dnx/FreeRTOS System Documentation

PRELIMINARY

dnx 1.1.0

Daniel Zorychta

October 24, 2013

Contents

1 Introduction **3**
1.1 Coding style . . . 3
1.2 Project download . . . 3

2 Project folder structure **4**
2.1 The *build* folder . . . 5
2.2 The *doc* folder . . . 5
2.3 The *extra* folder . . . 5
2.4 The *src* folder . . . 5
2.5 The *tools* folder . . . 5

3 Project build **6**
3.1 Build from command line . . . 6
3.2 Build from IDE . . . 6

4 Project configuration **8**
4.1 Makefile configuration . . . 8
4.2 The dnx configuration . . . 8
4.3 Driver configuration . . . 9
4.4 File system configuration . . . 9
4.5 Kernel configuration . . . 9

5 Driver development **10**
5.1 Interface . . . 10
5.1.1 The `API_MOD_INIT()` . . . 11
5.1.2 The `API_MOD_RELEASE()` . . . 11
5.1.3 The `API_MOD_OPEN()` . . . 11
5.1.4 The `API_MOD_CLOSE()` . . . 12
5.1.5 The `API_MOD_WRITE()` . . . 12
5.1.6 The `API_MOD_READ()` . . . 13
5.1.7 The `API_MOD_IOCTL()` . . . 13
5.1.8 The `API_MOD_FLUSH()` . . . 13
5.1.9 The `API_MOD_STAT()` . . . 14
5.2 Driver registration . . . 14
5.3 Example module . . . 14
5.3.1 Source file . . . 14
5.3.2 Module configuration header . . . 18
5.3.3 Module definitions header . . . 19

6 File System development **21**
6.1 Interface . . . 21
6.1.1 The API_FS_INIT() . . . 22
6.1.2 The API_FS_RELEASE() . . . 22
6.1.3 The API_FS_OPEN() . . . 23
6.1.4 The API_FS_CLOSE() . . . 23
6.1.5 The API_FS_WRITE() . . . 24
6.1.6 The API_FS_READ() . . . 24
6.1.7 The API_FS_IOCTL() . . . 24
6.1.8 The API_FS_FLUSH() . . . 25
6.1.9 The API_FS_FSTAT() . . . 25
6.1.10 The API_FS_MKDIR() . . . 25
6.1.11 The API_FS_MKNOD() . . . 26
6.1.12 The API_FS_OPENDIR() . . . 26
6.1.13 The API_FS_REMOVE() . . . 26
6.1.14 The API_FS_RENAME() . . . 27
6.1.15 The API_FS_CHMOD() . . . 27
6.1.16 The API_FS_CHOWN() . . . 27
6.1.17 The API_FS_STAT() . . . 28
6.1.18 The API_FS_STATFS() . . . 28
6.1.19 Helper functions . . . 28
6.2 File system registration . . . 29
6.3 Example . . . 29
6.3.1 Source file . . . 29

7 Program development **38**
7.1 Limitations . . . 38
7.2 Program registration . . . 38
7.3 Example . . . 39

8 Porting **41**
8.1 Makefile . . . 41
8.2 Kernel port selection . . . 41
8.3 dnx port create . . . 41

Chapter 1

Introduction

The dnx/FreeRTOS is a general purpose operating system based on FreeRTOS kernel. The dnx layer is modeled on well-known Unix architecture (everything is a file). Destination of the system are small microcontrollers supported by FreeRTOS kernel, especially 32-bit. System is easy scalable to user's needs, user can write own drivers, virtual devices, programs and so on. The programs layer is mostly compatibile with C standard.

Example project is designed for STM32F107RCT6 microcontroller because of development platform when dnx development started. System can be easily ported for other STM32 microcontrollers because all are similarly constructed.

System was designed to create small UNIX system on microcontrollers and to achieve fast time-to-market cycle. Project is an open source product licensed by GPL v2. In project exist software which are licensed on other licenses i.e. FATFS, FreeRTOS, etc. Those products are free for open source project. For commercial usage please contact with producer to get appropriate license. The dnx layer is licensed by GPL v2 license and there is NO WARRANTY.

1.1 Coding style

In the project is used Linux C coding style. See `https://www.kernel.org/doc/Documentation/CodingStyle` to obtain information about this style. Project is written using C99 standard.

1.2 Project download

The dnx project can be download from `git@bitbucket.org:devdnl/dnx.git` Project is stored in GIT repository.

Chapter 2

Project folder structure

Project was implemented with complex folder structure to categorize usage of created software. In the figure 2.1 on page 4 can see current folder structure of project.

```
|
|__ build ........................................ folder with build results
|   |__ stm32f1 ........................ folder with built binary of stm32f1 target
|   |__ * .............................. folder with built binary of ... target
|__ doc ........................................ folder with documentations
|   |__ dxn_doc.* ..................................... documentation files
|   |__ license.txt ................................ the GNU GPL v2 license
|__ extra ................................... folder with additional software
|   |__ programs ......................... folder with extra programs (if exists)
|   |__ templates ................................... folder with templates
|   |__ drivers ........................... folder with extra drivers (if exists)
|   |__ fs ............................ folder with extra filesystems (if exists)
|__ src .................................. folder with compilable source files
|   |__ programs ..................... folder with buil-in programs (user layer)
|   |   |__ * ....................................... folder with program
|   |   |__ Makefile.include ........... makefile with built-in program sources
|   |   |__ program_registration.c ....... system list used to register program
|   |__ system ........................................... folder with system
|       |__ config ..................................... system configuration
|       |__ core ................................... folder with core dnx files
|       |__ drivers ....................... folder with drivers implementation
|       |__ fs .......................... folder with file system implementation
|       |__ include ...................................... folder with headers
|       |__ kernel ............................... folder with kernel sources
|       |__ portable ............ folder with sources depends on architecture
|       |__ user ..................... folder with users's initd start daemon
|       |__ Makefile.include .................. makefile with system source files
|__ tools ................................... folder with miscellaneous tools
|__ changelog ..................................... release changelog file
|__ Makefile ....................................... GNU make Makefile
|__ README .............................................. readme file
```

Figure 2.1: Folder structure

2.1 The *build* folder

The *build* folder is created when new build process is started. After project was clone the folder does not exits. Folder contain all targets which were built in build process after type `make <target>`. Folder is not added to the version control system. Folder contains sub-folders named the same as target typed to the command line in compilation procedure.

2.2 The *doc* folder

The *doc* folder contains all documentation files for system in general or for specific file system or driver.

2.3 The *extra* folder

The *extra* folder contains additional software e.g. programs, drivers (modules), file systems; can also contains other files e.g. templates.

2.4 The *src* folder

The *src* folder contains main software. Software in folder is divided to *programs* and *system* directories. First one contains user's programs, second one full operating system. The *system* folder contains many other folders which contains specific software.

2.5 The *tools* folder

The *tools* folder contains all tool-scripts which are helpful in project development. For example folder contains script that start openOCD application (*runoocd.sh*), and script that is the most important for software developer – *flash.sh*, which programs microcontroller via openOCD software.

Chapter 3

Project build

In general, Makefile used in project is designed to support many different microcontroller's architectures and families. Makefile can be easly corrected to other or additional architectures. The main Makefile is localized in *./Makefile* file.

3.1 Build from command line

To build project go to project's root folder, and type `make <target>`, where `target` is existing microcontroller port. Nowadays, exist only stm32f1 port for STM32F107RCT6 microcontroller. This can be easily expanded to other STM32F1 microcontroller family. For other families based on ARM Cortex-M3 CPU, porting is also easy but more things shall be changed (see Porting section). By default, project is configured for *arm-none-eabi* toolchain. Make sure that this software exist in your system path. Toolchain can be downloaded from Mentor Graphics `http://www.mentor.com/embedded-software/codesourcery` – Code Sourcery; or from Linaro `http://www.linaro.org/downloads`. If you're using *arm-elf* toolchain please change *./Makefile* configuration.

Project starts building when `make stm32f1` was type in console. If compilation was finished successfully then on screen we see information as follow:

```
------------------------------------
| Compilation completed: 14:07:30 |
------------------------------------
```

In *build/stm32f1* directory, ready for flash programming, built files shall exist. Those files can be programmed using *./tools/flash.sh* script typing in console `./tools/flash.sh stm32f1`. Remember to run the openOCD (if your're using) application first to connect to the target board. To do this run *./tools/runoocd.sh* script in other terminal.

Default project is configured for specified development board. To use project in other development board or on target board, please reconfigure project, especially GPIO configuration (*./src/system/drivers/gpio/stm32f1/gpio_cfg.h*). To know more about project configuration read Project configuration section.

3.2 Build from IDE

In project exists Eclipse project files: *./.cproject_template* and *./.project_template*. To use those files user shall change names of those files to: *.cproject* and *.project*, then Eclipse can import project. To compile project user shall click to hammer icon.

If you're using other IDE you're on your own. Other IDEs are not supported by project.

Chapter 4

Project configuration

To configure project, first what user shall know is project organization. Each configurable modules contains **_cfg.h* file. In those files do possible configuration. Configuration files localized are in drivers, can be localized in file systems (currently none file system have configuration file), and other modules. Main system configuration is localized in *./src/system/config/config.h* file. Steps which user shall cross to configure whole system:

1. check if project currently contains port for your microcontroller, if yes go to next step, otherwise new port is needed;
2. make sure that correct linker script is used in *Makefile*. Linker script is depend on microcontroller architecture and family;
3. edit *./src/system/config/config.h* file to set correct settings;
4. add or remove drivers and configure it. Don't forget about GPIO driver;
5. add or remove file systems;
6. correct or write your own initd (*./src/system/user/initd.c*). You can use existing implementation but simpler implementation in most projects are recommended;
7. compile and try that project works correctly.

4.1 Makefile configuration

Makefile configuration is needed when you're using other microcontroller's family than used by default in project. In this case open file *./Makefile* and edit architecture configuration for specified target. Those configuration variables are localized at the beginning of file. Make sure that linker script exist, if not then create your own similar to existing.

4.2 The dnx configuration

Whole dnx configuration can be changed in the *./src/system/config/config.h* file. There are configuration sections like memory size, target frequency and all other.

Note that you shall carefully check interrupt priorities used by your processor. If those configuration are invalid then system can crash all time. The `CONFIG_RTOS_KERNEL_IRQ_PRIORITY` shall be the lowest possible priority. Next (higher) priority `CONFIG_RTOS_SYSCALL_IRQ_PRIORITY` shall be higher than first one. The `CONFIG_USER_IRQ_PRIORITY` priority shall be higher than previous one but this is only default user priority. In the drivers a higher priorities can be used.

4.3 Driver configuration

Driver configuration depends on module usage, construction and needs. Possible module configuration shall be localized in files with *_cfg.h* suffix. If module hasn't any configuration then file shall still exist, even be empty. Don't forgot add or remove driver's files in *./src/system/Makefile.include* file.

4.4 File system configuration

File systems can be configured in *_cfg.h* files. If file systems use external libraries then configuration can be done in the library configuration (e.g. fatfs). To add or remove file system from dnx, edit *./src/system/Makefile.include* file (`CSRC_CORE` variables).

4.5 Kernel configuration

Kernel configuration is done in the *./src/system/config/config.h* file using defined macros. Direct kernel configuration is not allowed, because system use specified kernel functions that have to be enabled or disabled to perform correct system behavior.

Chapter 5

Driver development

Drivers (modules) are based on common structure, other words, each driver have the same unified interfaces. Operating system use suitable interfaces to communicate with driver. Interfaces are defined as functions and all of those functions must exist in module.

5.1 Interface

Functions that create module interfaces:

- `API_MOD_INIT(MODULE_NAME, void **device_handle, u8_t major, u8_t minor)`
- `API_MOD_RELEASE(MODULE_NAME, void *device_handle)`
- `API_MOD_OPEN(MODULE_NAME, void *device_handle, int flags)`
- `API_MOD_CLOSE(MODULE_NAME, void *device_handle, bool force, const task_t *opened_by_task)`
- `API_MOD_WRITE(MODULE_NAME, void *device_handle, const u8_t *src, size_t count, u64_t *fpos)`
- `API_MOD_READ(MODULE_NAME, void *device_handle, u8_t *dst, size_t count, u64_t *fpos)`
- `API_MOD_IOCTL(MODULE_NAME, void *device_handle, int request, void *arg)`
- `API_MOD_FLUSH(MODULE_NAME, void *device_handle)`
- `API_MOD_STAT(MODULE_NAME, void *device_handle, struct vfs_dev_stat *device_stat)`

Functions are constructed using macros `API_MOD_x()`. This macro and first argument are used to generate specified function names. Interface functions returns values which say the system about operation status. In all functions, except `API_MOD_WRITE()` and `API_MOD_READ()`, is used `stdret_t` type that contain a two values:

- `STD_RET_OK` – operation finished successful;
- `STD_RET_NOT_OK` – operation finished with error.

This return type can be extended of other return values specific for module, but those basic values always must be preserved.

`API_MOD_WRITE()` and `API_MOD_READ()` functions return a number of written/read bytes.

5.1.1 The `API_MOD_INIT()`

Function is called when operating system, on user request (`init_driver()` function), initialize driver.
Function prototype:

```
API_MOD_INIT(MODULE_NAME, void **device_handle, u8_t major, u8_t minor)
```

Function arguments:

- `MODULE_NAME` – a module name, this is used to generate function name;
- `device_handle` – a pointer to memory allocated by module. This is an output pointer;
- `major` – a major driver/device number;
- `minor` – a minor driver/device number.

Function return:

- `STD_RET_OK` – an initialization finished successfully;
- `STD_RET_NOT_OK` – an initialization finished with error.

5.1.2 The `API_MOD_RELEASE()`

Function is called when user release driver using `release_driver()` function.
Function prototype:

```
API_MOD_RELEASE(MODULE_NAME, void *device_handle)
```

Function arguments:

- `MODULE_NAME` – a module name, this is used to generate function name;
- `device_handle` – a pointer to allocated memory region in initialization phase.

Function return:

- `STD_RET_OK` – an initialization finished successfully;
- `STD_RET_NOT_OK` – an initialization finished with error.

5.1.3 The `API_MOD_OPEN()`

Function is called when device is opening.
Function prototype:

```
API_MOD_OPEN(MODULE_NAME, void *device_handle, int flags)
```

Function arguments:

- `MODULE_NAME` – a module name, this is used to generate function name;
- `device_handle` – a pointer to allocated memory region in initialization phase;
- `flags` – a device open operation flags. Possible flags:
  `O_RDONLY` – a device opened for read only;
  `O_WRONLY` – a device opened for write only;
  `O_RDWR` – a device opened both for write and read.

Function return:

- `STD_RET_OK` – an initialization finished successfully;
- `STD_RET_NOT_OK` – an initialization finished with error.

5.1.4 The `API_MOD_CLOSE()`

Function is called when device is closing.
Function prototype:

```
API_MOD_CLOSE(MODULE_NAME, void *device_handle, bool force, const task_t
              *opened_by_task)
```

Function arguments:

- `MODULE_NAME` – a module name, this is used to generate function name;
- `device_handle` – a pointer to allocated memory region in initialization phase;
- `force` – a flag which means that device is force closing. Flag is controller by operating system and current task can be different than task which opens device;
- `opened_by_task` – a task handle, contain task number which opened device; valid only when `force` flag is `true`.

Function return:

- `STD_RET_OK` – an initialization finished successfully;
- `STD_RET_NOT_OK` – an initialization finished with error.

5.1.5 The `API_MOD_WRITE()`

Function is called when device write a data. Function returns number of written bytes.
Function prototype:

```
API_MOD_WRITE(MODULE_NAME, void *device_handle, const u8_t *src, size_t
              count, u64_t *fpos)
```

Function arguments:

- `MODULE_NAME` – a module name, this is used to generate function name;
- `device_handle` – a pointer to allocated memory region in initialization phase;
- `src` – a data source pointer;
- `count` – a number of bytes to write;
- `fpos` – a pointer to file position index. Can be read and modified.

Function return:

- `integer (size_t)` – a number of written bytes.

5.1.6 The `API_MOD_READ()`

Function is called when device read a data. Function returns number of read bytes.
Function prototype:

```
API_MOD_READ(MODULE_NAME, void *device_handle, u8_t *dst, size_t count,
             u64_t *fpos)
```

Function arguments:

- `MODULE_NAME` – a module name, this is used to generate function name;
- `device_handle` – a pointer to allocated memory region in initialization phase;
- `dst` – a data destination pointer;
- `count` – a number of bytes to read;
- `fpos` – a pointer to file position index. Can be read and modified.

Function return:

- `integer (size_t)` – a number of read bytes.

5.1.7 The `API_MOD_IOCTL()`

Function is called when non-standard operations are requested.
Function prototype:

```
API_MOD_IOCTL(MODULE_NAME, void *device_handle, int request, void *arg)
```

Function arguments:

- `MODULE_NAME` – a module name, this is used to generate function name;
- `device_handle` – a pointer to allocated memory region in initialization phase;
- `request` – a number of request. Function request numbers are generated using macros:
  `_IO(g, n)` – generate request without arguments;
  `_IOR(g, n, t)` – generate read request;
  `_IOW(g, n, t)` – generate write request;
  `_IOWR(g, n, t)` – generate read/write request.
- `arg` – a pointer to request's argument. Can be used as `int` value, input or output pointer.

Function return:

- `STD_RET_OK` – an initialization finished successfully;
- `STD_RET_NOT_OK` – an initialization finished with error.

5.1.8 The `API_MOD_FLUSH()`

Function is called when flush operation is requested.
Function prototype:

```
API_MOD_FLUSH(MODULE_NAME, void *device_handle)
```

Function arguments:

- `MODULE_NAME` – a module name, this is used to generate function name;
- `device_handle` – a pointer to allocated memory region in initialization phase.

Function return:

- `STD_RET_OK` – an initialization finished successfully;
- `STD_RET_NOT_OK` – an initialization finished with error.

5.1.9 The `API_MOD_STAT()`

Function is called when device status operation are requested.
Function prototype:

```
API_MOD_STAT(MODULE_NAME, void *device_handle, struct vfs_dev_stat
             *device_stat)
```

Function arguments:

- `MODULE_NAME` – a module name, this is used to generate function name;
- `device_handle` – a pointer to allocated memory region in initialization phase;
- `device_stat` – a pointer to device statistics structure.

Function return:

- `STD_RET_OK` – an initialization finished successfully;
- `STD_RET_NOT_OK` – an initialization finished with error.

5.2 Driver registration

To register driver in the system edit *./src/system/drivers/driver_registration.c* file and add specified sections. Registration steps:

- add `#include` statement with driver definitions (ioctl), for example:
  `#include "example_module_def.h"`
- import module external objects using `_IMPORT_MODULE()` macro, for example:
  `_IMPORT_MODULE(EXAMPLE_MODULE);`
- add registration to `_regdrv_module_name[]` array using `_USE_MODULE()` macro to enable module monitoring by system, for example:
  `_USE_MODULE(EXAMPLE_MODULE),`
- define module interfaces using `_USE_DRIVER_INTERFACE()` macro in the `_regdrv_driver_table[]` array, for example:
  `_USE_DRIVER_INTERFACE(EXAMPLE_DRIVER, "ex1", EX_MA_0, EX_MI_0),`

Those steps registers module in the system, don't forgot to add or remove module source files in to makefile (*./src/system/Makefile.include*). To share `ioctl()` definitions add `#include "example_module_def.h"` code to *./src/system/include/system/ioctl.h* file.

5.3 Example module

5.3.1 Source file

```
/*=========================================================================*//**
@file    genericmod.c

@author  Author

@brief   This driver support generic device.

@note    Copyright (C) year  Author <email>

         This program is free software; you can redistribute it and/or modify
         it under the terms of the GNU General Public License as published by
         the  Free Software  Foundation;  either version 2 of the License, or
         any later version.

         This  program  is  distributed  in the hope that  it will be useful,
         but  WITHOUT  ANY  WARRANTY;  without  even  the implied warranty of
         MERCHANTABILITY  or  FITNESS  FOR  A  PARTICULAR  PURPOSE.  See  the
         GNU General Public License for more details.

         You  should  have received a copy  of the GNU General Public License
         along  with  this  program;  if not,  write  to  the  Free  Software
         Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.


*//*==========================================================================*/

#ifdef __cplusplus
extern "C" {
#endif

/*==============================================================================
  Include files
==============================================================================*/
#include "system/dnxmodule.h"
#include "genericmod_cfg.h"
#include "genericmod_def.h"

/*==============================================================================
  Local macros
==============================================================================*/

/*==============================================================================
  Local object types
==============================================================================*/

/*==============================================================================
  Local function prototypes
==============================================================================*/

/*==============================================================================
  Local objects
==============================================================================*/

/*==============================================================================
  Exported objects
==============================================================================*/

/*==============================================================================
  External objects
==============================================================================*/

/*==============================================================================
  Function definitions
==============================================================================*/

//==============================================================================
/**
```

```
 * @brief Initialize device
 *
 * @param[out]          **device_handle        device allocated memory
 * @param[in ]            major                major device number
 * @param[in ]            minor                minor device number
 *
 * @retval STD_RET_OK
 * @retval STD_RET_ERROR
 */
//==============================================================================
API_MOD_INIT(GENERICMOD, void **device_handle, u8_t major, u8_t minor)
{
        STOP_IF(!device_handle);
        UNUSED_ARG(major);
        UNUSED_ARG(minor);

        return STD_RET_OK;
}

//==============================================================================
/**
 * @brief Release device
 *
 * @param[in ]          *device_handle          device allocated memory
 *
 * @retval STD_RET_OK
 * @retval STD_RET_ERROR
 */
//==============================================================================
API_MOD_RELEASE(GENERICMOD, void *device_handle)
{
        STOP_IF(!device_handle);

        return STD_RET_OK;
}

//==============================================================================
/**
 * @brief Open device
 *
 * @param[in ]          *device_handle          device allocated memory
 * @param[in ]           flags                  file operation flags (O_RDONLY,
    O_WRONLY, O_RDWR)
 *
 * @retval STD_RET_OK
 * @retval STD_RET_ERROR
 */
//==============================================================================
API_MOD_OPEN(GENERICMOD, void *device_handle, int flags)
{
        STOP_IF(!device_handle);

        return STD_RET_OK;
}

//==============================================================================
/**
 * @brief Close device
 *
 * @param[in ]          *device_handle          device allocated memory
 * @param[in ]           force                  device force close (true)
 * @param[in ]          *opened_by_task         task with opened this device (valid
     only if force is true)
 *
 * @retval STD_RET_OK
 * @retval STD_RET_ERROR
 */
```

```
//==============================================================================
API_MOD_CLOSE(GENERICMOD, void *device_handle, bool force, const task_t *opened_by_task)
{
        STOP_IF(!device_handle);

        return STD_RET_OK;
}

//==============================================================================
/**
 * @brief Write data to device
 *
 * @param[in ]          *device_handle          device allocated memory
 * @param[in ]          *src                    data source
 * @param[in ]           count                  number of bytes to write
 * @param[in ][out]     *fpos                   file position
 *
 * @return number of written bytes
 */
//==============================================================================
API_MOD_WRITE(GENERICMOD, void *device_handle, const u8_t *src, size_t count, u64_t *fpos)
{
        STOP_IF(!device_handle);
        STOP_IF(!src);
        STOP_IF(!fpos);

        return 0;
}

//==============================================================================
/**
 * @brief Read data from device
 *
 * @param[in ]          *device_handle          device allocated memory
 * @param[out]          *dst                    data destination
 * @param[in ]           count                  number of bytes to read
 * @param[in ][out]     *fpos                   file position
 *
 * @return number of read bytes
 */
//==============================================================================
API_MOD_READ(GENERICMOD, void *device_handle, u8_t *dst, size_t count, u64_t *fpos)
{
        STOP_IF(!device_handle);
        STOP_IF(!dst);
        STOP_IF(!fpos);

        return 0;
}

//==============================================================================
/**
 * @brief IO control
 *
 * @param[in ]          *device_handle          device allocated memory
 * @param[in ]           request                request
 * @param[in ][out]     *arg                    request's argument
 *
 * @retval STD_RET_OK
 * @retval STD_RET_ERROR
 * @retval ...
 */
//==============================================================================
API_MOD_IOCTL(GENERICMOD, void *device_handle, int request, void *arg)
{
        STOP_IF(!device_handle);

        switch (request) {
        default:
                return STD_RET_ERROR;
        }

        return STD_RET_OK;
}

//==============================================================================
/**
 * @brief Flush device
 *
 * @param[in ]          *device_handle          device allocated memory
 *
 * @retval STD_RET_OK
 * @retval STD_RET_ERROR
 */
//==============================================================================
API_MOD_FLUSH(GENERICMOD, void *device_handle)
{
        STOP_IF(!device_handle);

        return STD_RET_OK;
}

//==============================================================================
/**
 * @brief Device information
 *
 * @param[in ]          *device_handle          device allocated memory
 * @param[out]          *device_stat            device status
 *
 * @retval STD_RET_OK
 * @retval STD_RET_ERROR
 */
//==============================================================================
API_MOD_STAT(GENERICMOD, void *device_handle, struct vfs_dev_stat *device_stat)
{
        STOP_IF(!device_handle);
        STOP_IF(!device_stat);

        device_stat->st_size  = 0;
        device_stat->st_major = 0;
        device_stat->st_minor = 0;

        return STD_RET_OK;
}

#ifdef __cplusplus
}
#endif

/*==============================================================================
  End of file
==============================================================================*/
```

5.3.2 Module configuration header

```
/*=========================================================================*//**
@file    genericmod_cfg.h

@author  Author

@brief   This driver support generic device configuration.

```

```
@note    Copyright (C) year  Author <email>

         This program is free software; you can redistribute it and/or modify
         it under the terms of the GNU General Public License as published by
         the Free Software Foundation; either version 2 of the License, or
         any later version.

         This program is distributed in the hope that it will be useful,
         but WITHOUT ANY WARRANTY; without even the implied warranty of
         MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
         GNU General Public License for more details.

         You should have received a copy of the GNU General Public License
         along with this program; if not, write to the Free Software
         Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.


*//*==========================================================================*/

#ifndef _GENERICMOD_CFG_H_
#define _GENERICMOD_CFG_H_

#ifdef __cplusplus
extern "C" {
#endif

/*==============================================================================
  Include files
==============================================================================*/

/*==============================================================================
  Exported macros
==============================================================================*/

/*==============================================================================
  Exported object types
==============================================================================*/

/*==============================================================================
  Exported objects
==============================================================================*/

/*==============================================================================
  Exported functions
==============================================================================*/

/*==============================================================================
  Exported inline functions
==============================================================================*/

#ifdef __cplusplus
}
#endif

#endif /* _GENERICMOD_CFG_H_ */
/*==============================================================================
  End of file
==============================================================================*/
```

5.3.3 Module definitions header

```
/*=========================================================================*//**
@file    genericmod_def.h

@author  Author

@brief   This driver support generic device definitions (e.g. used in ioctl()).
```

```

@note    Copyright (C) year  Author <email>

         This program is free software; you can redistribute it and/or modify
         it under the terms of the GNU General Public License as published by
         the  Free Software  Foundation;  either version 2 of the License, or
         any later version.

         This  program  is  distributed  in the hope that  it will be useful,
         but  WITHOUT  ANY  WARRANTY;  without  even  the implied warranty of
         MERCHANTABILITY  or  FITNESS  FOR  A  PARTICULAR  PURPOSE.  See  the
         GNU General Public License for more details.

         You  should  have received a copy  of the GNU General Public License
         along  with  this  program;  if not,  write  to  the  Free  Software
         Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.


*//*==========================================================================*/

#ifndef _GENERICMOD_DEF_H_
#define _GENERICMOD_DEF_H_

#ifdef __cplusplus
extern "C" {
#endif

/*==============================================================================
  Include files
==============================================================================*/
#include "genericmod_cfg.h"
#include "system/ioctl_macros.h"

/*==============================================================================
  Exported macros
==============================================================================*/
#define GENERICMOD_IORQ_EX1         _IO(’M’, 0x00)
#define GENERICMOD_IORQ_EX2         _IOR(’M’, 0x01, int*)
#define GENERICMOD_IORQ_EX3         _IOW(’M’, 0x02, int)
#define GENERICMOD_IORQ_EX4         _IOWR(’M’, 0x03, int*)

/*==============================================================================
  Exported object types
==============================================================================*/

/*==============================================================================
  Exported objects
==============================================================================*/

/*==============================================================================
  Exported functions
==============================================================================*/

/*==============================================================================
  Exported inline functions
==============================================================================*/

#ifdef __cplusplus
}
#endif

#endif /* _GENERICMOD_DEF_H_ */
/*==============================================================================
  End of file
==============================================================================*/
```

Chapter 6

File System development

File systems are based on common API structure, other words, each file system have the same unified interfaces. Operating system (by VFS) use suitable interfaces to gets/gives data from/to device. Interfaces are defined as functions and all of those functions must exist in file system.

6.1 Interface

Functions that create module interfaces:

- `API_FS_INIT(FS_NAME, void **fs_handle, const char *src_path)`
- `API_FS_RELEASE(FS_NAME, void *fs_handle)`
- `API_FS_OPEN(FS_NAME, void *fs_handle, void **extra, fd_t *fd, u64_t *fpos, const char *path, int flags)`
- `API_FS_CLOSE(FS_NAME, void *fs_handle, void *extra, fd_t fd, bool force, const task_t *file_owner)`
- `API_FS_WRITE(FS_NAME, void *fs_handle,void *extra, fd_t fd, const u8_t *src, size_t count, u64_t *fpos)`
- `API_FS_READ(FS_NAME, void *fs_handle, void *extra, fd_t fd, u8_t *dst, size_t count, u64_t *fpos)`
- `API_FS_IOCTL(FS_NAME, void *fs_handle, void *extra, fd_t fd, int request, void *arg)`
- `API_FS_FLUSH(FS_NAME, void *fs_handle, void *extra, fd_t fd)`
- `API_FS_FSTAT(FS_NAME, void *fs_handle, void *extra, fd_t fd, struct vfs_stat *stat)`
- `API_FS_MKDIR(FS_NAME, void *fs_handle, const char *path)`
- `API_FS_MKNOD(FS_NAME, void *fs_handle, const char *path, const struct vfs_drv_interface *drv_if)`

- `API_FS_OPENDIR(FS_NAME, void *fs_handle, const char *path, DIR *dir)`
- `API_FS_REMOVE(FS_NAME, void *fs_handle, const char *path)`
- `API_FS_RENAME(FS_NAME, void *fs_handle, const char *old_name, const char *new_name)`
- `API_FS_CHMOD(FS_NAME, void *fs_handle, const char *path, int mode)`
- `API_FS_CHOWN(FS_NAME, void *fs_handle, const char *path, int owner, int group)`
- `API_FS_STAT(FS_NAME, void *fs_handle, const char *path, struct vfs_stat *stat)`
- `API_FS_STATFS(FS_NAME, void *fs_handle, struct vfs_statfs *statfs)`

Functions are constructed using `API_FS_x()` macros. The first argument of those macros are used to generate specified function names. Interface functions returns values which indicate the operating system about operation status. In all functions, except `API_FS_WRITE()` and `API_FS_READ()`, is used `stdret_t` type.

The `API_FS_WRITE()` and `API_FS_READ()` functions return a number of written/read bytes using `size_t` type.

6.1.1 The `API_FS_INIT()`

Function is used to initialize single instance of file system.
Function prototype:

```
API_FS_INIT(FS_NAME, void **fs_handle, const char *src_path)
```

Function arguments:

• `FS_NAME`	–	a file system name, argument used to generate function name;
• `fs_handle`	–	a pointer to memory allocated by file system. This is an output pointer;
• `src_path`	–	a path to device/file where file system data is stored.

Function return:

• `STD_RET_OK`	–	an initialization finished successfully;
• `STD_RET_NOT_OK`	–	an initialization finished with error.

6.1.2 The `API_FS_RELEASE()`

Function is called when file system is releasing (unmount process). Function shall return `STD_RET_OK` when all files used by file system are closed.
Function prototype:

```
API_FS_RELEASE(FS_NAME, void *fs_handle)
```

Function arguments:

• `FS_NAME`	–	a file system name, argument used to generate function name;
• `fs_handle`	–	a memory region allocated by file system at initialization phase.

Function return:

• `STD_RET_OK`	–	an initialization finished successfully;
• `STD_RET_NOT_OK`	–	an initialization finished with error.

6.1.3 The `API_FS_OPEN()`

Function is used to open or create file specified in path.
Function prototype:

```
API_FS_OPEN(FS_NAME, void *fs_handle, void **extra, fd_t *fd, u64_t *fpos,
            const char *path, int flags)
```

Function arguments:

- `FS_NAME` – a file system name, argument used to generate function name;
- `fs_handle` – a memory region allocated by file system at initialization phase;
- `extra` – an extra pointer to specified data created/existing in file open process. This is an output pointer, value defined by file system;
- `fd` – pointer to value which contains file descriptor (file number). This is an output pointer, value defined by file system;
- `fpos` – a file position index. This is an output pointer, value defined by file system depending on flags etc;
- `flags` – a flags which determine an operation at file open process. Possible flags:
  `O_RDONLY` – a file opened for read only;
  `O_WRONLY` – a file opened for write only;
  `O_RDWR` – a file opened both for write and read;
  `O_CREAT` – a file shall be created;
  `O_APPEND` – a file shall exist and is opened to append.

Function return:

- `STD_RET_OK` – an initialization finished successfully;
- `STD_RET_NOT_OK` – an initialization finished with error.

6.1.4 The `API_FS_CLOSE()`

Function is called when file is closing. When `force` flag is `true` then file system shall safe force close specified file even a current task is not the owner task.
Function prototype:

```
API_FS_CLOSE(FS_NAME, void *fs_handle, void *extra, fd_t fd, bool force,
             const task_t *file_owner)
```

Function arguments:

- `FS_NAME` – a file system name, argument used to generate function name;
- `fs_handle` – a memory region allocated by file system at initialization phase;
- `extra` – an extra pointer with file system data, pointer loaded at file open interface;
- `fd` – a file descriptor number created at file open process;
- `force` – a flag which indicate that file shall be force closed;
- `file_owner` – a task handle that is an owner of closing file; valid if `force` is `true`.

Function return:

- `STD_RET_OK` – an initialization finished successfully;
- `STD_RET_NOT_OK` – an initialization finished with error.

6.1.5 The `API_FS_WRITE()`

Function is called when data is write to the file.
Function prototype:

```
API_FS_WRITE(FS_NAME, void *fs_handle,void *extra, fd_t fd, const u8_t
            *src, size_t count, u64_t *fpos)
```

Function arguments:

• `FS_NAME`	–	a file system name, argument used to generate function name;
• `fs_handle`	–	a memory region allocated by file system at initialization phase;
• `extra`	–	an extra pointer with file system data, pointer loaded at file open interface;
• `fd`	–	a file descriptor number created at file open process;
• `src`	–	a pointer to source data;
• `count`	–	a number of bytes to write;
• `fpos`	–	a pointer to file position index.

Function return:

- `integer (size_t)` – a number of written bytes.

6.1.6 The `API_FS_READ()`

Function is called when data is read from the file.
Function prototype:

```
API_FS_READ(FS_NAME, void *fs_handle, void *extra, fd_t fd, u8_t *dst,
           size_t count, u64_t *fpos)
```

Function arguments:

• `FS_NAME`	–	a file system name, argument used to generate function name;
• `fs_handle`	–	a memory region allocated by file system at initialization phase;
• `extra`	–	an extra pointer with file system data, pointer loaded at file open interface;
• `fd`	–	a file descriptor number created at file open process;
• `dst`	–	a pointer to data destination;
• `count`	–	a number of bytes to read;
• `fpos`	–	a pointer to file position index.

Function return:

- `integer (size_t)` – a number of read bytes.

6.1.7 The `API_FS_IOCTL()`

Function is used to provide non-standard operations on files or devices if file is device node.
Function prototype:

```
API_FS_IOCTL(FS_NAME, void *fs_handle, void *extra, fd_t fd, int request,
            void *arg)
```

Function arguments:

- `FS_NAME` – a file system name, argument used to generate function name;
- `fs_handle` – a memory region allocated by file system at initialization phase;
- `extra` – an extra pointer with file system data, pointer loaded at file open interface;
- `fd` – a file descriptor number created at file open process;
- `arg` – a pointer to data or integer value.

Function return:

- `STD_RET_OK` – an initialization finished successfully;
- `STD_RET_NOT_OK` – an initialization finished with error.

6.1.8 The `API_FS_FLUSH()`

Function is used to flush file buffer to solid storage.
Function prototype:

```
API_FS_FLUSH(FS_NAME, void *fs_handle, void *extra, fd_t fd)
```

Function arguments:

- `FS_NAME` – a file system name, argument used to generate function name;
- `fs_handle` – a memory region allocated by file system at initialization phase;
- `extra` – an extra pointer with file system data, pointer loaded at file open interface;
- `fd` – a file descriptor number created at file open process.

Function return:

- `STD_RET_OK` – an initialization finished successfully;
- `STD_RET_NOT_OK` – an initialization finished with error.

6.1.9 The `API_FS_FSTAT()`

Function is used to getting file's information e.g size, times, etc.
Function prototype:

```
API_FS_FSTAT(FS_NAME, void *fs_handle, void *extra, fd_t fd, struct
             vfs_stat *stat)
```

Function arguments:

- `FS_NAME` – a file system name, argument used to generate function name;
- `fs_handle` – a memory region allocated by file system at initialization phase;
- `extra` – an extra pointer with file system data, pointer loaded at file open interface;
- `fd` – a file descriptor number created at file open process;
- `stat` – a pointer to object with file information to write.

Function return:

- `STD_RET_OK` – an initialization finished successfully;
- `STD_RET_NOT_OK` – an initialization finished with error.

6.1.10 The `API_FS_MKDIR()`

Function is used to create an directory in specified path and name.
Function prototype:

```
API_FS_MKDIR(FS_NAME, void *fs_handle, const char *path)
```

Function arguments:

- `FS_NAME` – a file system name, argument used to generate function name;
- `fs_handle` – a memory region allocated by file system at initialization phase;
- `path` – a path where directory shall be created.

Function return:

- `STD_RET_OK` – an initialization finished successfully;
- `STD_RET_NOT_OK` – an initialization finished with error.

6.1.11 The `API_FS_MKNOD()`

Function is used to created device node. If file system doesn't support devices then function shall finish with error.
Function prototype:

```
API_FS_MKNOD(FS_NAME, void *fs_handle, const char *path, const struct
             vfs_drv_interface *drv_if)
```

Function arguments:

- `FS_NAME` – a file system name, argument used to generate function name;
- `fs_handle` – a memory region allocated by file system at initialization phase;
- `path` – a file path where node shall be created;
- `drv_if` – a driver interface provided by VFS layer.

Function return:

- `STD_RET_OK` – an initialization finished successfully;
- `STD_RET_NOT_OK` – an initialization finished with error.

6.1.12 The `API_FS_OPENDIR()`

Function is used to open selected directory.
Function prototype:

```
API_FS_OPENDIR(FS_NAME, void *fs_handle, const char *path, DIR *dir)
```

Function arguments:

- `FS_NAME` – a file system name, argument used to generate function name;
- `fs_handle` – a memory region allocated by file system at initialization phase;
- `path` – a directory path;
- `dir` – a pointer to directory object which must be filled by file system.

Function return:

- `STD_RET_OK` – an initialization finished successfully;
- `STD_RET_NOT_OK` – an initialization finished with error.

6.1.13 The `API_FS_REMOVE()`

Function is used to remove file, directory or device node.
Function prototype:

```
API_FS_REMOVE(FS_NAME, void *fs_handle, const char *path)
```

Function arguments:

- `FS_NAME` – a file system name, argument used to generate function name;
- `fs_handle` – a memory region allocated by file system at initialization phase;
- `path` – a path to object which shall be removed.

Function return:

- `STD_RET_OK` – an initialization finished successfully;
- `STD_RET_NOT_OK` – an initialization finished with error.

6.1.14 The `API_FS_RENAME()`

Function is used to rename file, directory or device node.
Function prototype:

```
API_FS_RENAME(FS_NAME, void *fs_handle, const char *old_name, const char
              *new_name)
```

Function arguments:

- `FS_NAME` – a file system name, argument used to generate function name;
- `fs_handle` – a memory region allocated by file system at initialization phase;
- `old_name` – a path to current existing object which shall be renamed;
- `new_name` – a new object path/name.

Function return:

- `STD_RET_OK` – an initialization finished successfully;
- `STD_RET_NOT_OK` – an initialization finished with error.

6.1.15 The `API_FS_CHMOD()`

Function is used to change a mode of file, directory or device node.
Function prototype:

```
API_FS_CHMOD(FS_NAME, void *fs_handle, const char *path, int mode)
```

Function arguments:

- `FS_NAME` – a file system name, argument used to generate function name;
- `fs_handle` – a memory region allocated by file system at initialization phase;
- `path` – an object path which mode shall be changed;
- `mode` – a new object mode.

Function return:

- `STD_RET_OK` – an initialization finished successfully;
- `STD_RET_NOT_OK` – an initialization finished with error.

6.1.16 The `API_FS_CHOWN()`

Function is used to change file, directory or device node owner and group.
Function prototype:

```
API_FS_CHOWN(FS_NAME, void *fs_handle, const char *path, int owner, int
             group)
```

Function arguments:

- `FS_NAME` – a file system name, argument used to generate function name;
- `fs_handle` – a memory region allocated by file system at initialization phase;
- `path` – an object path which owner or group shall be changed;
- `owner` – an owner ID;
- `group` – a group ID.

Function return:

- `STD_RET_OK` – an initialization finished successfully;
- `STD_RET_NOT_OK` – an initialization finished with error.

6.1.17 The `API_FS_STAT()`

Function is used to getting file's information. Usage of function is the same as `API_FS_FSTAT()` function but user provide path to file, directory or device node.
Function prototype:

```
API_FS_STAT(FS_NAME, void *fs_handle, const char *path, struct vfs_stat
            *stat)
```

Function arguments:

- `FS_NAME` – a file system name, argument used to generate function name;
- `fs_handle` – a memory region allocated by file system at initialization phase;
- `path` – a path to object;
- `stat` – a pointer to object with file information to write.

Function return:

- `STD_RET_OK` – an initialization finished successfully;
- `STD_RET_NOT_OK` – an initialization finished with error.

6.1.18 The `API_FS_STATFS()`

Function is used to getting an information of file system (this file system).
Function prototype:

```
API_FS_STATFS(FS_NAME, void *fs_handle, struct vfs_statfs *statfs)
```

Function arguments:

- `FS_NAME` – a file system name, argument used to generate function name;
- `fs_handle` – a memory region allocated by file system at initialization phase;
- `statfs` – a pointer to object which shall be filled by file system.

Function return:

- `STD_RET_OK` – an initialization finished successfully;
- `STD_RET_NOT_OK` – an initialization finished with error.

6.1.19 Helper functions

Helper functions are file system's internal functions which are created to handle directory read and close functionality. Those functions are filled at directory open process. In file system the number of helper functions can be different because of specific of file system. Prototypes of helper functions:

```
static stdret_t closedir(void *fs_handle, DIR *dir)
static dirent_t readdir(void *fs_handle, DIR *dir)
```

6.2 File system registration

File system registration is mostly the same as driver registration. To register file system user shall edit *./src/system/fs/fs_registration.c* and add specified macros. Use `_IMPORT_FILE_SYSTEM(FS_NAME)` in external object section to import file system prototypes. To share file system interface use `_USE_FILE_SYSTEM_INTERFACE(FS_NAME)` macro in `_FS_table[]` array.

6.3 Example

6.3.1 Source file

```
/*=========================================================================*//**
@file    genericfs.c

@author  Author

@brief   This file support generic file system

@note    Copyright (C) year Author <email>

         This program is free software; you can redistribute it and/or modify
         it under the terms of the GNU General Public License as published by
         the Free Software Foundation; either version 2 of the License, or
         any later version.

         This program is distributed in the hope that it will be useful,
         but WITHOUT ANY WARRANTY; without even the implied warranty of
         MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
         GNU General Public License for more details.

         You should have received a copy of the GNU General Public License
         along with this program; if not, write to the Free Software
         Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.


*//*==========================================================================*/

#ifdef __cplusplus
extern "C" {
#endif

/*==============================================================================
  Include files
==============================================================================*/
#include "system/dnxfs.h"

/*==============================================================================
  Local macros
==============================================================================*/

/*==============================================================================
  Local object types
==============================================================================*/

/*==============================================================================
  Local function prototypes
```

```
=============================================================================*/
static stdret_t closedir(void *fs_handle, DIR *dir);
static dirent_t readdir (void *fs_handle, DIR *dir);

/*=============================================================================
  Local objects
=============================================================================*/

/*=============================================================================
  Exported objects
=============================================================================*/

/*=============================================================================
  External objects
=============================================================================*/

/*=============================================================================
  Function definitions
=============================================================================*/

//=============================================================================
/**
 * @brief Initialize file system
 *
 * @param[out]          **fs_handle             file system allocated memory
 * @param[in ]           *src_path              file source path
 *
 * @retval STD_RET_OK
 * @retval STD_RET_ERROR
 */
//=============================================================================
API_FS_INIT(genericfs, void **fs_handle, const char *src_path)
{
        STOP_IF(!fs_handle);
        STOP_IF(!src_path);

        return STD_RET_ERROR;
}

//=============================================================================
/**
 * @brief Release file system
 *
 * @param[in ]          *fs_handle              file system allocated memory
 *
 * @retval STD_RET_OK
 * @retval STD_RET_ERROR
 */
//=============================================================================
API_FS_RELEASE(genericfs, void *fs_handle)
{
        STOP_IF(!fs_handle);

        return STD_RET_OK;
}

//=============================================================================
/**
 * @brief Open file
 *
 * @param[in ]          *fs_handle              file system allocated memory
 * @param[out]          *extra                  file extra data
 * @param[out]          *fd                     file descriptor
 * @param[out]          *fpos                   file position
 * @param[in]           *path                   file path
 * @param[in]            flags                  file open flags (see vfs.h)
 *
 * @retval STD_RET_OK
 * @retval STD_RET_ERROR
 */
//==============================================================================
API_FS_OPEN(genericfs, void *fs_handle, void **extra, fd_t *fd, u64_t *fpos, const char *path, int flags)
{
        STOP_IF(!fs_handle);
        STOP_IF(!extra);
        STOP_IF(!fd);
        STOP_IF(!fpos);
        STOP_IF(!path);

        return STD_RET_ERROR;
}

//==============================================================================
/**
 * @brief Close file
 *
 * @param[in ]          *fs_handle              file system allocated memory
 * @param[in ]          *extra                  file extra data
 * @param[in ]           fd                     file descriptor
 * @param[in ]           force                  force close
 * @param[in ]          *file_owner             task which opened file (valid if force is true)
 *
 * @retval STD_RET_OK
 * @retval STD_RET_ERROR
 */
//==============================================================================
API_FS_CLOSE(genericfs, void *fs_handle, void *extra, fd_t fd, bool force, const task_t *file_owner)
{
        STOP_IF(!fs_handle);
        STOP_IF(!extra);
        STOP_IF(!file_owner);

        return STD_RET_ERROR;
}

//==============================================================================
/**
 * @brief Write data to the file
 *
 * @param[in ]          *fs_handle              file system allocated memory
 * @param[in ]          *extra                  file extra data
 * @param[in ]           fd                     file descriptor
 * @param[in ]          *src                    data source
 * @param[in ]           count                  number of bytes to write
 * @param[in ]          *fpos                   position in file

 * @return number of written bytes
 */
//==============================================================================
API_FS_WRITE(genericfs, void *fs_handle,void *extra, fd_t fd, const u8_t *src, size_t count, u64_t *fpos)
{
        STOP_IF(!fs_handle);
        STOP_IF(!extra);
        STOP_IF(!src);
        STOP_IF(!fpos);

        return 0;
}

//==============================================================================
/**
 * @brief Read data from file
 *
 * @param[in ]          *fs_handle              file system allocated memory
 * @param[in ]          *extra                  file extra data
 * @param[in ]           fd                     file descriptor
 * @param[out]          *dst                    data destination
 * @param[in ]           count                  number of bytes to read
 * @param[in ]          *fpos                   position in file

 * @return number of read bytes
 */
//==============================================================================
API_FS_READ(genericfs, void *fs_handle, void *extra, fd_t fd, u8_t *dst, size_t count, u64_t *fpos)
{
        STOP_IF(!fs_handle);
        STOP_IF(!extra);
        STOP_IF(!dst);
        STOP_IF(!fpos);

        return 0;
}

//==============================================================================
/**
 * @brief IO operations on files
 *
 * @param[in ]          *fs_handle              file system allocated memory
 * @param[in ]          *extra                  file extra data
 * @param[in ]           fd                     file descriptor
 * @param[in ]           request                request
 * @param[in ][out]     *arg                    request's argument
 *
 * @retval STD_RET_OK
 * @retval STD_RET_ERROR
 * @retval ...
 */
//==============================================================================
API_FS_IOCTL(genericfs, void *fs_handle, void *extra, fd_t fd, int request, void *arg)
{
        STOP_IF(!fs_handle);
        STOP_IF(!extra);

        return STD_RET_ERROR;
}

//==============================================================================
/**
 * @brief Flush file data
 *
 * @param[in ]          *fs_handle              file system allocated memory
 * @param[in ]          *extra                  file extra data
 * @param[in ]           fd                     file descriptor
 *
 * @retval STD_RET_OK
 * @retval STD_RET_ERROR
 */
//==============================================================================
API_FS_FLUSH(genericfs, void *fs_handle, void *extra, fd_t fd)
{
        STOP_IF(!fs_handle);
        STOP_IF(!extra);

        return STD_RET_ERROR;
}
```

```

//==============================================================================
/**
 * @brief Return file status
 *
 * @param[in ]          *fs_handle              file system allocated memory
 * @param[in ]          *extra                  file extra data
 * @param[in ]           fd                     file descriptor
 * @param[out]          *stat                   file status
 *
 * @retval STD_RET_OK
 * @retval STD_RET_ERROR
 */
//==============================================================================
API_FS_FSTAT(genericfs, void *fs_handle, void *extra, fd_t fd, struct vfs_stat *
    stat)
{
        STOP_IF(!fs_handle);
        STOP_IF(!extra);
        STOP_IF(!stat);

        return STD_RET_OK;
}

//==============================================================================
/**
 * @brief Create directory
 *
 * @param[in ]          *fs_handle              file system allocated memory
 * @param[in ]          *path                   name of created directory
 *
 * @retval STD_RET_OK
 * @retval STD_RET_ERROR
 */
//==============================================================================
API_FS_MKDIR(genericfs, void *fs_handle, const char *path)
{
        STOP_IF(!fs_handle);
        STOP_IF(!path);

        return STD_RET_ERROR;
}

//==============================================================================
/**
 * @brief Create node for driver file
 *
 * @param[in ]          *fs_handle              file system allocated memory
 * @param[in ]          *path                   name of created node
 * @param[in ]          *drv_if                 driver interface
 *
 * @retval STD_RET_OK
 * @retval STD_RET_ERROR
 */
//==============================================================================
API_FS_MKNOD(genericfs, void *fs_handle, const char *path, const struct
    vfs_drv_interface *drv_if)
{
        STOP_IF(!fs_handle);
        STOP_IF(!path);
        STOP_IF(!drv_if);

        return STD_RET_ERROR;
}

//==============================================================================
/**
```

```
 * @brief Open directory
 *
 * @param[in ]          *fs_handle              file system allocated memory
 * @param[in ]          *path                   name of opened directory
 * @param[in ]          *dir                    directory object
 *
 * @retval STD_RET_OK
 * @retval STD_RET_ERROR
 */
//==============================================================================
API_FS_OPENDIR(genericfs, void *fs_handle, const char *path, DIR *dir)
{
        STOP_IF(!fs_handle);
        STOP_IF(!path);
        STOP_IF(!dir);

        return STD_RET_ERROR;
}

//==============================================================================
/**
 * @brief Close directory
 *
 * @param[in ]          *fs_handle              file system allocated memory
 * @param[in ]          *dir                    directory object
 *
 * @retval STD_RET_OK
 * @retval STD_RET_ERROR
 */
//==============================================================================
static stdret_t closedir(void *fs_handle, DIR *dir)
{
        STOP_IF(!fs_handle);
        STOP_IF(!dir);

        return STD_RET_OK;
}

//==============================================================================
/**
 * @brief Read directory
 *
 * @param[in ]          *fs_handle              file system allocated memory
 * @param[in ]          *dir                    directory object
 *
 * @return directory entry description object
 */
//==============================================================================
static dirent_t readdir(void *fs_handle, DIR *dir)
{
        STOP_IF(!fs_handle);
        STOP_IF(!dir);

        dirent_t dirent;

        return dirent;
}

//==============================================================================
/**
 * @brief Remove file/directory
 *
 * @param[in ]          *fs_handle              file system allocated memory
 * @param[in ]          *path                   name of removed file/directory
 *
 * @retval STD_RET_OK
 * @retval STD_RET_ERROR
 */
//===============================================================================
API_FS_REMOVE(genericfs, void *fs_handle, const char *path)
{
        STOP_IF(!fs_handle);
        STOP_IF(!path);

        return STD_RET_ERROR;
}

//===============================================================================
/**
 * @brief Rename file/directory
 *
 * @param[in ]          *fs_handle               file system allocated memory
 * @param[in ]          *old_name                old object name
 * @param[in ]          *new_name                new object name
 *
 * @retval STD_RET_OK
 * @retval STD_RET_ERROR
 */
//===============================================================================
API_FS_RENAME(genericfs, void *fs_handle, const char *old_name, const char *new_name)
{
        STOP_IF(!fs_handle);
        STOP_IF(!old_name);
        STOP_IF(!new_name);

        return STD_RET_ERROR;
}

//===============================================================================
/**
 * @brief Change file's mode
 *
 * @param[in ]          *fs_handle               file system allocated memory
 * @param[in ]          *path                    file path
 * @param[in ]           mode                    new file mode
 *
 * @retval STD_RET_OK
 * @retval STD_RET_ERROR
 */
//===============================================================================
API_FS_CHMOD(genericfs, void *fs_handle, const char *path, int mode)
{
        STOP_IF(!fs_handle);
        STOP_IF(!path);

        return STD_RET_ERROR;
}

//===============================================================================
/**
 * @brief Change file's owner and group
 *
 * @param[in ]          *fs_handle               file system allocated memory
 * @param[in ]          *path                    file path
 * @param[in ]           owner                   new file owner
 * @param[in ]           group                   new file group
 *
 * @retval STD_RET_OK
 * @retval STD_RET_ERROR
 */
//===============================================================================
API_FS_CHOWN(genericfs, void *fs_handle, const char *path, int owner, int group)
{
```

```c
        STOP_IF(!fs_handle);
        STOP_IF(!path);

        return STD_RET_ERROR;
}

//==============================================================================
/**
 * @brief Return file/dir status
 *
 * @param[in ]          *fs_handle              file system allocated memory
 * @param[in ]          *path                   file path
 * @param[out]          *stat                   file status
 *
 * @retval STD_RET_OK
 * @retval STD_RET_ERROR
 */
//==============================================================================
API_FS_STAT(genericfs, void *fs_handle, const char *path, struct vfs_stat *stat)
{
        STOP_IF(!fs_handle);
        STOP_IF(!path);
        STOP_IF(!stat);

        stat->st_dev   = 0;
        stat->st_gid   = 0;
        stat->st_mode  = OWNER_MODE(MODE_R) | GROUP_MODE(MODE_R) | OTHER_MODE(MODE_R);
        stat->st_mtime = 0;
        stat->st_size  = 0;
        stat->st_uid   = 0;

        return STD_RET_OK;
}

//==============================================================================
/**
 * @brief Return file system status
 *
 * @param[in ]          *fs_handle              file system allocated memory
 * @param[out]          *statfs                 file system status
 *
 * @retval STD_RET_OK
 * @retval STD_RET_ERROR
 */
//==============================================================================
API_FS_STATFS(genericfs, void *fs_handle, struct vfs_statfs *statfs)
{
        STOP_IF(!fs_handle);
        STOP_IF(!statfs);

        statfs->f_bfree  = 0;
        statfs->f_blocks = 0;
        statfs->f_ffree  = 0;
        statfs->f_files  = 0;
        statfs->f_type   = 1;
        statfs->fsname   = "genericfs";

        return STD_RET_OK;
}

#ifdef __cplusplus
}
#endif

/*==============================================================================
  End of file
```

```
================================================================================*/
```

Chapter 7

Program development

A programs can be development similarly as in standard PC computer. The dnx provide standard C API functions that can be used in the same way as in PC programming. The dnx API doesn't contain all functions, some of functions are implemented in standard libraries and can be used as well.

7.1 Limitations

The dnx program implementation have many limitations:

- Global variables are provided by main global structure and access to those variables are provided by `global->...` macro. This is a pointer to independent program instance. To share global variables to other modules of the program, user shall share global variable structure. Global variable structure shall be between `GLOBAL_VARIABLES_SECTION_BEGIN` and `GLOBAL_VARIABLES_SECTION_END` macros.
- Global variables cannot be predefined at program start up. User shall initialize values of defined global variables if other than 0 values is required.
- Debugging of global variables maybe difficult because of global variables nature (pointer is requested from operating system).
- Its not necessary to include constant variables inside global variable structure because these are localized in ROM.
- Program main function shall be created using `PROGRAM_MAIN()` macro.
- The `<stdio.h>` library shall be **always** put first in include list.
- To register program in system, user shall edit *./src/programs/program_registration.c* file adding suitable statements. After registration, program will be visible in */proc/bin* directory after system start up and procfs mount.

7.2 Program registration

To register program in system, user shall edit *./src/programs/program_registration.c* file. Program registration is mostly the same as driver and file system registration. To add program to built-in program list, edit file adding: `_IMPORT_PROGRAM()` macro to import program objects, and configure program using `_PROGRAM_CONFIG()` macro in `_prog_table[]` variable. Don't forget to add source files to makefile (*./src/programs/-Makefile.include*).

7.3 Example

```
/*==========================================================================*//**
@file    helloworld.c

@author  Daniel Zorychta

@brief   The simple example program

@note    Copyright (C) 2013 Daniel Zorychta <daniel.zorychta@gmail.com>

         This program is free software; you can redistribute it and/or modify
         it under the terms of the GNU General Public License as published by
         the Free Software Foundation; either version 2 of the License, or
         any later version.

         This program is distributed in the hope that it will be useful,
         but WITHOUT ANY WARRANTY; without even the implied warranty of
         MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
         GNU General Public License for more details.

         You should have received a copy of the GNU General Public License
         along with this program; if not, write to the Free Software
         Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.


*//*==========================================================================*/

#ifdef __cplusplus
extern "C" {
#endif

/*==============================================================================
  Include files
==============================================================================*/
#include <stdio.h>
#include <string.h>
#include "system/dnx.h"

/*==============================================================================
  Local symbolic constants/macros
==============================================================================*/

/*==============================================================================
  Local types, enums definitions
==============================================================================*/

/*==============================================================================
  Local function prototypes
==============================================================================*/

/*==============================================================================
  Local object definitions
==============================================================================*/
GLOBAL_VARIABLES_SECTION_BEGIN
/* put here global variables */
GLOBAL_VARIABLES_SECTION_END

/*==============================================================================
  Exported object definitions
==============================================================================*/

/*==============================================================================
  Function definitions
==============================================================================*/

//==============================================================================
/**
 * @brief Program main function
 *
 * @param  argc         count of arguments
 * @param *argv[]       argument table
 *
 * @return program status
 */
//==============================================================================
PROGRAM_MAIN(helloworld, int argc, char *argv[])
{
        puts("Hello world!");
        printf("Free stack: %d\n", get_free_stack());
        printf("Static memory usage: %d\n", get_used_static_memory());
        printf("Memory size: %d\n", get_memory_size());
        printf("Free memory: %d\n", get_free_memory());

        printf("Program arguments:\n");
        for (int i = 0; i < argc; i++) {
                printf("%d: %s\n", i + 1, argv[i]);
        }

        return 0;
}

#ifdef __cplusplus
}
#endif

/*==============================================================================
  End of file
==============================================================================*/
```

Chapter 8

Porting

This section describe steps that user shall achieve to create own port for microcontroller that is supported by FreeRTOS kernel.

8.1 Makefile

To port project to other CPU, user shall create new target in makefile first. The best way is edit *./Makefile* and create new target similarly as example target `stm32f1`. User shall create new architecture configuration that define toolchain, liker script, C flags, etc. In next step user shall edit `all` target to show help when user doesn't type target. This is necessary only if user need this. The latest step is create target with list of dependencies, for example: `stm32f1 : dependencies buildobjects linkobjects hex status`. There are all steps that create output product (hex and bin files).

Sub-makefiles *./src/programs/Makefile.include* and *./src/system/Makefile.include* contains sources for all defined targets and core files. The core files are common for all targets. Only source files which depends on architecture shall be added to source list of specified architecture variables (`CSRC_new_target` `CXXSRC_new_target` `ASRC_new_target` `HDRLOC_new_target`). In the system may exists files that aren't designed for any architecture (e.g. TTY driver). For codes of this kind there are prepared `CSRC_noarch`, `CXXSRC_noarch` and `HDRLOC_noarch` variables.

8.2 Kernel port selection

The FreeRTOS kernel supports many CPUs and many compilers. To enable specified CPU, user shall edit *./src/system/Makefile.include* file and add `CSRC_new_target` variable with specified kernel port. An example:

```
CSRC_stm32f1 += kernel/FreeRTOS/Source/portable/GCC/ARM_CM3/port.c
```

To create new FreeRTOS port please learn more at http://www.freertos.org website.

8.3 dnx port create

To create new dnx port create a new directory with port name in *./src/system/portable* localization (for example stm32f0). To this folder add:

- startup code (C or asm);
- *cpuctl.c* and *cpuctl.h* files, those files contains all functions which are used to communication with system. For example see *./src/system/portable/stm32f1/cpuctl.h* and *./src/system/portable/stm32f1/cpuctl.h* files;
- create a file that handle CPU hooks (Hard fault, error and exception interrupts);
- create a file with CPU interrupt vectors;
- create linker script;
- create all other files which are needed to handle specified CPU.

The *cpuctl.c* and *cpuctl.h* are required files by system and those has defined API functions as follow:

- `void _cpuctl_init(void)`
  function is used to initialize CPU/MCU; function called before kernel start;
- `void _cpuctl_restart_system(void)`
  function is called by system to perform system reboot;
- `void _cpuctl_init_CPU_load_timer(void)`
  function is used to initialize CPU load timer;
- `u32_t _cpuctl_get_CPU_load_timer(void)`
  function return CPU load timer value;
- `void _cpuctl_clear_CPU_load_timer(void)`
  function clear CPU load timer;
- `u32_t _cpuctl_get_CPU_total_time(void)`
  function return CPU total time; time is encountered from last clear;
- `void _cpuctl_clear_CPU_total_time(void)`
  function clear CPU total time;
- `void _cpuctl_sleep(void)`
  function prepare and enter CPU in sleep mode; CPU from sleep mode shall be wake up from any interrupt source (especially context switch interrupt).