STMicroelectronics

OS21 for ARM

User manual

8083358 Rev C

May 2008

www.st.com

BLANK

User manual

OS21 for ARM

Introduction

The API defined in the *OS21 User manual* (ADCS 7358306) encapsulates the generic facilities offered by OS21 on all target platforms. However each processor implements certain features in different ways, and some processors offer facilities appropriate to their own specific API.

ARM specific APIs can be accessed by a single `#include`:

```
#include <os21/arm.h>
```

This include file is automatically included from `<os21.h>` when `__ARM__` is defined. The ARM GCC compiler always defines `__ARM__`; therefore `#include <arm.h>` is normally all that is necessary to include both the generic OS21 API and the ARM specific API.

Contents

Preface

Document identification and control

Each book carries a unique ADCS identifier of the form:

ADCS *nnnnnnnx*

where *nnnnnnn* is the document number, and *x* is the revision.

Whenever making comments on a document, the complete identification ADCS *nnnnnnnx* should be quoted.

Conventions used in this guide

General notation

The notation in this document uses the following conventions:

- `sample code`, `keyboard input` and `file names`,
- *variables* and *`code variables`*,
- *`code comments`*,
- **screens**, **windows** and **dialog boxes,**
- **instructions**.

Hardware notation

The following conventions are used for hardware notation:

- REGISTER NAMES and FIELD NAMES,
- PIN NAMES and SIGNAL NAMES.

Software notation

Syntax definitions are presented in a modified Backus-Naur Form (BNF). Briefly:

1. Terminal strings of the language, that is, strings not built up by rules of the language are printed in teletype font. For example, `void`.
2. Nonterminal strings of the language, that is, strings built up by rules of the language are printed in italic teletype font. For example, *`name`*.
3. If a nonterminal string of the language starts with a nonitalicized part, it is equivalent to the same nonterminal string without that nonitalicized part. For example, `vspace-`*`name`*.
4. Each phrase definition is built up using a double colon and an equals sign to separate the two sides (‘`::=`’).
5. Alternatives are separated by vertical bars (‘`|`’).
6. Optional sequences are enclosed in square brackets (‘`[`’ and ‘`]`’).
7. Items which may be repeated appear in braces (‘`{`’ and ‘`}`’).

Acknowledgements

ARM is a registered trademark of ARM Ltd.

1 Memory access

1.1 Memory access overview

On all ARM variants that are supported by OS21, memory regions can be given various characteristics such as cacheability and protection modes. These characteristics are controlled through an MMU.

Exact details of the MMU hardware can be found in the appropriate hardware manual.

OS21 provides two areas of memory access support.

- Cache support functions are available on all ARM variants. Details are given in the *OS21 User manual* (ADCS 7358306).
- MMU support functions are available on all ARM variants. These functions are provided using the Virtual Memory API. Details are given in the *OS21 User manual* (ADCS 7358306).

2 Timers

2.1 Timers overview

In order to run, OS21 requires at least three independent timers. Each of these must be capable of running as a free running, auto-reload counter, with interrupt on underflow. Each is programmed to count some fraction of the input clock.

The greatest accuracy is obtained by counting based on a large fraction of the input clock, and running that clock at high frequency.

2.2 Input clock frequency

The precise speed of the input clock is determined by the end user; it is a function of the board design and boot software. OS21 is not responsible for setting the input speed, therefore it has to be made aware of what it is.

This is done with the Board Support Package (BSP) using a function called `bsp_timer_input_clock_frequency_hz`(). Full details on how to use this function can be found in the *OS21 User manual* (ADCS 7358306), chapter 16.

2.3 OS21 tick duration

OS21 establishes the period of one tick when it boots. Based on the input clock frequency it selects an appropriate divisor to yield a tick which is approximately 10 microseconds.

2.4 ARM timer assignments

How OS21 uses the ARM timers depends upon the exact hardware available. *Table 1* shows an example of how the timers (four in this case) are assigned in a typical configuration.

Table 1. ARM timer assignments

Timer name	OS21 usage
`Timer0`	System timer
`Timer1`	Timeslice timer
`Timer2`	Timeout timer
`Timer3`	OS21 Profiling timer

The system timer is left free running and is used by `time_now()` to return the system time. On ARM platforms, the system time (`osclock_t`) is a 64-bit value. OS21 maintains the top 32 bits of the 64-bit time via an interrupt handler which is called each time the 32-bit timer reaches zero. The lower 32 bits of the system time are the value in the system timer.

The timeslice timer is programmed to run for the timeslice period before generating an interrupt and reloading. This is used to drive timeslice events into the task scheduler.

The timeout timer is programmed on demand to interrupt when the required number of ticks have elapsed. When multiple timeouts are requested OS21 orders which timeout should occur next, and programs the timeout timer appropriately.

When OS21 profiling is enabled, the profiling timer generates interrupts at regular intervals. The profiler samples the PC whenever this interrupt is fired.

3 Register context

3.1 Registers overview

The following registers are saved as part of each task's context.

On all platforms:

- R0 to R12
- R13 (stack pointer)
- R14 (link register)
- R15 (PC)
- CPSR (status register)

On platforms with a VFP present:

- F0 to F31
- FPINST2, FPINST, FPEXC, FPSCR (control and status registers)

On platforms with interrupt levels:

- priority mask register

4 Board support packages

4.1 Board support packages overview

OS21 Board Support Packages (BSPs) are supplied for all supported platforms as both pre-built libraries, and accompanying sources. The generic features of the BSPs can be found in the *OS21 User manual* (ADCS 7358306), chapter 16.

This section describes the platform-specific features of the BSP. For the ARM, this consists of the interrupt system description.

4.2 BSP interrupt system description

The BSP is responsible for describing the interrupt system to OS21. This coupled with the platform specific interrupt code implements OS21's generic interrupt API. On ARM platforms, this comprises the following elements:

- interrupt names
- interrupt table
- interrupt controller base address
- interrupt controller slave priority
- interrupt system initialization flags and settings

4.2.1 Interrupt names

A type is provided by OS21 called `interrupt_name_t`. Each interrupt is assigned a unique name (`interrupt_name_t`) which allows it to be identified both in the BSP interrupt tables that follow, and in the interrupt API. The BSP need only contain those interrupts that are used by other OS21 or the application code.

If any interrupts are missing then a linker error occurs. If interrupts are declared in the BSP but are subsequently not used, then this does no harm other than use memory. For example:

```
/* Define a DMA interrupt in the BSP */
interrupt_name_t OS21_INTERRUPT_DMA_0 = 21;
```

Header files are provided with OS21 which complement the interrupt description in the BSP. By including the appropriate header file, all the relevant external `interrupt_name_t` declarations are obtained. User code is also free to declare only those interrupt names that it requires. For example:

```
/* How to access the DMA interrupt in user code */
extern interrupt_name_t OS21_INTERRUPT_DMA_0;
```

The `interrupt_handle()` function takes an `interrupt_name_t` parameter and returns a handle to the given interrupt.

4.2.2 Interrupt table

The interrupt table describes the interrupt system to the OS21 platform-specific interrupt API implementation code. On ARM there is only one interrupt table. The format of the table depends upon whether or not interrupt priority levels are supported.

Interrupt table - no interrupt priority level support

Some ARM platforms do not support interrupt levels (for example, STn8815). In this case each entry in the interrupt table has four fields, as follows:

```
/* An entry in the interrupt table */
typedef struct interrupt_table_entry_s
{
  interrupt_name_t * namep;
  unsigned int       controller;
  unsigned int       reg_set;
  unsigned int       bit_set;

} interrupt_table_entry_t;
```

The table describes interrupts that are routed to specific interrupt controllers.

`namep` is a pointer to the name of the interrupt. `controller` specifies the interrupt controller where the interrupt arrives (only OS21_CTRL_INTC is supported on ARM platforms). `reg_set` is the number of the register set to which the interrupt is routed. Currently interrupts are routed to two registers on the interrupt controller; OS21 numbers these 0 or 1. `bit_set` is the bit number within this register.

This table enables OS21 to locate an appropriate bit in the INTC that maps to the named interrupt. For example:

```
interrupt_name_t OS21_MY_INTERRUPT = 21;
interrupt_table_entry_t my_interrupt =
               { &OS21_MY_INTERRUPT, OS21_CTRL_INTC, 1, 15 };
```

This describes an interrupt called `OS21_MY_INTERRUPT`, which is routed into bit 15 of register set 1 on the interrupt controller `OS21_CTRL_INTC`.

```
interrupt_table_entry_t bsp_interrupt_table [];
```

This describes the set of interrupts that arrive on the INTC. It comprises a list of `interrupt_table_entry_t` types. For example:

```
interrupt_table_entry_t bsp_interrupt_table [] =
{
  { &OS21_INTERRUPT_TIMER_0, OS21_CTRL_INTC, 0, 0 },
  { &OS21_INTERRUPT_TIMER_1, OS21_CTRL_INTC, 0, 1 },
  { &OS21_INTERRUPT_TIMER_2, OS21_CTRL_INTC, 0, 2 },
  { &OS21_INTERRUPT_TIMER_3, OS21_CTRL_INTC, 0, 3 }
};
```

This describes a basic system with four timer interrupts. Real systems are likely to be different to this.

Interrupt table - with interrupt priority level support

For those ARM platforms that do support interrupt levels (for example STn8820) each entry in the interrupt table has five entries:

```
/* An entry in the interrupt table */
typedef struct interrupt_table_entry_s
{
  interrupt_name_t * namep;
  unsigned int       controller;
  unsigned int       reg_set;
  unsigned int       bit_set;
  unsigned int       priority;

} interrupt_table_entry_t;
```

This table describes interrupts that are routed to the interrupt controllers.

`namep` is a pointer to the name of the interrupt. `controller` specifies the interrupt controller where the interrupt is directed (only OS21_CTRL_INTC is supported on ARM platforms). `reg_set` is the number of the register set to which the interrupt is routed. Currently interrupts are routed to two registers on the interrupt controller; OS21 numbers these 0 or 1. `bit_set` is the bit number within this register. `priority` is the priority or level of the given interrupt.

OS21 implements 16 interrupt levels on ARM platforms with level support. Level 1 is assigned to the lowest priority level, level 16 to the highest. This is not necessarily the same as implemented in hardware.

This table allows OS21 to locate an appropriate bit in the INTC which maps to the named interrupt. For example:

```
interrupt_name_t OS21_MY_INTERRUPT = 21;
interrupt_table_entry_t my_interrupt =
               { &OS21_MY_INTERRUPT, OS21_CTRL_INTC,1, 15, 4 };
```

This describes an interrupt called OS21_MY_INTERRUPT which is routed into bit 15 of register set 1 on the interrupt controller OS21_CTRL_INTC. The interrupt has a level of 4.

```
interrupt_table_entry_t bsp_interrupt_table [];
```

This describes the set of interrupts that arrive on the INTC. It comprises a list of interrupt_table_entry_t types. For example:

```
interrupt_table_entry_t bsp_interrupt_table [] =
{
  { &OS21_INTERRUPT_TIMER_0, OS21_CTRL_INTC, 0, 0, 1 },
  { &OS21_INTERRUPT_TIMER_1, OS21_CTRL_INTC, 0, 1, 2 },
  { &OS21_INTERRUPT_TIMER_2, OS21_CTRL_INTC, 0, 2, 3 },
  { &OS21_INTERRUPT_TIMER_3, OS21_CTRL_INTC, 0, 3, 4 }
};
```

This describes a basic system with four timer interrupts. The four interrupts have different priorities, with TIMER 3 being the highest priority. Real systems are likely to be different to this.

4.2.3 Interrupt controller base address

This variable informs OS21 of the base address of the interrupt controller.

For example:

```
void * bsp_intc_base_address  = (void *)(0x90410000);
```

This tells OS21 the interrupt controller registers commence at address 0x90410000.

4.2.4 Interrupt controller slave priority

On platforms that support interrupt levels, and where slave controllers drive master controllers, this variable informs OS21 of the priority of the slave controller interrupts relative to the interrupts on the master. For example:

```
unsigned int bsp_intc_slave_pri = 15;
```

This says that all interrupts on the slave controller are given a priority of 15 into the master controller.

Note: This item is not required on platforms that do not support interrupt priority levels.

4.2.5 Interrupt controller initialization flags

```
interrupt_init_flags_t bsp_interrupt_init_flags;
```

This is a set of flags that are used to control how OS21 initializes the interrupt subsystem. These may be combined by logically ORing the appropriate flags.

The normal value on ARM platforms is 0.

5 Revision history

Table 2. Document revision history

Date	Revision	Changes
9-May-2008	C	Moved MMU Mappings Description of the board support package to the *OS21 User manual* (ADCS 7358306R)
13-Nov-2007	B	Moved the generic description of the board support package to the *OS21 User manual* (ADCS 7358306Q).
14-Sep- 2007	A	Initial release.

Index

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY AN AUTHORIZED ST REPRESENTATIVE, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS "AUTOMOTIVE GRADE" MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER’S OWN RISK.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com