STM32H735G_DK

Table of Contents

1 Code deployment.

1.1 Fedora system setup.

1.1.1 ‘*dnf install openocd’*

1.1.2 ‘*dnf install telnet*’

1.1.3 ‘*dnf install g++*’

1.1.4 ‘*dnf install arm-none-eabi-gcc-cs*’

1.1.5 ‘*dnf install arm-none-eabi-gcc-cs-c++*’

1.1.6 ‘*dnf install arm-none-eabi-newlib*’

1.2 Build Makefile.

1.2.1 Run after downloading code from a terminal at ‘STM32H735G_DK’:

1.2.1.1 ‘*chmod +x make/*.sh*’

1.2.2 Run each time after adding / removing / moving source files from a terminal at ‘STM32H735G_DK’:

1.2.2.1 ‘*make/build_makefile.sh targets/TOUCH_SCREEN_STM32H735.target > /dev/null*’

1.3 Compile.

1.3.1 Run from a terminal at ‘STM32H735G_DK/bin’:

1.3.2 ‘*make -j8*’

1.4 Chip burning.

1.4.1 Run after connecting discovery board through the USB connection from a terminal at ‘STM32H735G_DK/openocd’:

1.4.1.1 ‘*openocd*’

1.4.2 Run from any other terminal:

1.4.2.1 ‘*telnet localhost 4444*’

1.4.2.2 ‘*reset init*’

1.4.2.3 ‘*flash erase_sector 0 0 last*’

1.4.2.4 ‘*flash write_bank 0* ***path2root/****STM32H735G_DK/bin/touch_screen_stm32h735_debug.bin*’.

1.4.2.5 ‘*reset*’

1.5 Demo*.*

1.5.1 The demo application uses leds, sdram, lcd, and touch screen. After burning the demo, you can follow a single, or double touches on the lcd screen. In case of triple touch, the code asserts (for demonstrating the assert mechanism).

2 Makefile.

2.1 Makefile generating code.

2.1.1 Directory ‘*make*’ holds the makefile generating code, and scripts. That code is not documented.

2.1.2 Code is divided to libraries, and modules.

2.1.3 A library directory holds a file named ‘*library.params*’.

2.1.4 A module directory holds a file name ‘*files.make*’.

2.1.5 The only exception is the *main* directory, which is both a library, and a module. It is also a module that receives the list of active modules (using the compiler -D option for params definition).

2.1.6 When adding a file to repository, you must add it to the ‘files.make’ list.

2.1.7 When adding a directory to the tree, you must add it also in the parent module / library.

2.1.8 Common compiler directives are to be found in ‘*target/XXX.compiler*’. (One such parameter is ‘-D ASSERT_LEVEL_ACTIVE=0x30U’ in ‘target/ARM_GPP_DEBUG.compiler’).

2.1.9 The code generates a single ‘*Makefile*’ at ‘*STM32H735G_DK/bin‘*. This makefile is mostly parallel, so we can use the ‘*make -jx*’ command effectively.

3 Directories structure.

3.1 Layout.

3.1.1 All source directories come under '*src*' path.

3.1.2 Directory '*src*/*COMMON*' holds the common files libraries. These are the files that are platform independent, and application independent.

3.1.3 Directory '*src/PHYSICAL*' holds the chip specific libraries. As an example, we have the '*src/PHYSICAL/STM32H735*' library for the STM32H735 chip. Here you can find mostly registers related code.

3.1.4 Directory 'src/PLATFORM' holds the platform specific libraries. As an example, we have the '*src/PLATFORM/DISCOVERY_735G*' library for the STM32H735G-DK board. Here you can find mostly actual gpio related implementations.

3.1.5 Directory '*src/APPLICATION*' holds the actual applications related code.

3.1.6 As a simple example, consider the I2C objects. '*I2cMaster*' is a '*COMMON*' member, mostly for defining the interface. '*H735I2cMaster*' is a '*PHYSICAL*' member, implementing the interface. '*Disc735I2cMaster4*' is a '*PLATFORM*' member, which initializes the i2c gpios for i2c4.

3.1.7 Libraries usually have a '*redirect*' path, for ease of use. We use the typedef directive for the chip dependent, and platform dependent files. A chip depended is prefixed by '*Physical*' (i.e. the '*H735I2cMaster*' file adds a typedef directive to '*PhysicalI2cMaster*', and redirected by '*PhysicalI2cMaster.h*' in the redirect path). For platforms dependent code we employ the '*Platform*' prefix.

4 Coding conventions.

4.1 Naming conventions.

4.1.1 For objects (class, struct, enum, typedef), we use a capital letter heading each word (followed by lower case). For example – '*ModuleManager*'.

4.1.2 Each file holds a single object. The filename is the same as the object name. (So the filename for the '*ModuleManager' object is ' ModuleManager.h').*

4.1.3 C++ implementations filenames are terminated by '.cc'.

4.1.4 Assembler implementations filenames are terminated by '.a'.

4.1.5 Defined directives are all capital (such as '*ASSERT_CRITICAL*').

4.1.6 Variables are all lower case (such as '*range_start*').

4.1.7 Class member variables are prefixed by '*m_*' (such as '*m_flash_manager*').

4.1.8 Class global variables are prefixed by '*g_*' (Such as '*g_ticks_count*').

4.1.9 Method, and function names are a lower cased word, followed by capital headed word (such as '*getClockControl*').

4.2 Asserts.

4.2.1 ‘ASSERT_TEMP’ – used during development for bug hunting. Should not be present in the code.

4.2.2 ‘ASSERT_DEV’ – used for testing any assumption that is always valid on wakeup (such as offset of a variable in a register structure).

4.2.3 ‘ASSERT_TEST’ – used for testing any state that may happen according to user actions, but would not cause an erroneous system behavior. It may still cause a system crash, such as a NULL pointer access.

4.2.4 ‘ASSERT_CRITICAL’ – used for identifying any possibly erroneous behavior. (an example would be unhandled case in a switch clause).

4.3 Switch cases.

4.3.1 When possible, use explicit case for each possible value, so that the compiler would issue a warning on any new (missing) case.

4.4 Inheritance.

4.4.1 This code prohibits multiple inheritance.

4.4.2 Only exception is the module managers, that may implement multiple pure virtual interfaces.