CODE TIME TECHNOLOGIES

Abassi RTOS

Introduction

Disclaimer

Code Time Technologies Inc. provides this document “AS IS” without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose.
Code Time Technologies Inc. does not warrant that the contents of this document will meet your requirements or that the document is error-free. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the document. Code Time Technologies Inc. may make improvements and/or changes in the product(s) and/or program(s) described in the document at any time. This document does not imply a commitment by Code Time Technologies Inc. to supply or make generally available the product(s) described herein.

Table of Contents

List of Figures

List of Tables

1 Introduction

This document is both a quick start guide and an "how to" reference guide for using Abassi[1] [R1] (including mAbassi [R2] and µAbassi [R3]).

1.1 Distribution Contents

All Abassi releases typically contain the following directories in the package. There could be extra directories or missing dome of the directory listed here. The directories in bold are the RTOS and the RTOS drivers directories:

Table 1-1 Package contents

```
. --- / --- readme.txt
      |
      / --- Abassi                  RTOS file + support software
      |
      / --- Drivers                 RTOS drivers
      |
      / --- FatFS-####              FAT32 stack : open source FatFS
      |
      / --- FullFAT-####            FAT32 stack : open source Full-FAT
      |
      / --- lwip-1.4.1              IP stack : open source lwIP V 1.4.1
      |
      / --- lwip-2.###              IP stack : open source lwIP Version 2.n.n
      |
      / --- lwip-if                 IP stack : I/F between lwIP & Drivers
      |
      / --- Abassi_proc_tool        Demo extra code, Abassi libraries, workspace
      |
      / --- Platform                Target specific code shared amongst toolchains
      |
      / --- Share                   Demo code shared amongst target / toolchains
      |
      / --- ueFAT-####              FAT32 stack : open source Ultra-Embedded-FAT
```

The most important thing to take onto account with the directory structure is this:

****** Never modify files directly in the released package ******

The proper way to create a new application(s) based on Abassi is to add an application directory with at least these 3 sub-directories:

- `inc`
- `src`
- `Workspace` or `Project`

In the build process or makefile, always put the application `inc` sub-directory as the first one in the include path to search. Same with the sub-directory `src`; if the make `VPATH` feature is used then put the as the first directory in the VPATH list the application `src` sub-directory. Having such an include path and VPATH set-up will always use the files in the application directories instead of the files with the same name from the released directories. So if any of the package files need to be modified, first copy the file in the application `inc` or `src` sub-directory and then modify the file in there, nor the one in the release package.

[1] When Abassi is mentioned in this document, unless explicitly stated, it always means Abassi, mAbassi and µAbassi.

1.1.1 RTOS files

The Abassi RTOS is composed if a few files only.

For the source package

`Abassi.h` / `mAbassi.h` (located in `./Abassi`)

`Abassi.c` / `mAbassi.c` (located in `./Abassi`)

The cache set-up code for target processor with caches (located in `./Abassi`)

A target platform / tool chain specific assembly file (located in `./Platform/src`)

For the library package or uAbassi:

`Abassi.h` / `mAbassi.h` / `uAbassi.h` (located in `./Abassi`)

A target platform / tool chain specific library (located in `./Platform/lib`)

Creating an application based on the Abassi RTOS is simple. For the source package, add all the RTOS source files in the build and select the method to define the build options (section 1.3). For the library package, link with the library file located in `./Platform/lib` and make sure to use method #2 (Section 1.3) to define the build options. For µAbassi, include the library file located in `./Platform/lib` and make sure to use method #6 (Section 1.3) to define the build options. In the application, always include `Abassi.h`, `mAbassi.h` or `uAbassi.h` whenever a RTOS service is used.

1.1.2 RTOS + Drivers

The 3 directories in bold, namely `Abassi`, `Drivers` and `Platform`, are the directory holding the RTOS and BSP code. The RTOS files are described in the previous section and are located in the directories `./Abassi` and `./Platform`. The drivers or BSP files are all in the `Drivers` directory.

The files in these 3 directories (`./Abassi`, `./Platform` and `./Drivers`) are typically guaranteed to be backward compatible when an update is provided. In the rare case when one or more files are not, the information is clearly stated when the updated package is provided.

These RTOS files should never be modified, nor copied in the application directory: all RTOS files are very tightly coupled and mixing files from different release could make the RTOS fail to build or to crash at run time.

1.1.3 Demo Code

Release packages are provided with demos. The project / workspace for the demos is located in the directory `Abassi_proc_tool` and the source code for the demos is mainly located in the directory `Share`, and possibly a few files in `Abassi_proc_tool` (platform and toolchain specific); for the Ethernet demo, `lwIP-IF` too. Files from all other directories are either RTOS, or Drivers, or Open Source software, or target platform manufacturer's BSP.

The code distributed in `Share`, `mAbassi_proc_tool` and in `lwIP-if` is likely <u>not</u> backward compatible between releases. One should never directly use files from these directories in an application. The safe way to proceed is again, copying the desired file into the application's `inc`, or `src` sub-directory and build the application using the copies. Doing so, when an upgraded package is provided, the application will never be affected by changes in the demo code.

1.1.4 Open Source SW

All open source software is provided as is with no modifications. The only possible change that may have been done with the open source code is a redistribution of the files in different directories.

None of the open source software is modified or changed between releases. Change may occur when a new version is included but a change of version is easy to notice as the root directory name has the version number in its name.

1.1.5 Upgrade

If an application has been built by using its local copy of the files from the distribution package, then when an upgraded package is provided, everything from the previous package can be replaced by the contents of the new package and there will be no impact to the application.

If new features were added in the RTOS + Driver files, the demos code likely supports the added feature. Looking at the file differences will highlight the changes and if desirable, the application could be upgraded by doing alike what is done in the demo code.

1.2 Platform.h & HWinfo.h

The file `Platform.h`, located in the `Platform/inc` directory, is essential as it hold the information about the mapping of the interrupts on the controller, i.e. what is the interrupt number for an interrupt coming from a specific peripheral and DMA trigger numbers. It also holds the information about which internal module is connected to the devices on development boards. The definitions for the two type of information are clearly grouped and separated when one looks into the file. The first grouping should never require modifications. The second should not be modified either, but when an application runs on a custom platform that is not compatible with any already supported platforms, a new entry must be added. The file `Platform.h` must first be copied in the application directory. Read through the `Platform.txt` and it becomes clear what value of `OS_PLATFORM` must be selected for a new platform. The selected value is important as using the wrong value will likely render the drivers unusable.

The file `HWinfo.h`, located in the `Platform/inc` directory, describes the capabilities of the device on a the support development boards, it is not used by the RTOS nor the drivers, only the demos rely on it. An application shouldn't require its use but if the application was partly built re-using the demo code it could be needed. As for `Platform.h`, if the application runs on an custom platform and `HWinfo.h` is needed, it must first be copied and modified by adding the required information in it. And as for `Platform.h`, the new platform must have a properly selected value and how to select the value for `OS_PLATFORM` is described in Platform.txt.

1.3 Build Options

There are 6 ways the RTOS build options can be specified and they are listed in order of precedence below:

1 – Define `OS_DEF_IN_INC`:

The file `AbassiDef.h` (it is `AbassiDef.h` for both Abassi and mAbassi) is included in `Abassi.h` / `mAbassi.h`. The file `AbassiDef.h` is not supplied in the package, it has to be created.

2 – Define `OS_DEMO` and set to a –ve value (`< 0`):

The file `AbassiLib.h` (it is `AbassiLib.h` for both Abassi and mAbassi) is included in `Abassi.h` / `mAbassi.h`. The file `AbassiLib.h` is located in the `Platform/inc` directory and contains the build option settings used for creating the library version of Abassi / mAbassi. Copy `AbassiLib.h` into the application directory and modify as needed (source code only).

This is how the library version must be used

3 – Define `OS_DEMO` and set to a non –ve value (`>= 0`):

The file `AbassiDemo.h` (It is `AbassiDemo.h` for both Abassi and mAbassi) is included in `Abassi.h` / `mAbassi.h`.

*** For internal use by Code Time, do not use

4 – Define `OS_TEST_HOOK`

The file `AbassiTestCfg.h` (it is `AbassiTestCfg.h` for both Abassi and mAbassi) is included in `Abassi.h` / `mAbassi.h`.

*** For internal use by Code Time, do not use

5 – Define `OS_DEF_IN_MAKE`:

Define the "C" token `OS_DEF_IN_MAKE` and define all the RTOS build options in the makefile or in the GUI

6 – None of the above

The build options must be set in `Abassi.h` / `mAbassi.h`. This is how µAbassi must be used

*** Legacy, do not use with Abassi / mAbassi

Important: with the library version of Abassi and mAbassi, the only useable method is to define `OS_DEMO` and set to to any negative value (method #3). Not doing so with the library version will make the `Abassi.h` / `mAbassi.h` include file incompatible with the library code. In the case of µAbassi, only method #6 can be used, i.e. do not define any of the token listed in method 1 to 5.

For the drivers, support software, etc. their build options must be set in the makefile or in the GUI. The reason is the RTOS definition file (`Abassi.h`) is not always included in the drivers and support software.

1.4 New Application

All packages are provided with a demo typically named `Template`. This demo is very basic and is provided because it can easily be used to create a new application from scratch. All it does is to create a few tasks that print "`hello world`" on `stdout`. The template shows how to start the RTOS, set-up the RTOS timer tick, initialize the system call layer and set-up the UART driver. For other drivers, looking into the appropriate demo in the directory `Share/src` is of great help to understand how to set-up and initialize a specific driver as the demo code is tailored to show that.

The typical initial sequence of operations an application based on Abassi goes through is:

- Optional H/W set-up
- Start the RTOS
- Initialize the System call layer if used
- Enable the interrupts
- Set-up the RTOS timer
    - Install the `TIMtick()` interrupt handler
    - Initialize the timer to trigger an interrupt every `OS_TIMER_US`
    - Enable the timer interrupt on the interrupt controller
- Install all drivers:
    - Initialize the driver
    - Install the interrupt handler
    - Enable the driver interrupt on the interrupt controller
- Create & resume the application tasks
- Optionally create and resume the debug / monitoring shell
- Self suspend or keep on processing

1.5 Debugging

Here's some information to ease the debugging of an application

1.5.1 Shell

The first resource to ease the debugging is to include the Debug Monitoring shell in the application. This is easily done adding in the application the files `Shell.c` and `SubShell.c`. Create a task that will run the debug / monitoring shell; the shell task function name is `OSshell`. Look into the header of `Shell.c` for explanations on how to set-up the Shell to fit the application requirements. There is a full help menu in the shell, to see all commands available, type `help` and to get detailed information about a command, e.g. the command `cmd`, type `help cmd`.

1.5.2 ARM processors

The ARM processors use error handlers and Abassi supplied error handlers are basically infinite loop in low power mode. On Thumb / Thumb2 and 32 instruction processors it is possible to go back to the instruction that has triggered the error by adding 2 or 4 to the PC register value and then single step. Thumb instruction code needs +2 and +4 for the 32 bit instructions.

The AArch64 architecture is a bit different. When an error occurred, the following registers hold the key information:

`X0 : ESR_ELn` (syndrome register)

`X1 : ELR_ELn` (return address)

`X2 : SPSR_ELn` (saved `Pstate` register)

`X3 : n` (Current exception level)

All unassigned interrupts are using the `OSinvalidISR()` do-nothing handler. To find interrupts that haven't been mapped using the `OSisrInstall()` component, put a breakpoint on `OSinvalidISR()` and the interrupt number can be found:.

On the M# processor line, right upon entry in `OSinvalidISR()`, the register `R0` holds the interrupt number.

On the A9, the first instruction in `OSinvalidISR()` is `"lsr r0, r0, #(31-9)"`, single step on it and the register `R0` will then hold the interrupt number.

On the A53, right upon entry in `OSinvalidISR()`, the register `X0` holds the interrupt number.

1.5.3 More info to be added in the futre

…

2 References

[R1] Abassi RTOS – User Guide, available at http://www.code-time.com
[R2] mAbassi RTOS – User Guide, available at http://www.code-time.com
[R3] µAbassi RTOS – User Guide, available at http://www.code-time.com

3 Revision History

Date	Version	Author/Editor	Description
2018.08.28	1.1	EV	First draft
2018.09.02	1.2	EV	Added debug info