

# ARMtwister Users Manual

V1.03a 01/20/05

## Table of Contents

ARMtwister Users Manual.....	1
The Midori Distribution.....	1
Download.....	1
Building the image.....	1
Log in to your board .....	2
Upload the image.....	2
Loading the kernel to RAM via network.....	2
Loading the uncompressed kernel to RAM via JTAG.....	2
Programming the flash.....	3
Hardware.....	3
PCMCIA Interface.....	4
Supported cards.....	4
EDO DRAM.....	4
Memory Map.....	5
Parallel I/O .....	6
Header pinout.....	7

## The Midori Distribution

### Download

Depending on your Linux installation, a couple of additional modules may be required. The Midori Distribution should work with most Linux Distributions. Until now, Debian and Redhat have been tested.

*Debian (woody):*

```
apt-get install wget ncftp flex bison patch unzip xlibs-dev libncurses5-dev zlib1g-dev bzip2
```

Now, create a working directory for downloading the sources, and get the required sources by cvs.

```
mkdir work
cd work

cvs -d :pserver:anoncvs@home.at:/home/cvs login (password: anoncvs)
cvs -z3 -d :pserver:anoncvs@home.at:/home/cvs co midori
```

If you are using the Source CD that is supplied with the ARMtwister, you can unpack the source archive from the CD instead:

```
mkdir work
cd work

< mount the CD >
tar zxf <CD mount point>/midori-1.03.tgz
```

On the source CD, all required original source archives are already available in *midori/repository/*, thus you can save the time that is otherwise required by downloading the sources from the Internet during the build process. The automatic downloading of source packages may sometimes break, either because the server is down or if the archive has been moved to a different location or removed from the server.

### Building the image

After downloading the CVS repository or unpacking the source, you are ready to start the web

configurator if you want to modify the midori setup.

```
cd midori
make webconfig
```

Point your web browser to <http://localhost:2000/>. If you want to do the configuration from a different location, you have to quit the configuration with `Ctrl-C`, and edit the line with `CONFIGURATOR_CLIENT` in `config`. Then, start the web configurator again with `make webconfig`.

When you are satisfied with the configuration, save the configuration in the web configurator and start the build with

```
make
```

The first build may take up to several hours. Internet access is required because the original source packages are downloaded by ftp and http during the build.

The original packages are placed into the directory `midori/repository`.

The packages that have been selected in the web configurator are unpacked in `midori/cache/build`, all necessary patches are applied automatically

The toolchain for cross compilation is built.

The kernel is configured and built, including kernel modules

A compressed (`zImage.u`) and a uncompressed kernel image (`Image.u`), containing the rom filesystem with the applications and configurations files, is built and placed into the `tftp` directory that is specified in the configuration (default: `midori/tftp/`).

## Log in to your board

all operations that are described below can be executed via telnet to your board, or by a terminal connected to the serial port. The IP for the board is 192.168.1.100. You can use any login name that you want, and the password `uClinux`.

*Serial connection:* 9600/8/1/N, any login name, password: uClinux

*Telnet session:* IP=192.168.1.100, any login name, password: uClinux

## Upload the image

to upload the new kernel images to your board, there are several choices available. If the flash content is destroyed and the board doesn't boot anymore, the only choice is the upload of an image by JTAG.

### **Loading the kernel to RAM via network**

this is the fastest method for testing. The image is first loaded into memory from the running linux kernel over Ethernet or WLAN, then the bootloader is started to execute the new image.

```
cd ~/work/midori

# mount NFS volume (substitute your NFS server path instead of "192.168.1.254:/home/user")
mount -o nolock 192.168.1.254:/home/user /var/mnt

# cd into the tftp directory on your NFS volume
cd /var/mnt/tftp

# load & execute the new image
# you can use the compressed (zImage.u) or the uncompressed kernel (Image.u) here.
loader zImage.u
```

### **Loading the uncompressed kernel to RAM via JTAG**

This can be done with the armtool monitor that is included in the Midori distribution. You need a JTAG interface for the parallel port, as described on <http://jtag-arm9.sourceforge.net/>. With this

method, only uncompressed images can be uploaded and executed from RAM.

# 'cd' into your midori directory,e.g.:

```
cd ~/work/midori

# add midori/cache/tools/bin to your PATH
export PATH=`pwd`/cache/tools/bin:$PATH

# configure the JTAG chain
export JTAG_CHAIN='0 0 1 10'

# initialize the board
sh cache/build/bootloader-0.1-uboot/src/init-rev6

# strip u-boot header from the kernel image
tail -c +65 Image.u > Image.bin

# upload the kernel image
armtool w 0x1000000 99999999 Image.bin

# execute the kernel
armtool x 0x1000000
```

## Programming the flash

After testing the new kernel in RAM, it can be written to flash so that the new kernel is booted after the next power cycle. The image that is written to flash is composed of the bootloader (4KB) followed by a compressed or uncompressed kernel image. First, the bootloader image must be prepared and copied to the tftp directory on the host system:

```
# change to midori working directory
cd ~/work/midori

# copy the bootloader to the tftp directory, filled up to 4KB
dd if=cache/build/bootloader-0.1-uboot/src/loader.bin bs=4k conv=sync of=tftp/loader_4k.bin
```

On the target system, mount the NFS volume with bootloader & kernel. Make sure that you have access to both files before you erase the flash! It is very important that the power is not interrupted during erasing and flash programming. Depending on the kernel size, the programming of the flash can require a couple of minutes.

```
# mount NFS volume (substitute your NFS server path instead of "192.168.1.254:/home/user")
mount -o nolock 192.168.1.254:/home/user /var/mnt

# cd into the tftp directory on your NFS volume
cd /var/mnt/tftp

# DO NOT TURN OFF THE POWER DURING THE NEXT STEPS
# erase the flash
erase /dev/mtd0 0 32

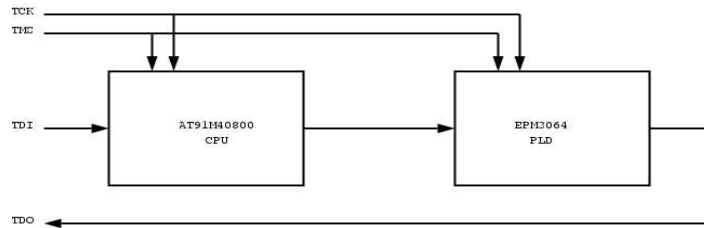
# write bootloader + kernel to flash
cat loader_4K.bin zImage.u > /dev/mtd0
```

The error message after the erase can be ignored. On the next reboot, the system is booted with the new kernel.

## Hardware

The JTAG chain

The AT91M40800 and the PLD are in the same JTAG chain. In order to communicate with a JTAG device in the chain, it is important to know the chain structure and the instruction register length for the non-used device. During the JTAG communication, the appropriate number of shift cycles is added to shift the data through the additional device. The JTAG chain allows programming of the systems flash without requiring any bootcode inside the flash.



## PCMCIA Interface

The PCMCIA socket is designed for both 3.3V and 5V 16-bit PCMCIA cards. To reduce cost, no host controller is used. Instead, address- and databus are directly connected to the socket. Only the control signals are generated by a PLD. The design does not provide hot plugging support. A power switching IC (MAX1602) is provided for the Vcc and Vpp voltage supply of the PCMCIA card. There are two pins (VS1\*, VS2\*) on the PCMCIA socket for detection of the card's supply voltage (3.3V/5V). The power switching IC selects the correct voltage that is applied to the sockets Vcc pins (17,51). If the card is designed for operation below 3.3V only, no power is applied to the socket to prevent damage.

### Supported cards

The hardware is designed for 16-bit PCMCIA cards only. Although 32-bit Cardbus cards fit into the socket mechanically, they have a completely different interface that is not available on the ARMTwister. The attempt to use such a card may damage the ARMTwister board as well as the card. The following 16-bit PCMCIA cards have been tested extensively:

<i>Manufacturer</i>	<i>card</i>	<i>comments</i>
D-Link	DFE-650	Ethernet
D-Link	DFE-650TXD	Ethernet
D-Link	DWL-650	WLAN
D-Link	DWL-660	WLAN
SMC	8041TX v.2	Ethernet
Netgear	FA-410	Ethernet
Netgear	FA-411	Ethernet, axnet driver
BENQ	AWL100	WLAN
Spectrum	LA-3021	2Mbps frequ. hopping WLAN
OvisLink	AirLive WL-1100PCM	WLAN
Conceptronic	CBT100C	Bluetooth® 100m card

The following CompactFlash Cards have been used successfully with a CF adapter:

<i>Manufacturer</i>	<i>card</i>	<i>comments</i>
D-Link	DCF-660W	WLAN
Linksys	WCF12	WLAN

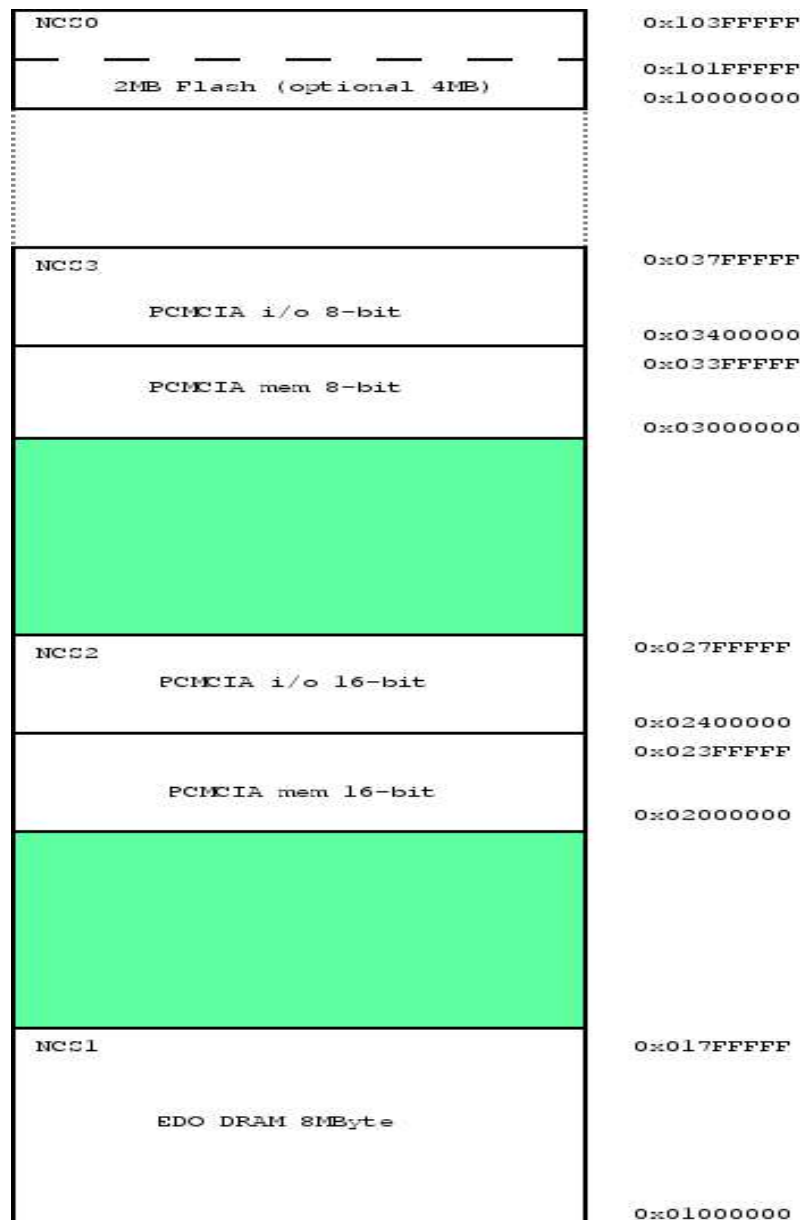
## EDO DRAM

The AT91M40xxx does not have a DRAM controller. The DRAM controller is implemented in the EPM3064 PLD, including the refresh logic. The refresh is triggered by the Timer output P7\_TIOA2. The timer must be programmed by the bootloader so that the refresh is working before the DRAM is accessed. TIOA2 is an output from Timer 2, this has the consequence that Timer 2

cannot be used for other purposes. On the AT91M40xxx, the peripherals continue to work even when the core is stopped. Thus, the DRAM refresh continues even when the processor is in Debug mode. The generation of the refresh trigger signal is only stopped by a hardware reset. The other two timers can be used without any restriction. For DRAM access, two standard wait states are used, giving a memory access time of  $3 \times 25\text{nsec} = 75\text{nsec}$  at 40MHz for read and write access. For consecutive DRAM access cycles an additional external wait cycle is applied because of a necessary DRAM precharge cycle following each memory access.

## Memory Map

For the PCMCIA socket, a fixed memory mapping is used. The PCMCIA address space is divided into four ranges of 4MB each, with the ability to use 8-bit/16-bit access as well as i/o and memory access. When the AT91 is reset, the flash memory is placed at address 0x00000000. The flash memory is accessed in 16-bit mode (BMS=0) after reset.



## Parallel I/O

<i>PIO controller</i>		<i>Peripheral</i>			<i>ARMTwister</i>	
<i>PIO bit</i>	<i>Port name</i>	<i>Signal description</i>	<i>Signal direction</i>	<i>Reset state</i>	<i>Name</i>	<i>Description</i>
P0	TCLK0	Timer 0 clock signal	input	PIO Input	E_IO3	general purpose I/O
P1	TIOA0	Timer 0 signal A	bi-directional	PIO Input	E_IO6	general purpose I/O
P2	TIOB0	Timer 0 signal B	bi-directional	PIO Input	E_IO7	general purpose I/O
P3	TCLK1	Timer 1 clock signal	input	PIO Input	E_IO5	general purpose I/O
P4	TIOA1	Timer 1 signal A	bi-directional	PIO Input	E_IO8	general purpose I/O
P5	TIOB1	Timer 1 signal B	bi-directional	PIO Input	E_IO9	general purpose I/O
P6	TCLK2	Timer 2 clock signal	input	PIO Input	E_IO4	general purpose I/O
P7	TIOA2	Timer 2 signal A	bi-directional	PIO Input	TIMER	DRAM Refresh
P8	TIOB2	Timer 2 signal B	bi-directional	PIO Input	E_IO10	general purpose I/O
P9	IRQ0	External Interrupt 0	input	PIO Input	P_IRQ	PCMCIA IRQ
P10	IRQ1	External Interrupt 1	input	PIO Input	-	-
P11	IRQ2	External Interrupt 2	input	PIO Input	A0VPP	PCMCIA Intel code
P12	FIQ	Fast Interrupt	input	PIO Input	E_IO1	general purpose I/O
P13	SCK0	USART 0 clock	bi-directional	PIO Input	A1VPP	PCMCIA Intel code
P14	TXD0	USART 0 transmit	output	PIO Input	TXD0	JP3/3 RS232 level
P15	RXD0	USART 0 receive	input	PIO Input	RXD0	JP3/2 RS232 level
P16	-	-	-	PIO Input	CTS0	JP3/8 RS232 level
P17	-	-	-	PIO Input	P_A24	PCMCIA A24
P18	-	-	-	PIO Input	P_A25	PCMCIA A25
P19	-	-	-	PIO Input	P_RST	PCMCIA Reset
P20	SCK1	USART 1 clock	bi-directional	PIO Input	E_IO2	general purpose I/O
P21	TXD1	USART 1 transmit	output	PIO Input	TXD1	JP2/19
P22	RXD1	USART 1 receive	input	PIO Input	RXD1	JP2/20
P23	-	-	-	PIO Input	P_WP	PCMCIA write protect
P24	-	-	-	PIO Input	RTS0	JP3/7 RS232 level
P25	MCK0	Master Clock Output	output	MCKO	E_IO0	general purpose I/O
P26	NCS2	Chip Select 2	output	NCS2	NCS2	PCMCIA I/O select
P27	NCS3	Chip Select 3	output	NCS3	NCS3	PCMCIA MEM select
P28	A20	Adress 20	output	A20	A20	address bus
P29	A21	Adress 21	output	A21	A21	address bus
P30	A22	Adress 22	output	A22	A22	address bus
P31	A23	Adress 23	output	A23	A23	address bus

## Header pinout

<b>JP1: JTAG connector</b>			
Vdd	1	2	Vdd
-	3	4	Ground
TDI	5	6	Ground
TMS	7	8	Ground
TCK	9	10	Ground
TCK	11	12	Ground
TDO	13	14	Ground
E_NRST	15	16	Ground
-	17	18	Ground
-	19	20	Ground

The pinout of JP1 is pin- and function-compatible to the JTAG connector on the Atmel evaluation boards (EB01/EB40/EB63)

<b>JP2: I/O connector</b>					
	Ground	1	2	Ground	
	Vdd (3.3V)	3	4	Vdd (3.3V)	
	E_5V (PCMCIA)	5	6	E_12V (PCMCIA)	
P25	E_IO0	7	8	E_IO1	P12
P20	E_IO2	9	10	E_IO3	P0
P6	E_IO4	11	12	E_IO5	P3
P1	E_IO6	13	14	E_IO7	P2
P4	E_IO8	15	16	E_IO9	P5
P8	E_IO10	17	18	-	
P21	TXD1	19	20	RXD1	P22

E\_5V is only required for 5V PCMCIA cards

E\_12V is only required for some old PCMCIA cards

RXD1/TXD1 from UART1 are without line drivers, they may be used as general purpose I/O lines if required.

<b>JP3: RS 232 connector</b>			
-	1	2	E_RXD0
E_TXD0	3	4	Pin 4 is connected to
GND	5	6	Pin6
E_RTS0	7	8	E_CTS0
RI	9	10	-