



# RTAI Port to MCF5329

## Developer's manual

M5329/RTAI3.8

Rev. 0.1 02/2010

# CONTENTS

1) Introduction	1
2) RTAI Features for MCF5329	1
2.1) RTAI's Services	1
2.1.1. Module rtai_hal	7
2.1.2. Module rtai_lxrt	7
2.1.3. Module rtai_sched	7
2.1.4. Module rtai_fifos	7
2.1.5. Module rtai_wd	7
2.1.6. Module rtai_msg	7
2.1.7. Module rtai_bits	8
2.1.8. Module rtai_mq	8
2.1.9. Module rtai_sem	8
2.1.10. Module rtai_netrpc	8
2.1.11. Module rtai_tbx	8
2.1.12. Module rtai_mbx	8
2.1.13. Module rtai_tasklets	8
2.1.14. Module rtai_shm	8
2.2) Related Libraries	2
3) RTAI Installation and Usage	9
3.1) RTAI General Services	9
3.1.1. Hard real time	9
3.1.2. RTAI and other real time projects	9
3.1.3. RTAI implementation	9
3.2) Linux #!P for MCF5329 Setup	5
3.3) RTAI Installation	5
3.6) RTAI Test Suite Run	2
3.5) Important Notes	6
3) Changes in the Linux Kernel and !P-pe for our, e Code	5

## Author's Document

This document describes setting up and usage of RTAI for MCF5329 installed into Linux embedded OS, and the changes in RTAI and Linux kernel source code, which allow using RTAI with MCF5329.

## Author

This document targets Linux software developers using the MCF5329 processor.

## References

- [1] MCF5329 Reference Manual Rev. 0
- [2] RTAI 3.4 User Manual Rev 0.3
- [3] Advanced Linux Programming. M. Mitchel, J. Oldham, A. Samuel

## Definitions, Acronyms and Abbreviations

The following list defines the acronyms and abbreviations used in this document.

ADEOS	Adaptive Domain Environment for Operating Systems, a nanokernel used by RTAI
FEC	ColdFire Fast Ethernet Controller
FIFO	First Input First Output
HAL	Hardware Abstraction Layer
I-Pipe	Interrupt Pipeline
LED	Light-Emitting Diode
OS	Operating System
RDTSC	Read Time Stamp Counter – the function returning number of ticks from the system start.
RTAI	Real Time Application Interface
RTC	Real Time Clock
SRQ	System Request
UART	Universal Asynchronous Receiver/Transmitter
BSP	Board Support Package

LTIB	Linux Target Image Builder
------	----------------------------

## )\* Intro+u , t-on

This document describes the Real Time Application Interface (RTAI), ported to the MCF5329. RTAI is a Linux kernel extension, that allows preemption of the Linux kernel at any time in order to perform real time operations with interrupt latencies in the microseconds range. The standard Linux kernel can have latencies of several milliseconds.

The document is divided logically into five parts.

The first part contains a general overview of RTAI.

The second part contains the description of its installation and usage.

The third part describes the changes, which were made in the Linux kernel 2.6.26 and Linux drivers. Mainly it contains information about modifications of the interrupt handling routines and timer routines.

The fourth part is a description of the changes made in the RTAI source code during porting.

The fifth part gives a short manual of creation of RTAI applications.

## )\*)\* RTAI Features /or MCF5329

- Correct execution of the real time tasks in periodic mode with the frequencies 3 kHz and less (In this mode real time task period have to be a multiple of the timer period).
- Correct execution of the real time tasks in oneshot mode with the frequencies 5 kHz and less (In this mode real time task period have to be a variable value based on the timer clock frequency).
- RTAI services are provided by 14 kernel modules, which allow hard real time, fully preemptive scheduling. These modules are: *rtai\_hal*, *rtai\_sched*, *rtai\_lxrt*, *rtai\_fifos*, *rtai\_wd*, *rtai\_msg*, *rtai\_bits*, *rtai\_mq*, *rtai\_sem*, *rtai\_netrpc*, *rtai\_tbx*, *rtai\_mbx*, *rtai\_tasklets*, *rtai\_shm*. Note that 15-th module, *rtai\_usi*, contains no code, so there is no reason to use it.



)<sup>2</sup>1\* Module rtai\_bits

It's RTAI event flags functions.

)<sup>2</sup>2\* Module rtai\_mq

It's POSIX-like message queues.

)<sup>2</sup>9\* Module rtai\_sem

It's RTAI semaphore functions.

)<sup>2</sup>)5\* Module rtai\_netrpc

It's a module for network real time communications.

)<sup>2</sup>))\* Module rta-<t : 4

It's RTAI message queues.

)<sup>2</sup>)2\* Module rta-<m : 4

It's RTAI mailbox functions.

)<sup>2</sup>)3\* Module rtai\_tasklets

It's an RTAI's implementation of tasklets. RTAI tasklets are used when functions are needed to be called from user- and kernel-space.

)<sup>2</sup>)6\* Module rtai\_shm

It's RTAI shared memory functions.

)<sup>3</sup>\* Related files

The following files are relevant to RTAI:

- rtai-3.8.tar.bz2 – RTAI 3.8 original package.

## 2\* RTAI Installation and "sa3e"

This chapter describes how to install and patch Linux BSP and RTAI and deploy it on M5329 evaluation board. It also contains a general overview of RTAI and information about RTAI installation.

### 2\*)\* RTAI 'eneral %verv-e0

#### 2\*)\*)\* &ar+ real t-me

True multi-tasking operating systems, such as Linux, are adopted for use in increasingly complex systems, where the need for hard real time often becomes apparent. "Hard real time" can be found in the systems, which are dependent from guaranteed system responses of thousandths or millionths of a second. Since these control deadlines can never be missed, a hard real time system cannot use average case performance to compensate for worst-case performance.

#### 2\*)\*2\* RTAI an+ ot8er real t-me pro-e, ts

There are four primary variants of hard real time Linux available: RTLinux, Xenomai and RTAI.

RTLinux was developed at the New Mexico Institute of Technology by Michael Barabanov under the direction of Professor Victor Yodaiken. Real Time Application Interface (RTAI) was developed at the Dipartimento di Ingegneria Aerospaziale, Politecnico di Milano by Professor Paolo Mantegazza. One of the main advantages of RTAI is the support of periodic mode scheduling and its performance. Xenomai, that was launched in 2001 provides slightly worse performance comparing to RTAI.

#### 2\*)\*3\* RTAI -mplementat-on

For the real time Linux scheduler the Linux OS kernel is an idle task. Therefore Linux executes only when the real time tasks aren't running and the real time kernel isn't active. RTAI 3.8 uses ADEOS nanokernel for managing interrupts. ADEOS provides Interrupt Pipeline (called I-Pipe), that delivers interrupts to domains. One of these domains is Linux, the second – RTAI. In hard real time mode (when there is a real time task running) Linux domain is in "stalled" state, which means it doesn't receive interrupts. So Linux kernel doesn't schedule, because timer interrupt never





## 2\*6\* RTAI test7su-te runn-n3

To launch RTAI test-suite, some additional steps must be performed:

1. Insert required modules:

```
# insmod rtai_hal.ko
# insmod rtai_sched.ko
# insmod rtai_sem.ko
# insmod rtai_fifos.ko
# insmod rtai_mbx.ko
# insmod rtai_msg.ko
```

2. Launch RTAI tests:

### **Kernel-space *latency* test in oneshot mode:**

```
# insmod latency_rt.ko
# ../testsuite/kern/latency/display
```

*Latency* test will start displaying its results, until you press Ctrl+C.

```
# rmmmod latency_rt.ko
```

### **Kernel-space *latency* test in periodic mode:**

```
# insmod latency_rt.ko timer_mode=1
# ../testsuite/kern/latency/display
```

*Latency* test will start displaying its results, until you press Ctrl+C.

```
# rmmmod latency_rt.ko
```

### **Kernel-space *preempt* test:**

```
# insmod preempt_rt.ko
# ../testsuite/kern/preempt/display
```

*Preempt* test will start displaying its results, until you press Ctrl+C.

```
# rmmmod preempt_rt.ko
```

### **Kernel-space switches test:**

```
# insmod switches_rt.ko
```

*Switches* test will display its results in a few seconds.

```
# rmmmod switches_rt.ko
```

### **User-space latency test in oneshot mode:**

```
# cd /usr/realtime/testsuite/user/latency
```

```
# ./latency&
```

```
# ./display
```

*Latency* test will start displaying its results, until you press ENTER.

### **User-space preempt test:**

```
# cd /usr/realtime/testsuite/user/preempt
```

```
# ./preempt&
```

```
# ./display
```

*Preempt* test will start displaying its results, until you press Ctrl+C. Also you will need to type

```
# killall preempt
```

in order to stop user-space preempt test.

### **User-space switches test:**

```
# cd /usr/realtime/testsuite/user/switches
```

```
# ./switches
```

*Switches* test will display its results in a few seconds.

*Note:* The description for each test can be found in README file in the test directory.

## 2\*5\* I7P-pe -mportant not-, es

### Notice 1:

You should enable I-Pipe (interrupt pipeline) in your kernel only if you want to use RTAI. I-Pipe slows down interrupt processing in Linux, that will be visible when working with drivers that use lots of interrupts for data transfers instead of DMA. If both I-Pipe and fast interrupt processing is required, then either use RTAI drivers (that should be written by you) or use hacks in I-Pipe to pass some interrupts directly to drivers (not via I-Pipe). The second way must be used very carefully and with full understanding of what you are doing.

### Notice 2:

If RTAI is active then Linux interrupts are not called immediately. Instead, each captured interrupt is *masked* and then hardware interrupts will be enabled. Linux handler (if it exists) will be called when all RTAI tasks will become inactive. The interrupt will be *unmasked* immediately after the Linux handler. This will not interfere with existing Linux BSP drivers, because all of them use kernel functions for masking/unmasking interrupts instead of working directly with interrupt controller registers. These kernel functions are modified by I-Pipe patch to make I-Pipe masking/unmasking described earlier transparent to drivers. If you are writing your own driver you must use these kernel functions (*enable\_vector* and *disable\_vector*) for masking and unmasking interrupts instead of direct access to interrupt controller registers.



