CMake

Using CMake (Project Users):

- Navigate to the build folder.

  ```
  $ cd /path/to/project/build
  ```

- Proceed as usual.

  ```
  $ make
  $ make install
  ```

- `cmake` apparantly needs to be run only the first time, before running `make`, though this will need some additional testing to verify.

Setting up a project with `CMake` scripts in place (New Instance of existing project):

- Acquire the project by `svn` or so.
- You should already have a `toolchain` file for the desired toolchain for your system. In not, a sample should be available in the project tree. Adapt it to match your system.
- Navigate to the build folder.

  ```
  $ cd /path/to/project/build
  ```

- Run `cmake` to generate the `makefile`.

  ```
  $ cmake --DCMAKE_TOOLCHAIN_FILE=<path_to_toolchain_file> ../src
  ```

Setting up CMake build scripts (Project Developers):

`CMake` is an ugly build system for C/C++. The only reason we're using it is because it's the natively supported buildsystem for both `KDevelop` and `CLion`, the most promising developing environments freely available. CLion is probably a year or two away, and hopefully they will have a community edition as well, similarly to `pycharm`. For the moment, KDevelop is the environment we hope to be able to use.

- Install cmake :

```
$ sudo aptitude install cmake cmake-qt-gui
```

- Standard Project Tree:

```
.
|-- build
|   `-- [build files]
`-- src
|-- CMakeLists.txt
|-- toolchain-<name>.cmake  (Sample)
`-- [source files]
```

- Every subfolder of SRC includes a `CMakeLists.txt` file declaring everything thats in the folder.
- For the moment, unless otherwise indicated let the root `src` folder not contain any actual source files.
- Create a folder called `application` in the root `src` folder. All application specific things should go there.
- Other folders within `src` should be designed to be (and, subsequently, treated as) independent reusable folders.

Notes on `cmake` files

- `cmake` silently ignores even the worst errors. Exercise caution.
- Strings can't be split across lines.
- Avoid depending on making manual changes to the CMakeCache.txt. Expect it to get nuked without notice.

Root `CMakeLists.txt` file:

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.8)

PROJECT(<project_name> C)

# Toolchain files should be written to use this value, or other mechanism
# must be used.
SET(SUPPORTED_DEVICES "<device_1>;<device_2>"
    CACHE STRING "Supported Target Devices")

# Include all the folders containing header files.
# A nicer method should perhaps be found.
INCLUDE_DIRECTORIES("${PROJECT_SOURCE_DIR}/<directory_name>
```

```
                        ${INCLUDE_DIRECTORIES}")
# ..

# Add all the subdirectories
ADD_SUBDIRECTORY(<directory_name>)
# ..
```

Library folder `CMakeLists.txt` file:

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.8)
# 'Standard' static library:
#ADD_LIBRARY(<library_name> STATIC  <source_files>)
# For special libraries, the toolchain/Platform files should contain
# wrappers that inject the needed extra flags.

ADD_MSP430_LIBRARY(<library_name> STATIC "<library_dependencies>" <source_files>

# Where:
# <library_dependencies> is a space separated list of dependencies.
#            It is a string enclosed by quotes, and must exist
#            even if empty. Only include immediate dependencies.
#            Examples: "" and "a b c"
# <source_files> is a space separated list of source files. This it not a string
#            and should not be enclosed by quotes. Example: a b c
```

Application folder `CMakeLists.txt` file:

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.8)

SET(application_SRCS <source_files>)
LIST(APPEND deps "<dependencies>")
# where:
# <dependencies> is a space separated list of dependencies the exe is to be
#            linked to, enclosed in quotes. Only include immediate
#            dependencies. Ex. "a b c"

# 'Standard' executable example:
# ADD_EXECUTABLE(<output_binary> ${application_SRCS})
# TARGET_LINK_LIBRARIES(<output_binary> <dependencies>)
# (where dependencies is not enclosed in quotes)
#
# For specialized executables, the toolchain / Platform file should contain
# a wrapper that adds the extra targets and fills in the extra flags.

ADD_MSP430_EXECUTABLE(<output_binary> ${deps} ${application_SRCS})
```

Toolchain Files

The toolchain file seems fairly messy to generalize. The basic information that is needed is that the `toolchain` file contains most of the toolchain specific information needed for the build. Every toolchain that has to be used will require it's own `toolchain` file. Every `cmake` enabled project (internally) should contain a `toolchain` sample file, which should be adapted to each machine. Conversely, whenever a toolchain is installed on a machine, the corresponding `toolchain` file can be created in a specific location, to be used by all projects using that toolchain.

For reference, the `toolchain` file for `msp430-gcc` toolchain compatible with the tools installed by standard `ubuntu` packages is listed here for reference. `(\)` in the file represent line breaks that must not exist in the live file.

```
# To be able to use Force Compiler macros.
include(CMakeForceCompiler)

# Add the location of your "toolchains" folder to the module path.
list(APPEND CMAKE_MODULE_PATH "/home/chintal/code/toolchains")

# Name should be 'Generic' or something for which a
# Platform/<name>.cmake (or other derivatives thereof, see cmake docs)
# file exists. The cmake installation comes with a Platform folder with
# defined platforms, and we add our custom ones to the "Platform" folder
# within the "toolchain" folder.
set(CMAKE_SYSTEM_NAME msp430-gcc)

# Compiler and related toochain configuration
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# This can be skipped to directly set paths below, or augmented with hints
# and such. See cmake docs of FIND_PROGRAM for details.
FIND_PROGRAM(MSP430_CC      msp430-gcc)
FIND_PROGRAM(MSP430_CXX     msp430-g++)
FIND_PROGRAM(MSP430_AR      msp430-ar)
FIND_PROGRAM(MSP430_AS      msp430-as)
FIND_PROGRAM(MSP430_OBJDUMP msp430-objdump)
FIND_PROGRAM(MSP430_OBJCOPY msp430-objcopy)
FIND_PROGRAM(MSP430_SIZE    msp430-size)
FIND_PROGRAM(MSP430_MSPDEBUG    mspdebug)

# Since compiler need a -mmcu flag to do anything, checks need to be bypassed
CMAKE_FORCE_C_COMPILER(${MSP430_CC}     GNU)
CMAKE_FORCE_CXX_COMPILER(${MSP430_CXX}  GNU)
```

```
set(AS      ${MSP430_AS} CACHE STRING "AS Binary")
set(AR      ${MSP430_AR} CACHE STRING "AR Binary")
set(OBJCOPY     ${MSP430_OBJCOPY} CACHE STRING "OBJCOPY Binary")
set(OBJDUMP     ${MSP430_OBJDUMP} CACHE STRING "OBJDUMP Binary")
set(SIZE    ${MSP430_SIZE} CACHE STRING "SIZE Binary")

IF(NOT CMAKE_BUILD_TYPE)
    SET(CMAKE_BUILD_TYPE RelWithDebInfo CACHE STRING
        "Choose the type of build, options are: None Debug Release (\)
            RelWithDebInfo MinSizeRel."
        FORCE)
ENDIF(NOT CMAKE_BUILD_TYPE)

set(MSPGCC_OPT_LEVEL    "0" CACHE STRING "MSPGCC OPT LEVEL")

set(MSPGCC_WARN_PROFILE "-Wall -Wshadow -Wpointer-arith -Wbad-function-cast (\)
              -Wcast-align -Wsign-compare -Waggregate-return (\)
              -Wstrict-prototypes -Wmissing-prototypes (\)
              -Wmissing-declarations -Wunused"
                CACHE STRING "MSPGCC WARNINGS")

set(MSPGCC_OPTIONS  "-fdata-sections -ffunction-sections"
                CACHE STRING "MSPGCC OPTIONS")

set(CMAKE_C_FLAGS   "${MSPGCC_WARN_PROFILE} ${MSPGCC_OPTIONS} (\)
              -O${MSPGCC_OPT_LEVEL}  -DGCC_MSP430" CACHE STRING "C Flags")

set(CMAKE_SHARED_LINKER_FLAGS   "-Wl,--gc-sections -Wl,--print-gc-sections"
                    CACHE STRING "Linker Flags")
set(CMAKE_EXE_LINKER_FLAGS  "-Wl,--gc-sections"
                    CACHE STRING "Linker Flags")

# Specify linker command. This is needed to use gcc as linker instead of ld
# This seems to be the preferred way for MSPGCC atleast, seemingly to avoid
# linking against stdlib.
set(CMAKE_CXX_LINK_EXECUTABLE
    "<CMAKE_C_COMPILER> ${CMAKE_EXE_LINKER_FLAGS} <LINK_FLAGS> <OBJECTS> (\)
                -o <TARGET> <LINK_LIBRARIES>"
    CACHE STRING "C++ Executable Link Command")

set(CMAKE_C_LINK_EXECUTABLE ${CMAKE_CXX_LINK_EXECUTABLE}
    CACHE STRING "C Executable Link Command")

# Programmer and related toochain configuration
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
set(PROGBIN     ${MSP430_MSPDEBUG} CACHE STRING "Programmer Application")
set(PROGRAMMER  tilib CACHE STRING "Programmer driver")
```

Platform File

```
# Helper macro for LIST_REPLACE
macro(LIST_REPLACE LISTV OLDVALUE NEWVALUE)
    LIST(FIND ${LISTV} ${OLDVALUE} INDEX)
    LIST(INSERT ${LISTV} ${INDEX} ${NEWVALUE})
    MATH(EXPR __INDEX "${INDEX} + 1")
    LIST(REMOVE_AT ${LISTV} ${__INDEX})
endmacro(LIST_REPLACE)

# Wrapper around ADD_EXECUTABLE, which adds the necessary -mmcu flags and
# sets up builds for multiple devices. Also creates targets to generate
# disassembly listings, size outputs, map files, and to upload to device.
# Also adds all these extra files created including map files to the clean
# list.
FUNCTION(add_msp430_executable EXECUTABLE_NAME DEPENDENCIES)
    SET(DEVICES ${SUPPORTED_DEVICES})

    SET(EXE_NAME ${EXECUTABLE_NAME})
    LIST(REMOVE_AT  ARGV    0)

    SET(DEPS ${DEPENDENCIES})
    SEPARATE_ARGUMENTS(DEPS)
    LIST(REMOVE_AT  ARGV    0)

    FOREACH(device ${DEVICES})

        SET(ELF_FILE ${EXE_NAME}-${device}.elf)
        SET(MAP_FILE ${EXE_NAME}-${device}.map)
        SET(LST_FILE ${EXE_NAME}-${device}.lst)

        ADD_EXECUTABLE(${ELF_FILE} ${ARGN})
        SET_TARGET_PROPERTIES(
            ${ELF_FILE} PROPERTIES
            COMPILE_FLAGS "-mmcu=${device}"
            LINK_FLAGS "-mmcu=${device} -Wl,-Map,${MAP_FILE}"
            )

        SET(DDEPS ${DEPS})
        LIST(REMOVE_DUPLICATES DDEPS)
        FOREACH(dep ${DDEPS})
            LIST_REPLACE(DDEPS "${dep}" "${dep}-${device}")
```

```
        ENDFOREACH(dep)
        TARGET_LINK_LIBRARIES(${ELF_FILE} ${DDEPS})

        ADD_CUSTOM_TARGET(
            ${EXE_NAME}-${device}.lst ALL
            ${MSP430_OBJDUMP} -h -S ${ELF_FILE} > ${LST_FILE}
            DEPENDS ${ELF_FILE}
            )

        ADD_CUSTOM_TARGET(
            ${EXE_NAME}-${device}-size ALL
            ${MSP430_SIZE} ${ELF_FILE}
            DEPENDS ${ELF_FILE}
            )

        ADD_CUSTOM_TARGET(
            ${EXE_NAME}-${device}-upload
            # TODO This needs to be better structured to allow
            # programmer change
            ${PROGBIN} -n ${PROGRAMMER} \"prog ${ELF_FILE}\"
            DEPENDS ${ELF_FILE}
            )

        LIST(APPEND all_lst_files   ${LST_FILE})
        LIST(APPEND     all_elf_files   ${ELF_FILE})
        LIST(APPEND all_map_files   ${MAP_FILE})

    ENDFOREACH(device)

    ADD_CUSTOM_TARGET(
        ${EXE_NAME} ALL
        DEPENDS ${all_elf_files}
        )

    GET_DIRECTORY_PROPERTY(clean_files ADDITIONAL_MAKE_CLEAN_FILES)
    LIST(APPEND clean_files ${all_map_files})
    LIST(APPEND clean_files ${all_lst_files})
    SET_DIRECTORY_PROPERTIES(PROPERTIES
        ADDITIONAL_MAKE_CLEAN_FILES "${clean_files}"
    )

ENDFUNCTION(add_msp430_executable)

# Wrapper around ADD_LIBRARY, which adds the necessary -mmcu flags and
# sets up builds for multiple devices.
FUNCTION(add_msp430_library LIBRARY_NAME LIBRARY_TYPE DEPENDENCIES)
```

```
    SET(DEVICES ${SUPPORTED_DEVICES})

    SET(LIB_NAME ${LIBRARY_NAME})
    LIST(REMOVE_AT  ARGV    0)

    SET(DEPS ${DEPENDENCIES})
    LIST(REMOVE_AT  ARGV    0)

    SET(TYPE ${LIBRARY_TYPE})
    LIST(REMOVE_AT  ARGV    0)

    FOREACH(device ${DEVICES})
        SET(LIB_FILE ${LIB_NAME}-${device})

        ADD_LIBRARY(${LIB_FILE} ${TYPE} ${ARGN})
        SET_TARGET_PROPERTIES(
                ${LIB_FILE} PROPERTIES
                COMPILE_FLAGS "-mmcu=${device}"
                LINK_FLAGS "-mmcu=${device}"
            )

        SET(DDEPS ${DEPS})
        FOREACH(dep ${DEPS})
            LIST_REPLACE(DDEPS "${dep}" "${dep}-${device}")
        ENDFOREACH(dep)
        TARGET_LINK_LIBRARIES(${LIB_FILE} ${DDEPS})
    ENDFOREACH(device)
ENDFUNCTION(add_msp430_library)
```

Useful References

- http://www.vtk.org/Wiki/CMake_Cross_Compiling
- http://www.cmake.org/Wiki/CMake_Useful_Variables
- http://www.elpauer.org/stuff/learning_cmake.pdf
- http://voices.canonical.com/jussi.pakkanen/2013/03/26/a-list-of-common-cmake-antipatterns/

Syntax reference for editing this readme:

- http://johnmacfarlane.net/pandoc/README.html