CODE TIME TECHNOLOGIES

µAbassi RTOS

User's Guide

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 Tables

1 Introduction

This document is the User's Guide for the µAbassi RTOS, a minimalist version of mAbassi, and it provides all the information the reader requires to configure and use the RTOS on multi-processor devices. It is an addendum to the User's Guide for the Abassi RTOS [R1] and the User's Guide for mAbassi [R2]. mAbassi is upward compatible with µAbassi, meaning an application build on µAbassi can be used as-is with mAbassi.

1.1 Glossary

BMP	Bounded Multi-Processing.
Load Balancing	Distribute workload across multiple processing units.
SMP	Symmetric Multi-Processing.

2 Features

µAbassi is a minimalist version of mAbassi, which supports tasks, semaphores and mutexes only. It can be considered a toolkit for multi-core as it is possible to create almost every service and component by using these three basic services.

The feature set of µAbassi is as follows:

- Tasks
- Semaphores (priority ordering only; no FCFS)
- Mutexes (priority ordering only; no FCFS)
- SMP / BMP - Packed and normal load balancing
- Run-time safe services creation
- All service creation use the standard "C" library function `malloc()`

The availability of semaphores and mutexes is sufficient to customize almost any types of new services. The semaphores are used as the basic blocking mechanism and the mutexes are used to guarantee mutually exclusive access to a resource.

2.1 Limitations

µAbassi is a minimalist version of mAbassi. As such, many features and services of mAbassi are not available in µAbassi.

3 Overview

This section gives an overview of the µAbassi RTOS. The design choices are explained so as to give the reader an understanding of the decisions made when mAbassi was architected and implemented.

From the designer point of view, the only difference between the single core Abassi and the multi-core mAbassi is the name of the RTOS source files. In the standard multicore RTOS mAbassi, the names of the files are `mAbassi.c` and `mAbassi.h`, but in the multi-core toolkit µAbassi, the names were changed to `uAbassi.c` and `uAbassi.h`.

3.1 Design choices

Same as mAbassi, see [R2].

3.1.1 Portability

Same as mAbassi, see [R2].

3.1.2 Feature set

A sub-set of mAbassi, see [R2]. All info on the set of features supported by µAbassi is given in section 6.

3.1.3 Scalability

Same as mAbassi, see [R2].

3.1.4 Code Size

Same as mAbassi, see [R2].

3.1.5 Data Size

Same as mAbassi, see [R2].

3.1.6 Interrupts not disabled

Same as mAbassi, see [R2].

3.2 Services

A sub-set of mAbassi, see [R2]. All info on the set of services supported by µAbassi is given in section 6.

3.2.1 Tasks

Same as mAbassi, see [R1].

3.2.1.1 Load Balancing

Same as mAbassi, see [R2].

3.2.1.2 Multi-Processor Mode

Same as mAbassi, see [R2].

3.2.2 Semaphores

Same as mAbassi, see [R2], except only priority mode; first come first served is not available.

3.2.3 Mutexes

Same as mAbassi, see [R2], except only priority mode; first come first served is not available.

3.2.4 Event Flags

Not available in µAbassi.

3.2.5 Mailboxes

Not available in µAbassi.

3.2.6 Timer

Same as mAbassi, see [R2], except the timer callback is done at every timer tick.

3.2.7 Interrupts Handlers

Same as mAbassi, see [R2].

3.3 Constraints and Don'ts

Same as mAbassi, see [R2].

3.3.1 Idle Task

Not needed.

3.3.2 Interrupts

Same as mAbassi, see [R2].

3.3.3 Task Suspension

Same as mAbassi, see [R2].

3.3.4 Single Task per Priority

Same as mAbassi, see [R2].

3.4 Distribution Contents

The mAbassi RTOS source code distribution always has a minimum of 4 files:

`uAbassi.h`	The µAbassi RTOS definition file
`uAbassi.c`	The µAbassi RTOS code
`uAbassi_???_???.?`	The processor / compiler specific assembly file
`uAbassiCfg.c`	Configuration file for build options remapped to variables

The mAbassi RTOS object code distribution always has a minimum of 4 files:

`uAbassi.h`	The µAbassi RTOS definition file
`uAbassi_`*`TOOL_TARGET`*`.o`	The µAbassi RTOS code
`uAbassi_???_???.o`	The processor / compiler specific assembly file
`uAbassiCfg.c`	Configuration file for build options remapped to variables

Most of the distributions are supplied with code examples for specific hardware platforms, and some processor/compiler ports may also include device drivers. Consult the processor/compiler port document that applies to your target application.

4 Configuration

Most of the build options required in mAbassi [R2] are not needed in µAbassi because the µAbassi kernel has been custom tailored for a minimalist set of services and components.

4.1 Build Options

The following sub-sections describe the build options that must be defined for µAbassi, and the meaning of their values. Another sub-section explains the implicit values of the Abassi / mAbassi build options that are used in µAbassi; the implicit built options are ignore by µAbassi as the code is custom tailored.

4.1.1 µAbassi's Build Options

4.1.1.1 OS_MAX_PEND_RQST

Same as mAbassi, see [R2]. When µAbassi is used in object form, the value of `OS_MAX_PEND_RQST` is fixed and set to a fairly large value according to the target platform. Refer to the target specific document for the value set for `OS_MAX_PEND_RQST`.

4.1.1.2 OS_MP_TYPE

`OS_MP_TYPE` is not really used as a build option inside µAbassi, instead, the value of the `OS_MP_TYPE` definition is used to set the global variable `G_OS_MP_TYPE` which is then used internally by µAbassi. Doing so allows full configurability of µAbassi when used in object form.

NOTE: `G_OS_MP_TYPE` must be declared `const` as dynamic changes of the `G_OS_MP_TYPE` value can possibly deliver un-predictable results.

4.1.1.3 OS_N_CORE

Same as mAbassi, see [R2], but single core is not supported. Therefore the `OS_N_CORE` build option must be set to a value greater than 1. When µAbassi is used in object form, the value of `OS_N_CORE` is fixed and set according to the target platform.

4.1.1.4 OS_NESTED_INTS

Same as mAbassi, see [R2]. When µAbassi is used in object form, the value of `OS_NESTED_INTS` is fixed and set according to the target platform. Refer to the target specific document for the value set for `OS_NESTED_INTS`.

4.1.1.5 OS_SPINLOCK_BASE

Same as mAbassi, see [R2]. When µAbassi is used in object form, the value of `OS_SPINLOCK_BASE` is fixed and set according to the target platform. Refer to the target specific document for the value set for `OS_SPINLOCK_BASE`.

4.1.1.6 OS_START_STACK

`OS_START_STACK` is not really used as a build option inside µAbassi, instead, the value of the `OS_START_STACK` definition is used to set the global variable `G_OS_START_STACK` which is then used internally by µAbassi. Doing so allows full configurability of µAbassi when used in object form.

4.1.1.7 OS_TIMER_US

`OS_TIMER_US` is not really used as a build option inside µAbassi, instead, the value of the `OS_TIMER_US` definition is used to set the global variable `G_OS_TIME_US` which is then used internally by µAbassi. Doing so allows full configurability of µAbassi when used in object form.

4.1.2 mAbassi's Unused Build Options

The following table lists all the build options of mAbassi that have been eliminated in µAbassi. This means these build options are ignored when using µAbassi. The table gives the information on what is the equivalent value that would create the same code in mAbassi as it is in µAbassi. The table rows that are highlighted show the features that are available / supported in µAbassi.

Table 4-1 mAbassi unused build option

Build Option	Value	Description
OS_ALLOC_SIZE	0	The standard "C" library `malloc()` is used for all memory allocation in µAbassi
OS_COOPERATIVE	0	Cooperative mode is not supported in µAbassi
OS_EVENTS	0	The Event Flags service is not available in µAbassi
OS_FCFS	0	First Come First Served blocking ordering is not supported by µAbassi. Blocking is always in a priority order
OS_IDLE_STACK	0	The `IdleTask` is not needed nor used in µAbassi
OS_LOGGING_TYPE	0	Logging is not supported in µAbassi
OS_MAILBOX	0	The Mailboxes services is not available in µAbassi
OS_MEM_BLOCK	0	The Memory Block services is not available in µAbassi
OS_MIN_STACK_USE	0	The stack usage minimization feature is not available in µAbassi
OS_MTX_DEADLOCK	0	Mutex deadlock detection is not supported in µAbassi
OS_MTX_INVERSION	0	Mutex priority inversion protection is not supported by µAbassi
OS_MTX_OWN_UNLOCK	0	Mutex unlocking restriction is not supported by µAbassi
OS_NAMES	1	Names are supported by all services in µAbassi
OS_OUT_OF_MEM	1	Out of memory detection is supported by trapping a `NULL` return value from `malloc()` in µAbassi
OS_PERF_MON	0	Performance monitoring is not supported by µAbassi
OS_PRE_CONTEXT	0	Callback before the context switch is not supported by µAbassi
OS_PRIO_CHANGE	0	Run-time priority change is not supported by µAbassi
OS_PRIO_MIN	32	33 priority levels are handled by µAbassi
OS_PRIO_SAME	1	Multiple task at the same priority is supported by µAbassi
OS_ROUND_ROBIN	0	Round Robin is not supported by µAbassi
OS_RUNTIME	-1	All services are created at run time; static service creation is not available in µAbassi
OS_SEARCH_ALGO	0	Simple table traversing is supported by µAbassi
OS_STACK_CHECK	0	Stack overflow checking is not supported by µAbassi
OS_STARVE_PRIO	0	Starvation protection is not supported by µAbassi
OS_STARVE_RUN_MAX	0	Starvation protection is not supported by µAbassi
OS_STARVE_WAIT_MAX	0	Starvation protection is not supported by µAbassi
OS_STATIC_BUF_MBLK	0	Memory blocks are not supported by µAbassi

OS_STATIC_MBLK	0	Memory blocks are not supported by µAbassi
OS_STATIC_BUF_MBX	0	Mailboxes are not supported by µAbassi
OS_STATIC_MBX	0	Mailboxes are not supported by µAbassi
OS_STATIC_NAME	0	All services memory is obtained from OSalloc()/malloc() in µAbassi
OS_STATIC_SEM	0	All services memory is obtained from OSalloc()/malloc() in µAbassi
OS_STATIC_STACK	0	All services memory is obtained from OSalloc()/malloc() in µAbassi
OS_STATIC_TASK	0	All services memory is obtained from OSalloc()/malloc() in µAbassi
OS_STATIC_TIM_SRV	0	All services memory is obtained from OSalloc()/malloc() in µAbassi
OS_TASK_SUSPEND	1	Task suspension / resuming is supported in µAbassi
OS_TASK_XTRA_FIELD	0	Task descriptor scratch area is not supported by µAbassi
OS_TIM_TICK_MULTI	0	Multi timer tick interrupt handling is not supported in µAbassi
OS_TIMEOUT	1	Service timeouts are supported in µAbassi
OS_TIMER_CB	1	The callback function is called at every timer tick in µAbassi
OS_TIMER_SRV	0	Timer services are not available in µAbassi
OS_USE_TASK_ARG	0	Task arguments are not supported by µAbassi

4.2 Build Option Selection

Same as mAbassi, see [R2].

4.3 Build Examples

Same as mAbassi, see [R2].

5 Quick Start

Same as mAbassi, see [R2].

6 Components

This section lists all the components available in µAbassi. If an Abassi / mAbassi component is not listed in here, this means the service / feature related to the non-listed component is not supported by µAbassi.

6.1 System Components

- `OSstart()`

6.2 Tasks Components

- `TSKcreate()`
- `TSKgetID()`
- `TSKgetPrio()`
- `TSKisBlk()`
- `TSKisRdy()`
- `TSKisSusp()`
- `TSKmyID()`
- `TSKresume()`
- `TSKselfSusp()`
- `TSKsetCore`
- `TSKsetPrio()`
- `TSKsleep()`
- `TSKstate()`
- `TSKsuspend()`
- `TSKtimeoutKill()`

6.3 Semaphores Components

- `SEMopen()`
- `SEMpost()`
- `SEMwait()`
- `SEMwaitBin()`

6.4 Mutexes Components

- `MTXlock()`
- `MTXopen()`
- `MTXowner()`
- `MTXunlock()`

6.5 Timer Components

- `OS_TIM_TICK_ACK()`
- `G_OStimCnt`
- `TIMcallBack()`
- `OS_HMS_TO_TICK()`
- `OS_MS_TO_TICK()`
- `OS_SEC_TO_TICK()`
- `OS_TICK_PER_SEC`

6.6 Interrupt Components

- `OSdint()`
- `OSeint()`
- `OSisrInstall()`

6.7 Mix Bag Components

- `G_OSmutex`
- `G_OSnoName`
- `OSalloc()`

6.8 Core Components

- `COREgetID()`
- `CORElock()`
- `COREunlock()`
- `SPINlock()` Added in µAbassi version 1.46.37
- `SPINtrylock()` Added in µAbassi version 1.46.37
- `SPINunlock()` Added in µAbassi version 1.46.37

7 Appendix A: mAbassi Start-up

Same as mAbassi, see [R2].

8 Appendix B: BMP

Same as mAbassi, see [R2].

9 Appendix C: Load Balancing

Same as mAbassi, see [R2].

10 Appendix D: Custom Service Example

As stated in Section 2, with only the semaphore and mutex services, it is fairly straightforward to custom create new services. Here, an example on how to create a simple Event Flags service is given, where multiple tasks are allowed to set the event flags, but only one task can get and block on the event flags:

Table 10-1 Event Get example

```
int EventGetOR(EvtDesc, ORmask)
{
int EvtRecv;                                /* Return value                          */

  MTXlock(EvtDesc->Mutex, -1)               /* Get exclusive access to descriptor    */
  while ((EvtDesc->EvtReg & ORmask) == 0) { /* Check if selected flags are set       */
    EvtDesc->IsBlocked = 1;                 /* Inform writer(s) I am blocked         */
    MTXunlock(EvtDesc->Mutex);              /* Release exclusive access              */
    SEMwaitBin(EvtDesc->Sema, -1);          /* Block, waiting to get flags sets      */
    MTXlock(EvtDesc->Mutex, -1);            /* Get exclusive access to descriptor    */
    EvtDesc->IsBlocked = 0;                 /* Report we are not blocked anymore     */
  }                                         /* Redo as long as selected flags not set*/
  EvtRecv = EvtDesc->EvtReg & ORmask;       /* Grab the flags                        */
  EvtDesc->EvtReg &= ~ORmask;               /* Remove selected flags accumulaton     */
  MTXunlock(EvtDesc->Mutex);                /* Release exclusive access              */

  return(EvtRecv);
}
```

Table 10-2 Event Set Example

```
void EventSet(EvtDesc, Event)
{
  MTXlock(EvtDesc->Mutex, -1);              /* Get exclusive access to descriptor    */
  EvtDesc->EvtReg |= Event;                 /* Set the event flags                   */
  MTXunlock(EvtDesc->Mutex);                /* Release exclusive access              */
  if (EvtDesc->IsBlocked != 0) {            /* Is the event reader blocked?          */
    SEMpost(EvtDesc->Sema);                 /* When blocked, post blocking semaphore */
  }

  return;
}
```

The two examples above are functionally correct but they are not as real-time efficient as the event flag service provided by mAbassi. Any custom services based on the semaphore and mutex services will suffer from inefficiencies when compared to the same service internally handled in the mAbassi kernel. µAbassi is considered a low cost multi-core SMP tool kit; as such, almost everything can be created with the basic services it provides but this is obtained at the cost of real-time efficiency compared to the same functionality by mAbassi.

11 References

[R1] Abassi RTOS – User's Guide, available at http://www.code-time.com
[R2] mAbassi RTOS – User's Guide, available at http://www.code-time.com
[R3] http://en.wikipedia.org/wiki/Spinlock, Spinlock description