Yocto

AM335x BSP Manual

Document No.: **L-818e_3**

Release No.: **AM335x PD16.2.x**

Yocto 2.1.2

Edition: February 2017

A product of a PHYTEC Technology Holding company

Copyrighted products are not explicitly indicated in this manual. The absence of the trademark (™, or ®) and copyright (©) symbols does not imply that a product is not protected. Additionally, registered patents and trademarks are similarly not expressly indicated in this manual.

The information in this document has been carefully checked and is considered to be entirely reliable. However, PHYTEC Messtechnik GmbH assumes no responsibility for any inaccuracies. PHYTEC Messtechnik GmbH neither gives any guarantee nor accepts any liability whatsoever for consequential damages resulting from the use of this manual or its associated product. PHYTEC Messtechnik GmbH reserves the right to alter the information contained herein without prior notification and accepts no responsibility for any damages that might result.

Additionally, PHYTEC Messtechnik GmbH offers no guarantee nor accepts any liability for damages arising from the improper usage or improper installation of the hardware or software. PHYTEC Messtechnik GmbH further reserves the right to alter the layout and/or design of the hardware without prior notification and accepts no liability for doing so.

© Copyright 2017 PHYTEC Messtechnik GmbH, D-55129 Mainz.
Rights - including those of translation, reprint, broadcast, photomechanical or similar reproduction and storage or processing in computer systems, in whole or in part - are reserved. No reproduction may occur without the express written consent from PHYTEC Messtechnik GmbH.

	EUROPE	NORTH AMERICA	FRANCE
Address:	PHYTEC Messtechnik GmbH Robert-Koch-Str. 39 D-55129 Mainz GERMANY	PHYTEC America LLC 203 Parfitt Way SW Bainbridge Island, WA 98110 USA	PHYTEC France 17, place Saint-Etienne F-72140 Sillé-le-Guillaume FRANCE
Sales:	+49 6131 9221-32 sales@phytec.de	+1 800 278-9913 sales@phytec.com	+33 2 43 29 22 33 info@phytec.fr
Technical Support:	+49 6131 9221-31 support@phytec.de	+1 206 780-9047 support@phytec.com	support@phytec.fr
Fax:	+49 6131 9221-33	+1 206 780-9135	+33 2 43 29 22 34
Web Site:	http://www.phytec.de http://www.phytec.eu	http://www.phytec.com	http://www.phytec.fr

	INDIA	CHINA
Address:	PHYTEC Embedded Pvt. Ltd. #438, 1st Floor, 18th Main, 6th Block, Oppt. BMTC Bus Depot, Koramangala, Bangalore-560095 INDIA	PHYTEC Information Technology (Shenzhen) Co. Ltd. 2106A, Block A, Tianxia Jinniu Square, Taoyuan Road, Nanshan District, 518026 Shenzhen CHINA
Sales:	+91-80-4086 7046/48 sales@phytec.in	+86-755-6180-2110 sales@phytec.cn
Technical Support:	+91-80-4086 7047/50 support@phytec.in	support@phytec.cn
Fax:		
Web Site:	http://www.phytec.in	http://www.phytec.cn

3rd Edition February 2017

List of Figures

Conventions, Abbreviations and Acronyms

This AM335x BSP Manual describes the *Linux* BSP accompanying our hardware products. It is based on The *Yocto* Project, extended with hardware support for our products. We give a brief introduction to *Yocto* in general and the specific changes and additions made by Phytec.

Conventions

The conventions used in this manual are as follows:

- Text in *blue italic* indicates a hyperlink within, or external to the document. Click these links to quickly jump to the applicable URL, part, chapter, table, or figure.
- Text in ***bold italic*** indicates an interaction by the user, which is defined on the screen.
- Text in `Consolas` indicates an input by the user, without a premade text or button to click on.
- Text in *italic* indicates proper names of development tools and corresponding controls (windows, tabs, commands, file paths, etc.) used within the development tool, no interaction takes place.
- White Text on black background shows the result of any user interaction (command, program execution, etc.)

⚠	This is a warning. It helps you to avoid annoying problems.
	You can find useful supplementary information about the topic.

1 Introduction to Yocto

Please read the *Yocto* Reference Manual (L-813e_x) for a better understanding of *Yocto* and this BSP.

2 Introduction to the BSP

2.1 Supported Hardware

For information which boards and modules are supported by the release of Phytec's AM335x unified BSP described herein, visit our web page at *http://www.phytec.de/produkte/software/yocto/phytec-unified-yocto-bsp-releases/*.

Click the corresponding BSP release and look for the ordering number of your module in the column "Hardware Article Number". Now you can find the correct machine name in the corresponding cell under "Machine Name".

3 Building the BSP

This section will guide you through the general build process of the unified AM335x BSP using the phyLinux script. If you want to use our software without phyLinux and the *Repo* tool managed environment instead, you can find all *Git* repositories on

https://git.phytec.de

Used *barebox* repository:

https://git.phytec.de/barebox

Our *barebox* version is based on the *barebox* mainline and adds only a few patches which will be sent upstream in future. Used *Linux* kernel repository:

https://git.phytec.de/linux-ti

Our AM335x kernel is based on the kernel maintained by TI. TI has several thousand patches on top of the mainline kernel. We added another 50 patches on top of it. TI's kernel repository can be found at:

git://git.ti.com/ti-linux-kernel/ti-linux-kernel.git

To find out which tag is used for a specific board, have a look at your checked out BSP source folder under:

meta-phytec/recipes-bsp/barebox/barebox_*.bb
meta-phytec/recipes-kernel/linux/linux-ti_*.bb

3.1 Get the BSP

- Create a fresh project directory, e.g.

  ```
  host$ mkdir ~/yocto
  ```

- Download and run the phyLinux script

  ```
  host$ cd ~/yocto
  host$ wget ftp://ftp.phytec.de/pub/Software/Linux/Yocto/Tools/phyLinux
  host$ chmod  +x  phyLinux
  host$./phyLinux init
  ```

3.2 Basic Set-Up

There are a few important steps which have to be done, before the main build process.

- Setting up the host, see *Yocto* Reference Manual "Setting up the Host"
- Setting up the *Git* configuration, see *Yocto* Reference Manual "*Git* Configuration"

3.3 Finding the right Software Platform

The AM335x BSP is a unified BSP, which means it supports a set of different Phytec carrier boards (CB) with different Systems on Module (SOMs). Sometimes it is not easy to find the right software for your Phytec board. So if you need to figure out the corresponding machine name of your board, have a look at *http://www.phytec.de/produkte/software/yocto/phytec-unified-yocto-bsp-releases/* and click on the corresponding BSP release, or refer to the files in the source folder of the BSP:

meta-phytec/conf/machine/.conf*

where you can find the platform name to the corresponding product IDs. All this information is also displayed by the phyLinux script.

E.g.: *phycore-am335x-1.conf* machine configuration file:

```
#@TYPE:  Machine
#@NAME: phycore-am335x-1
#@DESCRIPTION:  PCM-051-12102F0C.A1/KPCM-953  (Kit)
```

Machine *phycore-am335x-1* represents the PCM-953 (Kit) CB with the PCM-051-12102F0C.A1 SOM.

3.4 Selecting a Software Platform

- To select the correct SoC, BSP version and platform call:

  ```
  host$ ./phyLinux init
  ```

It is also possible to pass this information directly using command line parameters:

```
host$ ./phyLinux init -p  am335x -r PD15.1-rc1 -m  phycore-am335x-1
```

Please read section "Initialization" in the *Yocto* Reference Manual for more information.

3.5 Starting the Build Process

Refer to *Yocto* Reference Manual "Start the Build".

3.6 BSP Images

All images generated by *Bitbake* are deployed to *yocto/build/deploy/images/<machine>*.

The following list shows for example all files generated for the AM335x SoC, *phycore-am335x-1* machine:

- *Barebox*: barebox.bin
- *Barebox* configuration: barebox.config
- *Barebox* PBL
  - for memory boot devices (MMC, NAND): MLO
  - for memory boot devices (SPI): MLO.spi
  - for memory boot devices (MMC, NAND): MLO
  - for peripheral boot devices (EMAC, UART): MLO.per
- Kernel: zImage
- Kernel device tree file: zImage-am335x-phycore-rdk.dtb
- Kernel configuration: zImage.config
- Root filesystem: phytec-qt5demo-image-phycore-am335x-1.tar.gz,
  phytec-qt5demo-image-phycore-am335x-1.ubifs,
  phytec-qt5demo-image-phycore-am335x-1.ext4
- SD card image: phytec-qt5demo-image-phycore-am335x-1.sdcard

4 Booting the System

The default boot source for phyCORE-AM335x modules is the NAND Flash. The easiest way to get started with your freshly created images, is writing them to an SD card and setting the boot configuration accordingly. For information how to set the correct boot configuration refer to the corresponding hardware manual for your Phytec board.

4.1 Booting from NAND Flash

NAND is the default boot source. To update the software of the NAND Flash see *5 "Updating the Software"*.

4.2 Booting from SD Card

Booting from SD card is useful in several situations, e.g. if the board does not start any more due to a damaged bootloader. To boot from SD card the SD card must be formatted in a special way, because the AM335x does not use file systems. Instead it is hard coded at which sectors of the SD card the AM335x expects the bootloader.

There are two ways to create a bootable SD card. You can either use:

- a single prebuild SD card image, or
- a script which creates a formatted SD card allowing to copy the four individual images (*MLO*, *barebox.bin*, *linuximage* and *oftree* file) manually

4.2.1 Using a single, prebuild SD Card Image

The first possibility is to use the SD card image built by *Bitbake*, a tool integrated in *Yocto*. This image has the ending **.sdcard* and can be found under *build/deploy/images/<MACHINE>/<IMAGENAME>-<MACHINE>.sdcard*. It contains all BSP files in already correctly formatted partitions and can be copied to the SD card easily using the single *Linux* command *dd*.

You can also find ready-to-use **.sdcard* images on our FTP server: *ftp://ftp.phytec.de/pub/Software/Linux/BSP-Yocto-AM335x/*.

⚠	To create your bootable SD card with the *dd* command you must have root privileges. Because of that you must be very careful when selecting the destination device for the *dd* command! All files on the selected destination device will be erased immediately without any further query! Consequently, having selected the wrong device can also erase your hard drive!

To create your bootable SD card you must first find out the correct device name of your SD card and possible partitions and then unmount the partitions before you start copying the image to the SD card.

- In order to get the correct device name first remove your SD card and execute `ls /dev.`
- Now insert your SD card and execute `ls /dev` again.
- Compare the two outputs to find the new device name(s) listed in the second output. These are the device names of the SD card (device, and partitions if the SD card is formatted).
- In order to verify the device names found, execute the command `dmesg`. Within the last lines of its output you should also find the device names, for example *sde* (depending on your system).

Now that you have the device name */dev/<your_device>* (e.g. */dev/sde*) you can recognize the partitions which must be unmounted if the SD card is formatted, too. In this case you will also find */dev/<your_device>* with an appended number (e.g. */dev/sde**1***) in the output. These represent the partition(s) to be unmounted.

- Unmount all partitions with:

  ```
  host$ umount /dev/<your_device><number>
  ```

- After having unmounted all devices with an appended number (*<your_device><number>)*, you can create your bootable SD card with

  ```
  host$ sudo dd if=<IMAGENAME>-<MACHINE>.sdcard of=/dev/<your_device>
                                                       bs=1MB conv=fsync
  ```

  using the device name (<your_device>) without appended number (e.g. *sde)* which stands for the whole device.

The parameter *conv=fsync* forces a sync operation on the device before *dd* returns. This ensures that all blocks are written to the SD card and are not still in memory.

4.2.2 Using a Script and the four individual Images (*MLO*, *barebox.bin*, *linuximage* and *oftree* file)

It is also possible to make the steps described in the previous section manually by using a script to create a proper formatted SD card. The script creates two partitions. The first one is formatted with *vfat* and the second one is formatted with *ext4*.

```
#!/bin/bash
if [ ! "$1" = "/dev/sda" ] ; then
   unset LANG
   DRIVE=$1
   umount $DRIVE"1"
   umount $DRIVE"2"
   if [ -b "$DRIVE" ] ; then
      dd if=/dev/zero of=$DRIVE bs=1024 count=1024
      SIZE=`fdisk -l $DRIVE | grep Disk | awk '{print $5}'`
      echo DISK SIZE - $SIZE bytes
      CYLINDERS=`echo $SIZE/255/63/512 | bc`
      echo CYLINDERS - $CYLINDERS
      {
      echo ,25,0x0C,*
      echo ,,,-
      } | sfdisk -D -H 255 -S 63 -C $CYLINDERS $DRIVE
      mkfs.vfat -F 32 -n "boot" ${DRIVE}1
      mke2fs -t ext4 -L "rootfs" ${DRIVE}2
   fi
fi
```

- Copy this script into a file and make it executable with:

```
host$ chmod +x copyscript.sh
```

- Execute it with:

```
host$ sudo ./copyscript.sh /dev/<sddevice>
```

Afterwards the images need to be copied to the SD card.

- Mount the first partition (*vfat*) and copy the *MLO*, *barebox.bin*, *linuximage* and *oftree* file to it.

```
host$ mount /dev/sd<X>
host$ cp yocto/build/deploy/images/<MACHINE>/MLO /mnt/MLO
host$ cp yocto/build/deploy/images/<MACHINE>/barebox.bin
         /mnt/barebox.bin
host$ cp yocto/build/deploy/images/<MACHINE>/zImage /mnt/linuximage
host$ cp yocto/build/deploy/images/<MACHINE>/zImage-am335x-phycore-
         rdk.dtb /mnt/oftree
```

⚠	Make sure that the images are named as mentioned before, as the bootloader expects them exactly like that.

	The <IMAGENAME>-<MACHINE>.ubifs rootfs image is actually not used for booting but in order to allow updating the NAND root filesystem you should copy it, too.

- In case you want to boot the whole *Linux* from SD card, also mount the *ext4* partition.
- Then untar <IMAGENAME>-<MACHINE>.tgz rootfs image to it:

```
host$ sudo mount /dev/sd<X>2 /media/rootfs
host$ sudo tar zxf yocto/build/deploy/images/<MACHINE>/<IMAGENAME>-
                                        <MACHINE>.tgz -C /media/rootfs
```

- Do not forget to properly unmount the SD card with:

```
host$ sudo umount /media
```

4.3 Booting from UART

If no SD card slot is available, it is also useful to be able to boot at least the bootloader from UART. This is possible with the UART0 interface of the AM335x. Make sure that the boot mode is set correctly to serial. The boot mode is set correctly, if you see a lot of "C" characters on the console output after booting.

The following script facilitates booting from serial. But this can also be done directly from a serial terminal program like *minicom* by loading first the *MLO.per* (first stage bootloader for peripheral boot) and then the *barebox.bin* file over *xmodem*. When using the script, please close all serial terminal programs.

```
#!/bin/sh
SERIAL=/dev/<ttyXX>
FILEPATH=<path>
MLO=$FILEPATH/MLO.per
BAREBOX=$FILEPATH/barebox.bin

stty -F $SERIAL 115200
sx -vv $MLO < $SERIAL > $SERIAL
sx -vv $BAREBOX < $SERIAL > $SERIAL
minicom -D $SERIAL
```

Change the SERIAL variable to the correct *tty* device on your host and adapt the FILEPATH to your needs.

After the *MLO.per* and *barebox.bin* files are transferred over serial to the target, the *minicom* starts with showing the *barebox.bin* output and you will get to the *barebox* prompt.

4.4 Booting from SPI NOR Flash

Most of the AM335x modules (e.g. phyCORE-AM335x and phyFLEX-AM335x) are optionally equipped with SPI NOR Flash. To boot from SPI Flash, select the correct boot mode as described in the hardware manual of your Phytec board. The SPI Flash is usually quite small so that only the two bootloaders, the *Linux* kernel and the device tree can be stored. The root filesystem is taken from NAND Flash by default. How to flash the SPI NOR is described in section *5 "Updating the Software"*.

4.5 Booting a Bootloader from Network

AM335x's ROM code can boot an MLO from the first port of the EMAC interface, using the standard TCP/IP network boot protocols BOOTP and TFTP.

Note: Not all AM335x boards are able to boot from network.

The following tools will be needed to boot the MLO (*barebox*) from Ethernet:

1. a BOOTP/DHCP server
2. a TFTP server and
3. a tool for starting/stopping a service.

- For *Ubuntu* install:

```
host$ sudo apt-get install isc-dhcp-server tftpd-hpa xinetd
```

After the installation of the packages you have to configure the DHCP and TFTP server.

Configurations for the DHCP server:

- Edit */etc/dhcp/dhcpd.conf*.

```
# Basic boot sequence: AM335x ROM -> MLO -> barebox -> kernel
subnet 192.168.3.0 netmask 255.255.255.0
{
  # The filenames must correspond to the barebox and MLO files which
are placed in the /tftpboot directory
  range dynamic-bootp 192.168.3.11 192.168.3.100;
  if substring (option vendor-class-identifier, 0, 10) = "AM335x ROM"
  {
    filename "MLO";
  }
  elsif substring (option vendor-class-identifier, 0, 18) = "am335x
barebox-mlo"
  {
    filename "barebox.bin";
  }
  range 192.168.3.101 192.168.3.199;
}
```

Set up for the TFTP server:

- Edit */etc/xinetd.d/tftp*.

  ```
  service tftp
  {
      protocol = udp
      port = 69
      socket_type = dgram
      wait = yes
      user = root
      server = /usr/sbin/in.tftpd
      server_args = -s /tftpboot
      disable = no
  }
  ```

- Create a directory to store the TFTP files:

  ```
  host$ sudo mkdir /tftpboot
  host$ sudo chmod -R 777 /tftpboot
  host$ sudo chown -R nobody /tftpboot
  ```

- Edit */etc/default/isc-dhcp-server* to bind the server to a single network interface:

  ```
  INTERFACES="eth1"
  ```

- Configure a static IP address for the appropriate interface:

  ```
  host$ ifconfig eth1
  ```

  You will receive:

  ```
  eth1      Link encap:Ethernet  HWaddr 00:11:6b:98:e3:47
            inet addr:192.168.3.10  Bcast:192.168.3.255
                                         Mask:255.255.255.0
  ```

- Copy your *barebox* images to the */tftpboot* directory.

- Restart the services to pick-up the configuration changes:

  ```
  host$ sudo service isc-dhcp-server restart
  host$ sudo service tftpd-hpa restart
  ```

- Now connect the first port of the board to your host system, configure the board to network boot and start it.

A more detailed network boot description can be found on the following web page.
http://processors.wiki.ti.com/index.php/Ubuntu_12.04_Set_Up_to_Network_Boot_an_AM335x_Based_Platform

4.6 Booting the Kernel from Network

In this case booting from network means to load the kernel over TFTP and the root filesystem over NFS. The bootloader itself must already be loaded from any other boot device available.

4.6.1 Development Host Preparations

On the development host a TFTP server must be installed and configured. An example how to set up a TFTP server can be found in the section above *"Booting a Bootloader from Network"*. Usually TFTP servers are using the */tftpboot* directory to fetch files from. If you built your own images, please copy them from the BSP's build directory to there.

We also need a network connection between the embedded board and the TFTP server. The server should be set to IP 192.168.3.10 and netmask 255.255.255.0, or a DHCP server can be used instead. Setting up a DHCP server is also explained in the section *"Booting a Bootloader from Network"*.

After the installation of the TFTP server, an NFS server needs to be installed, too. The NFS server is not restricted to a certain file system location, so all we have to do on most distributions is to modify the file */etc/exports* and export our root filesystem to the embedded network. In this example file the whole work directory is exported, and the ”lab network” address of the development host is 192.168.3.10, so the IP addresses have to be adapted to the local needs:

```
/home/<user>/<rootfspath>
     192.168.3.10/255.255.255.0(rw,no_root_squash,sync,no_subtree_check)
```

Where <*user*> must be replaced with your home directory name. The <*rootfspath*> can be set to a folder which contains a rootfs *tar.gz* image extracted with *sudo*.

4.6.2 Preparations on the Embedded Board

- To find out the Ethernet settings in the bootloader of the target, type:

```
bootloader$ ifup eth0
bootloader$ devinfo eth0
```

With your development host set to IP 192.168.3.10 and netmask 255.255.255.0, the target should return:

```
ipaddr=192.168.3.11
netmask=255.255.255.0
gateway=192.168.3.10
serverip=192.168.3.10
```

- If you need to change something, type :

```
bootloader$ edit /env/network/eth0
```

Here you can also change the IP address to DHCP instead of using a static one.

- Just configure: `ip=dhcp`
- Edit the settings if necessary and save them by leaving the editor with ***CTRL+D***.
- Type *saveenv* if you made any changes.
- Set up the paths for TFTP and NFS in the file */env/boot/net.*

4.6.3 Booting the Embedded Board

- To boot from network call

  ```
  bootloader$ boot net
  ```

  or restart the board and press ***m*** to stop autoboot.

  You will get a menu:

  ```
  Welcome to Barebox
  1: Boot: nand (UBI)
  2: Boot: kernel & rootfs on SD card
  3: Boot: network (tftp, nfs)
  4: Boot: SPI NOR Flash
  5: Settings
  6: Shell
  7: Reset
  ```

- Press ***3*** and then ***Enter*** in order to boot the board from network.

4.7 Custom Boot Setup

You may have custom boot requirements that are not covered by the four available boot files (*nand*, *net*, *mmc*, *spi*). If this is the case you can create your own custom boot entry specifying the kernel and root filesystem location.

- First create your own boot entry in *barebox*, for example named "custom":

  ```
  bootloader$ edit /env/boot/custom
  ```

- Use the following template to specify the location of the *Linux* kernel and root filesystem.

  ```
  #!/bin/sh

  global.bootm.image="<kernel_loc_bootm.image>"
  global.bootm.oftree="<dts_loc_bootm.oftree>"

  nfsroot="<nfs_root_path>"
  bootargs-ip
  /env/config-expansions

  global.linux.bootargs.dyn.root="<rootfs_loc_dyn.root>"
  ```

Please note that the text in <> such as <*kernel_loc_bootm.image*>, <*rootfs_loc_dyn.root*>, and <*nfs_root_path*> are intended to be replaced with user specific values as described in the following.

- <kernel_loc_bootm.image> specifies the location of the *Linux* kernel image and can be:

  ```
  /dev/nand0.root.ubi.kernel - To boot the Linux kernel from NAND Flash
  /dev/m25p0.kernel - To boot the Linux kernel form SPI NOR Flash
  /mnt/tftp/linuximage - To boot the Linux kernel via TFTP
  /boot/linuximage - To boot the Linux kernel from SD/MMC card
  ```

- <dts_loc_bootm.oftree> specifies the location of the device tree binary and can be:

  ```
  /dev/nand0.root.ubi.oftree - To boot the device tree binary from NAND
                               Flash
  /dev/m25p0.oftree - To boot the device tree binary from SPI NOR Flash
  /mnt/tftp/oftree - To boot the device tree binary via TFTP
  /boot/oftree - To boot the device tree binary from SD/MMC card
  ```

- <rootfs_loc_dyn.root> specifies the location of the root filesystem and can be:

  ```
  root=ubi0:root ubi.mtd=root rootfstype=ubifs - To mount the root file-
                                                 system from NAND Flash
  root=/dev/nfs nfsroot=$nfsroot,vers=3,udp rw consoleblank=0
                        - To mount the root filesystem via NFS
  root=/dev/mmcblk0p2 rootwait - To mount the root filesystem from
                                 SD/MMC card
  ```

- <nfs_root_path> is only required if mounting the root filesystem from NFS is desired. Replace with the following:

  ```
  nfsroot="/home/${global.user}/nfsroot/${global.hostname}"
  ```

- After completing the modifications exit the editor using ***CTRL+D*** and save the environment:

  ```
  bootloader$ saveenv
  ```

- To run your custom boot entry from the *barebox* shell enter:

  ```
  bootloader$ boot custom
  ```

If you want to configure the bootloader for booting always from "custom", you need to create the */env/nv/boot.default* file. Here you can just insert "custom" and save it. Otherwise the boot sources and boot order is defined in */env/init/bootsource*.

5 Updating the Software

In this chapter we explain how to use the *barebox* bootloader to update the images in NAND and SPI NOR Flash.

5.1 Updating from Network

AM335x boards that have an Ethernet connector can be updated over network. As the bootloader only supports the first Ethernet port, the development host has to be connected to the first Ethernet port of the target. Be sure to set up the development host correctly. The IP needs to be set to 192.168.3.10, the netmask to 255.255.255.0, and a TFTP server needs to be available.

- Boot the system using any boot device available.
- Press any key to stop autoboot, then type:

```
bootloader$ ifup eth0
bootloader$ devinfo eth0
```

The Ethernet interfaces should be configured like this:

```
ipaddr=192.168.3.11
netmask=255.255.255.0
gateway=192.168.3.10
serverip=192.168.3.10
```

If a DHCP server is available, it is also possible to set:

```
ip=dhcp
```

If you need to change something:

- Type:

```
bootloader$ edit /env/network/eth0
```

- Edit the settings, save them by leaving the editor with ***CTRL+D*** and type:

```
bootloader$ saveenv
```

- Reboot the board.

5.1.1 Updating NAND Flash from Network

To update the second stage bootloader MLO you may use the *barebox_update* command. This provides a handler which automatically erases and flashes copies of the MLO image into the first four blocks of the NAND Flash. This makes the system more robust against ECC issues. If one block is corrupted the ROM loader does use the next block.

- Type:

```
bootloader$ barebox_update -t MLO.nand /mnt/tftp/MLO
```

On startup the TFTP server is automatically mounted to */mnt/tftp*. So copying an image from TFTP to flash can be done in one step. Do not get confused when doing an *ls* on the */mnt/tftp* folder. The TFTP protocol does not support anything like *ls* so the folder will appear to be empty.

The *barebox* as third stage bootloader can also be updated with a *barebox_update* command. But it is only stored once in the flash.

- Type:

  ```
  bootloader$ barebox_update -t nand /mnt/tftp/barebox.bin
  ```

We recommend to also erase the environment of the old *barebox*. Otherwise the new *barebox* would use the old environment.

- First erase the old environment with:

  ```
  bootloader$ erase /dev/nand0.bareboxenv.bb
  ```

After erasing the environment, you have to reset your board. Otherwise the *barebox* still uses the old environment.

- To reset your board in order to get the new *barebox* running type:

  ```
  bootloader$ reset
  ```

- Create UBI volumes for *Linux* kernel, *oftree* and root filesystem in NAND:

  ```
  bootloader$ ubiformat /dev/nand0.root
  bootloader$ ubiattach /dev/nand0.root

  bootloader$ ubimkvol -t static /dev/nand0.root.ubi kernel 8M
  bootloader$ ubimkvol -t static /dev/nand0.root.ubi oftree 1M
  bootloader$ ubimkvol -t dynamic /dev/nand0.root.ubi root 0
  ```

- Now get the *Linux* kernel and *oftree* from your TFTP server and store it also into the NAND Flash with:

  ```
  bootloader$ ubiupdatevol /dev/nand0.root.ubi.kernel
                                                     /mnt/tftp/linuximage
  bootloader$ ubiupdatevol /dev/nand0.root.ubi. oftree /mnt/tftp/oftree
  ```

- For flashing *Linux*'s root filesystem to NAND, please use:

  ```
  bootloader$ cp -v /mnt/tftp/root.ubifs /dev/nand0.root.ubi.root
  ```

- Change the boot configuration of your board to NAND boot if necessary, and reset your board.

5.1.2 Updating SPI NOR Flash from Network

The MLO for the SPI NOR Flash has to be byte-swapped. There is a handler for the *barebox_update* command to make this on the fly. Thus a special MLO image is not required.

- To update the second stage bootloader in the SPI Flash from network do the following:

  ```
  bootloader$ barebox_update -t MLO.spi /mnt/tftp/MLO
  ```

- After that, use the following command to flash the *barebox*:

  ```
  bootloader$ barebox_update -t spi /mnt/tftp/barebox.bin
  ```

We recommend to also erase the environment of the old *barebox*. Otherwise the new *barebox* would use the old environment.

- First erase the old environment with:

  ```
  bootloader$ erase /dev/m25p0.bareboxenv
  ```

After erasing the environment, you have to reset your board. Otherwise the *barebox* still uses the old environment.

- To reset your board in order to get the new *barebox* running type:

  ```
  bootloader$ reset
  ```

- The kernel and *oftree* are then updated with the regular *erase* and *cp* commands:

  ```
  bootloader$ erase /dev/m25p0.kernel
  bootloader$ cp /mnt/tftp/linuximage /dev/m25p0.kernel
  bootloader$ erase /dev/m25p0.oftree
  bootloader$ cp /mnt/tftp/oftree /dev/m25p0.oftree
  ```

- Change the boot configuration of your board to NOR boot if necessary, and reset your board.

The root filesystem is too big to fit into the SPI NOR Flash. So the default configuration when booting from SPI is to take the root filesystem from NAND Flash.

When booting from SPI Flash the AM335x expects the MLO in a big endian format. The *barebox_update* command with the MLO.spi type, takes the MLO image and converts it on the fly to the correct format and flashes it to SPI. If you can not use the *barebox_update* command you can take directly the *MLO.spi* file which also drops out of the image build.

5.2 Updating from SD Card

To update an AM335x board from SD card the SD card used for updating needs to be mounted after the board is powered and the boot sequence is stopped on the bootloader prompt. If the board is booted from SD card the card is already mounted automatically under */boot*. In case the board is booted from any other device the SD card needs to be mounted manually with:

```
bootloader$ mkdir /boot
bootloader$ mmc0.probe=1
bootloader$ mount /dev/mmc0.0 /boot
```

5.2.1 Updating NAND Flash from SD Card

To update the images on the NAND Flash from SD card basically the same commands as updating from TFTP are used with just the path parameters adapted.

- Type:

  ```
  bootloader$ barebox_update -t MLO.nand /boot/MLO
  bootloader$ barebox_update -t nand /boot/barebox.bin
  ```

We recommend to also erase the environment of the old *barebox*. Otherwise the new *barebox* would use the old environment.

- First erase the old environment with:

  ```
  bootloader$ erase /dev/nand0.bareboxenv.bb
  ```

After erasing the environment, you have to reset your board. Otherwise the *barebox* still uses the old environment.

- To reset your board in order to get the new *barebox* running type:

  ```
  bootloader$ reset
  ```

- Create UBI volumes for *Linux* kernel, *oftree* and root filesystem in NAND:

  ```
  bootloader$ ubiformat /dev/nand0.root
  bootloader$ ubiattach /dev/nand0.root
  ```

  ```
  bootloader$ ubimkvol -t static /dev/nand0.root.ubi kernel 8M
  bootloader$ ubimkvol -t static /dev/nand0.root.ubi oftree 1M
  bootloader$ ubimkvol -t dynamic /dev/nand0.root.ubi root 0
  ```

- Now get the *Linux* kernel and *oftree* from your SD card and store it also into the NAND Flash with:

  ```
  bootloader$ ubiupdatevol /dev/nand0.root.ubi.kernel /boot/linuximage
  bootloader$ ubiupdatevol /dev/nand0.root.ubi. oftree /boot/oftree
  ```

- For flashing *Linux*'s root filesystem to NAND, please use:

  ```
  bootloader$ cp /boot/root.ubifs /dev/nand0.root.ubi.root
  ```

- Change the boot configuration of your board to NAND boot if necessary, and reset your board.

5.2.2 Updating SPI NOR Flash from SD Card

To update the images on the SPI Flash from SD card basically the same commands as updating from TFTP are used with just the path parameters adapted.

- Type:

```
bootloader$ barebox_update -t MLO.spi /boot/MLO
bootloader$ barebox_update -t spi /boot/barebox.bin
```

We recommend to also erase the environment of the old *barebox*. Otherwise the new *barebox* would use the old environment.

- First erase the old environment with:

```
bootloader$ erase /dev/m25p0.bareboxenv
```

After erasing the environment, you have to reset your board. Otherwise the *barebox* still uses the old environment.

- To reset your board in order to get the new *barebox* running type:

```
bootloader$ reset
```

- The kernel and *oftree* are then updated with the regular *erase* and *cp* commands:

```
bootloader$ erase /dev/m25p0.kernel
bootloader$ cp /boot/linuximage /dev/m25p0.kernel
bootloader$ erase /dev/m25p0.oftree
bootloader$ cp /boot/oftree /dev/m25p0.oftree
```

- Change the boot configuration of your board to NOR boot if necessary, and reset your board.

The root filesystem is too big to fit into the SPI NOR Flash. So the default configuration when booting from SPI is to take the root filesystem from NAND Flash.

When booting from SPI Flash the AM335x expects the MLO in a big endian format. The *barebox_update* command with the MLO.spi type, takes the MLO image and converts it on the fly to the correct format and flashes it to SPI. If you can not use the *barebox_update* command you can take directly the *MLO.spi* file which also drops out of the image build.

5.3 Updating from USB Flash Drive

It is not possible to boot an AM335x board from an USB Flash Drive. However, the *barebox* bootloader does support USB Flash Drives and is thus also able to update NAND and SPI NOR Flashes from them.

- To update from USB Flash Drive boot the system from any bootable device.
- Press any key to stop autoboot.
- Plug in an USB Flash Drive containing the images. *Barebox* supports *vfat* and *ext4* (read only).
- Check the USB bus with following command:
  ```
  bootloader$ usb
  ```

This should print some output like:

```
USB: scanning bus for devices...
Bus 001 Device 001: ID 058f:6387 Intenso Rainbow Line
Using index 0 for the new disk
1 USB Device(s) found
```

If the USB Flash Drive is inserted during boot, it might not be detected the first time. In this case execute the *usb* command twice. This will create a device *disk0* under */dev* and the partitions which can be listed with:

```
bootloader$ ls /dev/disk0*
```

You should see:

```
/dev/disk0            /dev/disk0.0
```

- Now mount the single partitions with:
  ```
  bootloader$ mkdir /mnt/disk
  bootloader$ mount /dev/disk0.0 /mnt/disk/
  ```

5.3.1 Updating NAND Flash from USB Flash Drive

To update the images on the NAND Flash from USB Flash Drive basically the same commands as updating from TFTP are used with just the path parameters adapted.

- Type:

```
bootloader$ barebox_update -t MLO.nand /mnt/disk/MLO
bootloader$ barebox_update -t nand /mnt/disk/barebox.bin
```

We recommend to also erase the environment of the old *barebox*. Otherwise the new *barebox* would use the old environment.

- First erase the old environment with:

```
bootloader$ erase /dev/nand0.bareboxenv.bb
```

After erasing the environment, you have to reset your board. Otherwise the *barebox* still uses the old environment.

- To reset your board in order to get the new *barebox* running type:

```
bootloader$ reset
```

- Create UBI volumes for *Linux* kernel, *oftree* and root filesystem in NAND:

```
bootloader$ ubiformat /dev/nand0.root
bootloader$ ubiattach /dev/nand0.root

bootloader$ ubimkvol -t static /dev/nand0.root.ubi kernel 8M
bootloader$ ubimkvol -t static /dev/nand0.root.ubi oftree 1M
bootloader$ ubimkvol -t dynamic /dev/nand0.root.ubi root 0
```

- Now get the *Linux* kernel and *oftree* from your USB Flash Drive and store it also into the NAND Flash with:

```
bootloader$ ubiupdatevol /dev/nand0.root.ubi.kernel
                                                  /mnt/disk/linuximage
bootloader$ ubiupdatevol /dev/nand0.root.ubi. oftree /mnt/disk/oftree
```

- For flashing *Linux*'s root filesystem to NAND, please use:

```
bootloader$ cp /mnt/disk/root.ubifs /dev/nand0.root.ubi.root
```

- Change the boot configuration of your board to NAND boot if necessary, and reset your board.

5.3.2 Updating SPI NOR Flash from USB Flash Drive

To update the images on the SPI Flash from USB Flash Drive basically the same commands as updating from TFTP are used with just the path parameters adapted.

- Type:

```
bootloader$ barebox_update -t MLO.spi /mnt/disk/MLO
bootloader$ barebox_update -t spi /mnt/disk/barebox.bin
```

We recommend to also erase the environment of the old *barebox*. Otherwise the new *barebox* would use the old environment.

- First erase the old environment with:

```
bootloader$ erase /dev/m25p0.bareboxenv
```

After erasing the environment, you have to reset your board. Otherwise the *barebox* still uses the old environment.

- To reset your board in order to get the new *barebox* running type:

```
bootloader$ reset
```

- The kernel and *oftree* are then updated with the regular *erase* and *cp* commands:

```
bootloader$ erase /dev/m25p0.kernel
bootloader$ cp /mnt/disk/linuximage /dev/m25p0.kernel

bootloader$ erase /dev/m25p0.oftree
bootloader$ cp /mnt/disk/oftree /dev/m25p0.oftree
```

- Change the boot configuration of your board to NOR boot if necessary, and reset your board.

The root filesystem is too big to fit into the SPI NOR Flash. So the default configuration when booting from SPI is to take the root filesystem from NAND Flash.

When booting from SPI Flash the AM335x expects the MLO in a big endian format. The *barebox_update* command with the MLO.spi type, takes the MLO image and converts it on the fly to the correct format and flashes it to SPI. If you can not use the *barebox_update* command you can take directly the *MLO.spi* file which also drops out of the image build.

5.4 Troubleshooting NAND Flash Update

5.4.1 Updating from BSP-Yocto-AM335x-PD15.x.x or BSP-Yocto-phyCORE-AM335x-R2-PD16.1.x

In the BSP-Yocto-AM335x-PD16.2.0 the NAND Flash's partition layout has changed. Instead of having a separate *Linux* image and *oftree* partition the images are now part of the root filesystem partition. Inside the root UBI two new static volumes are being created. So the kernel and *oftree* are now kept more secure and protected with the wear leveling of the UBI. There is also a *barebox_backup* partition being introduced in the BSP-Yocto-AM335x-PD16.2.0 release. To update the BSP use one of the following two ways:

- Boot the new BSP from a different bootsource than NAND and follow the update instructions in the previous sections.
- Update the BSP using an older release. Update the MLO and the *barebox* with the *barebox_update* command and delete the environment partition as described in the update sections above. Reboot the board from NAND and run again the *barebox_update* command for the *barebox.bin* to ensure that the new *barebox_backup* partition is being written. Continue with updating the kernel, device tree and root filesystem as described in the previous sections.

5.4.2 Updating from a PTXdist BSP

When updating the images on the NAND Flash from a *PTXdist*-based release (*Linux Kernel 3.2*) to a *Yocto*-based release, there might be an *UBIFS* issue coming up when mounting the root filesystem. The old *Linux* kernel had an issue with subpage write which required to set the VID header offset to 2048. This is not needed any more. But because of that the UBI parameters changed which requires a different call:

```
bootloader$ ubiformat -y -f /dev/nand0.root -s 512
bootloader$ ubiattach /dev/nand0.root
bootloader$ ubimkvol /dev/nand0.root.ubi root 0
bootloader$ cp /mnt/disk/root.ubifs /dev/nand0.root.ubi.root
```

6 Device Tree (DT)

6.1 Introduction

The following text describes briefly the Device Tree and can be found in the *Linux* kernel (*linux/Documentation/devicetree/usage-model.txt*).

"The "Open Firmware Device Tree", or simply Device Tree (DT), is a data structure and language for describing hardware. More specifically, it is a description of hardware that is readable by an operating system so that the operating system doesn't need to hard code details of the machine.
Structurally, the DT is a tree, or acyclic graph with named nodes, and nodes may have an arbitrary number of named properties encapsulating arbitrary data. A mechanism also exists to create arbitrary links from one node to another outside of the natural tree structure.
Conceptually, a common set of usage conventions, called 'bindings', is defined for how data should appear in the tree to describe typical hardware characteristics including data busses, interrupt lines, GPIO connections, and peripheral devices."

The kernel is a really good source for a DT introduction. An overview of the device tree data format can be found on the device tree usage page at devicetree.org:
http://devicetree.org/Device_Tree_Usage

6.2 Phytec AM335x BSP Device Tree Concept

The following sections explain some rules we have defined on how to set up device trees in first place for our AM335x SoC based boards.

6.2.1 Basic DT Structure

The module include file *Modul .dtsi* contains all devices which are mounted on the module, such as NAND Flash and RAM. Devices which come from the AM335x SoC but are just routed down to the carrier board are not part of the *Module .dtsi,* but are included in the *Carrierboard .dtsi*. The *Board .dts* includes the carrier board and module nodes. It also adds partition tables and enables all hardware configurable nodes of the carrier board or the module. I.e. the *Board .dts* shows the special characteristics of the board configuration. For example, there are phyCORE-AM335x SOMs which may or may not have an SPI NOR Flash mounted. Hence, the SPI NOR Flash is enabled (if available) in the *Board .dts* and not in the *Module .dtsi*.

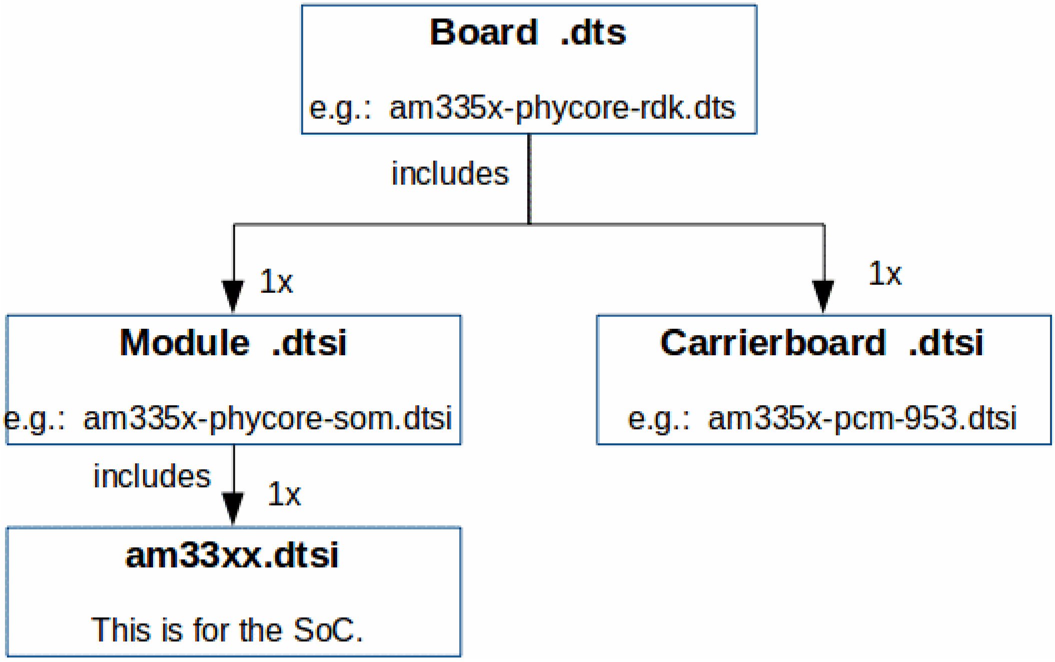


Figure 1: Basic DT Structure of Phytec AM335x Boards

To make starting of the development easier, we have also created different device tree include files for the various module options. E.g. there is already an *am335x-phycore-som-eeprom.dtsi* which has the EEPROM enabled but not the SPI NOR Flash and the RTC. So you can include the appropriate *Module .dtsi* to start directly with the development of the *Carrierboard .dtsi*.

6.2.2 Expansion Boards and Displays

Different expansion boards can be mounted on different carrier boards. Some carrier boards may have several connectors to support more than one expansion board at a time. This requires a separate device tree for every expansion board / carrier board combination.

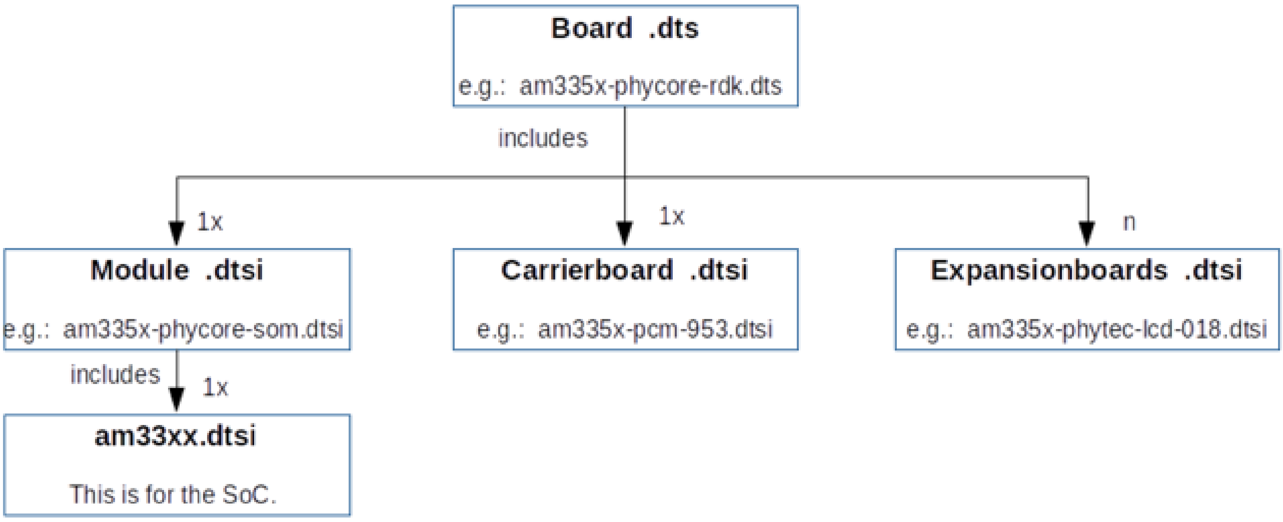


Figure 2: DTS Structure of Expansion Boards

To facilitate the use of expansion boards we created an *Expansionboards .dtsi* for every expansion board/ carrier board / module combination. And then included all possible extensions to a board file, but disabled them. To use an expansion board it can be enabled in the *Board .dts,* or from the bootloader.

6.2.3 Switching Expansion Boards and Displays

Disconnect all power before connecting an expansion board. After you plugged in the board, the software support can be activated in the bootloader without recompiling and flashing the images.

Here is a simple example on how to switch from *am335x-wega-peb-av-01* (HDMI Expansion Board) to *am335x-phytec-lcd-018-peb-av-02* (Display Expansion Board) on the phyBOARD-Wega using the *barebox* bootloader.

The configuration for the currently selected expansion board *am335x-wega-peb-av-01* can be found in *env/config-expansions*.

```
#!/bin/sh
. /env/expansions/am335x-wega-peb-av-01
```

To switch to another expansion board the file *config-expansions* in the *barebox* environment must be edited.

- To switch to *am335x-phytec-lcd-018-peb-av-02*, modify the *env/config-expansions* file and add, or uncomment the text according to the expansion board used. E.g. for the PEB-AV-02 on the phyBOARD-Wega:

  ```
  #!/bin/sh
  . /env/expansions/am335x-phytec-lcd-018-peb-av-02
  ```

config-expansions is called within each bootsource script (*/env/boot/**) and will be executed before every boot process. This will cause the *barebox* to modify the DT used before the boot process. Information on which DT nodes are necessary for the expansion board can be found in the expansion configuration files.

am335x-wega-peb-av-01:

```
of_fixup_status /hdmi
of_fixup_status /ocp/lcdc@0x4830e000
```

am335x-phytec-lcd-018-peb-av-02:

```
of_fixup_status /panel
of_fixup_status /backlight
of_fixup_status /ocp/lcdc@0x4830e000/
of_fixup_status /ocp/epwmss@48304000/
of_fixup_status /ocp/epwmss@48304000/ecap@48304100/
of_fixup_status /ocp/i2c@44e0b000/touchscreen@38/
```

of_fixup_status is a *barebox* command and will enable a given DT node.

6.2.4 Handle the Different Displays

If you have chosen a display as expansion you have to select the appropriate display timings in the device tree.

- Look at the following lines at the end of *config-expansions*:

  ```
  #7.0" display
  #of_display_timings -S /panel/display-timings/ETM0700G0DH6

  #5.7" display
  #of_display_timings -S /panel/display-timings/ETMV570G2DHU

  #4.3" display
  #of_display_timings -S /panel/display-timings/ETM0430G0DH6
  ```

- Uncomment the *of_display_timings -S* ... command for your screen size and save the file and environment.

Beside the display timings, you have to choose the right touchscreen type. Capacitive touchscreens are the standard touchscreens used with our boards. Thus, if you want to use a resistive touchscreen, you have to choose a modified board file in the *env/config-expansions*. The board files for resistive touchscreens all have the suffix *-res*.

Example: If you have the *am335x-phytec-lcd-018-peb-av-02* board, change the *env/config-expansions* from:

```
#!/bin/sh
. /env/expansions/am335x-phytec-lcd-018-peb-av-02
```

to:

```
#!/bin/sh
. /env/expansions/am335x-phytec-lcd-018-peb-av-02-res
```

6.2.5 Pre-Bootloader's DT Handling

The pre-bootloader (MLO) uses only a very simple and small device tree, which also has most nodes deactivated by default. The MLO detects the boot device and activates this node only to load the *barebox* bootloader. The RAM setup is done in an early stage were no device tree support is available.

6.2.6 Bootloader's DT Modifications

The bootloader loads the device tree blob, a separate binary which includes the hardware description (*section 7*). Then it will modify the memory node loaded at runtime. Thus, there is no need to handle different RAM sizes in the device tree. However, there should be always a memory node to ensure that the RAM size is set even if e.g. a different bootloader without this feature is used. For this purpose a memory node dummy which sets the RAM size to the size of the RDK module's memory is added in the *Module.dsti*.

```
/* This is a dummy node. Will be set by a bootloader */
memory {
        device_type = "memory";
        reg = <0x80000000 0x20000000>; /* 512 MB */
};
```

The memory node is not the only node, which is modified by the bootloader. Display nodes and nodes of expansion boards can be enabled (*section 6.2.3 "Switching Expansion Boards and Displays"*), whereas the partition table in the kernel device tree is modified by the bootloader. Because of that it is not possible to use our device trees without modifications with another bootloader, than the *barebox* bootloader from our unified BSP.

The bootloader itself contains also a device tree in the bootloader image which differs from the one used by the kernel. So the *barebox* loads to start *Linux*, the kernel image and a second device tree blob file.

> The unified BSP contains device tree (DT) sources for the *barebox* bootloader and for the *Linux* kernel. The bootloader DT holds only the absolutely necessary hardware description for the basic board bring-up and is also using a different DT model. Make sure you are working with the right DT.

The following boot script for booting from SD card is an example for the *barebox* configuration:

- Display the script with:

```
bootloader$ cat env/boot/mmc
```

```
#!/bin/sh

global.bootm.image=/boot/linuximage # kernel image
global.bootm.oftree=/boot/oftree   # DTB
```

7 Accessing Peripherals

The following sections provide an overview of the supported hardware components and their corresponding operating system drivers. Further changes can be ported upon customer request.

To find out which boards and modules are supported by the release of Phytec's AM335x unified BSP described herein, visit our web page at *http://www.phytec.de/produkte/software/yocto/phytec-unified-yocto-bsp-releases/* and click the corresponding BSP release. Now you can find all hardware supported in the columns "Hardware Article Number" and the correct machine name in the corresponding cell under "Machine Name".

To achieve maximum software re-use, the *Linux* kernel offers a sophisticated infrastructure, layering software components into board specific parts. The BSP tries to modularize the kit features as far as possible, which means that when a customized baseboard, or even a customer specific module is developed, most of the software support can be re-used without error-prone copy-and-paste. The kernel code corresponding to the supported hardware can be found in the device trees (DT) *linux/arch/arm/boot/dts/*.dts[i]*.

In fact, software re-use is one of the most important features of the *Linux* kernel and especially of the ARM implementation, which always had to fight with an insane number of possibilities of the System-on-Chip CPUs.

The whole board specific hardware is described in DTs and is not part of the kernel image itself. The hardware description is in its own separate binary, called Device Tree Blob (DTB) (*section 6.2.6 "Bootloader's DT Modifications"*).

Please read also section *6.2 "Phytec AM335x BSP Device Tree Concept"* to get an understanding of our unified AM335x BSP device tree model.

The following sections provide an overview of the supported hardware components and their operating system drivers on the AM335x platform.

7.1 AM335x Pin Muxing

The AM335x SoC contains many peripheral interfaces. In order to reduce package size and lower overall system cost while maintaining maximum functionality, many of the AM335x terminals can multiplex up to eight signal functions. Although there are many combinations of pin multiplexing that are possible, only a certain number of sets, called IO sets, are valid due to timing limitations. These valid IO sets were carefully chosen to provide many possible application scenarios for the user.

The pin muxing tool helps developers to find the correct IO sets for their application (*http://www.ti.com/tool/pinmuxtool*).

Also see the following datasheet for a detailed pin configuration description (*http://www.ti.com/lit/ds/symlink/am3359.pdf*).

AM335x's pin register addresses can be found in the AM335x Technical Reference Manual, chapter 9 "Control Module" (*http://www.ti.com/lit/ug/spruh73o/spruh73o.pdf*).

The IO set configuration, also called muxing, is done in the Device Tree. The driver *pinctrl-single* reads the DT's node "pinctrl-single,pins" and does the appropriate pin muxing.

The following is an example of the pin muxing of the UART0 device in *am335x-pcm-953.dtsi*:

```
&am33xx_pinmux {
        uart0_pins: pinmux_uart0 {
                pinctrl-single,pins = <
                        0x170 (PIN_INPUT_PULLUP | MUX_MODE0)
                                                /* uart0_rxd.uart0_rxd */
                        0x174 (PIN_OUTPUT_PULLDOWN | MUX_MODE0)
                                                /* uart0_txd.uart0_txd */
                >;
        };
};
```

All pin configuration nodes have to be a declared in the *am33xx_pinmux* node. The pin configuration for the *pinctrl-single* nodes is specified by combined pairs of pinctrl register offset and value within *pinctrl-single,pins*. Only the bits specified in *pinctrl-single,function-mask* are updated.

- E.g., to set a pin for a device you can write:

  ```
  pinctrl-single,pins = <0xdc 0x118>;
  ```

  Where 0xdc is the offset from the pinctrl register base address for the device pinctrl register, and 0x118 contains the desired value of the pinctrl register.

AM335x's pinctrl register base address is 0x44e10800 (*am33xx.dtsi)*.

More information about the *pinctrl-single* driver binding can be found in the kernel documentation *linux/Documentation/devicetree/bindings/pinctrl/pinctrl-single.txt*

7.2 Serial TTYs

The AM335x SoC provides up to 6 so called UART units. Phytec boards support different numbers of these UART units.

tty00 is always used as the standard console output.

- From the command line prompt of *Linux* user space you can easily check the availability of other UART interfaces with:

  ```
  target$ echo "test" > /dev/tty01
  ```

  Be sure that the baud rate is set correctly on host and target side.

In order to get the currently configured baud rate, you can use the command *stty* on the target. The following example shows how to copy all serial settings from tty00 (the standard console on most AM335x boards) to tty01.

- First get the current parameters with:

  ```
  target$ stty -F /dev/tty00 -g
  ```

```
5500:5:1cb2:a3b:3:1c:7f:15:4:0:1:0:11:13:1a:0:12:f:17:16:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0:0
```

- Now use the output from the *stty* command above as argument for the next command:

  ```
  target$ stty -F /dev/ tty01
  5500:5:1cb2:a3b:3:1c:7f:15:4:0:1:0:11:13:1a:0:12:f:17:16:0:0:0:0:0:0:0
  :0:0:0:0:0:0:0:0:0
  ```

Here is an example from *am335x- pcm-953.dtsi*:

```
/* UARTs */
&am33xx_pinmux {
        uart0_pins: pinmux_uart0 {
                pinctrl-single,pins = <
                        0x170 (PIN_INPUT_PULLUP | MUX_MODE0)
                                                /* uart0_rxd.uart0_rxd */
                        0x174 (PIN_OUTPUT_PULLDOWN | MUX_MODE0)
                                                /* uart0_txd.uart0_txd */
                >;
        };

        uart1_pins: pinmux_uart1 {
                pinctrl-single,pins = <
                        0x180 (PIN_INPUT_PULLUP | MUX_MODE0)
                                                /* uart1_rxd.uart1_rxd */
                        0x184 (PIN_OUTPUT_PULLDOWN | MUX_MODE0)
                                                /* uart1_txd.uart1_txd */
```

```
                        0x178 (PIN_OUTPUT_PULLDOWN | MUX_MODE0)
                                              /* uart1_ctsn.uart1_ctsn */
                        0x17C (PIN_INPUT_PULLUP | MUX_MODE0)
                                              /* uart1_rtsn.uart1_rtsn */
                >;
        };

        uart2_pins: pinmux_uart2 {
                pinctrl-single,pins = <
                        0x12C (PIN_INPUT_PULLUP | MUX_MODE1)
                                              /* mii1_tx_clk.uart2_rxd */
                        0x130 (PIN_OUTPUT_PULLDOWN | MUX_MODE1)
                                              /* mii1_rx_clk.uart2_txd */
                >;
        };

        uart3_pins: pinmux_uart3 {
                pinctrl-single,pins = <
                        0x134 (PIN_INPUT_PULLUP | MUX_MODE1)
                                                /* mii1_rxd3.uart3_rxd */
                        0x138 (PIN_OUTPUT_PULLDOWN | MUX_MODE1)
                                                /* mii1_rxd2.uart3_txd */
                >;
        };
};

&uart0 {
        pinctrl-names = "default";
        pinctrl-0 = <&uart0_pins>;
};

&uart1 {
        pinctrl-names = "default";
        pinctrl-0 = <&uart1_pins>;
};

&uart2 {
        pinctrl-names = "default";
        pinctrl-0 = <&uart2_pins>;
};

&uart3 {
        pinctrl-names = "default";
        pinctrl-0 = <&uart3_pins>;
};
```

7.2.1 RS-485

The phyBOARD-Regor provides one RS-485 interface derived from UART1. The following code snippet can be found in the *am335x-regor.dtsi*:

```
/* RS485 – UART1 */
&am33xx_pinmux {
       uart1_rs485_gpio_pin: pinmux_uart1_rs485_pins {
               pinctrl-single,pins = <
                       0x180 (PIN_INPUT_PULLUP | MUX_MODE0)
                                                /* uart1_rxd.uart1_rxd */
                       0x184 (PIN_OUTPUT_PULLDOWN | MUX_MODE0)
                                                /* uart1_txd.uart1_txd */
                       0x17C (PIN_INPUT_PULLUP | MUX_MODE7)
                                                /* uart1_rtsn.gpio0_13 */
               >;
       };
};

&uart1 {
        pinctrl-names = "default";
        pinctrl-0 = <&uart1_rs485_pins>;
        status = "okay";
        rs485-rts-active-high;
        rts-gpio = <&gpio0 13 GPIO_ACTIVE_HIGH>;
        rs485-rts-delay = <0 0>;
        linux,rs485-enabled-at-boot-time;
};
```

For first easy testing the RS-485 port must be configured.

- Execute:

  ```
  target$ stty -F /dev/ttyO2 -echo -echoe -echok -echoctl -echoke 115200
  ```

Now, you can apply the *echo* and *cat* command to /dev/ttyO2.

7.3 Network

The Ethernet features provided by our modules and boards vary (e.g.: 1 x 10/100 Mbit, 2 x 10/100 Mbit, gigabit or both).

However, all interfaces offer a standard *Linux* network port which can be programmed using the BSD socket interface.

The whole network configuration is handled by the *systemd-networkd* daemon. The relevant configuration files can be found on the target in */lib/systemd/network/* and also in the BSP in *meta-yogurt/recipes-core/systemd/systemd/*.

IP addresses can be configured within **.network* files. The default IP addresses and netmasks for eth0 and eth1 are :
eth0: 192.168.3.11/24
eth1: 192.168.4.11/24

In our configuration both interfaces are handled with separate MAC addresses. Thus, both interfaces have to be connected to different subnets. A different configuration, like switch mode is also possible.

The DT Ethernet setup is mostly split into two files, the module DT and the board specific DT.

Example: RDK phyCORE- AM335x

Module DT, *am335x-phycore-som.dtsi* :

```
/* Ethernet */
&am33xx_pinmux {
        ethernet0_pins: pinmux_ethernet0 {
                pinctrl-single,pins = <
                        0x10c (PIN_INPUT_PULLDOWN | MUX_MODE1)
                                              /* mii1_crs.rmii1_crs_dv */
                        0x110 (PIN_INPUT_PULLDOWN | MUX_MODE1)
                                             /* mii1_rxerr.rmii1_rxerr */
                        0x114 (PIN_OUTPUT | MUX_MODE1)
                                               /* mii1_txen.rmii1_txen */
                        0x124 (PIN_OUTPUT | MUX_MODE1)
                                               /* mii1_txd1.rmii1_txd1 */
                        0x128 (PIN_OUTPUT | MUX_MODE1)
                                               /* mii1_txd0.rmii1_txd0 */
                        0x13c (PIN_INPUT_PULLDOWN | MUX_MODE1)
                                               /* mii1_rxd1.rmii1_rxd1 */
                        0x140 (PIN_INPUT_PULLDOWN | MUX_MODE1)
                                               /* mii1_rxd0.rmii1_rxd0 */
                        0x144 (PIN_INPUT_PULLDOWN | MUX_MODE0)
                                          /* rmii1_refclk.rmii1_refclk */
                >;
        };

        mdio_pins: pinmux_mdio {
                pinctrl-single,pins = <
                        /* MDIO */
                        0x148 (PIN_INPUT_PULLUP | SLEWCTRL_FAST |
                               MUX_MODE0)      /* mdio_data.mdio_data */
                        0x14c (PIN_OUTPUT_PULLUP | MUX_MODE0)
                                                  /* mdio_clk.mdio_clk */
                >;
        };
};

&cpsw_emac0 {
        phy_id = <&davinci_mdio>, <0>;
        phy-mode = "rmii";
        dual_emac_res_vlan = <1>;
};

&davinci_mdio {
        pinctrl-names = "default";
        pinctrl-0 = <&mdio_pins>;
        status = "okay";
};

&mac {
        slaves = <1>;
        pinctrl-names = "default";
        pinctrl-0 = <&ethernet0_pins>;
        status = "okay";
};

&phy_sel {
        rmii-clock-ext;
};
```

Board specific DT, *am335x-pcm-953.dtsi* :

```
/* Ethernet */
&am33xx_pinmux {
        ethernet1_pins: pinmux_ethernet1 {
                pinctrl-single,pins = <
                        0x40 (PIN_OUTPUT_PULLDOWN | MUX_MODE2)
                                                /* gpmc_a0.rgmii2_tctl */
                        0x44 (PIN_INPUT_PULLDOWN | MUX_MODE2)
                                                /* gpmc_a1.rgmii2_rctl */
                        0x48 (PIN_OUTPUT_PULLDOWN | MUX_MODE2)
                                                 /* gpmc_a2.rgmii2_td3 */
                        0x4c (PIN_OUTPUT_PULLDOWN | MUX_MODE2)
                                                 /* gpmc_a3.rgmii2_td2 */
                        0x50 (PIN_OUTPUT_PULLDOWN | MUX_MODE2)
                                                 /* gpmc_a4.rgmii2_td1 */
                        0x54 (PIN_OUTPUT_PULLDOWN | MUX_MODE2)
                                                 /* gpmc_a5.rgmii2_td0 */
                        0x58 (PIN_OUTPUT_PULLDOWN | MUX_MODE2)
                                                /* gpmc_a6.rgmii2_tclk */
                        0x5c (PIN_INPUT_PULLDOWN | MUX_MODE2)
                                                /* gpmc_a7.rgmii2_rclk */
                        0x60 (PIN_INPUT_PULLDOWN | MUX_MODE2)
                                                 /* gpmc_a8.rgmii2_rd3 */
                        0x64 (PIN_INPUT_PULLDOWN | MUX_MODE2)
                                                 /* gpmc_a9.rgmii2_rd2 */
                        0x68 (PIN_INPUT_PULLDOWN | MUX_MODE2)
                                                /* gpmc_a10.rgmii2_rd1 */
                        0x6c (PIN_INPUT_PULLDOWN | MUX_MODE2)
                                                /* gpmc_a11.rgmii2_rd0 */
                >;
        };
};

&cpsw_emac1 {
        phy_id = <&davinci_mdio>, <2>;
        phy-mode = "rgmii";
        dual_emac_res_vlan = <2>;

        /* Register 260 (104h) - RGMII Clock and Control Pad Skew */
        rxc-skew-ps = <1400>;
        rxdv-skew-ps = <0>;
        txc-skew-ps = <1400>;
        txen-skew-ps = <0>;

        /* Register 261 (105h) - RGMII RX Data Pad Skew */
        rxd3-skew-ps = <0>;
        rxd2-skew-ps = <0>;
        rxd1-skew-ps = <0>;
        rxd0-skew-ps = <0>;
        /* Register 262 (106h) - RGMII TX Data Pad Skew */
        txd3-skew-ps = <0>;
        txd2-skew-ps = <0>;
        txd1-skew-ps = <0>;
        txd0-skew-ps = <0>;

        status = "okay";
};

&mac {
        slaves = <2>;
        pinctrl-names = "default";
        pinctrl-0 = <&ethernet0_pins &ethernet1_pins>;
        dual_emac;
};
```

7.4 CAN Bus

The phyCORE-AM335x provides a Controller Area Network (CAN) interface, which is supported by drivers using the proposed *Linux* standard CAN framework *SocketCAN*. Using this framework, CAN interfaces can be programmed with the BSD socket API.

The CAN bus offers a low-bandwidth, prioritized message fieldbus for serial communication between microcontrollers. Unfortunately, CAN was not designed with the ISO/OSI layer model in mind, so most CAN APIs available throughout the industry do not support a clean separation between the different logical protocol layers, as for example known from Ethernet.

The *SocketCAN* framework for *Linux* extends the BSD socket API concept towards CAN bus. Because of that, using this framework, the CAN interfaces can be programmed with the BSD socket API and behaves like an ordinary *Linux* network device, with some additional features special to CAN.

- E.g., use:

  ```
  target$ ifconfig -a
  ```

  to see if the interface is up or down, but the given MAC and IP addresses are arbitrary and obsolete.

- To get the information on can0 (which represents AM335x's CAN1) such as bit rate and error counters type:

  ```
  target$ ifconfig can0
  ```

The information for can0 will look like the following:

```
# ifconfig can0
can0     Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
         UP RUNNING NOARP  MTU:16  Metric:1
         RX packets:0 errors:0 dropped:0 overruns:0 frame:0
         TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:10
         RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
         Interrupt:71
```

The output contains a standard set of parameters also shown for Ethernet interfaces, so not all of these are necessarily relevant for CAN (for example the MAC address). The following output parameters contain useful information:

Field	Description
can0	Interface Name
NOARP	CAN cannot use ARP protocol
MTU	Maximum Transfer Unit
RX packets	Number of Received Packets
TX packets	Number of Transmitted Packets
RX bytes	Number of Received Bytes
TX bytes	Number of Transmitted Bytes
errors...	Bus Error Statistics

The CAN configuration is done in the *systemd* configuration file */lib/systemd/systemd-machine-units/can0.service*.

For a persistent change of e.g. the default bitrates change the configuration in the BSP under *meta-yogurt/recipes-core/systemd/systemd/can0.service.* For temporarily modifications change the *systemd* file in the root filesystem instead and rebuild the root filesystem.

```
[Unit]
Description=can0 interface setup

[Service]
Type=simple
RemainAfterExit=yes
ExecStart=/sbin/ip link set can0 up type can bitrate 500000
ExecStop=/sbin/ip link set can0 down

[Install]
WantedBy=basic.target
```

The *can0.service* is started after boot by default. You can start and stop it with:

```
target$ systemctl stop can0.service
target$ systemctl start can0.service
```

You can send messages with *cansend* or receive messages with *candump*:

```
target$ cansend can0 123#45.67
target$ candump can0
```

See `cansend --help` and `candump --help` messages for further information on options and usage.

To generate random CAN traffic for testing purpose use *cangen*:

```
target$ cangen
```

The corresponding kernel part can be found within the board specific DT, e.g. *am335x-pcm-953.dtsi*

```
/* CAN */
&am33xx_pinmux {
        dcan1_pins: pinmux_dcan1 {
                pinctrl-single,pins = <
                        0x180 (PIN_OUTPUT_PULLUP | MUX_MODE2)
                                            /* uart1_rxd.dcan1_tx_mux2 */
                        0x184 (PIN_INPUT_PULLUP | MUX_MODE2)
                                            /* uart1_txd.dcan1_rx_mux2 */
                >;
        };
};

&dcan1 {
        pinctrl-names = "default";
        pinctrl-0 = <&dcan1_pins>;
};
```

7.5 MMC/SD Card

All AM335x kits support a slot for Secure Digital Cards and Multi Media Cards to be used as general purpose block devices. These devices can be used in the same way as any other block device.

⚠	This kind of devices are hot pluggable, nevertheless you must pay attention not to unplug the device while it is still mounted. This may result in data loss.

After inserting an MMC/SD card, the kernel will generate new device nodes in /dev. The full device can be reached via its */dev/mmcblk0* device node, MMC/SD card partitions will show up in the following way:

```
/dev/mmcblk0p<Y>
```

<*Y*> counts as the partition number starting from 1 to the max. count of partitions on this device.

The partitions can be formatted with any kind of file system and also handled in a standard manner, e.g. the *mount* and *umount* command work as expected.

	• These partition device nodes will only be available if the card contains a valid partition table (”hard disk” like handling). If it does not contain one, the whole device can be used as a file system (”floppy” like handling). In this case */dev/mmcblk0* must be used for formatting and mounting. • The cards are always mounted as being writable.

DT configuration for the MMC/SD interface:

```
/* MMC */
&am33xx_pinmux {
        mmc1_pins: pinmux_mmc1_pins {
                pinctrl-single,pins = <
                        0x0F0 (PIN_INPUT_PULLUP | MUX_MODE0)
                                                /* mmc0_dat3.mmc0_dat3 */
                        0x0F4 (PIN_INPUT_PULLUP | MUX_MODE0)
                                                /* mmc0_dat2.mmc0_dat2 */
                        0x0F8 (PIN_INPUT_PULLUP | MUX_MODE0)
                                                /* mmc0_dat1.mmc0_dat1 */
                        0x0FC (PIN_INPUT_PULLUP | MUX_MODE0)
                                                /* mmc0_dat0.mmc0_dat0 */
                        0x100 (PIN_INPUT_PULLUP | MUX_MODE0)
                                                /* mmc0_clk.mmc0_clk   */
                        0x104 (PIN_INPUT_PULLUP | MUX_MODE0)
                                                /* mmc0_cmd.mmc0_cmd   */
                        0x160 (PIN_INPUT_PULLUP | MUX_MODE7)
                                                /* spi0_cs1.mmc0_sdcd  */
                >;
        };
};

&mmc1 {
        vmmc-supply = <&vcc3v3>;
        bus-width = <4>;
        pinctrl-names = "default";
        pinctrl-0 = <&mmc1_pins>;
        cd-gpios = <&gpio0 6 GPIO_ACTIVE_HIGH>;
        status = "okay";
};
```

7.6 NAND Flash

Phytec AM335x modules are equipped with raw NAND memory, which is used as media for storing *Linux*, DTB and root filesystem, including applications and their data files.

The NAND Flash is connected to the GPMC interface of the AM335x. The NAND Flash type and size is automatically being detected via the Open NAND Flash Interface (ONFI) during boot.

This type of media is managed by the UBI file system. This file system uses compression and decompression on the fly to increase the quantity of data stored.

From *Linux* user space the NAND Flash partitions start with */dev/mtdblock5*. Only the */dev/mtdblock13* on the Phytec modules has a file system, meaning that the other partitions cannot be mounted to the root filesystem. The only way to access them is by flashing a prepared flash image into the corresponding */dev/mtd* device node.

The partitions of a NAND Flash are defined in all DTs, but the *barebox* bootloader overwrites only the partitions of the kernel device tree. Thus, changing the partitions has to be done either in the *barebox* DT, or in the *barebox* environment. How to modify the partitions during runtime in the *barebox* environment is described in section *8.1 "Changing MTD Partitions"*.

Adding new partitions can be done with creating a new partition node in the *Module.dtsi*. The property *label* defines the name of the partition and the *reg* value the offset and size of a partition. Do not forget to update all following partitions when adding a partition, or changing a partition's size.

The kernel image and device tree are stored in the first 9 MB of the root partition (section *5.2.1 "Updating NAND Flash from SD Card"*)

The partitions are defined in the DT, e.g. *am335x-phytec-phycore-som.dtsi* in the *barebox*:

```
&nandflash {
        partition@0 {
                label = "xload";
                reg = <0x0 0x20000>;
        };
         partition@20000 {
                label = "xload_backup1";
                reg = <0x20000 0x20000>;
        };
        partition@40000 {
                label = "xload_backup2";
                reg = <0x40000 0x20000>;
        };
        partition@60000 {
                label = "xload_backup3";
                reg = <0x60000 0x20000>;
        };
        partition@80000 {
                label = "barebox";
                reg = <0x80000 0x80000>;
        };
         partition@100000 {
                label = "barebox_backup";
                reg = <0x100000 0x80000>;
        };
         partition@180000 {
                label = "bareboxenv";
                reg = <0x180000 0x40000>;
        };
         partition@1C0000 {
                label = "root";
                /*
                 * setting size to 0x0 here, size will be extended to
                 * end of nand flash while booting.
                 */
                reg = < 1C0000 0x0>;
        };
};
```

We also kept the partition nodes in the *Linux* Kernel DT as fallback.

7.7 GPIOs

Phytec boards have often a set of pins specially dedicated as user I/Os. Those pins are connected directly to AM335x pins. The processor has organized its GPIOs into four banks (GPIO0 – GPIO3) of 32 GPIOs each. Pins connected directly to the AM335x are muxed as GPIOs and are directly usable in *Linux* user space. *gpiochip0*, *gpiochip32*, *gpiochip64* and *gpiochip96* are the sysfs representation of these internal AM335x GPIO banks GPIO0 – GPIO3.

The GPIOs are identified as GPIO<X>_<Y> (e.g. GPIO3_7). <X> identifies the GPIO bank and counts from 0 to 3, while <Y> stands for the GPIO within the bank. <Y> is being counted from 0 to 31 (32 GPIOs on each bank).

By contrast, the *Linux* kernel uses a single integer to enumerate all available GPIOs in the system. The formula to calculate the right number is

Linux GPIO number <N> = <X> * 32 + <Y>

Accessing GPIOs from user space will be done using the sysfs path */sys/class/gpio/*.

- First you have to register the GPIO that you want to use by writing its numbers into the file *export, e.g.*:

```
target$ echo <N> > /sys/class/gpio/export
```

This will create a new subdirectory *gpio<N>* (for GPIO<X>_<Y> of the controller). The two files *direction* and *value* in the new subdirectory allow to control the GPIO.

- For example, to use the newly created GPIO <N> as input execute the following commands:

```
target$ echo in > /sys/class/gpio/gpio<N>/direction
target$ cat /sys/class/gpio/gpio<N>/value
```

⚠	Some of the user IOs are used for special internal functions on the carrier boards (e.g. GPIO1_8 and GPIO1_9 on the PCM-953 CB). Before using a user IO refer to the schematic, or the hardware manual of your board to ensure that it is not already in use. Otherwise please do not touch them to avoid malfunctioning of the CB.

7.7.1 Keys

With *gpio-keys* the *Linux* kernel can interpret GPIO signals as virtual keyboard events. Some carrier boards have buttons, which can be used with the *gpio-keys* driver. By pushing a button an interrupt is triggered which causes the system to handle the corresponding keyboard event.

- To display the key events in ASCII format use *evtest, e.g.*:

  ```
  target$ evtest /dev/input/event0
  ```

- With the *cat* command the raw output can be printed, e.g.:

  ```
  target$ cat /dev/input/event0
  ```

GPIO-Keys configuration in *am335x-pcm-953.dtsi*:

```
/* Misc */
&am33xx_pinmux {
        pinctrl-names = "default";
        pinctrl-0 = <&cb_gpio_pins>,

        cb_gpio_pins: pinmux_cb_gpio {
                pinctrl-single,pins = <
                        0x168 (PIN_OUTPUT_PULLDOWN | MUX_MODE7)
                                                 /* uart0_ctsn.gpio1_8 */
                        0x16C (PIN_OUTPUT_PULLDOWN | MUX_MODE7)
                                                 /* uart0_rtsn.gpio1_9 */
                >;
        };
};

/* User IO */
&am33xx_pinmux {
        user_buttons_pins: pinmux_user_buttons {
                pinctrl-single,pins = <
                        0x1E4 (PIN_INPUT_PULLUP | MUX_MODE7)
                                                       /* emu0.gpio3_7 */
                        0x1E8 (PIN_INPUT_PULLUP | MUX_MODE7)
                                                       /* emu1.gpio3_8 */
                >;
        };
};

&user_buttons {
        pinctrl-names = "default";
        pinctrl-0 = <&user_buttons_pins>;
        #address-cells = <1>;
        #size-cells = <0>;
```

```
        button@0 {
                label = "home";
                linux,code = <KEY_HOME>;
                gpios = <&gpio3 7 GPIO_ACTIVE_HIGH>;
                gpio-key,wakeup;
        };

        button@1 {
                label = "menu";
                linux,code = <KEY_MENU>;
                gpios = <&gpio3 8 GPIO_ACTIVE_HIGH>;
                gpio-key,wakeup;
        };
};
```

7.7.2 LEDs

In case that LEDs are being connected to GPIOs, you have the possibility to access them by a special LED driver interface. All LEDs will be accessible through the */sys/class/leds/* directory, where they appear with their DT label. Several attributes, such as the maximum brightness (*max_brightness*), or the current brightness (*brightness*), which can be every positive number less or equal to the maximum brightness, are assigned to each LED. Since most LEDs do not support hardware brightness they will be turned on by all non-zero brightness settings.

Here is a simple example for the PCM-953 CB:

- To get all LEDs available, type

  ```
  target$ ls /sys/class/leds
  ```

  which will result in:

  ```
  green:user      yellow:user
  ```

- To toogle the LEDs use

  ```
  target$ echo 255 > /sys/class/leds/green\:user/brightness
  ```

  to turn it ON, and

  ```
  target$ echo 0 > /sys/class/leds/green\:user/brightness
  ```

  to turn it OFF.

User I/O configuration in *am335x-pcm-953.dtsi*:

```
&am33xx_pinmux {
        user_leds_pins: pinmux_user_leds {
                pinctrl-single,pins = <
                        0x80 (PIN_OUTPUT_PULLDOWN | MUX_MODE7)
                                                 /* gpmc_csn1.gpio1_30 */
                        0x84 (PIN_OUTPUT_PULLDOWN | MUX_MODE7)
                                                 /* gpmc_csn2.gpio1_31 */
                >;
        };
};

&user_leds {
        pinctrl-names = "default";
        pinctrl-0 = <&user_leds_pins>;

        green {
                label = "green:user";
                gpios = <&gpio1 30 GPIO_ACTIVE_HIGH>;
                linux,default-trigger = "gpio";
                default-state = "on";
        };

        yellow {
                label = "yellow:user";
                gpios = <&gpio1 31 GPIO_ACTIVE_LOW>;
                linux,default-trigger = "gpio";
                default-state = "on";
        };
};
```

7.8 SPI Master

Most Phytec boards are equipped with a NOR Flash which connects to the AM335x's McSPI interface. The NOR Flash is suitable for booting (section "*Booting from SPI NOR Flash*").

From *Linux* user space the NOR Flash partitions start with *dev/mtdblock0*. There are currently five partitions: *barebox*, *barebox*-environment, *oftree*, *MLO* and kernel. Please note that there is no root file system partition on the NOR Flash.

The partitions of an SPI Flash are defined in all DTs, but the *barebox* bootloader overwrites only the partitions of the kernel device tree. Thus, changing the partitions has to be done either in the *barebox* DT, or in the *barebox* environment. How to modify the partitions during runtime in the *barebox* environment is described in section *8.1 "Changing MTD Partitions"*.

Adding new partitions can be done with creating a new partition node in the *Module.dts*. The property *label* defines the name of the partition and the *reg* value the offset and size of a partition. Do not forget to update all following partitions when adding a partition, or changing a partition's size.

The *serial_flash* node is defined inside of the SPI master node in the module DTs. The SPI node contains all devices connected to this SPI bus which is in this case only the SPI NOR Flash.

Definition of the SPI master node, e.g. in *am335x-phycore-som.dtsi:*

```
/* SPI Busses */
&am33xx_pinmux {
        spi0_pins: pinmux_spi0 {
                pinctrl-single,pins = <
                        0x150 (PIN_INPUT_PULLDOWN | MUX_MODE0)
                                                  /* spi0_clk.spi0_clk */
                        0x154 (PIN_INPUT_PULLDOWN | MUX_MODE0)
                                                  /* spi0_d0.spi0_d0   */
                        0x158 (PIN_INPUT_PULLUP | MUX_MODE0)
                                                  /* spi0_d1.spi0_d1   */
                        0x15c (PIN_INPUT_PULLUP | MUX_MODE0)
                                                  /* spi0_cs0.spi0_cs0 */
                >;
        };
};

&spi0 {
        pinctrl-names = "default";
        pinctrl-0 = <&spi0_pins>;
        status = "okay";
        serial_flash: m25p80@0 {
                compatible = "jedec,spi-nor";
                spi-max-frequency = <48000000>;
                reg = <0x0>;
                m25p,fast-read;
                #address-cells = <1>;
                #size-cells = <1>;

                partition@0 {
                        label = "xload";
                        reg = <0x0 0x20000>;
                };

                partition@1 {
                        label = "barebox";
                        reg = <0x20000 0x80000>;
                };

                partition@2 {
                        label = "bareboxenv";
                        reg = <0xa0000 0x20000>;
                };

                partition@3 {
                        label = "oftree";
                        reg = <0xc0000 0x20000>;
                };

                partition@4 {
                        label = "kernel";
                        reg = <0xe0000 0x0>;
                };
        };
};
```

The *am335x-phycore-som.dtsi* also includes an example for an *spidev* device. This node has been enabled in the phyBOARD-Wega RDK. *spidev* allows to access an SPI device directly from user space.

```
spidev0: spi@0 {
        compatible = "spidev";
        reg = <0x0>;
        spi-max-frequency = <48000000>;
        status = "disabled";
};
```

The partition layout is also available in the kernel as fallback.

7.9 I^{2}C Bus

The AM335x contains three multimaster fast-mode I^{2}C modules called I2C0, I2C1, and I2C2. Phytec boards provide plenty of different I^{2}C devices connected to the three I^{2}C modules of the AM335x. This chapter will describe the basic device usage and its DT representation of some of the I^{2}C devices integrated on our RDK boards.

General I^{2}C bus configuration (e.g. *am335x-phycore-som.dtsi*):

```
/* I2C Busses */
&am33xx_pinmux {
        i2c0_pins: pinmux_i2c0 {
                pinctrl-single,pins = <
                        0x188 (PIN_INPUT_PULLUP | MUX_MODE0)    /*
i2c0_sda.i2c0_sda */
                        0x18c (PIN_INPUT_PULLUP | MUX_MODE0)    /*
i2c0_scl.i2c0_scl */
                >;
        };
};

&i2c0 {
        pinctrl-names = "default";
        pinctrl-0 = <&i2c0_pins>;
        clock-frequency = <400000>;
        status = "okay";
        /* ... */
};
```

7.9.1 EEPROM

It is possible to read and write to the device directly in */sys/class/i2c-adapter/i2c-0/0-0052/eeprom*.

- E.g. to read and print the first 1024 bytes of the EEPROM as hex number, execute:

```
target$ dd if=/sys/class/i2c-adapter/i2c-0/0-0052/eeprom bs=1
                                              count=1024  | od -x
```

- E.g. to fill the whole EEPROM with zeros use:

```
target$ dd if=/dev/zero of=/sys/class/i2c-adapter/i2c-0/0-0052/eeprom
                                                      bs=4096 count=1
```

This operation takes some time, because the EEPROM is relatively slow.

DT representation, e.g. in *am335x-phycore-som.dtsi*:

```
&i2c0 {
        i2c_eeprom: eeprom@52 {
                compatible = "atmel,24c32";
                pagesize = <32>;
                reg = <0x52>;
                status = "disabled";
        };
};
```

7.9.2 RTC

RTCs can be accessed via */dev/rtc**. Because Phytec boards have often more than one RTC, there might be more than one RTC device file.

- To find out the name of the RTC device you can read its sysfs entry with:
  ```
  target$ cat /sys/class/rtc/rtc*/name
  ```

You will get for example:

```
rv4162c7
44e3e000.rtc
```

> This will list all RTCs including the non-I^{2}C RTCs. *Linux* assigns RTC devices IDs based on the device tree /aliases entries if present.

am335x-phycore-som.dtsi:

```
aliases {
        rtc0 = &i2c_rtc;
        rtc1 = &da830rtc;
};
```

As the time is set according to the value of rtc0 during system boot rtc0 should be always the RTC that is being backed up.

Date and time can be manipulated with the *hwclock* tool, using the *-w* (systohc) and *-s* (hctosys) options.

To set the date first use *date* and then run *hwclock -w -u* to store the new date into the RTC. For more information about this tool refer to the manpage of *hwclock*.

> In case you want to use the interrupt of the RV-4162-C7 RTC while working with the PCM-953 CB, jumper JP23 on the CB must be closed. Remember that the interrupt can only be applied to the RTC if it is not already in use for the LCD-018 touch.

DT representation for I^2C RTCs, e.g. in *am335x-phycore-som.dtsi:*

```
&i2c0 {
        i2c_rtc: rtc@68 {
                compatible = "mc,rv4162c7";
                reg = <0x68>;
                status = "disabled";
        };
};
```

7.9.3 Capacitive Touchscreen

The capacitive touchscreen is a part of the display module.

- For a simple test of this feature start our demo application with:

  ```
  target$ QtDemo
  ```

  This application also includes a *Multitouch Demo*.

- To start another more simple test application type:

  ```
  target$ qt5-opengles2-test
  ```

- To test the basic input handling of the touchscreen use *evtest* after selecting an input device:

  ```
  target$ evtest
  ```

  The raw touch input events will be displayed.

As the touchscreen is part of the display the touchscreen DT entry is required for example for the LCD-018 module. For the LCD-018 module, our DT model expects an *i2c_ts* node. Please see *am335x-phytec-lcd.dtsi*.

> The display DT representation for all displays and touchscreens implemented so far, are summarized in *am335x-phytec-lcd.dtsi*.

DT representation, e.g. *am335x-pcm-953.dtsi*:

```
/* Defined in am335x-pcm-953.dtsi, board specific part */
&am33xx_pinmux {
        ts_irq_pin: pinmux_ts_irq_pin {
                pinctrl-single,pins = <
                        0x1B4 (PIN_INPUT_PULLUP | MUX_MODE7)
                                          /* xdma_event_intr1.gpio0_20 */
                >;
        };
};

&i2c0 {
        i2c_ts: touchscreen@38 {
                compatible = "edt,edt-ft5x06";
                reg = <0x38>;
                pinctrl-names = "default";
                pinctrl-0 = <&ts_irq_pin>;
                interrupt-parent = <&gpio0>;
                interrupts = <20 0>;
                status = "disabled";
        };
};
```

7.9.4 Temperature Sensor

The phyCORE-AM335x-R2 has a TMP102 temperature sensor optionally mounted on the module. The temperature can be read out of the sensor over the sysfs.

- Type

  ```
  target$ cat /sys/class/hwmon/hwmon0/temp1_input
  ```

  to get the temperature in millicelsius.

DT representation, e.g. *am335x-phycore-som.dtsi*:

```
&i2c0 {
        i2c_tmp102: temp@4b {
                compatible = "ti,tmp102";
                reg = <0x4b>;
                status = "disabled";
        };
};
```

7.10 USB Host Controller

The USB controller of the AM335x SoC provides a low-cost connectivity solution for numerous consumer portable devices by providing a mechanism for data transfer between USB devices with a line/bus speed up to 480 Mbps. The USB subsystem has two independent USB 2.0 modules built around two Mentor USB OTG controllers (musbmhdrc).

The unified BSP includes support for mass storage devices and keyboards. Other USB related device drivers must be enabled in the kernel configuration on demand.

Due to *udev*, all mass storage devices connected get unique IDs and can be found in */dev/disks/by-id*. These IDs can be used in */etc/fstab* to mount the different USB memory devices in different ways.

User USB1 (host) configuration in *am335x-pcm-953.dtsi*:

```
/* USB */
&am33xx_pinmux {
        usb_pins: pinmux_usb_pins {
                pinctrl-single,pins = <
                        0x21c (PIN_OUTPUT_PULLDOWN | MUX_MODE0) /*
usb0_drvvbus.usb0_drvvbus */
                        0x234 (PIN_OUTPUT_PULLDOWN | MUX_MODE0) /*
usb1_drvvbus.usb1_drvvbus */
                >;
        };
};

&cppi41dma {
        status = "okay";
};

&ctrl_mod {
        status = "okay";
};

&usb {
        pinctrl-names = "default";
        pinctrl-0 = <&usb_pins>;
        status = "okay";
};

&usb1 {
        status = "okay";
        dr_mode = "host";
};

&usb1_phy {
        status = "okay";
};
```

7.11 USB OTG

Most Phytec boards provide an USB OTG interface. USB OTG ports can act as USB device, or USB host. The mode depends on the USB hardware attached to the USB OTG port. If, for example, an USB mass storage device is attached to the USB OTG port, the device may show up as */dev/sda*. To automatically switch between modes a USB gaget driver has to be loaded. If only USB OTG in host mode is needed, the module *g_zero* may be used.

7.11.1 USB Device

In order to connect the board as USB device to an USB host port (for example a PC), you need to configure the appropriate USB gadget. With *USB configfs* you can define the parameters and functions of the USB gadget. The BSP includes *USB configfs* support as kernel module.

- Type

  ```
  target$ modprobe libcomposite
  ```

  to load the module.

Example:

- First define the parameters such as the USB vendor and product IDs, and set the information strings for the english (0x409) language:

  ```
  target$ cd /sys/kernel/config/usb_gadget/
  target$ mkdir g1
  target$ cd g1/
  target$ echo "0x1d6b" > idVendor
  target$ echo "0x0104" > idProduct
  target$ mkdir strings/0x409
  target$ echo "0123456789" > strings/0x409/serialnumber
  target$ echo "Foo Inc." > strings/0x409/manufacturer
  target$ echo "Bar Gadget" > strings/0x409/product
  ```

- Next create a file for the mass storage gadget:

  ```
  target$ dd if=/dev/zero of=/tmp/file.img bs=1M count=64
  ```

- Now you should create the functions you want to use:

  ```
  target$ cd /sys/kernel/config/usb_gadget/g1
  target$ mkdir functions/acm.GS0
  target$ mkdir functions/ecm.usb0
  target$ mkdir functions/mass_storage.0
  target$ echo /tmp/file.img > functions/mass_storage.0/lun.0/file
  ```

  - *acm*: Serial gadget, creates a serial interface like */dev/ttyGS0*.
  - *ecm*: Ethernet gadget, creates an Ethernet interface, e.g. *usb0*
  - *mass_storage*: The host can partition, format and mount the gadget mass storage the same way as any other USB mass storage.

- Bind the defined functions to a configuration with:

  ```
  target$ cd /sys/kernel/config/usb_gadget/g1
  target$ mkdir configs/c.1
  target$ mkdir configs/c.1/strings/0x409
  target$ echo "CDC ACM+ECM+MS" > configs/c.1/strings/0x409/configuration
  target$ ln -s functions/acm.GS0 configs/c.1/
  target$ ln -s functions/ecm.usb0 configs/c.1/
  target$ ln -s functions/mass_storage.0 configs/c.1/
  ```

- Finally start the USB gadget with the following commands:

  ```
  target$ cd /sys/kernel/config/usb_gadget/g1
  target$ ls /sys/class/udc/
  musb-hdrc.0.auto
  ```

- Now use the output from the previous command:

  ```
  target$ echo "musb-hdrc.0.auto" >UDC
  ```

If your system has more than one USB Device or OTG port, you can pass the correct one to the USB Device Controller (UDC).

- To stop the USB gadget and unbind the used functions execute:

  ```
  target$ echo "" > /sys/kernel/config/usb_gadget/g1/UDC
  ```

User USB0 (OTG) configuration in *am335x-pcm-953.dtsi*:

```
&usb0 {
        status = "okay";
};

&usb0_phy {
        status = "okay";
};
```

7.12 Audio

On Phytec products you will find different audio chips. This chapter should be applicable to most audio chips without modification.

Audio support on Phytec AM335x boards is done via the I^{2}S interface and controlled via I^{2}C.

> The I^{2}S port does not always strictly follow the I^{2}S specification, but can use other clock and bit alignments. However, the common ground is the synchronous serial transmission.

- To check if your soundcard driver is loaded correctly and what the device is called type:

```
target$ aplay -lL
```

- Use *scp* to copy a wav file to the board and play it through the sound interface with:

```
target$ aplay -vv file.wav
```

Not all wave formats are supported by the sound interface. Use *Audacity* on your host to convert any file into *44100:S16_LE* wave format which should be supported on all platforms.

- Run *speaker-test* to identify channel numbers:

```
target$ speaker-test -c 2 -t wav
```

- An external audio source can be connected to the input, in order to record a sound file which can then be played back:

```
target$ arecord -c 2 -r 441000 -f S16_LE test.wav
target$ aplay test.wav
```

- To inspect your soundcards capabilities call

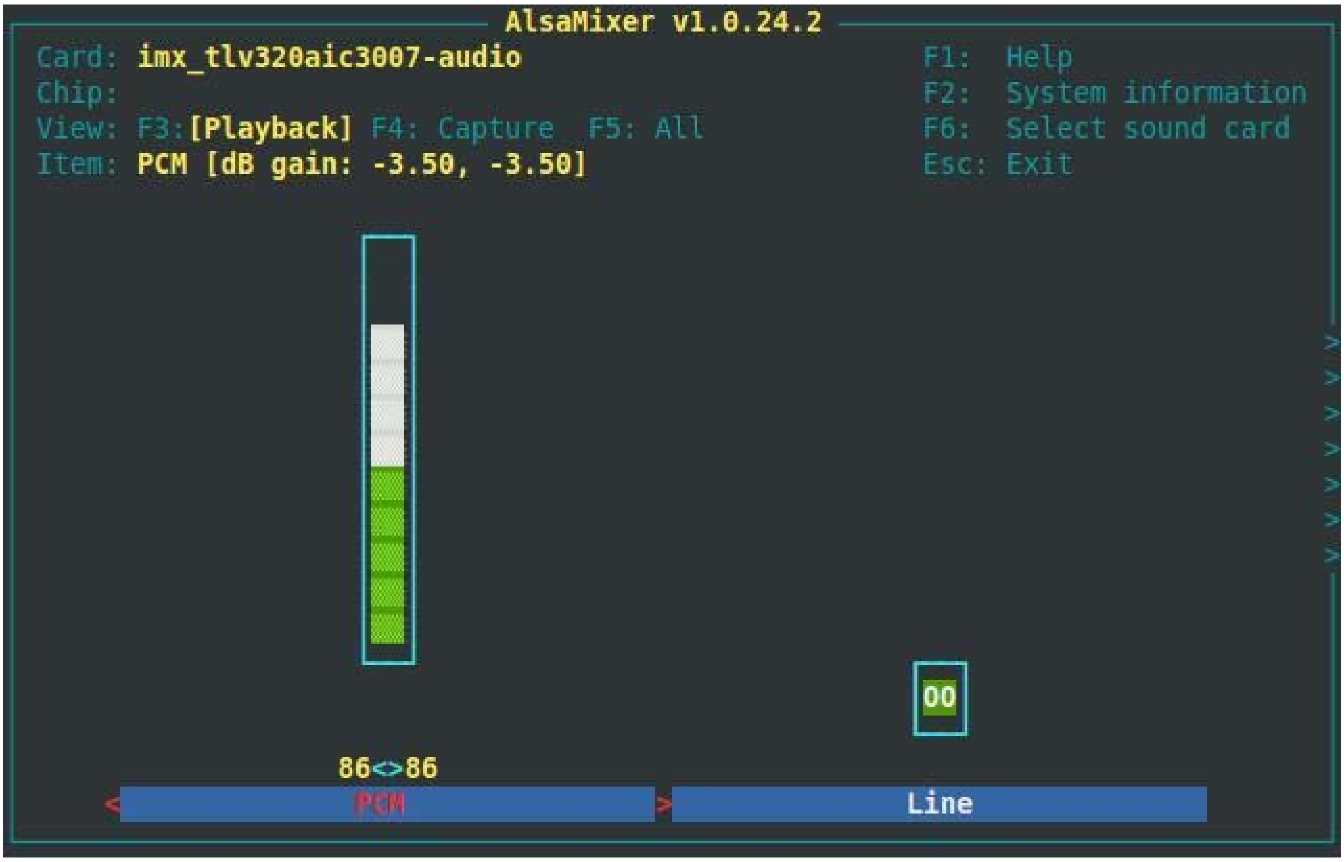
```
target$ alsamixer
```

You should see a lot of options as the audio-ICs have many features you can play with. It might be better to open *alsamixer* via *ssh* instead of the serial console, as the console graphical effects could be better. You have either mono or stereo gain controls for all mix points. "*MM*" means the feature is muted, which can be toggled by hitting ***m***.

For more advanced audio usage, you need to have an audio server. This is required e.g. if you want to playback from different applications at the same time, e.g. you a have a notification app which makes a beep every now and then, plus you have a browser running which should be able to play sounds embedded in websites. The notification app has no possibility to open the sound device as long as the browser is running. The notifications will be suppressed. The standard sound server of *Linux* is *pulseaudio*. It is not installed per default at the moment, though.

7.12.1 Audio Sources and Sinks

Enabling and disabling input and output channels can be done with the *alsamixer* program. ***F3*** selects *Screen Playback* and ***F4*** *Screen Capture*. With the ***Tabulator*** key you can switch between these screens. To enable, or disable switchable controls press ***m*** (mute).



```
                       AlsaMixer v1.0.24.2
Card: imx_tlv320aic3007-audio                 F1:  Help
Chip:                                         F2:  System information
View: F3:[Playback] F4: Capture  F5: All      F6:  Select sound card
Item: PCM [dB gain: -3.50, -3.50]             Esc: Exit

                                                        00
               86<>86
         <          PCM         >           Line
```

Figure 3: Screenshot of alsamixer

With the keys cursor left and cursor right you can step through the different channels. There are much more channels than fit onto one screen, so they will scroll if your cursor reaches the right or left edge of it. In case you get trouble in the display during scrolling, please use *ssh* instead of *microcom*.

alsamixer can be left by pressing the ***ESC*** key. The settings are saved automatically on shutdown and restored on boot by the systemd service *alsa-restore*. If you want to save the mixer settings manually you can execute *alsactl store*. The settings are saved in */var/lib/alsa/asound.state*.

7.12.2 Playback

To playback simple audio streams, you can use *aplay*. For example:

```
target$ aplay /usr/share/sounds/alsa/Front_Center.wav
```

The file formats *.ogg*, and *.flac* can be played back using *ogg123*. MP3 playback is currently not supported per default, because of licensing issues. If you are going to deliver a product including *.mp3* files, please check the royalty fees.

7.12.3 Capture

arecord is a command line tool for capturing audio streams which uses *Line In* as default input source.

To select a different audio source you can use *alsamixer*. For example, switch on *Right PGA Mixer Mic3R* and *Left PGA Mixer Mic3L* in order to capture the audio from the microphone input.

It is a known error that you need to choose *Playback screen* (***F3***) instead of *Capture screen* (***F4***) for accessing these two controls.

The following example will capture the current stereo input source with a sample rate of 48000 Hz and will create an audio file in WAV format (signed 16 bit per channel, 32 bit per sample):

```
target$ arecord -t wav -c 2 -r 48000 -f S16_LE test.wav
```

Capturing can be stopped again using ***CTRL-C***.

7.12.4 Texas Instruments TLV320AIC3007 (phyBOARD-Wega)

The TI chip has a lot of mixing features as you will notice when opening *alsamixer*. To get an idea of the possibilities, refer to the datasheet from TI. One important feature is the analog mixing capability. You can route input channels to output channels either active, or passive without transferring data back and forth to the SoC. This is usually known as zero latency monitoring in terms of PC Audio cards. To control this feature open *alsamixer* and go to *Playback section* (F3). There will be one group of selectors for each output channel, e.g. *Right Line Mixer*. For most of the input paths you can select between "passive through", or you can activate a programmable gain signal path. To route the line input to the line output activate both *Right Line Mixer PGAR Bypass* and *Left Line Mixer PGAL Bypass*. You should now hear the input signal at the line out.

As an example application we can route the audio stream through the whole system. First, we deactivate the direct loop from line in to line out in *alsamixer* and activate the ADC for the analog input mixer and the DAC for the analog output mixer. We can now close the signal path in the user space by using a pipe and the *ALSA* tools:

```
target$ arecord -c 2 -r 44100 -f S16_LE | aplay
```

We created the following audio routing path:
analog in -> ADC -> AM335x -> kernel -> (userspace -> kernel ->)[1] DAC -> analog out

You should now hear the input signal at the line output again. But this time, the routing through the entire system added a latency of 750 ms to the signal. The line mixer on the other side will add about 4 µs delay.

This pipe routing is a crude example of how to use audio. If you need good audio performance you should use an audio server. There are two prominent choices available, *pulseaudio* and *jack*.

As a third option you can route the audio from line in to the internal ADC. But instead of taking the signal path through the complete system, you can directly use the internal digital mixer of the TLV to route the signal back to the DAC and then to the output analog mixer and line out. This signal path is record only. So you do not have the option of full duplex playback in this mode, but you can use all the available DSP features of the TLV, the automatic gain, the high-pass filter, EQ and de-emphasis filters for the record path. To activate this mode you have to use the TI Windows tool to look up the specific I^{2}C commands. The software can be found on the TI webpage and is called *TLV320AIC3107EVM-K - GUI* Software. The 3107 is register compatible in all available features to the 3007. The software is used for both chips.

7.12.5 Wolfson WM8974 (phyCORE-AM335x Carrier Board - PCM-953)

The WM8974 is a low power mono codec with a speaker out. The codec has an internal PLL, so different clocking modes can be support. On the PCM-953 we achieved the best values for the Total Harmonic Distortion (THD) with an external quartz (OZ1 on the PCM-953) and the WM8974 used in I^{2}S and master mode. The BSP is per default configured this way. Hence, if the codec is not working first ensure that jumper JP6 is closed at 1+2 to enable the external quartz.

The microphone input is configured in single ended mode. Test Pad TP1 can be used when connecting a balanced microphone. MONO_OUT is the line out mono signal. HEADPHONES is a stereo out jack with ground on the sleeve and the differential mono signal connected to the tip (positive power out) and the ring of the jack (negative power out). Hence, an adapter is required in order to attach a stereo headphone to the jack. The adapter must connect the ring of the boards jack (negative power out) to the ground of the headphone

[1]: twice for the pipe

and the tip of the jack (positive power out) to both, left and right signal input of the stereo headphone.

7.13 Framebuffer

This driver gains access to displays connected to Phytec carrier boards via device node */dev/fb0*.

- To run a simple test of the framebuffer feature execute:

  ```
  target$ fbtest
  ```

This will show various pictures on the display.

- Information about the framebuffer's resolution can be obtained with

  ```
  target$ fbset
  ```

  which will return:

```
mode "800x480-0"
    # D: 0.000 MHz, H: 0.000 kHz, V: 0.000 Hz
    geometry 800 480 800 480 32
    timings 0 0 0 0 0 0 0
    accel true
    rgba 8/16,8/8,8/0,0/0
endmode
```

> *fbset* cannot be used to change display resolution or color depth. Depending on the framebuffer device different kernel commands are mostly needed to do this. Some documentation can be found in the kernel documentation at
> *https://www.kernel.org/doc/Documentation/fb/modedb.txt*.
> Please also refer to the manual of your display driver for more details.

- To query the color depth of the framebuffer emulation type:

  ```
  target$ cat /sys/class/graphics/fb0/bits_per_pixel
  ```

The result can be, for example:

```
16
```

The display DT representation can be found in *am335x-phytec-lcd.dtsi*. We have split the display configuration into two parts. In a generic display module *am335x-phytec-lcd.dtsi,* which includes the display DT representations for all displays and touchscreens implemented so far, and a board specific part, which can be found in all board DTs. The *am335x-phytec-lcd.dtsi* file is included within the board DTs.

Board specific part, e.g. *am335x-pcm-953.dtsi*

```
/* Display */
&am33xx_pinmux {
        ecap0_pins: pinmux_ecap0 {
                pinctrl-single,pins = <
                        0x164 (PIN_OUTPUT_PULLDOWN | MUX_MODE0)
                      /* ecap0_in_pwm0_out.ecap0_in_pwm0_out */
                >;
        };

        ts_irq_pin: pinmux_ts_irq_pin {
                pinctrl-single,pins = <
                        0x1B4 (PIN_INPUT_PULLUP | MUX_MODE7)
                              /* xdma_event_intr1.gpio0_20 */
                >;
        };
};

&ecap0 {
        pinctrl-names = "default";
        pinctrl-0 = <&ecap0_pins>;
        status = "disabled";
};


&i2c0 {
        i2c_ts: touchscreen@38 {
                compatible = "edt,edt-ft5x06";
                reg = <0x38>;
                pinctrl-names = "default";
                pinctrl-0 = <&ts_irq_pin>;
                interrupt-parent = <&gpio0>;
                interrupts = <20 0>;
                status = "disabled";
        };
};

&tscadc {
        status = "disabled";
        tsc {
                ti,wires = <4>;
                ti,x-plate-resistance = <200>;
                ti,coordinate-readouts = <5>;
                ti,wire-config = <0x00 0x11 0x22 0x33>;
                ti,charge-delay = <0x400>;
        };
};

#include "am335x-phytec-lcd.dtsi"
```

7.13.1 Backlight Control

If a display is connected to the Phytec board, you can control its backlight with the *Linux* kernel sysfs interface. All available backlight devices in the system can be found in the folder */sys/class/backlight*. Reading the appropriate files, and writing to them allows to control the backlight.

- To get, for example, the maximum brightness level (max_brightness) execute

  ```
  target$ cd /sys/class/backlight/backlight/
  target$ cat max_brightness
  ```

  which will result in:

  ```
  7
  ```

The valid values for the brightness level are 0 to <max_brightness>.

- To obtain the current brightness level type

  ```
  target$ cat brightness
  ```

  you will get for example:

  ```
  2
  ```

- Write to the file *brightness* to change the brightness. E.g.,

  ```
  target$ echo 0 > brightness
  ```

  turns the backlight off,

  ```
  target$ echo 6 > brightness
  ```

  sets the brightness to the second highest brightness level.

For documentation of all files see
https://www.kernel.org/doc/Documentation/ABI/stable/sysfs-class-backlight.

If dimming does not work by writing to the file *brightness*, check the DIP switch settings on the bottom side of the display module (LCD-018-xxx). The correct setting of DIP switch *S1* is 1=OFF, 2=OFF, **3=ON** and 4=OFF.

This note applies to phyFLEX Kits with one of the following display modules:
LCD-018-035-KAP, LCD-018-043-KAP, LCD-018-057-KAP, and
LCD-018-070-KAP.

7.13.2 Resistive Touchscreens

Most of the Phytec boards support connecting a resistive touchscreen which requires *tslib* support in general. But *tslib* support is also needed for the *Qt* framework for use of a resistive touchscreen, because *tslib* and *Qt5* do not work together out of the box.

- To set the right environment when starting a *Qt* application, start it with our *qtLauncher*:

```
target$ qtLauncher QtDemo [optional QtDemo parameters]
```

The launcher will check, whether the touchscreen is calibrated or not. If not, it will automatically start the calibration program before starting your application.

- Recalibration of the touchscreen can be done after closing the *Qt* application by executing:

```
target$ ts_calibrate
```

7.14 Watchdog

The AM335x SoC includes a hardware watchdog which is able to reset the board when the system hangs. Support for this is already partly activated in the AM335x BSP. This chapter will explain how to enable full support based on a root filesystem using *systemd*. The kernel driver expects a user space to "ping" the watchdog in regular bases. As user space setups may handle this different, we have not activated this for the whole system.

7.14.1 Watchdog Support in the Barebox Bootloader

The watchdog is enabled when the boot command is launched. The timeout of the watchdog is set with the persistent *nv* variable in *boot.watchdog_timeout*. The default value is 60 s.

- If the value should be changed edit the file *boot.watchdog_timeout* with:

```
bootloader$ edit /env/nv/boot.watchdog_timeout
```

- Or change the variable with the *nv* command:

```
bootloader$ nv boot.watchdog_timeout=<time, e.g. 30s>
```

- Then save the changes with *saveenv*.

It is also possible to enable the watchdog manually at the *barebox* prompt using the *wd <timeout>* command. Executing the *wd* command without any parameter retriggers the watchdog. Disabling the watchdog again can be done with *wd 0*.

7.14.2 Watchdog Support in the Linux Kernel

After the kernel has started the watchdog is disabled again when the *omap_wdt* driver is loaded. This is the default behavior of the watchdog driver. To keep the watchdog enabled after the driver has been probed, the kernel command line needs to be extended with:

```
omap_wdt.early_enable omap_wdt.timer_margin=60
```

The first parameter enables the watchdog in the kernel and the second one defines a new timeout in seconds.

The bootargs can also be set in the *barebox* environment.

- To set the bootargs edit the file *linux.bootargs.base*:

  ```
  bootloader$ edit /env/nv/linux.bootargs.base
  ```

- Then save the changes with *saveenv*.

⚠	The kernel driver is not able to handle a timeout and to refresh the watchdog. This must be done in user space. Otherwise the board will reset after the timeout runs out.

7.14.3 Watchdog Support in systemd

Systemd does include hardware watchdog support since version 183.

- To activate watchdog support the file *system.conf* in */etc/systemd/* has to be adapted by enabling the options: `RuntimeWatchdogSec=60s` and `ShutdownWatchdogSec=10min`

RuntimeWatchdogSec defines the timeout value of the watchdog, while *ShutdownWatchdogSec* defines the timeout when the system is rebooted.

For more detailed information about hardware watchdogs under *systemd* refer to *http://0pointer.de/blog/projects/watchdog.html*

7.15 WLAN Modules

7.15.1 Supported Wi-Fi Modules and Software used

Currently there is only one module supported. The BSP described herein allows to run the TiWi-BLE *Bluetooth* and Wi-Fi combo module in client and Access Point (AP) mode using *WEP*, *WPA* and *WPA2* encryption. More information about the module can be found at *https://www.lsr.com/embedded-wireless-modules/wifi-plus-bluetooth-module/tiwi-ble*

This BSP is using the most current firmware from the linux-firmware repository.
git://git.kernel.org/pub/scm/linux/kernel/git/firmware/linux-firmware.git

The following binaries are used:
wl127x-fw-5-mr.bin (multi role)
wl127x-fw-5-sr.bin (single role)
wl127x-fw-5-plt.bin (used for the calibration process)

The *Linux* kernel driver will select the right firmware.

> We are using a self generated *nvs* binary file to allow the TiWi-BLE combo module to operate in compliance with the modular certification for *FCC* and *ETSI*. We have also set the MAC in the *nvs* file to 00:00:00:00:00:00. This causes the driver to use the MAC from the BT BD address that is burned into the wl127x chip.

The *wl1271-nvs.bin* can be generated by using TI's calibrator tool and our *ini* file in */usr/share/wl127x-inis/tiwi-ble-fcc-etsi.ini.*

Please see section *7.15.3 "Calibration"* for more information.

7.15.2 Enable Wi-Fi Expansion Boards

All Wi-Fi expansion boards can be enabled by sourcing the *am335x-wlan* expansion file in *env/expansions/*.

am335x-wlan:

```
of_fixup_status /fixedregulator@2
of_fixup_status /ocp/mmc@47810000
of_fixup_status /ocp/mmc@47810000/wlcore@0
```

This file will be sourced in *env/config-expansions*.

After enabling the Wi-Fi module on the expansion board and booting the kernel, you should see an output on your console similar to the following:

```
cfg80211: Calling CRDA to update world regulatory domain
wlcore: loaded
```

7.15.3 Calibration

Calibration is a process of determining radio parameters for a specific chip. These configuration parameters are suitable to the specific chip with its unique design. Therefore, the calibration parameters are sent back to the driver to be stored in non-volatile memory for later use. Upon initialization, the wL12xx driver loads an *nvs* file, where it expects to read those calibration parameters to send them to the chip. The *nvs* file includes two main parts: one stores the calibration parameters and the other stores the initialization information required for the wL12xx driver. The calibration is described also in *http://processors.wiki.ti.com/index.php/WL12xx_NLCP_Calibration_Process* and *http://linuxwireless.org/en/users/Drivers/wl12xx/calibrator/#wl12xx_Calibration*.

TI provides a calibration tool which can be found in the 18xx-ti-utils repository:
git://git.ti.com/wilink8-wlan/18xx-ti-utils.git

The calibrator version used is 0.80.

- Before starting the calibration, the WiFi module has to be set to PLT mode. For this purpose open the menuconfig in your *Yocto* directory:
  ```
  <yocto_dir>/build$ bitbake virtual/kernel -c menuconfig
  ```
- Activate the *NL80211_TESTMODE* configuration:
  ```
  Networking support > Wireless > nl80211 testmode command
  ```
- Press ***F6*** to save the configurations and ***F9*** to leave. Rebuild the kernel:
  ```
  <yocto_dir>/build$ bitbake phytec-headless-image
  ```
- Flash the new BSP image to the board (*section 5*) and reboot the module.
- To start the calibration first generate an *nvs* reference file:
  ```
  target$ calibrator set ref_nvs /usr/share/wl127x-inis/tiwi-ble-fcc-
                                                                 etsi.ini
  target$ cp new-nvs.bin /lib/firmware/ti-connectivity/wl1271-nvs.bin
  ```
- Reload the driver:
  ```
  target$ ifconfig wlan0 down
  target$ rmmod wlcore_sdio
  target$ modprobe wlcore_sdio
  ```
- Now perform the first calibration:
  ```
  target$ calibrator plt calibrate
  ```
- Copy the newly created file to the proper location:
  ```
  target$ cp new-nvs.bin /lib/firmware/ti-connectivity/wl1271-nvs.bin
  ```
- Finally set the WLAN MAC address:
  ```
  target$ calibrator set nvs_mac /lib/firmware/ti-connectivity/wl1271-
                                            nvs.bin 00:00:00:00:00:00
  ```
- Restart the module to get the new MAC address:
  ```
  target$ rmmod wlcore_sdio
  target$ modprobe wlcore_sdio
  ```

7.16 Power Management

7.16.1 CPU Core Frequency Scaling

The CPU of the AM335x SoC is able to scale the clock frequency and the voltage. This is used to save power when the full performance of the CPU is not needed. Scaling the frequency and the voltage is referred to as 'Dynamic Voltage and Frequency Scaling' (DVFS).

The AM335x BSP supports the DVFS feature. The *Linux* kernel provides a DVFS framework that allows each CPU core to have a min/max frequency and a governor that governs it. Depending on the AM335x variant used several different frequencies are supported.

- Type
  ```
  target$ cat
      /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_frequencies
  ```
  to get a complete list.

In case you have for example an AM3359 with a maximum of 800 MHz the result will be:

```
 300000 600000 720000 800000
```

The voltages are scaled according to the setup of the frequencies.

You can decrease the maximum frequency (e.g. to 720000) with

```
target$ echo 720000 >
                /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
```

or increase the minimum frequency (e.g. to 600000)

```
target$ echo 600000 >
                /sys/devices/system/cpu/cpu0/cpufreq/scaling_min_freq
```

- To ask for the current frequency type:
  ```
  target$ cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
  ```

So called governors are selecting one of this frequencies in accordance to their goals, automatically.

- List all governors available with the following command:
  ```
  target$ cat
        /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_governors
  ```

The result will be:

```
conservative userspace powersave ondemand performance
```

conservative is much like the *ondemand* governor. It differs in behavior in that it gracefully increases and decreases the CPU speed rather than jumping to max speed the moment there is any load on the CPU.

userspace allows the user or user space program running as root to set a specific frequency (e.g. to 600000). Type:

```
target$ echo 600000 >
        /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed
```

powersave always selects the lowest possible CPU core frequency.

ondemand switches between possible CPU core frequencies in reference to the current system load. When the system load increases above a specific limit it increases the CPU core frequency immediately. This is the default governor when the system starts up.

performance always selects the highest possible CPU core frequency.

- In order to ask for the current governor, type:

```
target$ cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

You will normally get:

```
ondemand
```

- Switching over to another governor (e.g. *userspace*) is done with:

```
target$ echo userspace >
                /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

For more detailed information about the governors refer to the *Linux* kernel documentation in: *linux/Documentation/cpu-freq/governors.txt.*

7.16.2 Power Saving Modes

The phyCORE-AM335x-R2 supports two different Suspend-to-RAM (STR) modes, standby and deep sleep. They switch the SoC into a power saving mode and turn off different power domains.

7.16.2.1 Standby

This mode has the following behavioral:

- DDR in self-refresh
- CPU shutdown
- Peripheral is on

- To turn on this STR mode write *standby* to */sys/power/state*:

```
echo -n standby > /sys/power/state
```

7.16.2.2 Deep Sleep

This mode has the following behavioral:

- DDR in self-refresh
- CPU shutdown
- Peripheral is off

- To turn on this STR mode write *mem* to */sys/power/state*:

```
echo -n mem > /sys/power/state
```

8 Customizing the BSP

8.1 Changing MTD Partitions

For Memory Technology Devices (MTD) such as NAND Flash or SPI NOR Flash the partitions are usually defined in the device trees, i.e. they are defined in the device tree of the MLO, the *barebox* and the kernel. When changing the partition table all those parts need to be touched. Newer *barebox* versions (v2015.07.0 and newer) have a mechanism to overwrite partitions at runtime.

The *barebox* holds an internal list with partitions which is initialized with the partition table out of the *barebox* device tree. This list is used later on to fix up the device tree of the kernel and can be overwritten in the *barebox* shell, or with a script before booting the kernel.

- To print the partitions just type:

  ```
  bootloader$ echo $<mtddevice>.partitions
  ```

Example:

```
bootloader$ echo $nand0.partitions
```

can, for example, result in:

```
128k(xload),128k(xload_backup1),128k(xload_backup2),128k(xload_backup3),512k(bar
ebox),512k(barebox_backup),256k(bareboxenv),129280k(root)
```

- To overwrite the partitions just change the partitions variable:

  ```
  target$ m25p0.partitions=128k(xload),512k(barebox),128k(bareboxenv),
                           128k(oftree),7296k(kernel)
  ```

Adding and deleting partitions by overwriting the partitions variable is possible. But do **not** touch the *xload*, *xload_backup**, *barebox*, *barebox_backup* and *bareboxenv* partitions. Those should **not** be changed at runtime.

9 Revision History

Date	Version #	Changes in this manual
18.02.2016	Manual L-818e_1	First edition. Describes the Phytec BSP release PD15.2.x for AM335x products with *Yocto* 1.8.1
27.07.2016	Manual L-818e_2	Second edition. Describes the Phytec BSP release phyCORE-AM335x R2 PD16.1.x with *Yocto* 2.0.2
24.02.2017	Manual L-818e_3	Third edition. Describes the Phytec BSP release AM335x PD16.2.x with *Yocto* 2.1.2

Document:	**Yocto AM335x BSP Manual**
Document number:	**L-818e_3, February 2017**

How would you improve this manual?

Did you find any mistakes in this manual? **page**

Submitted by:

Customer number: ____________________

Name: ____________________

Company: ____________________

Address: ____________________

____

Return to:

PHYTEC Messtechnik GmbH
Postfach 100403
D-55135 Mainz, Germany
Fax : +49 (6131) 9221-33

Published by

PHYTEC

© PHYTEC Messtechnik GmbH 2017

Ordering No. L-818e_3
Printed in Germany