clang")
      set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -fabi-version=6")
    endif()
    message(STATUS "Enabling AVX512 in tests/examples")
  endif()

  option(EIGEN_TEST_AVX512DQ "Enable/Disable AVX512DQ in tests/examples" OFF)
  if(EIGEN_TEST_AVX512DQ)
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -mavx512dq")
    if (NOT "${CMAKE_CXX_COMPILER_ID}" STREQUAL "Clang")
      set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -fabi-version=6")
    endif()
    message(STATUS "Enabling AVX512DQ in tests/examples")
  endif()

  option(EIGEN_TEST_F16C "Enable/Disable F16C in tests/examples" OFF)
```

```
  if(EIGEN_TEST_F16C)
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -mf16c")
    message(STATUS "Enabling F16C in tests/examples")
  endif()

  option(EIGEN_TEST_ALTIVEC "Enable/Disable AltiVec in tests/examples" OFF)
  if(EIGEN_TEST_ALTIVEC)
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -maltivec -mabi=altivec")
    message(STATUS "Enabling AltiVec in tests/examples")
  endif()

  option(EIGEN_TEST_VSX "Enable/Disable VSX in tests/examples" OFF)
  if(EIGEN_TEST_VSX)
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -m64 -mvsx")
    message(STATUS "Enabling VSX in tests/examples")
  endif()

  option(EIGEN_TEST_MSA "Enable/Disable MSA in tests/examples" OFF)
  if(EIGEN_TEST_MSA)
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -mmsa")
    message(STATUS "Enabling MSA in tests/examples")
  endif()

  option(EIGEN_TEST_NEON "Enable/Disable Neon in tests/examples" OFF)
  if(EIGEN_TEST_NEON)
    if(EIGEN_TEST_FMA)
      set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -mfpu=neon-vfpv4")
    else()
      set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -mfpu=neon")
    endif()
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -mfloat-abi=hard")
    message(STATUS "Enabling NEON in tests/examples")
  endif()

  option(EIGEN_TEST_NEON64 "Enable/Disable Neon in tests/examples" OFF)
  if(EIGEN_TEST_NEON64)
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS}")
    message(STATUS "Enabling NEON in tests/examples")
  endif()

  option(EIGEN_TEST_Z13 "Enable/Disable S390X(zEC13) ZVECTOR in tests/examples"
OFF)
  if(EIGEN_TEST_Z13)
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -march=z13 -mzvector")
    message(STATUS "Enabling S390X(zEC13) ZVECTOR in tests/examples")
  endif()

  option(EIGEN_TEST_Z14 "Enable/Disable S390X(zEC14) ZVECTOR in tests/examples"
OFF)
  if(EIGEN_TEST_Z14)
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -march=z14 -mzvector")
    message(STATUS "Enabling S390X(zEC13) ZVECTOR in tests/examples")
  endif()

  check_cxx_compiler_flag("-fopenmp" COMPILER_SUPPORT_OPENMP)
  if(COMPILER_SUPPORT_OPENMP)
    option(EIGEN_TEST_OPENMP "Enable/Disable OpenMP in tests/examples" OFF)
    if(EIGEN_TEST_OPENMP)
      set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -fopenmp")
      message(STATUS "Enabling OpenMP in tests/examples")
    endif()
  endif()

else()

  # C4127 - conditional expression is constant
  # C4714 - marked as __forceinline not inlined (I failed to deactivate it
selectively)
  #         We can disable this warning in the unit tests since it is clear that it
occurs
  #         because we are oftentimes returning objects that have a destructor or
may
  #         throw exceptions - in particular in the unit tests we are throwing
extra many
  #         exceptions to cover indexing errors.
  # C4505 - unreferenced local function has been removed (impossible to deactivate
selectively)
  set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} /EHsc /wd4127 /wd4505 /wd4714")

  # replace all /Wx by /W4
  string(REGEX REPLACE "/W[0-9]" "/W4" CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS}")

  check_cxx_compiler_flag("/openmp" COMPILER_SUPPORT_OPENMP)
  if(COMPILER_SUPPORT_OPENMP)
    option(EIGEN_TEST_OPENMP "Enable/Disable OpenMP in tests/examples" OFF)
    if(EIGEN_TEST_OPENMP)
      set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} /openmp")
      message(STATUS "Enabling OpenMP in tests/examples")
    endif()
  endif()

  option(EIGEN_TEST_SSE2 "Enable/Disable SSE2 in tests/examples" OFF)
  if(EIGEN_TEST_SSE2)
    if(NOT CMAKE_CL_64)
      # arch is not supported on 64 bit systems, SSE is enabled automatically.
      set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} /arch:SSE2")
    endif()
    message(STATUS "Enabling SSE2 in tests/examples")
  endif()

  option(EIGEN_TEST_AVX "Enable/Disable AVX in tests/examples" OFF)
  if(EIGEN_TEST_AVX)
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} /arch:AVX")
    message(STATUS "Enabling AVX in tests/examples")
  endif()

  option(EIGEN_TEST_FMA "Enable/Disable FMA/AVX2 in tests/examples" OFF)
  if(EIGEN_TEST_FMA AND NOT EIGEN_TEST_NEON)
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} /arch:AVX2")
    message(STATUS "Enabling FMA/AVX2 in tests/examples")
  endif()

endif()

option(EIGEN_TEST_NO_EXPLICIT_VECTORIZATION "Disable explicit vectorization in
tests/examples" OFF)
option(EIGEN_TEST_X87 "Force using X87 instructions. Implies no vectorization."
OFF)
```

```
option(EIGEN_TEST_32BIT "Force generating 32bit code." OFF)

if(EIGEN_TEST_X87)
  set(EIGEN_TEST_NO_EXPLICIT_VECTORIZATION ON)
  if(CMAKE_COMPILER_IS_GNUCXX)
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -mfpmath=387")
    message(STATUS "Forcing use of x87 instructions in tests/examples")
  else()
    message(STATUS "EIGEN_TEST_X87 ignored on your compiler")
  endif()
endif()

if(EIGEN_TEST_32BIT)
  if(CMAKE_COMPILER_IS_GNUCXX)
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -m32")
    message(STATUS "Forcing generation of 32-bit code in tests/examples")
  else()
    message(STATUS "EIGEN_TEST_32BIT ignored on your compiler")
  endif()
endif()

if(EIGEN_TEST_NO_EXPLICIT_VECTORIZATION)
  add_definitions(-DEIGEN_DONT_VECTORIZE=1)
  message(STATUS "Disabling vectorization in tests/examples")
endif()

option(EIGEN_TEST_NO_EXPLICIT_ALIGNMENT "Disable explicit alignment (hence 
vectorization) in tests/examples" OFF)
if(EIGEN_TEST_NO_EXPLICIT_ALIGNMENT)
  add_definitions(-DEIGEN_DONT_ALIGN=1)
  message(STATUS "Disabling alignment in tests/examples")
endif()

option(EIGEN_TEST_NO_EXCEPTIONS "Disables C++ exceptions" OFF)
if(EIGEN_TEST_NO_EXCEPTIONS)
  ei_add_cxx_compiler_flag("-fno-exceptions")
  message(STATUS "Disabling exceptions in tests/examples")
endif()

set(EIGEN_CUDA_COMPUTE_ARCH 30 CACHE STRING "The CUDA compute architecture level to 
target when compiling CUDA code")

include_directories(${CMAKE_CURRENT_SOURCE_DIR})

# Backward compatibility support for EIGEN_INCLUDE_INSTALL_DIR
if(EIGEN_INCLUDE_INSTALL_DIR)
  message(WARNING "EIGEN_INCLUDE_INSTALL_DIR is deprecated. Use INCLUDE_INSTALL_DIR 
instead.")
endif()

if(EIGEN_INCLUDE_INSTALL_DIR AND NOT INCLUDE_INSTALL_DIR)
  set(INCLUDE_INSTALL_DIR ${EIGEN_INCLUDE_INSTALL_DIR}
      CACHE PATH "The directory relative to CMAKE_INSTALL_PREFIX where Eigen header 
files are installed")
else()
  set(INCLUDE_INSTALL_DIR
      "${CMAKE_INSTALL_INCLUDEDIR}/eigen3"
      CACHE PATH "The directory relative to CMAKE_INSTALL_PREFIX where Eigen header 
files are installed"
```

```
      )
endif()
set(CMAKEPACKAGE_INSTALL_DIR
    "${CMAKE_INSTALL_DATADIR}/eigen3/cmake"
    CACHE PATH "The directory relative to CMAKE_INSTALL_PREFIX where
Eigen3Config.cmake is installed"
    )
set(PKGCONFIG_INSTALL_DIR
    "${CMAKE_INSTALL_DATADIR}/pkgconfig"
    CACHE PATH "The directory relative to CMAKE_INSTALL_PREFIX where eigen3.pc is
installed"
    )

foreach(var INCLUDE_INSTALL_DIR CMAKEPACKAGE_INSTALL_DIR PKGCONFIG_INSTALL_DIR)
  # If an absolute path is specified, make it relative to "{CMAKE_INSTALL_PREFIX}".
  if(IS_ABSOLUTE "${${var}}")
    file(RELATIVE_PATH "${var}" "${CMAKE_INSTALL_PREFIX}" "${${var}}")
  endif()
endforeach()

# similar to set_target_properties but append the property instead of overwriting
it
macro(ei_add_target_property target prop value)

  get_target_property(previous ${target} ${prop})
  # if the property wasn't previously set, ${previous} is now "previous-NOTFOUND"
which cmake allows catching with plain if()
  if(NOT previous)
    set(previous "")
  endif()
  set_target_properties(${target} PROPERTIES ${prop} "${previous} ${value}")
endmacro()

install(FILES
  signature_of_eigen3_matrix_library
  DESTINATION ${INCLUDE_INSTALL_DIR} COMPONENT Devel
  )

if(EIGEN_BUILD_PKGCONFIG)
    configure_file(eigen3.pc.in eigen3.pc @ONLY)
    install(FILES ${CMAKE_CURRENT_BINARY_DIR}/eigen3.pc
        DESTINATION ${PKGCONFIG_INSTALL_DIR}
        )
endif()

install(DIRECTORY Eigen DESTINATION ${INCLUDE_INSTALL_DIR} COMPONENT Devel)


option(EIGEN_BUILD_DOC "Enable creation of Eigen documentation" ON)
if(EIGEN_BUILD_DOC)
  add_subdirectory(doc EXCLUDE_FROM_ALL)
endif()


option(BUILD_TESTING "Enable creation of Eigen tests." ON)
if(BUILD_TESTING)
  include(EigenConfigureTesting)

  if(EIGEN_LEAVE_TEST_IN_ALL_TARGET)
```

```
    add_subdirectory(test) # can't do EXCLUDE_FROM_ALL here, breaks CTest
  else()
    add_subdirectory(test EXCLUDE_FROM_ALL)
  endif()

  add_subdirectory(failtest)
endif()

if(EIGEN_LEAVE_TEST_IN_ALL_TARGET)
  add_subdirectory(blas)
  add_subdirectory(lapack)
else()
  add_subdirectory(blas EXCLUDE_FROM_ALL)
  add_subdirectory(lapack EXCLUDE_FROM_ALL)
endif()

# add SYCL
option(EIGEN_TEST_SYCL "Add Sycl support." OFF)
option(EIGEN_SYCL_TRISYCL "Use the triSYCL Sycl implementation (ComputeCPP by
default)." OFF)
if(EIGEN_TEST_SYCL)
  set (CMAKE_MODULE_PATH "${CMAKE_ROOT}/Modules" "cmake/Modules/" "$
{CMAKE_MODULE_PATH}")
  find_package(Threads REQUIRED)
  if(EIGEN_SYCL_TRISYCL)
    message(STATUS "Using triSYCL")
    include(FindTriSYCL)
  else()
    message(STATUS "Using ComputeCPP SYCL")
    include(FindComputeCpp)
    set(COMPUTECPP_DRIVER_DEFAULT_VALUE OFF)
    if (NOT MSVC)
      set(COMPUTECPP_DRIVER_DEFAULT_VALUE ON)
    endif()
    option(COMPUTECPP_USE_COMPILER_DRIVER
      "Use ComputeCpp driver instead of a 2 steps compilation"
      ${COMPUTECPP_DRIVER_DEFAULT_VALUE}
    )
  endif(EIGEN_SYCL_TRISYCL)
  option(EIGEN_DONT_VECTORIZE_SYCL "Don't use vectorisation in the SYCL tests."
OFF)
  if(EIGEN_DONT_VECTORIZE_SYCL)
    message(STATUS "Disabling SYCL vectorization in tests/examples")
    # When disabling SYCL vectorization, also disable Eigen default vectorization
    add_definitions(-DEIGEN_DONT_VECTORIZE=1)
    add_definitions(-DEIGEN_DONT_VECTORIZE_SYCL=1)
  endif()
endif()

add_subdirectory(unsupported)

add_subdirectory(demos EXCLUDE_FROM_ALL)

# must be after test and unsupported, for configuring buildtests.in
add_subdirectory(scripts EXCLUDE_FROM_ALL)

# TODO: consider also replacing EIGEN_BUILD_BTL by a custom target "make btl"?
if(EIGEN_BUILD_BTL)
  add_subdirectory(bench/btl EXCLUDE_FROM_ALL)
```

```
endif()

if(NOT WIN32)
  add_subdirectory(bench/spbench EXCLUDE_FROM_ALL)
endif()

configure_file(scripts/cdashtesting.cmake.in cdashtesting.cmake @ONLY)

if(BUILD_TESTING)
  ei_testing_print_summary()
endif()

message(STATUS "")
message(STATUS "Configured Eigen ${EIGEN_VERSION_NUMBER}")
message(STATUS "")

string(TOLOWER "${CMAKE_GENERATOR}" cmake_generator_tolower)
if(cmake_generator_tolower MATCHES "makefile")
  message(STATUS "Available targets (use: make TARGET):")
else()
  message(STATUS "Available targets (use: cmake --build . --target TARGET):")
endif()
message(STATUS "---------
+--------------------------------------------------------------")
message(STATUS "Target   |   Description")
message(STATUS "---------
+--------------------------------------------------------------")
message(STATUS "install  | Install Eigen. Headers will be installed to:")
message(STATUS "         |     <CMAKE_INSTALL_PREFIX>/<INCLUDE_INSTALL_DIR>")
message(STATUS "         |   Using the following values:")
message(STATUS "         |     CMAKE_INSTALL_PREFIX: ${CMAKE_INSTALL_PREFIX}")
message(STATUS "         |     INCLUDE_INSTALL_DIR:  ${INCLUDE_INSTALL_DIR}")
message(STATUS "         |   Change the install location of Eigen headers using:")
message(STATUS "         |     cmake . -DCMAKE_INSTALL_PREFIX=yourprefix")
message(STATUS "         |   Or:")
message(STATUS "         |     cmake . -DINCLUDE_INSTALL_DIR=yourdir")
message(STATUS "doc      | Generate the API documentation, requires Doxygen &
LaTeX")
if(BUILD_TESTING)
  message(STATUS "check    | Build and run the unit-tests. Read this page:")
  message(STATUS "         |   http://eigen.tuxfamily.org/index.php?title=Tests")
endif()
message(STATUS "blas     | Build BLAS library (not the same thing as Eigen)")
message(STATUS "uninstall| Remove files installed by the install target")
message(STATUS "---------
+--------------------------------------------------------------")
message(STATUS "")


set ( EIGEN_VERSION_STRING ${EIGEN_VERSION_NUMBER} )
set ( EIGEN_VERSION_MAJOR  ${EIGEN_WORLD_VERSION} )
set ( EIGEN_VERSION_MINOR  ${EIGEN_MAJOR_VERSION} )
set ( EIGEN_VERSION_PATCH  ${EIGEN_MINOR_VERSION} )
set ( EIGEN_DEFINITIONS "")
set ( EIGEN_INCLUDE_DIR "${CMAKE_INSTALL_PREFIX}/${INCLUDE_INSTALL_DIR}" )
set ( EIGEN_ROOT_DIR ${CMAKE_INSTALL_PREFIX} )

include (CMakePackageConfigHelpers)
```

```
# Imported target support
add_library (eigen INTERFACE)
add_library (Eigen3::Eigen ALIAS eigen)
target_compile_definitions (eigen INTERFACE ${EIGEN_DEFINITIONS})
target_include_directories (eigen INTERFACE
  $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}>
  $<INSTALL_INTERFACE:${INCLUDE_INSTALL_DIR}>
)

# Export as title case Eigen
set_target_properties (eigen PROPERTIES EXPORT_NAME Eigen)

install (TARGETS eigen EXPORT Eigen3Targets)

configure_package_config_file (
  ${CMAKE_CURRENT_SOURCE_DIR}/cmake/Eigen3Config.cmake.in
  ${CMAKE_CURRENT_BINARY_DIR}/Eigen3Config.cmake
  PATH_VARS EIGEN_INCLUDE_DIR EIGEN_ROOT_DIR
  INSTALL_DESTINATION ${CMAKEPACKAGE_INSTALL_DIR}
  NO_CHECK_REQUIRED_COMPONENTS_MACRO # Eigen does not provide components
)
# Remove CMAKE_SIZEOF_VOID_P from Eigen3ConfigVersion.cmake since Eigen does
# not depend on architecture specific settings or libraries. More
# specifically, an Eigen3Config.cmake generated from a 64 bit target can be
# used for 32 bit targets as well (and vice versa).
set (_Eigen3_CMAKE_SIZEOF_VOID_P ${CMAKE_SIZEOF_VOID_P})
unset (CMAKE_SIZEOF_VOID_P)
write_basic_package_version_file (Eigen3ConfigVersion.cmake
                                  VERSION ${EIGEN_VERSION_NUMBER}
                                  COMPATIBILITY SameMajorVersion)
set (CMAKE_SIZEOF_VOID_P ${_Eigen3_CMAKE_SIZEOF_VOID_P})

# The Eigen target will be located in the Eigen3 namespace. Other CMake
# targets can refer to it using Eigen3::Eigen.
export (TARGETS eigen NAMESPACE Eigen3:: FILE Eigen3Targets.cmake)
# Export Eigen3 package to CMake registry such that it can be easily found by
# CMake even if it has not been installed to a standard directory.
export (PACKAGE Eigen3)

install (EXPORT Eigen3Targets NAMESPACE Eigen3:: DESTINATION $
{CMAKEPACKAGE_INSTALL_DIR})

install ( FILES ${CMAKE_CURRENT_SOURCE_DIR}/cmake/UseEigen3.cmake
                ${CMAKE_CURRENT_BINARY_DIR}/Eigen3Config.cmake
                ${CMAKE_CURRENT_BINARY_DIR}/Eigen3ConfigVersion.cmake
          DESTINATION ${CMAKEPACKAGE_INSTALL_DIR} )

# Add uninstall target
add_custom_target ( uninstall
    COMMAND ${CMAKE_COMMAND} -P
${CMAKE_CURRENT_SOURCE_DIR}/cmake/EigenUninstall.cmake)

if (EIGEN_SPLIT_TESTSUITE)
  ei_split_testsuite("${EIGEN_SPLIT_TESTSUITE}")
endif()
```