

BLUEGIGA APX4 SOFTWARE DEVELOPMENT KIT

USER GUIDE

Wednesday, 10 April 2013

Version 5.0.0-beta10



Copyright © 2001 - 2012 Bluegiga Technologies

Bluegiga Technologies reserves the right to alter the hardware, software, and/or specifications detailed herein at any time without notice, and does not make any commitment to update the information contained herein. Bluegiga Technologies assumes no responsibility for any errors which may appear in this manual. Bluegiga Technologies' products are not authorized for use as critical components in life support devices or systems.

Bluegiga Access Server, Access Point, APx4, AX4, BSM, iWRAP, BGScript and WRAP THOR are trademarks of Bluegiga Technologies.

The *Bluetooth* trademark and logo are registered trademarks and are owned by the Bluetooth SIG, Inc.

ARM and ARM9 are trademarks of ARM Ltd.

Linux is a trademark of Linus Torvalds.

All other trademarks listed herein belong to their respective owners.

TABLE OF CONTENTS

1. Introduction	4
1.1 Glossary	4
1.2 SDK directory layout	5
1.3 Meta layers	5
1.4 Other interesting links	6
1.5 Sources for recipes	6
2. Getting started	7
2.1 System requirements	7
2.2 Installing the SDK	8
2.2.1 Step 1: Install required packages	8
2.2.2 Step 2: Extract the SDK package	8
2.2.3 Step 3: Start building	8
2.2.4 Step 4: SDK installation completed	10
2.2.5 Step 5: Copy Re-flash package to target	10
2.2.6 Troubleshooting	10
3. Compiling and installing Hello World	11
3.1 Simple own Hello World application	11
3.2 Autotools-based Hello World application	12
3.3 More information	12
4. Customizing file system image to include additional programs	13
5. Configuring Linux kernel	14
5.1 Enabling features or drivers inside the kernel	14
5.2 Compiling kernel	14
5.3 Building external kernel module	14
5.4 Configuring LCD display	
6. Modifying the software to support your own motherboard	15
6.1 Linux kernel	15
6.1.1 GPIO	15
6.1.2 I2C	18
6.2 Libraries	
7. Eclipse support	19
7.1 Setting up the development environment	19
7.1.1 Step 1 - Install the Bluegiga SDK	19
7.1.2 Step 2 - Prepare IDE support	19
7.1.3 Step 3 - Install the Eclipse Juno IDE	19
7.1.4 Step 4 - Install required plugins	19
7.1.5 Step 5 - Configure the Eclipse Yocto Plugin	20
7.2 Creating a new project	20
7.3 Building and deploying the project	20
7.4 Debugging the project	21
8. Contact information	22

1 Introduction

Bluegiga SDK is based on Yocto project build system. Yocto offers powerful tools for cross compiling and debugging embedded applications and is also used to prepare custom distributions, like the one used as the default for the APx4. Yocto build system is very flexible and it is delivered with hundreds of open source applications.

1.1 Glossary

Bitbake is the application cooker. It handles package dependencies and executes build commands.

Recipe contains building instructions for bitbake. It has information about url of source code package, rules of compiling and package dependencies. All applications have their own recipes.

Layer is a Yocto term to indicate a set of recipes and configuration files. Recipes are divided into different layers such as board support and distribution meta layers.

.bpk indicates Bluegiga package format. It can contain complete filesystem images for the APx4 or scripts that are run when the package is installed.

Reflash is a term used in the name of special *.bpk* packages containing full file system images. A **reflash*.bpk* can be installed using the web interface or by other ways, for example by uploading it to the **/tmp/autoinstall** directory.

.ipk is a package format used by the OPKG package management system.

1.2 SDK directory layout

+-bluegiga-sdk	
+-build	build folder, bitbake command is executed in this folder
+-conf	
+-local.conf	local configuration file, e.g. target machine is selected here
+-bblayers.conf	bitbake layer config file, meta layers are selected here
+-tmp	
+-deploy	
+-images	file system images and u-boot binaries are here after building
+-ipk	location for ipk packages, packages are divided into subfolders
+-all	
+-armv5te	
+-apx4devkit	
+-work	work directory where applications are patched, built and packaged
+-conf	templates for local.conf and bblayers.conf files
+-downloads	source file download cache
+-sstate-cache	sstate cache (pre-compiled binaries)
+-images	location for pre-built file system images and u-boot binaries
+-meta	meta layers for Bluegiga SDK
+-bluegiga-bsp	layer which provides apx4 hardware support
+-bluegiga-distro	layer containing open source applications
+-bluegiga-binary-utils	layer with Bluegiga closed source applications
+-bluegiga-examples	layer that contains example applications
+-yocto	location for Yocto meta layers and bitbake command

1.3 Meta layers

Bluegiga SDK contains following meta layers.

Layer name	Description
yocto/meta	Recipes from Yocto
meta/bluegiga-bsp	Bluegiga hardware Board Support Package (BSP) for apx4devkit
meta/bluegiga-distro	Bluegiga Linux distribution related recipes (add-on software on top of Yocto base installation)
meta/bluegiga-binary-utils	Bluegiga utils and helper applications
meta/bluegiga-examples	Bluegiga example applications

Yocto is base of whole SDK and all other layers are depending on it. Following figure shows all layers and their dependencies.



1.4 Other interesting links

- Yocto documentation: <http://www.yoctoproject.org/documentation>
- New Developer Screencast Tutorial by Yocto: <http://www.yoctoproject.org/blogs/jeff/2012/new-yocto-project-introductory-screencast>
- Frequently asked questions: <https://wiki.yoctoproject.org/wiki/FAQ>

1.5 Sources for recipes

If you want to use an application but did find it from the prebuild packages, please look for below source of recipes. The sources are listed in an order where the recipes found from the first ones are most likely to compile without modifications. The recipes found from the links that are further down are more likely to require some tuning before they work.

1. oe-core, yocto is fetching data from this repository:

<http://git.openembedded.org/openembedded-core/tree/>

2. (these should be fully compatible with latest yocto)

<http://git.openembedded.org/meta-openembedded/tree/>

3. (tens of community driven meta layers)

<http://www.openembedded.org/wiki/LayerIndex>

4. (deprecated openembedded classic, these recipes are not fully compatible)

<http://git.openembedded.org/openembedded/tree/>

2 Getting started

Bluegiga SDK is tested to work with Ubuntu 12.04 LTS. Separate SDK packages are delivered for 32-bit and 64-bit host operating systems. (64-bit version not yet available at the time of writing, but 32-bit SDK can be used in 64-bit OS installations.)

2.1 System requirements

- Ubuntu 12.04 LTS (other Ubuntu distributions should work as well)
- 10 GB of free hard disk space
- 2 GB of RAM
- Internet access

In order to check the Ubuntu version, use the following command:

```
~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description: Ubuntu 12.04 LTS
Release: 12.04
Codename: precise
```

In order to check whether you have 32-bit or 64-bit OS, use the following command:

```
~$ uname -a
Linux csteam-ThinkPad-Z60m 3.0.0-16-generic #29-Ubuntu SMP Tue Feb 14 12:49:42
UTC 2012 i686 i686 i386 GNU/Linux
```


And finally in order to check the amount of free disk space and RAM, use the following commands:

```
~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1       90G   9.8G   76G  12% /
udev            996M   4.0K  996M   1% /dev
tmpfs           401M   788K  401M   1% /run
none            5.0M     0   5.0M   0% /run/lock
none            1003M  200K  1003M   1% /run/shm

~$ free -m
              total        used        free      shared    buffers     cached
Mem:           7985         5352         2633           0          152         1346
-/+ buffers/cache:      3852         4133
Swap:          8188           0         8188
~$
```

2.2 Installing the SDK

The SDK installation is a fairly straightforward process and requires only a few steps. This section walks you through the SDK installation and compilation of your first Re-flash packet.


 System wide root (or sudo) access is required to do some of the installation steps.

2.2.1 Step 1: Install required packages

Bluegiga SDK requires some additional packages in order to work properly. Packages can be installed running the following commands.

```
$ sudo apt-get update
$ sudo apt-get install sed wget cvs subversion git-core coreutils \
  unzip texi2html texinfo libsdl1.2-dev docbook-utils gawk \
  python-pysqlite2 diffstat help2man make gcc build-essential \
  g++ desktop-file-utils chrpath libgl1-mesa-dev libglu1-mesa-dev \
  mercurial autoconf automake groff
```


The command above will require the download of about 550-650MB (depending on the Ubuntu installation) from the Internet and this will take a while depending on your Internet connection speed.

 You will be asked if you want to download the packages from the Internet and you need to answer "Yes" to the question.

2.2.2 Step 2: Extract the SDK package

After the compressed archive containing the SDK files is received from Bluegiga and is copied to the local file system, you will have to extract its contents into your home directory by running the following commands:

```
$ cd ~/
$ tar xvf /path/to/bluegiga-sdk-sstate_5.0.0-beta2_apx4devkit.tgz
```

 Before uncompressing the archive, make sure once again that there is enough space under your home directory with command `df -h`

2.2.3 Step 3: Start building

First you need to prepare the necessary environment variables and create a build folder. You can do this by running the script **source init-sdk-env** as seen in the example below (after this command is executed, the current directory will be changed to `~/bluegiga-sdk/build` which is the directory where bitbake must be run always.)

Then you can experiment building your first reflash package with Yocto's command called **bitbake**. Example ready target recipes are `bluegiga-console-image` (to obtain a full installable filesystem, like the one installed by default in APx4s, using recipes from all meta layers as from graph in chapter 1.3) or `bluegiga-base-image` (to obtain a minimal distribution spanning only meta layers Yocto, Bluegiga BSP and a reduced Bluegiga Distro.) By

default bitbake will search for the given recipe in all meta layers directories defined in `~/bluegiga-sdk/build/conf/bblayers.conf` and for example the `bluegiga-console-image` happens to be under `~/bluegiga-sdk/meta/bluegiga-distro/recipes-image/bluegiga/`



Building a Reflash package should take less than 15 minutes with a quad core CPU with 4 GB of RAM, but please keep in mind that it might take a lot longer with older machines.

Example:

```
$ cd ~/bluegiga-sdk
$ source init-sdk-env
$ bitbake bluegiga-console-image
```

A successful build operation will deliver the following output:

```
Pseudo is not present but is required, building this first before the main
build
Parsing recipes: 100% |#####| Time:
00:00:53
Parsing of 883 .bb files complete (0 cached, 883 parsed). 1162 targets, 38
skipped, 0 masked, 0 errors.

...

NOTE: package bluegiga-console-image-1.0-r0: task do_rootfs: Started
NOTE: package bluegiga-console-image-1.0-r0: task do_rootfs: Succeeded
NOTE: Running task 1977 of 1978 (ID: 8, /tmp/sdk/bluegiga-sdk/yocto/./meta/
bluegiga-distro/recipes-reflash/bluegiga/bluegiga-console-reflash.bb,
do_build_reflash)
NOTE: package bluegiga-console-reflash-1.0-r0: task do_build_reflash: Started
NOTE: package bluegiga-console-reflash-1.0-r0: task do_build_reflash:
Succeeded
NOTE: Running noexec task 1978 of 1978 (ID: 5,
/tmp/sdk/bluegiga-sdk/yocto/./meta/
bluegiga-distro/recipes-reflash/bluegiga/bluegiga-console-reflash.bb,
do_build)
NOTE: Tasks Summary: Attempted 1978 tasks of which 1910 didn't need to be
rerun and all succeeded.
```

After the building tasks are completed the reflash package can be found in the following directory:

`~/bluegiga-sdk/build/tmp/deploy/images/`

The file itself will be called `bluegiga-console-reflash_5.0.0-beta*_apx4devkit.bpk` and in that same directory also a symbolic link called `bluegiga-console-reflash_apx4devkit.bpk` will exist, always pointing to the last compiled package.

2.2.4 Step 4: SDK installation completed

Congratulations, you have now successfully installed the Bluegiga APx4 SDK and created your first file system image delivered in an installable *reflash*.bpk package.

2.2.5 Step 5: Copy Re-flash package to target

You can now install the compiled Re-flash package into the APx4. To install the Re-flash package you could use SCP application for uploading it into the APx4.

The syntax for copying a file using SCP is

```
$ scp file/you/wish/to/copy remoteuser@remote_ip:/destination/dir
```

If you omit the destination directory the file ends up in default location at the remote side, normally:

```
/home/remoteuser
```

The installation of the package starts automatically if it is uploaded to the correct directory, that is, /tmp/autoinstall/

```
$ cd ~/bluegiga-sdk/build/tmp/deploy/images/  
$ scp bluegiga-console-reflash_5.0-beta1-20120328105830_apx4devkit.bpk  
root@10.1.1.127:/tmp/autoinstall/
```

2.2.6 Troubleshooting

In case the installation of the SDK did not succeed, for example you were not able to compile the reflash package, please contact Bluegiga technical support for assistance. Include in your request the details of your PC as well as all installation and error logs.

3 Compiling and installing Hello World

Applications are compiled and *.ipk* packages are built by **bitbake** command. User has to provide the meta data of the application and a set of building rules for bitbake. These data and rules are written to a text file called a recipe which file extension is *.bb*. More information about recipes and bitbake command is available at [Bitbake User Manual](#) and [Openembedded User Manual](#).

3.1 Simple own Hello World application

Bluegiga SDK offers very simple and world famous Hello World application as an example. The application files are located at *~/bluegiga-sdk/meta/bluegiga-examples/recipes/helloworld/* The Hello World application consists of a recipe file and a source code file written in C language. Recipe file is named *helloworld_1.0.bb* where *helloworld* is the project name and *1.0* is version of the project.

```
~/bluegiga-sdk/meta/bluegiga-examples/helloworld/helloworld_1.0.bb
```

```
PR = "r1"
LICENSE = "CLOSED"

S = "${WORKDIR}"
FILES_${PN} = "/"

SRC_URI = "file://helloworld.c"

do_compile() {
    ${CC} helloworld.c -o helloworld
}

do_install() {
    install -d ${D}${bindir}
    install -m 0755 helloworld ${D}${bindir}
}
```

The recipe contains meta data for the creation of the *.ipk* package, list of source files and instructions for compiling binary and building package. Hello world application is then built using **bitbake** command as shown below. Remember that the "source init-sdk-env" must always be launched when a new terminal window is opened.

```
$ cd ~/bluegiga-sdk
$ source init-sdk-env
$ bitbake helloworld
```

The ready *.ipk* package will be copied to *~/bluegiga-sdk/build/tmp/deploy/ipk/armv5te/helloworld_1.0-r1_armv5te.ipk* by bitbake.

To install the hello world to your APx4 you have multiple choices:

- Using the web interface -> Software upload
- Using SCP: **scp ~/bluegiga-sdk/build/tmp/deploy/ipk/armv5te/helloworld_1.0-r1_armv5te.ipk root@[APx4 IP address]:/tmp/autoinstall/**

Once installed you can run the program at the shell prompt of the APx4: */usr/bin/helloworld* where also */usr/bin/* is defined in the recipe (see http://docs.openembedded.org/usermanual/html/directories_installation.html)

3.2 Autotools-based Hello World application

Bluegiga SDK is designed for building autotools-based applications (see http://en.wikipedia.org/wiki/GNU_build_system). User needs only to define meta data and URL of source package. SDK is capable of handling the rest, like compiling binaries and installing files. A basic example of autotools based hello world application is at `~/bluegiga-sdk/meta/bluegiga-examples/helloworld/hello_2.7.bb`.

```
~/bluegiga-sdk/meta/bluegiga-examples/helloworld/hello_2.7.bb
```

```
DESCRIPTION = "GNU Helloworld application"
SECTION = "examples"
LICENSE = "GPLv3"
LIC_FILES_CHKSUM = "file://COPYING;md5=d32239bcb673463ab874e80d47fae504"
DEPENDS = "virtual/gettext"

SRC_URI = "http://ftp.gnu.org/gnu/${PN}/${PN}-${PV}.tar.gz"

SRC_URI[md5sum] = "fc01b05c7f943d3c42124942a2a9bb3a"
SRC_URI[sha256sum] =
"fd593b5bcf6d1bb6d7d1bb7eefdccdc0010cf2c4985ccb445ef490f768b927c0"

inherit autotools
```

3.3 More information

More information about writing recipes and using bitbake is available at:

- Yocto Reference Manual: <http://www.yoctoproject.org/docs/current/poky-ref-manual/poky-ref-manual.html>
- Recipe & Patch Style Guide: https://wiki.yoctoproject.org/wiki/Recipe_%26_Patch_Style_Guide
- OpenEmbedded User Manual (partly outdated but might be very useful): <http://docs.openembedded.org/usermanual/html/>

4 Customizing file system image to include additional programs

If you want to add your own applications, like helloworld and hello into the file system image it is possible to do so. File system images can be customized adding new packages to *IMAGE_INSTALL* variable of recipe.

meta/bluegiga-distro/recipes-image/bluegiga/customized-console-image.bb

```
IMAGE_FEATURES += "bluegiga-apps-console bluegiga-apps-connectivity"
LICENSE = "MIT"

inherit bluegiga-image

IMAGE_INSTALL += "helloworld hello"
```

Build customized file system image using **bitbake** command.

```
$ cd ~/bluegiga-sdk
$ source init-sdk-env
$ bitbake customized-console-image
```

More information about image customization can be found from documentation of Yocto:
<http://www.yoctoproject.org/docs/latest/dev-manual/dev-manual.html#usingpoky-extend-customimage>

5 Configuring Linux kernel

5.1 Enabling features or drivers inside the kernel

To enable features or drivers from kernel.org kernel:

SDK shell session

```
$ cd ~/bluegiga-sdk
$ source init-sdk-env
$ bitbake linux -c menuconfig
$ bitbake linux -c devshell
```

New devshell console window will appear:

SDK shell session

```
$ cp .config
~/bluegiga-sdk/meta/bluegiga-bsp/recipes-kernel/linux/linux-3.2/apx4devkit/defco
exit
```

5.2 Compiling kernel

SDK shell session

```
$ cd ~/bluegiga-sdk
$ source init-sdk-env
$ bitbake linux
```

5.3 Building external kernel module

Build strategy differs from a module to module as it depends how it is meant to be build. Please refer to instructions found inside the package and apply those to the SDK.

We provide very simple hello world kernel module example under `~/bluegiga-sdk/meta/bluegiga-examples/recipes/hellokernel`. Please take a look to files under that directory and modify those to your needs. Source code of the module is under the files directory.

6 Modifying the software to support your own motherboard

6.1 Linux kernel

6.1.1 GPIO

All interfaces defined as LCD, SDIO, SAIF, UARTs etc. in default software can be changed to be used as GPIOs.

Following three lines are not routed to the connector.

Pin MX28_PAD_PWM3__PWM_3 is the APX4 module's LED next to power LED.

Pin MX28_PAD_GPMI_CE1N_GPIO_0_17 is reserved as Bluetooth reset and is an active low signal.

MX28_PAD_GPMI_RDY1_GPIO_0_21 is reserved as Wi-Fi reset and is also an active low signal.

Also lines for UART1, MMC2 and FEC0 are reserved for specific purposes and are not routed to the module's connector.

arch/arm/mach-mxs/mach-apx4devkit.c

```
static const iomux_cfg_t apx4devkit_pads[] __initconst = {
    /* LED */
    MX28_PAD_PWM3__PWM_3 | MXS_PAD_CTRL,
    /* LCD Backlight */
    MX28_PAD_PWM4__PWM_4 | MXS_PAD_CTRL,

    /* GPIO */
    MX28_PAD_GPMI_CE1N_GPIO_0_17 | MXS_PAD_CTRL, /* Bluetooth reset (active low)
*/
    MX28_PAD_GPMI_RDY1_GPIO_0_21 | MXS_PAD_CTRL, /* Wifi reset (active low) */
    MX28_PAD_LCD_RESET_GPIO_3_30 | MXS_PAD_CTRL, /* LCD reset (active low) */
    MX28_PAD_SAIF0_MCLK_GPIO_3_20 | MXS_PAD_CTRL, /* USB reset (active low) */
    MX28_PAD_JTAG_RTCK_GPIO_4_20 | MXS_PAD_CTRL, /* Factory reset button (active
low) */

    /* MXSFB */
#define LCD_MODE (MXS_PAD_3V3 | MXS_PAD_4MA)
    MX28_PAD_LCD_D00__LCD_D0 | LCD_MODE,
    MX28_PAD_LCD_D01__LCD_D1 | LCD_MODE,
    MX28_PAD_LCD_D02__LCD_D2 | LCD_MODE,
    MX28_PAD_LCD_D03__LCD_D3 | LCD_MODE,
    MX28_PAD_LCD_D04__LCD_D4 | LCD_MODE,
    MX28_PAD_LCD_D05__LCD_D5 | LCD_MODE,
    MX28_PAD_LCD_D06__LCD_D6 | LCD_MODE,
    MX28_PAD_LCD_D07__LCD_D7 | LCD_MODE,
    MX28_PAD_LCD_D08__LCD_D8 | LCD_MODE,
    MX28_PAD_LCD_D09__LCD_D9 | LCD_MODE,
    MX28_PAD_LCD_D10__LCD_D10 | LCD_MODE,
    MX28_PAD_LCD_D11__LCD_D11 | LCD_MODE,
    MX28_PAD_LCD_D12__LCD_D12 | LCD_MODE,
    MX28_PAD_LCD_D13__LCD_D13 | LCD_MODE,
    MX28_PAD_LCD_D14__LCD_D14 | LCD_MODE,
    MX28_PAD_LCD_D15__LCD_D15 | LCD_MODE,
    MX28_PAD_LCD_D16__LCD_D16 | LCD_MODE,
    MX28_PAD_LCD_D17__LCD_D17 | LCD_MODE,
    MX28_PAD_LCD_D18__LCD_D18 | LCD_MODE,
    MX28_PAD_LCD_D19__LCD_D19 | LCD_MODE,
    MX28_PAD_LCD_D20__LCD_D20 | LCD_MODE,
```

```

MX28_PAD_LCD_D21__LCD_D21 | LCD_MODE,
MX28_PAD_LCD_D22__LCD_D22 | LCD_MODE,
MX28_PAD_LCD_D23__LCD_D23 | LCD_MODE,
MX28_PAD_LCD_RD_E__LCD_VSYNC | LCD_MODE,
MX28_PAD_LCD_WR_RWN__LCD_HSYNC | LCD_MODE,
MX28_PAD_LCD_RS__LCD_DOTCLK | LCD_MODE,
MX28_PAD_LCD_CS__LCD_ENABLE | LCD_MODE,

/* DUART */
MX28_PAD_PWM1__DUART_TX | MXS_PAD_CTRL,
MX28_PAD_PWM0__DUART_RX | MXS_PAD_CTRL,

/* UART0 */
MX28_PAD_AUART0_TX__AUART0_TX,
MX28_PAD_AUART0_RX__AUART0_RX,
MX28_PAD_AUART0_CTS__AUART0_CTS,
MX28_PAD_AUART0_RTS__AUART0_RTS,

/* UART1 (Bluetooth) */
MX28_PAD_AUART1_TX__AUART1_TX,
MX28_PAD_AUART1_RX__AUART1_RX,

/* UART2 */
MX28_PAD_SSP2_MOSI__AUART2_TX,
MX28_PAD_SSP2_SCK__AUART2_RX,

/* UART3 */
MX28_PAD_SSP2_SS0__AUART3_TX,
MX28_PAD_SSP2_MISO__AUART3_RX,

/* I2C */
MX28_PAD_I2C0_SCL__I2C0_SCL,
MX28_PAD_I2C0_SDA__I2C0_SDA,

/* MMC0 */
MX28_PAD_SSP0_DATA0__SSP0_D0 |
(MXS_PAD_8MA | MXS_PAD_3V3 | MXS_PAD_PULLUP),
MX28_PAD_SSP0_DATA1__SSP0_D1 |
(MXS_PAD_8MA | MXS_PAD_3V3 | MXS_PAD_PULLUP),
MX28_PAD_SSP0_DATA2__SSP0_D2 |
(MXS_PAD_8MA | MXS_PAD_3V3 | MXS_PAD_PULLUP),
MX28_PAD_SSP0_DATA3__SSP0_D3 |
(MXS_PAD_8MA | MXS_PAD_3V3 | MXS_PAD_PULLUP),
MX28_PAD_SSP0_CMD__SSP0_CMD |
(MXS_PAD_8MA | MXS_PAD_3V3 | MXS_PAD_PULLUP),
MX28_PAD_SSP0_DETECT__SSP0_CARD_DETECT |
(MXS_PAD_8MA | MXS_PAD_3V3 | MXS_PAD_NOPULL),
MX28_PAD_SSP0_SCK__SSP0_SCK |
(MXS_PAD_12MA | MXS_PAD_3V3 | MXS_PAD_NOPULL),

/* MMC2 (WiFi) */
MX28_PAD_SSP0_DATA4__SSP2_D0 |
(MXS_PAD_8MA | MXS_PAD_3V3 | MXS_PAD_PULLUP),
MX28_PAD_SSP2_SS1__SSP2_D1 |
(MXS_PAD_8MA | MXS_PAD_3V3 | MXS_PAD_PULLUP),
MX28_PAD_SSP2_SS2__SSP2_D2 |
(MXS_PAD_8MA | MXS_PAD_3V3 | MXS_PAD_PULLUP),
MX28_PAD_SSP0_DATA5__SSP2_D3 |
(MXS_PAD_8MA | MXS_PAD_3V3 | MXS_PAD_PULLUP),
MX28_PAD_SSP0_DATA6__SSP2_CMD |

```



```

(MXS_PAD_8MA | MXS_PAD_3V3 | MXS_PAD_PULLUP),
MX28_PAD_SSP0_DATA7__SSP2_SCK |
(MXS_PAD_12MA | MXS_PAD_3V3 | MXS_PAD_NOPULL),

#define MXS_PAD_FEC (MXS_PAD_8MA | MXS_PAD_3V3 | MXS_PAD_PULLUP)
/* fec0 */
MX28_PAD_ENET0_MDC__ENET0_MDC | MXS_PAD_FEC,
MX28_PAD_ENET0_MDIO__ENET0_MDIO | MXS_PAD_FEC,
MX28_PAD_ENET0_RX_EN__ENET0_RX_EN | MXS_PAD_FEC,
MX28_PAD_ENET0_RXD0__ENET0_RXD0 | MXS_PAD_FEC,
MX28_PAD_ENET0_RXD1__ENET0_RXD1 | MXS_PAD_FEC,
MX28_PAD_ENET0_TX_EN__ENET0_TX_EN | MXS_PAD_FEC,
MX28_PAD_ENET0_TXD0__ENET0_TXD0 | MXS_PAD_FEC,
MX28_PAD_ENET0_TXD1__ENET0_TXD1 | MXS_PAD_FEC,
MX28_PAD_ENET_CLK__CLKCTRL_ENET | MXS_PAD_FEC,

/* saif0 & saif1 */
MX28_PAD_SAI0_LRCLK__SAIF0_LRCLK |
(MXS_PAD_12MA | MXS_PAD_3V3 | MXS_PAD_PULLUP),
MX28_PAD_SAI0_BITCLK__SAIF0_BITCLK |
(MXS_PAD_12MA | MXS_PAD_3V3 | MXS_PAD_PULLUP),
MX28_PAD_SAI0_SDATA0__SAIF0_SDATA0 |
(MXS_PAD_12MA | MXS_PAD_3V3 | MXS_PAD_PULLUP),
MX28_PAD_SAI1_SDATA0__SAIF1_SDATA0 |

```

```
(MXS_PAD_12MA | MXS_PAD_3V3 | MXS_PAD_PULLUP),  
};
```

6.1.2 I2C

I2C (pins `MX28_PAD_I2C0_SCL_I2C0_SCL` and `MX28_PAD_I2C0_SDA_I2C0_SDA`) are used for RTC on the module. On the APX4 Development Kit also SGT15000 audio codec is also connected to it. I2C is a shared bus so you don't need to change pin assignments. Just add your devices to the list below:

`arch/arm/mach-mxs/mach-apx4devkit.c`

```
static struct i2c_board_info apx4devkit_i2c_boardinfo[] __initdata = {  
    { I2C_BOARD_INFO("sgt15000", 0x0a) }, /* ASoC */  
    { I2C_BOARD_INFO("pcf8563", 0x51) }, /* RTC */  
};
```

First string is driver's I2C name. In this case "pcf8563" comes from `drivers rtc/rtc-pcf8563.c`. 0x51 is device's I2C address. Please refer to I2C device's datasheet for that one.

7 Eclipse support

7.1 Setting up the development environment

This section describes how to install and configure the Bluegiga SDK and Eclipse IDE.

7.1.1 Step 1 - Install the Bluegiga SDK

If you have Bluegiga SDK already installed, you may skip this step.

Please follow the installation instructions found in the "[Getting started](#)" section.

7.1.2 Step 2 - Prepare IDE support

Enable IDE support in the Bluegiga SDK by issuing the following commands:

```
$ cd ~/bluegiga-sdk
$ source init-sdk-env
$ bitbake meta-ide-support
```

7.1.3 Step 3 - Install the Eclipse Juno IDE

If you already have the Eclipse Juno (4.2) IDE installed on your system, you may skip this step.

Download the Eclipse Juno (4.2) tarball: from <http://www.eclipse.org/downloads> choose the Eclipse Classic version. Once you have the tarball downloaded, extract it into your home directory. For example, if you downloaded it into `~/Downloads`, executing the following commands will extract the content of the tarball into `~/eclipse`.

```
$ cd ~
$ tar xvf ~/Downloads/eclipse-SDK-4.2*-linux-gtk*.tar.gz
```

7.1.4 Step 4 - Install required plugins

First you need to install official Eclipse plugins which are required for building, executing and debugging applications in APx4. You may do so by following the steps below:

1. Start the Eclipse IDE using commands "cd ~/eclipse" and "./eclipse". Make sure you have Java installed by using the command "java -version". In case you need to install Java, use for example the command "sudo apt-get install openjdk-7-jre-headless" as suggested by the output of the "java -version" command.
2. Select a workspace.
3. From the "*Help*" menu, select "*Install New Software*".
4. Select `_"Juno - http://download.eclipse.org/releases/juno"_` from the "*Work with*" drop-down list.
5. Expand the "*Linux Tools*" entry when it is retrieved from the internet and select (checkbox) the "*LTTng - Linux Tracing Toolkit*" box.
6. Expand the "*Mobile and Device Development*" entry and check the following boxes:
 - "*C/C++ Remote Launch*"
 - "*Remote System Explorer End-user Runtime*"
 - "*Remote System Explorer User Actions*"

- *"Target Management Terminal"*
 - *"TCF Remote System Explorer add-in"*
 - *"TCF Target Explorer"*
7. Expand the *"Programming Languages"* entry and select the *"Autotools Support for CDT"* box.
 8. Click "Next" buttons and complete the installation.
 9. Restart the Eclipse IDE.

In addition to the standard plugins, a Yocto plugin needs to be installed. It streamlines the configuration of the cross-compiling toolchain. Next steps explain how to install the plugin:

1. Start up the Eclipse IDE and select the workspace.
2. From the *"Help"* menu, select *"Install New Software"*.
3. Click *"Add..."* in the *"Work with"* area.
4. Enter *"Yocto"* in the *"Name"* field and *_"http://downloads.yoctoproject.org/releases/eclipse-plugin/1.3"_* in the URL field, and click OK.
5. Make sure *_"Yocto - http://downloads.yoctoproject.org/releases/eclipse-plugin/1.3"_* is selected in the *"Work with"* drop-down list.
6. Check the box next to *"Development tools & SDKs for Yocto Linux"*.
7. Complete the installation (discarding any message about software containing unsigned content) and restart the Eclipse IDE.

7.1.5 Step 5 - Configure the Eclipse Yocto Plugin

The last step in setting up the development environment is to configure the cross-compiler and target options to be used in the Yocto plugin. These options will become the default settings for new projects.

1. Start up the Eclipse IDE and select the workspace.
2. From the *"Window"* menu, select *"Preferences"*.
3. Select *"Yocto Project ADT"* from the list on the left side of the preferences dialog box.
4. Configure the *"Cross Compiler Options"* section as follows:
 - a. Make sure that *"Build system derived toolchain"* is selected.
 - b. Enter the path to the *build* directory inside the Bluegiga SDK root into the *"Toolchain Root Location"* field.
 - c. Enter the path to the *build/tmp/sysroots/apx4devkit* directory inside the Bluegiga SDK root into the *"Sysroot Location"* field.
 - d. Make sure *"armv5te-bluegiga-linux-gnueabi"* is selected from the *"Target Architecture"* drop-down list.
5. Configure the *"Target Options"* section as follows:
 - a. Make sure that *"External HW"* is selected.
6. Click the OK button.

7.2 Creating a new project

This section describes how to create a new Autotools-based project within the Eclipse IDE.

1. Start up the Eclipse IDE and select the workspace.
2. Select *"File" -> "New" -> "Project"*.
3. Expand the *"C/C++"* entry and double-click *"C Project"*.
4. Put the name of your project into the *"Project name"* field (do not use "-" character).
5. Expand the *"Yocto Project ADT Project"* and select *"Hello World ANSI C Autotools Project"*.
6. Click *"Next"*.
7. Add information in the *"Author"*, *"Copyright notice"* and *"License"* fields.
8. Click *"Finish"*.
9. If the *"Open Associated Perspective"* prompt appears, click "Yes" to get into the C/C++ perspective.
10. You can access your project files using the navigation pane on the left-hand side of the screen.

7.3 Building and deploying the project

First you should build your project to make sure that it is syntactically correct.

1. Make sure that your project is selected in the left-hand side navigation pane.
2. From the *"Project"* menu, select *"Build Project"*.

If the building succeeds, you may proceed to configure the remote system where you want the program to be copied to and executed.

1. From the "Run" menu, select "Run Configurations..."
2. Select "C/C++ Remote Application" and click the "New" icon in the top row of the dialog.
3. In the "Connection" area, click "New..."
4. Select "SSH Only" as the remote system type and click "Next".
5. Insert the IP address of your APx4 development kit into the "Host name" field, and click "Finish".
6. In the "C/C++ Application" area, click "Search Project".
7. Double click on the binary file in the "Binaries" list.
8. Insert path and name of the executable, according to how you want it to be copied to the remote system, into the "Remote Absolute File Path for C/C++ Application" field, e.g. "/usr/bin/helloworld".
9. Click "Run".
10. Insert your username and password for the APx4 into the corresponding fields.
11. Click "OK".

You should see the output of the program in the bottom of the screen in the "Console" tab. If you modify your application and want to see the changes in effect, it is enough to click on the menubar "Run" -> "Run History" and click on the first item in the list: the project will be built again and executable sent to remote device.

7.4 Debugging the project

In order to debug the project, you need to install the *gdbserver* package on the remote system. Log in to the remote system as root and issue the following command:

```
$ opkg update
$ opkg install gdbserver
```

1. From the "Run" menu, select "Debug Configurations".
2. The run configuration you created previously should be selected on the list on the left-hand side of the window.
3. Select the "Debugger" tab.
4. Put "arm-bluegiga-linux-gnueabi-gdb" into the "GDB debugger" field.
5. Click "Debug".
6. If the "Open Associated Perspective" prompt appears, click "Yes" to get into the debug perspective.

The debug perspective should open up and show the first line of the main function of your application. The next time you want to debug your application, it is enough to click on the menubar "Run" -> "Debug History" and click on the first item in the list.

8 Contact information

Sales: sales@bluegiga.com

Technical support: support@bluegiga.com
<http://techforum.bluegiga.com>

Orders: orders@bluegiga.com

WWW: <http://www.bluegiga.com>
<http://www.bluegiga.hk>

Head Office / Finland: Phone: +358-9-4355 060
Fax: +358-9-4355 0660
Sinikalliontie 5 A
02630 ESPOO
FINLAND

Head address / Finland: P.O. Box 120
02631 ESPOO
FINLAND

Sales Office / USA: Phone: +1 770 291 2181
Fax: +1 770 291 2183
Bluegiga Technologies, Inc.
3235 Satellite Boulevard, Building 400, Suite 300
Duluth, GA, 30096, USA

Sales Office / Hong-Kong: Phone: +852 3182 7321
Fax: +852 3972 5777
Bluegiga Technologies, Inc.
Unit 10-18, 32/F, Tower 1, Millennium City 1,
388 Kwun Tong Road, Kwun Tong, Kowloon,
Hong Kong