Creative Systems Engineering

AMRG

Advanced Modem Router Gateway

Development Platform User Manual

Rev 2.0

Table Of Contents

List of Tables

List of Figures

Figure **Page**

1 INTRODUCTION

CSE's AMRG is an integrated gateway development platform, featuring ADSL2/2+ interface, five Ethernet ports, USB 2.0 port and WiFi module.

This document presents the features and procedures that allow the AMRG to be used for applications development.

2 SYSTEM OVERVIEW

2.1 HARDWARE OVERVIEW

2.1.1 Block diagram

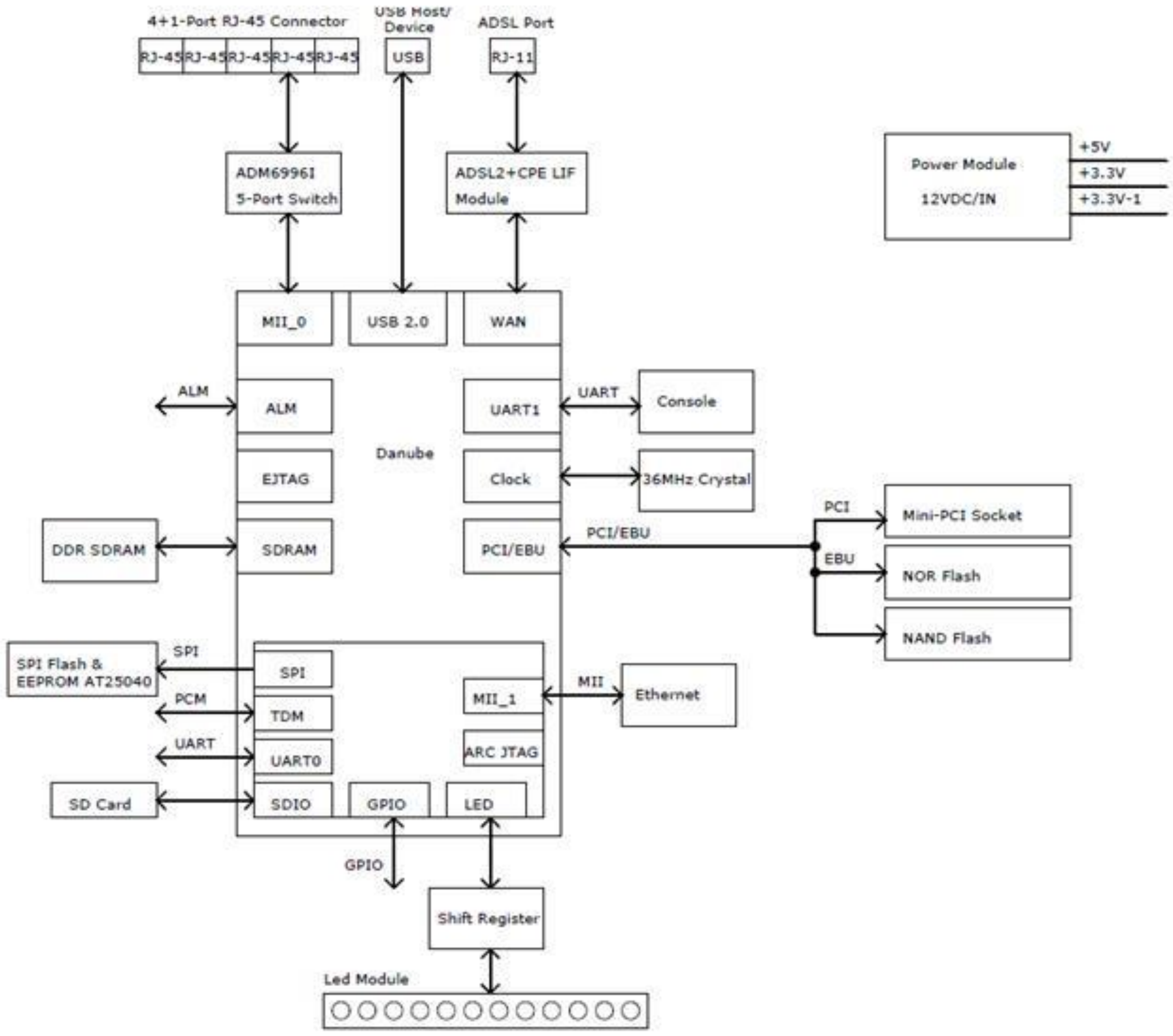

Figure 1 AMRG block diagram

2.1.2 Board Description

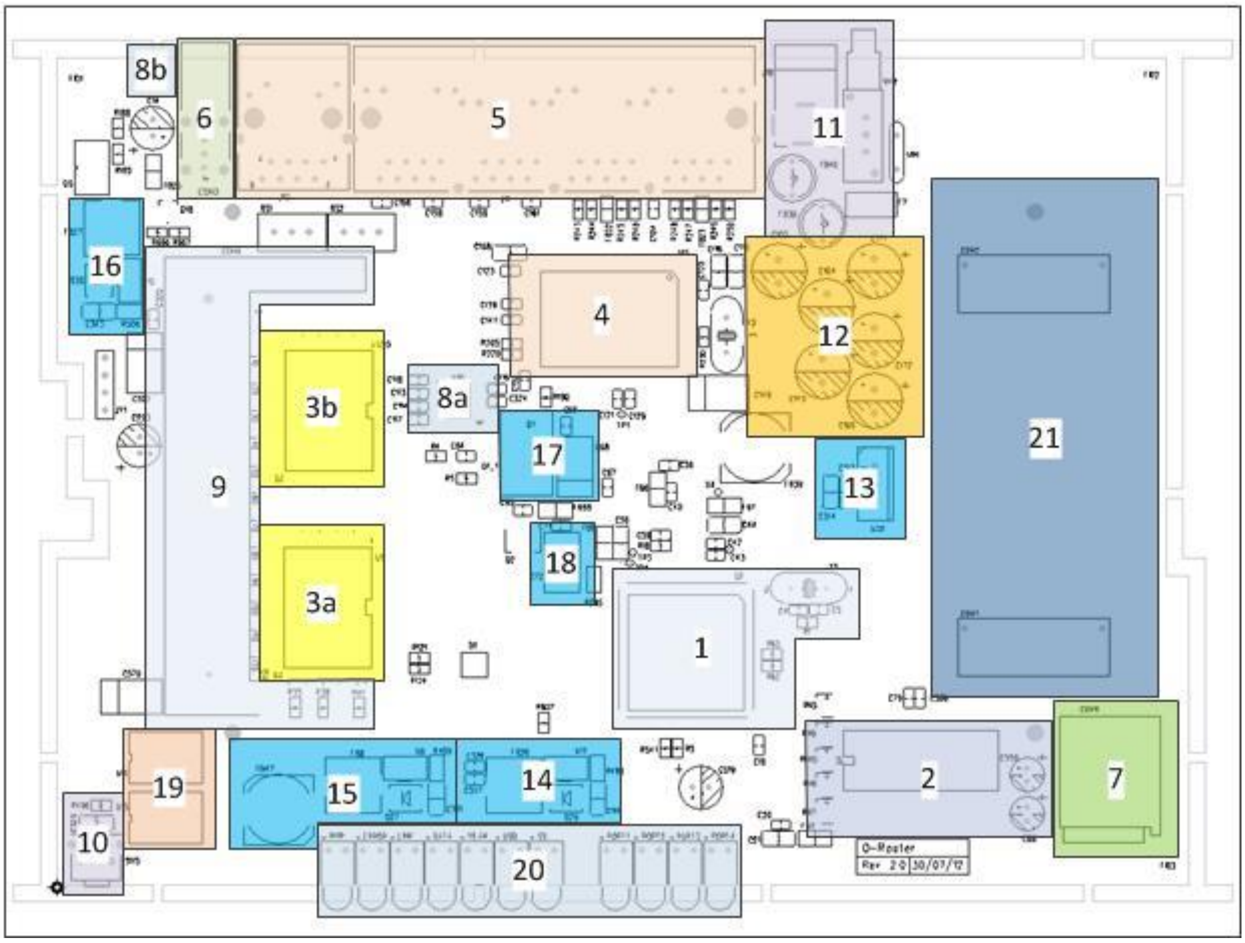

Figure 2: AMRG Board Top view

1. Danube ADSL2+ controller IC
2. DDR SDRAM
3. a. NOR Flash, b. NAND Flash
4. Samurai Ethernet controller IC
5. Gateway Ethernet ports
6. Peripheral USB Host port
7. Peripheral SD card slot
8. a. UART transceiver (2V1) or UART to USB converter, b. UART console connector (2V2)
9. Mini PCI extension
10. Reset button
11. Power jack and On/Off button
12. 12VDC power in and hold-up capacitors
13. 5V_A linear regulator and choke
14. 3.3V_BT switching regulator and choke

15. 3.3V switching regulator and choke
16. USB 5V_IN switching regulator and choke
17. Danube 1.5V transistor based regulator
18. Danube and DDR SDRAM 2.5V transistor based regulator
19. Control signals Shift registers
20. Front panel LEDs
21. ADSL Hybrid circuit add-on board connectors - Change

2.1.3 Equipment Overview

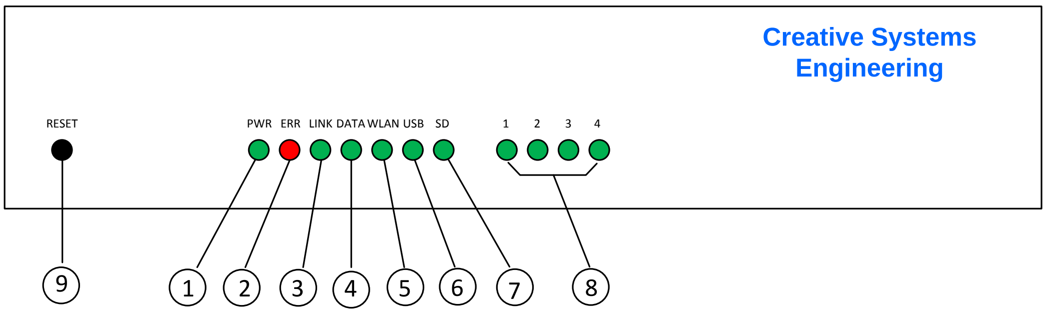

Figure 3: Front view

1. Power LED
2. Error LED
3. ADSL Link LED
4. LAN Data LED
5. Wireless LAN LED
6. USB LED
7. SD card LED (optional)
8. User defined LEDs
9. Reset button

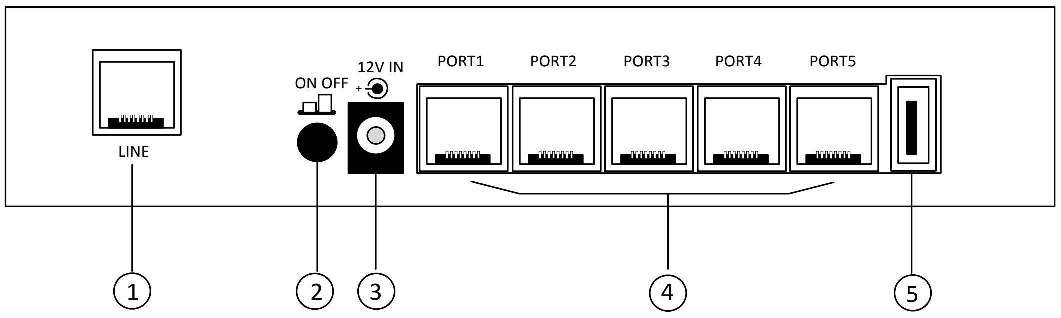

Figure 4: Rear view

1. ADSL line connector
2. On/Off button
3. Power input
4. Ethernet connectors

5. USB host connector

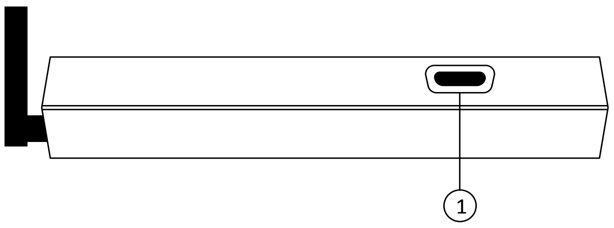

Figure 5: Side view with RS-232 connector

2.1.4 Add-on boards

2.1.4.1 ADSL Front End

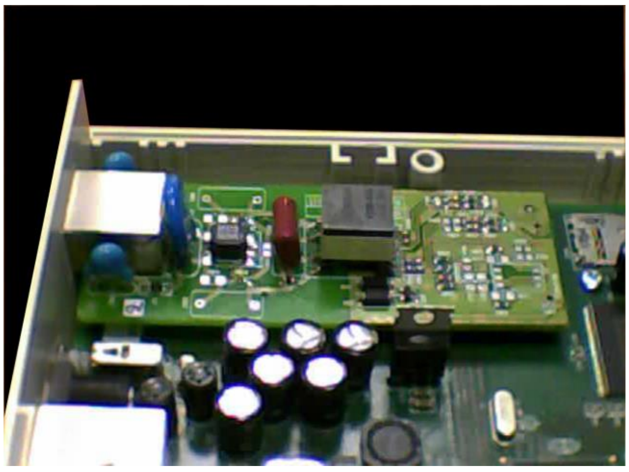

Figure 6: ADSL Front End Add-On board

The ADSL Front End circuits of the AMRG 2V0 are provided as an add-on module that is attached to the main board with two SMD connectors

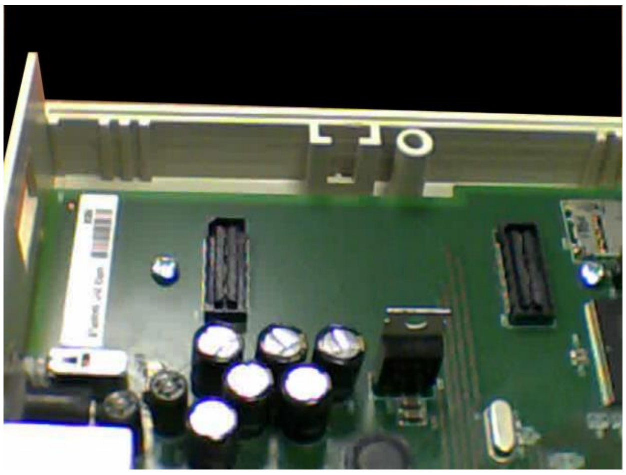

Figure 7: ADSL module connectors

2.1.4.2 WiFi board

The AMRG 2V0 is provided with a Winstron CM9 miniPCI WiFi card featuring the Atheros chipset. The miniPCI connector can be used to host a variety of WiFi cards, according to the user's needs.

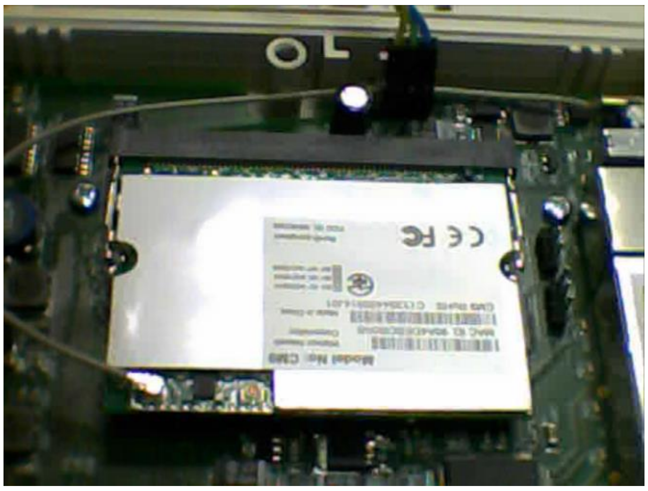

Figure 8: WiFi card in the miniPCI connector

2.1.5 Jumpers

The AMRG 2V0 main board provides the capability to alter the boot mode of the processor via two dedicated jumpers (BS1 and BS2):

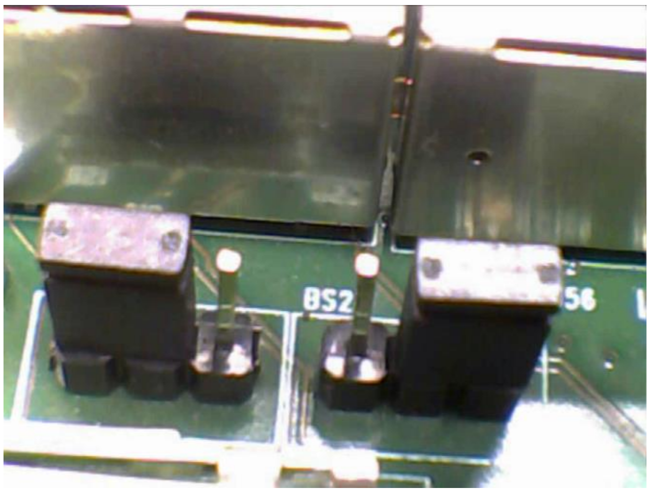

Figure 9: Boot mode jumpers placed for UART boot

The available options are:

Position	Boot function
BS1 and BS2 placed as in Figure 9	Boot from UART
BS1 and BS2 removed	Boot from Flash

Table 1: AMRG boot mode jumpers

3 LINUX KERNEL

The device comes with kernel Version 2.6.30

4 DEVELOPMENT ENVIRONMENT AND TOOLCHAIN

4.1 TOOLCHAIN OVERVIEW

AMRG is based on the Lantiq Danube PSB50702 V1.3 processor. The networking processor implemented in Danube is a MIPS24KeC core.

The Ethernet controller is Lantiq Samurai ADM6996I.

The toolchain that is used for compiling and building the firmware consists of the following packages:

- GCC 4.3.3
- Binutils 2.19.1
- uClibc 0.9.30.1
- Linux Kernel 2.6.30.10

The bootloader of the board is based on u-boot-2009.11.

The CPU is Big Endian.

4.2 FIRMWARE COMPILATION

The full image (bootloader, kernel and applications) deployed on AMRG is built by use of the OpenWRT development environment (https://openwrt.org) since the hardware configuration of the board is supported by the distribution. This eases the building and installation of the toolchain as well as the configuration of the operating system of the board.

The complete SW package as used in the AMRG 2V0 is available from support@creativese.eu

The version of the OpenWRT distribution that has been used is Backfire (10.03.1-RC5, r27450).

In the sequel, the setup and build instructions along with any modifications that were performed will be presented.

The environment of OpenWRT is available via svn:

```
svn checkout svn://svn.openwrt.org/openwrt/branches/backfire -r 27450
```

Initial configuration:

```
make menuconfig
```

and select:

```
Target System (Infineon Mips)
Target Profile (Atheros WiFi (default))
Boot loaders --> U-boot Lantiq --> Enable RAM boot image
```

execute make to have the toolchain compiled.

4.2.1 Modifications

Once the toolchain has been compiled the sources for the kenrel and applications have been downloaded as well. Navigate to backfire/package/uboot-lantiq folder, and replace in file backfire/package/uboot-lantiq/patches/100-ifx_targets.patch:

```
DDR=$(subst DDR,,$(filter DDR%,$(subst _, ,$@))); \}
```

with

```
DDR=111M; \
```

In file backfire/package/uboot-lantiq/files/include/configs/ifx-common.h (line 61) the environment variables of u-boot should be reefined as follows:

```
#define CONFIG_EXTRA_ENV_SETTINGS                    \
   "bootcmd=run flash_flash\0"\
   "bootdelay=5\0"\
   "baudrate=115200\0"\
   "loads_echo \0"\
   "preboot=echo;echo Type \"run flash_nfs\" to mount root filesystem over NFS;echo\0"\
   "ram_addr=0x80500000\0"\
   "kernel_addr=0xb0050000\0"\
   "mtdparts=mtdparts=ifx-nor:256k(uboot)ro,64k(uboot_env)ro,64k(kernel),-(rootfs)\0"\
   "flashargs=setenv bootargs rootfstype=squashfs,jffs2\0"\
   "nfsargs=setenv bootargs root=/dev/nfs rw nfsroot=${serverip}:${rootpath} \0"\
   "addip=setenv bootargs ${bootargs}
ip=${ipaddr}:${serverip}:${gatewayip}:${netmask}:${hostname}:${netdev}:off\0"\
   "addmisc=setenv bootargs ${bootargs} init=/etc/preinit console=ttyS1,115200
ethaddr=${ethaddr} ${mtdparts}\0"\
   "flash_flash=run flashargs addip addmisc;bootm ${kernel_addr}\0"\
   "flash_nfs=run nfsargs addip addmisc;bootm ${kernel_addr}\0"\
   "net_flash=run load_kernel flashargs addip addmisc;bootm ${ram_addr}\0"\
   "net_nfs='run load_kernel nfsargs addip addmisc;bootm ${ram_addr}\0"\
   "load_kernel=tftp ${ram_addr} ${tftppath}openwrt-ifxmips-uImage\0"\
   "update_uboot=tftp 0x80500000 ${tftppath}u-boot.bin;era 0xb0000000 +${filesize};cp.b
0x80500000 0xb0000000 ${filesize}\0"\
   "update_openwrt=tftp ${ram_addr} ${tftppath}openwrt-ifxmips-squashfs.image;era
${kernel_addr} +${filesize};cp.b ${ram_addr} ${kernel_addr} ${filesize}\0"\
   "ethact=lq_cpe_eth\0"\
   "ethaddr=X:X:X:X:X:X\0"\
   "rootpath=/opt/export/fs\0"\
   "filesize=30000\0"\
   "fileaddr=80500000\0"\
   "ipaddr=X.X.X.X\0"\
   "serverip=X.X.X.X\0"\
   "stdin=serial\0"\
   "stdout=serial\0"\
   "stderr=serial\0"}
```

Pay attention so that proper values for ethaddr, ipaddr, serverip are assigned.

In file build_dir/linux-ifxmips/linux-2.6.30.10/drivers/net/phy/adm6996.c, in the init function prefer the following settings:

```
/* initialize port and vlan settings */
w16(pdev, adm_portcfg[0], 0x840F);
w16(pdev, adm_portcfg[1], 0x840F);
```

```
w16(pdev, adm_portcfg[2], 0x840F);
w16(pdev, adm_portcfg[3], 0x840F);
w16(pdev, adm_portcfg[4], 0x840F);
w16(pdev, adm_portcfg[5], 0x840F);
w16(pdev, adm_portcfg[5]+8, 0xFF00);
```

instead of:

```
for (i = 0; i < ADM_PHY_PORTS; i++) {
w16(pdev, adm_portcfg[i], ADM_PORTCFG_INIT |
ADM_PORTCFG_PVID((i == ADM_WAN_PORT) ? 1 : 0));
}
w16(pdev, adm_portcfg[5], ADM_PORTCFG_CPU);
}
```

In file backfire/build_dir/linux-ifxmips/linux-2.6.30.10/drivers/mtd/chips/cfi_cmdset_0002.c apply the following modification as indicated by the following diff output.

```
1367c1367
< int z, words;
---
> int z, words, y
1405,1406c1405,1410
<
< map_write(map, datum, adr + z);
---
> #ifdef CONFIG_IFXMIPS
> y = adr+z;
> y ^= 2;
> #endif
> map_write(map, datum, y);
> //map_write(map, datum, adr + z);
```

4.3 FIRST BOOT

Select UART Boot as indicated by the jumper settings in Figure 9.

- Connect the USB-to-serial cable on the 9-pin connector
- Open (gtkterm, cutecom) /dev/ttyUSB0 at 115200-8n1 from Linux Host
- Power-up
- Send as raw file the ram bootloader located at
  backfire/bin/ifxmips/uboot-lantiq/easy50712_DDR166M/u-boot.asc
- Install a tftp server on a Linux host and make available the file
  backfire/bin/ifxmips/uboot-lantiq/easy50712_DDR166M/u-boot.bin
- From the u-boot environment execute (assuming that IP addresses for both linux host and board are properly set)
- protect off all
- run update_uboot

4.4 SECOND BOOT

Select flash-boot by removing the jumpers BS1 and BS2

- Power-up
- Press enter to interrupt boot process enter the u-boot environment
- Make available via tftp the file located at bin/ifxmips/openwrt-ifxmips-squashfs.image
- Execute: run update-openwrt
- Once the process complete execute: boot

4.5 LED MANAGEMENT

LEDs can be set on and off in Kernel mode by use of the functions exposed in:

```
#include <asm/mach-ifxmips/ifxmips_led.h>

void ifxmips_led_set(unsigned int led);
void ifxmips_led_clear(unsigned int led);
```

The argument in this functions is a 2 bytes value.

The following table indicates the values that can be safely used in the above functions and the labels of the associated LEDs.

Value	LED Label
0x0001	4
0x0002	SD
0x0004	USB
0x0008	3
0x1000	2
0x2000	1
0x4000	ERR
0x8000	WLAN

Table 2: LED management values

The remaining LEDs (PWR, LINK, DATA) are already managed by the aDSL firmware and the power circuit and therefore it is advised that no intervention is attempted.

4.6 USB AND NAND SUPPORT

USB and NAND drivers can be provided by CSE upon request.

4.7 JAVA SUPPORT

A version of PhoneME Java can be built and used on AMRG. The source is available via svn (account can be created at http://java.net/projects/phoneme):

```
svn co https://svn.java.net/svn/phoneme~svn/components/cdc/trunk cdc
svn co https://svn.java.net/svn/phoneme~svn/components/tools/trunk tools
```

create a script (build.sh) containing in the parent folder of the cdc and tools replacing the path prefix according to the toolchain location:

```
#!/bin/sh
make -C cdc/build/linux-mips-openwrt/ -f GNUmakefile \
J2ME_CLASSLIB=foundation \
CVM_JIT=true \
CVM_OPTIMIZED=true \
CVM_PRELOAD_LIB=true \
CVM_BUILD_SUBDIR_NAME=cdc-fp \
JDK_HOME={path to your java installation} \
CVM_TARGET_TOOLS_PREFIX={path-to-}/backfire/staging_dir/toolchain-mips_r2_gcc-
4.3.3+cs_uClibc-0.9.30.1/usr/bin/mips-openwrt-lin
```

Apply the following changes to file cdc/linux/javavm/runtime/globals_md.c

```
===================================================================
--- src/linux/javavm/runtime/globals_md.c (revision 20547)
+++ src/linux/javavm/runtime/globals_md.c (working copy)
@@ -185,11 +185,16 @@
linuxNetInit();
- sigignore(SIGPIPE);
+// sigignore(SIGPIPE);
+ struct sigaction ignore_action;
+ ignore_action.sa_handler = SIG_IGN;
+ ignore_action.sa_flags = SA_RESTART;
+ sigaction(SIGPIPE, &ignore_action, NULL);
#ifdef __VFP_FP__
/* TODO: Needed for armboard5. Should be moved to ARM specific code. */
- sigignore(SIGFPE);
+ //sigignore(SIGFPE);
+ sigaction(SIGFPE, &ignore_action, NULL);
#endif
{
```

Execute the build.sh script in the same folder.

Create a folder under backfire/packages (in the OpenWRT dev folder):

```
mkdir backfire/package/phoneme
```

create the following folder/files tree in backfire/packages/phoneme

- backfire/package/phoneme/files/usr
- backfire/package/phoneme/files/usr/java
- backfire/package/phoneme/files/usr/java/cdc
- backfire/package/phoneme/files/usr/java/cdc/testclasses.zip
- backfire/package/phoneme/files/usr/java/cdc/democlasses.jar
- backfire/package/phoneme/files/usr/java/cdc/bin
- backfire/package/phoneme/files/usr/java/cdc/bin/cvm

- backfire/package/phoneme/files/usr/java/cdc/lib

from the files located at the output folder of the phoneME compilation:
cdc/build/linux-mips-openwrt/cdc-fp
Create a Makefile in backfire/packages/phoneme containing:

```
#
# Copyright (C) 2006-2010 OpenWrt.org
#
# This is free software, licensed under the GNU General Public License v2.
# See /LICENSE for more information.
#
include $(TOPDIR)/rules.mk
PKG_NAME:=phoneme
PKG_VERSION:=1.1.2
PKG_RELEASE:=4
#PKG_BUILD_DIR:=$(BUILD_DIR)/$(PKG_NAME)
include $(INCLUDE_DIR)/package.mk
define Package/phoneme
SUBMENU:=Java
SECTION:=lang
CATEGORY:=Languages
TITLE:=A compact Java Virtual Machine
DEPENDS:=+libpthread
endef
define Package/phoneME/description
phoneME is an open source project based on Java Micro Edition (Java ME) \
technology. phoneME Advanced is a phoneME project based on Java ME CDC \
technology targeting resource-constrained devices like smartphones, \
set-top boxes and office equipment. See https://phoneme.dev.java.net \
and http://java.sun.com/products/cdc
endef
define Build/Compile
endef
define Package/phoneme/install
$(INSTALL_DIR) $(1)/usr/java/cdc
$(CP) files/usr/java/cdc $(1)/usr/java
endef
$(eval $(call BuildPackage,phoneme))
```

In this way the pre-built phoneME can be selected from menuconfig so that it can be included in the next image. Otherwise an ipk package can be created by executing: `make package/phoneme/install`, that will result in a file

bin/ifxmips/packages/phoneme_1.1.2-4_ifxmips.ipk

that can be uploaded and installed independently on the board by the opkg manager.

However, since the JVM is a relatively large file, it is suggested that the NAND flash is used for storing the JVM executable (cvm).

4.8 AVAILABLE SOFTWARE (LINKS TO OPENWRT)

The operating system and the applications on the MRG can be configured via the **make menuconfig** console of the OpenWRT environment. There is a default configuration containing a wide range of the usual Linux utilities, tools and libraries. Additional packages can be selected to be built as part of an upgrade image or independently and the packages to be installed via the opkg manager.

Moreover, custom software can be compiled and prepared for distribution and installation via the package cross compilation option of the OpenWRT environment:

http://wiki.openwrt.org/doc/packages).

4.9 CREATING PACKAGES

A typical package directory contains two things:

- package/Makefile
- package/patches

The patches directory is optional and typically contains bug fixes or optimizations to reduce the size of the executable.

The package makefile is the important item because it provides the steps actually needed to download and compile the package.

Here for example, is package/bridge/Makefile:

```
include $(TOPDIR)/rules.mk

        PKG_NAME:=bridge
        PKG_VERSION:=1.0.6
        PKG_RELEASE:=1

        PKG_BUILD_DIR:=$(BUILD_DIR)/bridge-utils-$(PKG_VERSION)
        PKG_SOURCE:=bridge-utils-$(PKG_VERSION).tar.gz
        PKG_SOURCE_URL:=@SF/bridge
        PKG_MD5SUM:=9b7dc52656f5cbec846a7ba3299f73bd
        PKG_CAT:=zcat

        include $(INCLUDE_DIR)/package.mk

        define Package/bridge
          SECTION:=base
          CATEGORY:=Network
          DEFAULT:=y
          TITLE:=Ethernet bridging configuration utility
          #DESCRIPTION:=This variable is obsolete. use the Package/name/description define
        instead!
          URL:=http://bridge.sourceforge.net/
        endef
```

```
define Package/bridge/description
 Ethernet bridging configuration utility
 Manage ethernet bridging; a way to connect networks together to
 form a larger network.
endef

define Build/Configure
  $(call Build/Configure/Default,--with-linux-headers=$(LINUX_DIR))
endef

define Package/bridge/install
	$(INSTALL_DIR) $(1)/usr/sbin
	$(INSTALL_BIN) $(PKG_BUILD_DIR)/brctl/brctl $(1)/usr/sbin/
endef

$(eval $(call BuildPackage,bridge))
```

4.10 BUILDPACKAGE VARIABLES

- PKG_NAME - The name of the package, as seen via menuconfig and ipkg
- PKG_VERSION - The upstream version number that we're downloading
- PKG_RELEASE - The version of this package Makefile
- PKG_BUILD_DIR - Where to compile the package
- PKG_SOURCE - The filename of the original sources
- PKG_SOURCE_URL - Where to download the sources from
- PKG_MD5SUM - A checksum to validate the download
- PKG_CAT - How to decompress the sources (zcat, bzcat, unzip)
- PKG_BUILD_DEPENDS - Packages that need to be built before this package, but are not required at runtime. Uses the same syntax as DEPENDS below.
- PKG_INSTALL - Setting it to "1" will call the package's original "make install" with prefix set to PKG_INSTALL_DIR
- PKG_INSTALL_DIR - Where "make install" copies the compiled files

The PKG_* variables define where to download the package from; @SF is a special keyword for downloading packages from sourceforge. The md5sum is used to verify the package was downloaded correctly and PKG_BUILD_DIR defines where to find the package after the sources are uncompressed into $(BUILD_DIR). PKG_INSTALL_DIR defines where the files will be copied after calling "make install" (set with the PKG_INSTALL variable).

"BuildPackage" is a macro setup by the earlier include statements. BuildPackage only takes one argument directly – the name of the package to be built, in this case "bridge". All other information is taken from the define blocks. This is a way of providing a level of verbosity, it's inherently clear what the DESCRIPTION variable in Package/bridge is, which wouldn't be the case if we passed this information directly as the Nth argument to BuildPackage.

4.11 BUILDPACKAGE DEFINES

Package/

matches the argument passed to buildroot, this describes the package the menuconfig and ipkg entries. Within Package/ one can define the following variables:

- SECTION - The type of package (currently unused)
- CATEGORY - Which menu it appears in menuconfig
- TITLE - A short description of the package
- DESCRIPTION - (deprecated) A long description of the package
- URL - Where to find the original software
- MAINTAINER - (optional) Who to contact concerning the package
- DEPENDS - (optional) Which packages must be built/installed before this package. See “es Dependency Types” below for the syntax.

Package/conffiles (optional)

A list of config files installed by this package, one file per line.

Package/description

A free text description of the package

Build/Prepare (optional)

A set of commands to unpack and patch the sources. You may safely leave this undefined.

Build/Configure (optional)

You can leave this undefined if the source doesn't use configure or has a normal config script, otherwise you can put your own commands here or use "$(call Build/Configure/Default,)" as above to pass in additional arguments for a standard configure script.

Build/Compile (optional)

How to compile the source; in most cases you should leave this undefined.

Package/install

A set of commands to copy files out of the compiled source and into the ipkg which is represented by the $(1) directory.

Package/preinst

The actual text of the script which is to be executed before installation. Don’t forget to include the #!/bin/sh. If you need to abort installation have the script return false.

Package/postinst

The actual text of the script which is to be executed after installation. Don’t forget to include the #!/bin/sh.

Package/prerm

The actual text of the script which is to be executed before removal. Don’t forget to include the #!/bin/sh. If you need to abort removal have the script return false.

Package/postrm

The actual text of the script which is to be executed after removal. Don't forget to include the #!/bin/sh.

4.12 DEPENDENCY TYPES

Various types of dependencies can be specified, which require a bit of explanation for their differences.

Type	Description
+<foo>	Package will depend on package <foo> and will select it when selected.
<foo>	Package will depend on package <foo> and will be invisible until <foo> is selected.
@FOO	Package depends on the config symbol CONFIG_FOO and will be invisible unless CONFIG_FOO is set. This usually used for depending on certain Linux versions or targets, e.g. @TARGET_foo will make a package only available for target foo. You can also use boolean expressions for complex dependencies, e.g. @(!TARGET_foo&&!TARGET_bar) will make the package unavailable for foo and bar.
+FOO:<bar>	Package will depend on <bar> if CONFIG_FOO is set, and will select <bar> when it is selected itself. The typical use case would be if there compile time options for this package toggling features that depend on external libraries. Note that the + replaces the @.
@FOO:<bar>	Package will depend on <bar> if CONFIG_FOO is set, and will be invisible until <bar> is selected when CONFIG_FOO is set.

Some typical config symbols for (conditional) dependencies are:

Symbol	Description
TARGET_<foo>	Target <foo> is selected
TARGET_<foo>_<bar>	If the target <foo> has subtargets, subtarget <foo> is selected. If not, profile <foo> is selected. This is in addition to TARGET_<foo>
TARGET_<foo>_<bar>_<baz>	Target <foo> with subtarget <bar> and profile <baz> is selected.
LINUX_3_X	Linux version used is 3.x
LINUX_2_6_X	Linux version used is 2.6.x.* (:1: only used for backfire and earlier)
LINUX_2_4	Linux version is 2.4 (only used in backfire and earlier, and only for target brcm-2.4)
USE_UCLIBC, USE_GLIBC, USE_EGLIBC	To (not) depend on a certain libc
BROKEN	Package doesn't build or work, and should only be visible if "Show broken targets/packages" is selected. Prevents the package from failing builds by accidentally selecting it
IPV6	IPv6 support in packages is selected

NOTES

All variables in your pre/post install/removal scripts should have double ($$) instead of a single ($) string characters. This will inform "make" to not interpret the value as a variable, but rather just ignore the string and replace the double $$ by a single $.

After you've created your package Makefile, the new package will automatically show in the menu the next time you run "make menuconfig" and if selected will be built automatically the next time "make" is run.

DESCRIPTION is obsolete, use Package/PKG_NAME/description.

4.13 ADDING CONFIGURATION OPTIONS

If you would like configure your package installation/compilation in the menuconfig you can do the following: Add MENU:=1 to your package definition like this:

```
define Package/mjpg-streamer
  SECTION:=multimedia
  CATEGORY:=Multimedia
  TITLE:=MJPG-streamer
  DEPENDS:=@!LINUX_2_4 +libpthread-stubs +jpeg
  URL:=http://mjpg-streamer.wiki.sourceforge.net/
  MENU:=1
endef
```

Create a config key in the Makefile:

```
define Package/mjpg-streamer/config
        source "$(SOURCE)/Config.in"
endef
```

Create a Config.in file directory where the Makefile is located with the content like this:

```
# Mjpg-streamer configuration
menu "Configuration"
depends on PACKAGE_mjpg-streamer

config MJPEG_STREAMER_AUTOSTART
      bool "Autostart enabled"
      default n

      menu "Input plugins"
      depends on PACKAGE_mjpg-streamer
      config MJPEG_STREAMER_INPUT_FILE
            bool "File input plugin"
      help
      You can stream pictures from jpg files on the filesystem
                        default n
```

```
        config MJPEG_STREAMER_INPUT_UVC
                bool "UVC input plugin"
        help
        You can stream pictures from an Universal Video Class compatible webcamera
                        default y

        config MJPEG_STREAMER_FPS
        depends MJPEG_STREAMER_INPUT_UVC
                     int "Maximum FPS"
                        default 15

        config MJPEG_STREAMER_PICT_HEIGHT
        depends MJPEG_STREAMER_INPUT_UVC
                      int "Picture height"
                        default 640

        config MJPEG_STREAMER_PICT_WIDTH
        depends MJPEG_STREAMER_INPUT_UVC
                       int "Picture width"
                        default 480

        config MJPEG_STREAMER_DEVICE
        depends MJPEG_STREAMER_INPUT_UVC
                        string "Device"
                     default /dev/video0

        config MJPEG_STREAMER_INPUT_GSPCA
            bool "GSPCA input plugin"
        help
        You can stream pictures from a gspca supported webcamera Note this module is
        deprecated, use the UVVC plugin instead
                                  default n
        endmenu
endmenu
```

Above you can see examples for various type config parameters.

And finally you can check your configuration parameters in your Makefile in the following way: (Note that you can reference to the parameters value with it name prefixed with CONFIG_)

```
ifeq ($(CONFIG_MJPEG_STREAMER_INPUT_UVC),y)
$(CP) $(PKG_BUILD_DIR)/input_uvc.so $(1)/usr/lib
endif
```

4.14 CREATING PACKAGES FOR KERNEL MODULES

One can also add kernel modules which are not part of the linux source distribution. In this case, a kernel module appears in the package/ directory, just as any other package does. The package/Makefile uses KernelPackage/xxx definitions in place of Package/xxx

For example, here is package/madwifi/Makefile:

```
#
# Copyright (C) 2006 OpenWrt.org
#
# This is free software, licensed under the GNU General Public License v2.
# See /LICENSE for more information.
#
# $Id$

include $(TOPDIR)/rules.mk
include $(INCLUDE_DIR)/kernel.mk

PKG_NAME:=madwifi
PKG_VERSION:=0.9.2
PKG_RELEASE:=1

PKG_SOURCE:=$(PKG_NAME)-$(PKG_VERSION).tar.bz2
PKG_SOURCE_URL:=@SF/$(PKG_NAME)
PKG_MD5SUM:=a75baacbe07085ddc5cb28e1fb43edbb
PKG_CAT:=bzcat

PKG_BUILD_DIR:=$(KERNEL_BUILD_DIR)/$(PKG_NAME)-$(PKG_VERSION)

include $(INCLUDE_DIR)/package.mk

RATE_CONTROL:=sample

ifeq ($(ARCH),mips)
  HAL_TARGET:=mips-be-elf
endif
ifeq ($(ARCH),mipsel)
  HAL_TARGET:=mips-le-elf
endif
ifeq ($(ARCH),i386)
  HAL_TARGET:=i386-elf
endif
ifeq ($(ARCH),armeb)
  HAL_TARGET:=xscale-be-elf
```

```
endif
ifeq ($(ARCH),powerpc)
  HAL_TARGET:=powerpc-be-elf
endif

BUS:=PCI
ifneq ($(CONFIG_LINUX_2_4_AR531X),)
  BUS:=AHB
endif
ifneq ($(CONFIG_LINUX_2_6_ARUBA),)
  BUS:=PCI AHB# no suitable HAL for AHB yet.
endif

BUS_MODULES:=
ifeq ($(findstring AHB,$(BUS)),AHB)
  BUS_MODULES+=$(PKG_BUILD_DIR)/ath/ath_ahb.$(LINUX_KMOD_SUFFIX)
endif
ifeq ($(findstring PCI,$(BUS)),PCI)
  BUS_MODULES+=$(PKG_BUILD_DIR)/ath/ath_pci.$(LINUX_KMOD_SUFFIX)
endif

MADWIFI_AUTOLOAD:= \
          wlan \
          wlan_scan_ap \
          wlan_scan_sta \
          ath_hal \
       ath_rate_$(RATE_CONTROL) \
          wlan_acl \
          wlan_ccmp \
          wlan_tkip \
          wlan_wep \
          wlan_xauth

ifeq ($(findstring AHB,$(BUS)),AHB)
  MADWIFI_AUTOLOAD += ath_ahb
endif
ifeq ($(findstring PCI,$(BUS)),PCI)
   MADWIFI_AUTOLOAD += ath_pci
endif

define KernelPackage/madwifi
  SUBMENU:=Wireless Drivers
  DEFAULT:=y if LINUX_2_6_BRCM | LINUX_2_6_ARUBA | LINUX_2_4_AR531X |
LINUX_2_6_XSCALE, m if ALL
```

```
  TITLE:=Driver for Atheros wireless chipsets
  DESCRIPTION:=\
This package contains a driver for Atheros 802.11a/b/g chipsets.
  URL:=http://madwifi.org/
  VERSION:=$(LINUX_VERSION)+$(PKG_VERSION)-$(BOARD)-$(PKG_RELEASE)
  FILES:= \
      $(PKG_BUILD_DIR)/ath/ath_hal.$(LINUX_KMOD_SUFFIX) \
                  $(BUS_MODULES) \

$(PKG_BUILD_DIR)/ath_rate/$(RATE_CONTROL)/ath_rate_$(RATE_CONTROL).$(LINU
X_KMOD_SUFFIX) \

         $(PKG_BUILD_DIR)/net80211/wlan*.$(LINUX_KMOD_SUFFIX)
  AUTOLOAD:=$(call AutoLoad,50,$(MADWIFI_AUTOLOAD))
endef

MADWIFI_MAKEOPTS= -C $(PKG_BUILD_DIR) \
PATH="$(TARGET_PATH)" \
ARCH="$(LINUX_KARCH)" \
CROSS_COMPILE="$(TARGET_CROSS)" \
TARGET="$(HAL_TARGET)" \
TOOLPREFIX="$(KERNEL_CROSS)" \
TOOLPATH="$(KERNEL_CROSS)" \
KERNELPATH="$(LINUX_DIR)" \
LDOPTS=" " \
ATH_RATE="ath_rate/$(RATE_CONTROL)" \
DOMULTI=1

ifeq ($(findstring AHB,$(BUS)),AHB)
  define Build/Compile/ahb
$(MAKE) $(MADWIFI_MAKEOPTS) BUS="AHB" all
  endef
endif

ifeq ($(findstring PCI,$(BUS)),PCI)
  define Build/Compile/pci
$(MAKE) $(MADWIFI_MAKEOPTS) BUS="PCI" all
  endef
endif

define Build/Compile
          $(call Build/Compile/ahb)
          $(call Build/Compile/pci)
endef
```

```
define Build/InstallDev
$(INSTALL_DIR) $(STAGING_DIR)/usr/include/madwifi
$(CP) $(PKG_BUILD_DIR)/include $(STAGING_DIR)/usr/include/madwifi/
$(INSTALL_DIR) $(STAGING_DIR)/usr/include/madwifi/net80211
$(CP) $(PKG_BUILD_DIR)/net80211/*.h $(STAGING_DIR)/usr/include/madwifi/net80211/
endef

define KernelPackage/madwifi/install
        $(INSTALL_DIR) $(1)/etc/init.d
        $(INSTALL_DIR) $(1)/lib/modules/$(LINUX_VERSION)
        $(INSTALL_DIR) $(1)/usr/sbin
        $(INSTALL_BIN) ./files/madwifi.init $(1)/etc/init.d/madwifi
        $(CP)
$(PKG_BUILD_DIR)/tools/{madwifi_multi,80211debug,80211stats,athchans,athctrl,athdeb
ug,athkey,athstats,wlanconfig} $(1)/usr/sbin/
endef

$(eval $(call KernelPackage,madwifi))
```

INSTALL_DIR, INSTALL_BIN, INSTALL_DATA are used for creating a directory, copying an executable, or a data file. +x is set on the target file for INSTALL_BIN, independent of its mode on the host.

Package/<name>/install:

A set of commands to copy files out of the compiled source and into the ipkg which is represented by the $(1) directory. Note that there are currently 4 defined install macros:

```
INSTALL_DIR
install -d -m0755
INSTALL_BIN
install -m0755
INSTALL_DATA
install -m0644
INSTALL_CONF
install -m0600
```

4.15 PACKAGING A SERVICE

If you want to install a service, (something that should start/stop at boot time, that has a /etc/init.d/blah script), you should make sure that the init.d script can be run on the host. At image build time, all init.d scripts found are run on the host, looking for the START=20/STOP=99 lines.

This is what installs the symlinks in /etc/rc.d, so they are only created when you rebuild the entire image. If you want the symlinks to be created when a package is installed, such as via opkg, you should add a postinstall script which runs

```
/etc/init.d/foo enable
```

if $IPKG_INSTROOT is empty. when $IPKG_INSTROOT is defined, you run within the buildroot, if it is empty you run on the target.

Example makefile snippet to install/remove symlinks.

```
define Package/mrelay/postinst
#!/bin/sh
# check if we are on real system
if [ -z "$${IPKG_INSTROOT}" ]; then
	echo "Enabling rc.d symlink for mrelay"
	/etc/init.d/mrelay enable
fi
exit 0
endef

define Package/mrelay/prerm
#!/bin/sh
# check if we are on real system
if [ -z "$${IPKG_INSTROOT}" ]; then
	echo "Removing rc.d symlink for mrelay"
	/etc/init.d/mrelay disable
fi
exit 0
endef
```

Very basic example of a suitable init.d script

```
#!/bin/sh /etc/rc.common

START=80
APP=mrelay
PID_FILE=/var/run/$APP.pid

start() {
	start-stop-daemon -S -x $APP -p $PID_FILE -m -b
}

stop() {
	start-stop-daemon -K -n $APP -p $PID_FILE -s TERM
	rm -rf $PID_FILE
}
```