phyCORE-AM57x RDK Linux Quickstart PD15.1-rc3

This Quickstart provides you with the tools and know-how to install and work with the Linux Board Support Package (BSP) for the phyCORE-AM57x Rapid Development Kit. This Quickstart shows you how to do everything from installing the appropriate tools and source, to building custom kernels, to deploying the OS, to exercising the software and hardware. Please refer to the phyCORE-AM57x Hardware Manual for specific information on board-level features such as jumper configuration, memory mapping and pin layout for the phyCORE-AM57x System on Module (SOM) and baseboard. Additionally, gain access to the SOM and baseboard schematics for the phyCORE-AM57x Rapid Development Kit by registering at the following: **http://phytec.com/support/registration/**.

- 1 Requirements
  - 1.1 Hardware
  - 1.2 Software
- 2 Getting Started With Binary Images
  - 2.1 Connector Interfaces
  - 2.2 Booting the Pre-built Images
- 3 About the Yocto BSP
- 4 Development Host Setup
  - 4.1 Host Debian Packages
  - 4.2 Repo Tool
  - 4.3 Git Setup
  - 4.4 Server Setup (Optional)
    - 4.4.1 TFTP
    - 4.4.2 NFS
    - 4.4.3 Samba
- 5 Building the BSP from Source
  - 5.1 Setup the BSP Directory:
  - 5.2 Install the Linaro Toolchain:
  - 5.3 Download the BSP Meta Layers
  - 5.4 Start the Build
  - 5.5 Built Images
  - 5.6 Build Time Optimizations
- 6 Customizing the BSP
  - 6.1 Appending Recipes
  - 6.2 Adding Packages to the build
  - 6.3 Configuring the Kernel
  - 6.4 Customizing the Device Tree
- 7 Creating a Bootable SD Card
  - 7.1 Root Filesystem
  - 7.2 Kernel
  - 7.3 Bootloader
- 8 Boot Configurations
  - 8.1 Selecting Boot Modes
    - 8.1.1 SD Card
    - 8.1.2 eMMC
  - 8.2 Basic Settings
    - 8.2.1 Network Settings
    - 8.2.2 Saving Configurations
  - 8.3 Boot Options
    - 8.3.1 Stand-Alone SD/MMC Card Boot
    - 8.3.2 Remote Boot
    - 8.3.3 Stand-Alone eMMC Boot
    - 8.3.4 Custom Boot
- 9 Flashing Images to eMMC
  - 9.1 Partition eMMC from U-boot
  - 9.2 U-Boot
  - 9.3 Root Filesystem

Requirements

The following system requirements are necessary to successfully complete this Quickstart. Deviations from these requirements may suffice, or may have other workarounds.

Hardware

- phyCORE-AM57x System on Module (PCM-057)
- phyCORE-AM57x Baseboard (PCM-948)
- Serial cable (RS-232)
- Ethernet cable
- AC adapter supplying 12VDC / min. 2A

Software

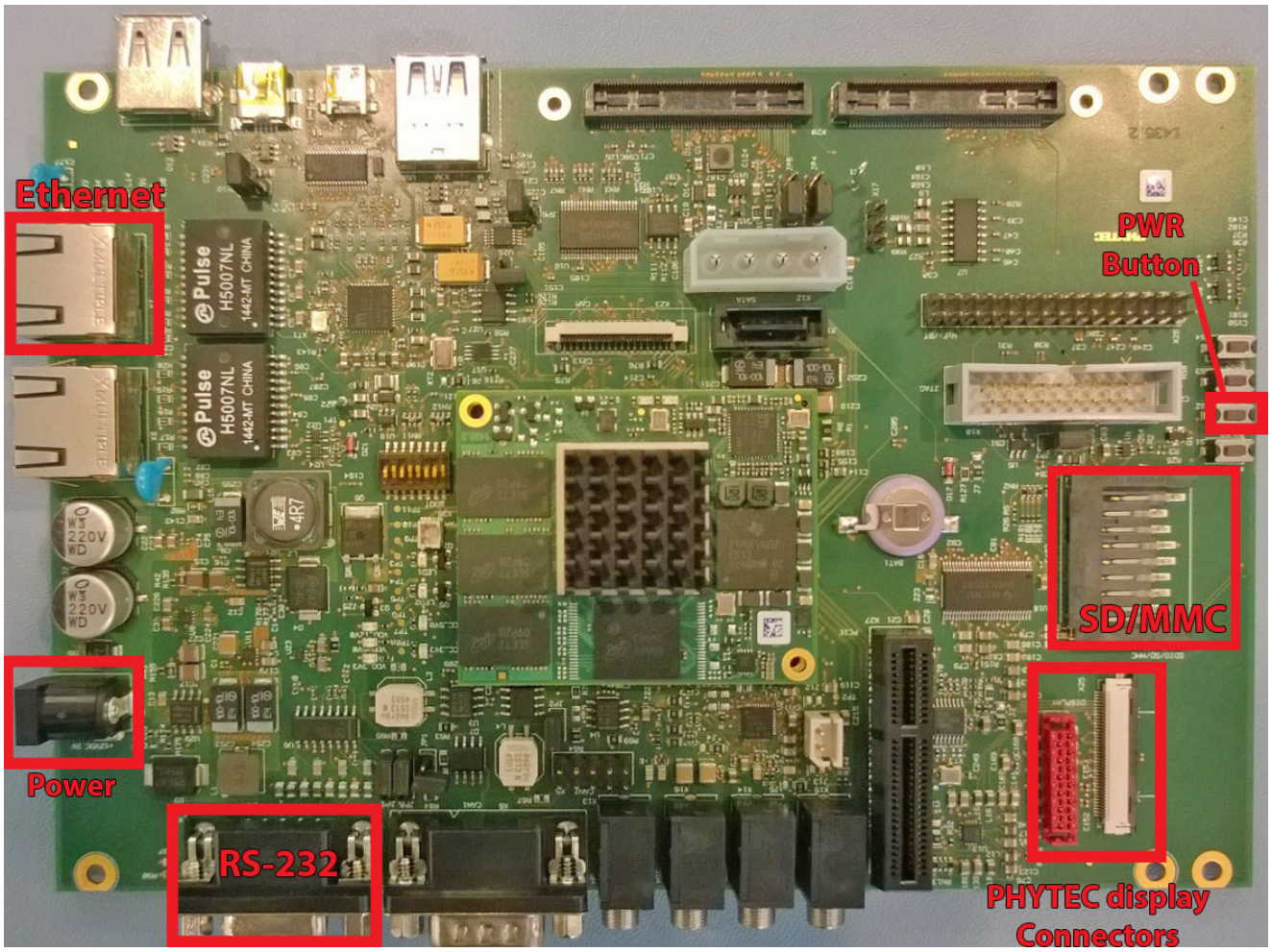
- A modern GNU/Linux Operating host system either natively or via a virtual machine:
  - Ubuntu 14.04 LTS 64-bit recommended. Other distributions will likely work, please note that some setup information as well as OS-specific commands and paths may differ.
  - If using a virtual machine, VMWare Workstation, VMWare Player, and VirtualBox are all viable solutions.
- Root access to your Linux Host PC. Some commands in the Quickstart will not work if you don't have sudo access (ex. package installation, formatting SD card).
- At least 110GB free on target build partition.
- SD card reader operational under Linux.
  - If you do not have SD card access under Linux then formatting, copying the bootloader, and mounting the root file system on an SD card will not be possible.
- Active Internet connection

Getting Started With Binary Images

This section is designed to get the board up-and-running with pre-built images.

Connector Interfaces

Use the following as a reference for the connector interfaces on the phyCORE-AM57x Rapid Development Kit that will be used in this Quickstart.

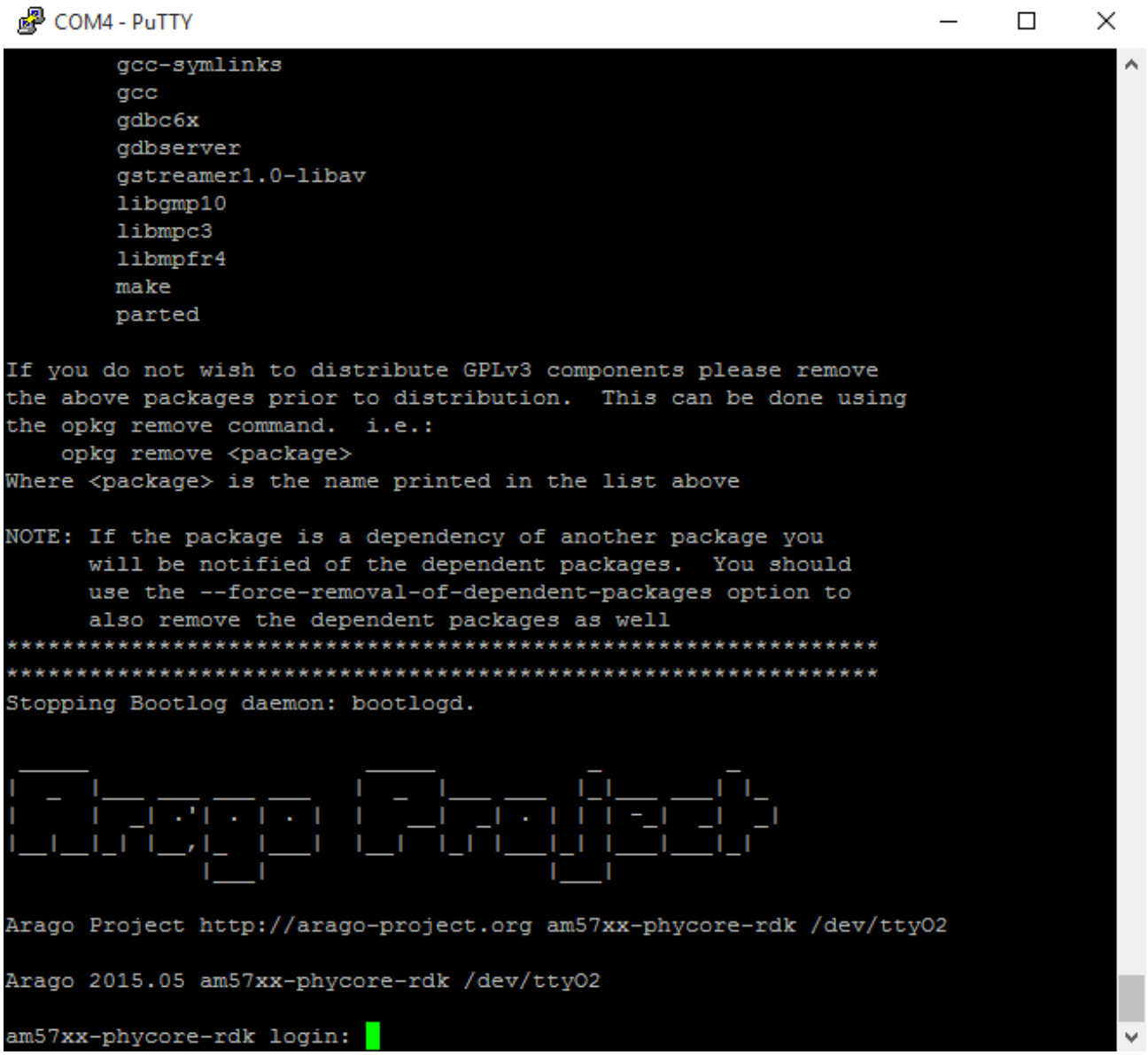


Booting the Pre-built Images

The section was designed to show you how to boot the phyCORE-AM57x Rapid Development Kit with the pre-built demo images.

1. Connect the kit supplied serial cable from a free serial port on your host PC to the DB9 connector X18 on the carrier board. This is the UART3 communication channel with the AM57x at RS-232 levels.
2. Connect the kit supplied Ethernet cable from the Ethernet connector X7 on the carrier board to your network hub, router, or switch. If you do not have an Ethernet connection you can postpone this step, Linux will boot without the need for Ethernet connectivity but having the connection will significantly reduce your boot time.
3. Start your favorite terminal software (such as Minicom or TeraTerm) on your host PC and configure it for 115200 baud, 8 data bits, no parity, and 1 stop bit (8n1) with no handshake.
4. Plug the kit supplied 12 V power adapter into the power connector X4 on the carrier board. You will instantly see power LEDs VCC_5V0 and VCC_3V3 on the carrier board light up solid green.

5. Press the power button S2 on the carrier board. You will now see power LEDs VDD_3V3, VDD_5V0, and VDD_12V0 on the carrier board light up a solid green. You will also start to see console output on your terminal window. If everything was done correctly the board should boot completely into Linux, arriving at a *am57xx-phycore-rdk* prompt. The default login account is *root* with an empty password. Note that the first time the board is booted it will takes a little while for the SSH server to generate new keys. Subsequent boots should be faster.



Troubleshooting

Not seeing any output on the console?

- Check that you have setup the terminal software correctly per step 5.
- **Make sure to press the power button S2 on the carrier board.** Unlike some other PHYTEC boards, the phyCORE-AM57x RDK does not get powered on simply by plugging in the power supply.
- **Create a Bootable SD Card** with the release images from the **PHYTEC ARTIFACTORY**, then configure the board to boot from SD/MMC (**Selecting Boot Modes**). After booting, you can restore your eMMC contents by following the **Flashing Images to eMMC** section.

About the Yocto BSP

The Yocto Project is a Linux embedded development environment which provides layers of meta data and tools. PHYTEC's AM57x Yocto BSP is based on the Arago Project, which contains BSP, distro, and application recipes and tools for TI platforms based on ARM processors. The layers that provide this are meta-ti, meta-arago, and meta-processor-sdk. The openEmbedded meta layer is also included in this BSP and is made up of a collection of meta layers which provide recipes for many software packages. The **meta-phytec** layer leverages the Arago project as a base and contains recipes and configurations developed by PHYTEC. This layer defines configurations for u-boot, the kernel, and software specific to the phyCORE-AM57xx.

In order to help get started with PHYTEC's Yocto BSP structure, the repo tool can be used to obtain all the BSP sources relevant to your hardware configuration without interfacing with git. Detailed information on building this BSP from source is provided following the Development Host Setup section.

Development Host Setup

Host Debian Packages

Yocto development requires certain packages to be installed. Run the following commands to ensure you have the packages installed:

```
sudo apt-get install git build-essential diffstat texinfo gawk chrpath doxygen dos2unix
```

If using a 64-bit machine, the following libraries are also required:

```
sudo apt-get install lib32stdc++6 lib32ncurses5 lib32z1 libc6-dev-i386
```


The above is the recommended package installation for development on a Ubuntu 14.04 LTS Linux distribution. For a breakdown of the packages as well as a list of packages required for other Linux distributions, see the "Required Packages for the Host Development System" section in the Yocto Project Reference Manual: **http://www.yoctoproject.org/docs/1.8/ref-manual/ref-manual.html#required-packages-for-the-host-development-system**

Verify that the preferred shell for your Host PC is "bash" and not "dash":

```
sudo dpkg-reconfigure dash
# Respond "No" to the prompt asking "Install dash as /bin/sh?"
bash
```

Repo Tool

Download and install the *repo* tool. This tool is used to obtain Yocto source from Git.

```
cd /opt
sudo mkdir bin

# /opt/ directory has root permission, change the permissions so your user account can access this folder. In
the following replace <user> with your specific username
sudo chown -R <user>: bin

cd bin
curl http://commondatastorage.googleapis.com/git-repo-downloads/repo > ./repo
#add directory that contains repo to your path
chmod a+x repo
```

Add the *repo* directory in your PATH, using **export** from the command line or permanently by including it in *.bashrc*:

```
PATH=/opt/bin/:$PATH
```

Git Setup

If you have not yet configured your Git environment on this machine, please execute the following commands to set your user name and email address. See here for more information on getting started with Git.

```
git config --global user.email "your@email.com"
git config --global user.name "Your Name"
git config --global http.sslcainfo /etc/ssl/certs/ca-certificates.crt
```

Server Setup (Optional)

The following steps describe the setup for TFTP, NFS, and Samba servers. Server setup is not required for working with the board, however they will significantly reduce time and are highly recommended during the building and development phase.

TFTP

TFTP is a "trivial" file transfer protocol used to transfer individual files across a network. Setting up a TFTP server on your Linux Host PC will allow you to exchange files with the target board. Some examples where this will be advantageous include:

- Modifying and doing development on the Linux kernel. Barebox can be configured to remotely boot the kernel so you have access to the latest build without needing to continually reflash the target board.
- Updating images from the bootloader. Transferring files over a network in Barebox is an alternative to using an SD card which eliminates some time consuming steps such as formatting an SD card.
- Individual file transfer to the root fileystem. When Linux has been fully booted you may want to copy a specific file from your Host PC to the target board (images, application executables).

Install the TFTP server on your Host PC:

```
sudo apt-get install tftpd-hpa
```

Specify a folder where the files will reside on your Host PC by replacing the folder path for "TFTP_DIRECTORY" with whatever folder you wish to use as your TFTP file storage location, or leave the folder as the default.

```
sudo gedit /etc/default/tftpd-hpa
# /etc/default/tftpd-hpa
TFTP_USERNAME="tftp"
TFTP_DIRECTORY="/var/lib/tftpboot"
TFTP_ADDRESS="0.0.0.0:69"
TFTP_OPTIONS="--secure"
```

If you made any changes to the settings of the TFTP server, you need to restart it for them to take effect.

```
sudo restart tftpd-hpa
```

If you would like to grant every user on the system permission to place files in the TFTP directory, use the following command, replacing "<TFTP_DIRECTORY>" with your chosen location.

```
sudo chmod ugo+rwx <TFTP_DIRECTORY>
```

Files in the "<TFTP_DIRECTORY>" on your Host PC can now be accessed from another machine on the same network such as the target board by simply using the IP address of the Host PC. Take note of this IP address, in a typical wired connection this will be "inet addr" listed under "eth0".

```
ifconfig
```

NFS

A network filesystem (NFS) server gives other systems the ability to mount a filesystem stored on the Host PC and exported over the network. Setting up an NFS server on your Linux Host PC gives you access to the target boards root filesystem which will allow you to quickly test applications and evaluate different filesystem setups for the target board. That is, the root filesystem for the board will actually be located on the remote host Linux machine. This enables easy access and modifications to the root filesystem during development.

Install the NFS server on your Host PC:

```
sudo apt-get install nfs-kernel-server
```

Exported filesystems are designated in the "/etc/exports" file and allow you to choose both the directory to be exported and many settings for accessing the exports. Below is an example for exporting a folder called "nfs_export-ex" located in a user's home directory.

```
sudo gedit /etc/exports
# /etc/exports
/home/<user>/nfs_export-ex *(rw,sync,no_root_squash,no_subtree_check)
```

The options (rw, sync, no_root_squash, no_subtree_check) for this folder are essential in setting up the NFS export correctly. For more information on additional options, refer to the man page for 'exports'.

- **rw enables:** read and write access when the directory is mounted
- **sync:** tells the file-system to handle local access calls before remote access
- **no_root_squash:** allows root access when mounting the file-system
- **no_subtree_check:** reduces the number of checks the server must make to ensure that an exported sub-directory is within an exported tree and also enables access to root files in conjunction with no_root_squash

After modifying this file, in order to mount the directories as an NFS, you must force the NFS server to export all of the directories listed in "/etc/exports".

```
sudo /usr/sbin/exportfs -va
```

Samba

Samba servers are an excellent way to access a Linux file-system on a Windows machine via a network connection. Using a Samba server, it is quick and easy to transfer files between systems.

To install a Samba server, use the following command:

```
sudo apt-get install samba
```

Before the Samba share can be mounted on another machine it's necessary to modify the configuration file to allow write access and access to home directories. Start by editing the "/etc/samba/smb.conf" file.

```
sudo gedit /etc/samba/smb.conf
```

Inside this file there are four specific things that need to be uncommented (remove the ';' at the beginning of the line) to enable the sharing of home folders and write access. Below is the section that must be modified:

```
#======================= Share Definitions =======================
# Un-comment the following (and tweak the other settings below to suit)
# to enable the default home directory shares. This will share each
# user's home directory as \\server\username
;[homes]
; comment = Home Directories
; browseable = yes
# By default, the home directories are exported read-only. Change the
# next parameter to 'no' if you want to be able to write to them.
; read only = no
```

The outcomes after the changes are made follow:

```
#======================= Share Definitions =======================
# Un-comment the following (and tweak the other settings below to suit)
# to enable the default home directory shares. This will share each
# user's home directory as \\server\username
[homes]
comment = Home Directories
browseable = yes
# By default, the home directories are exported read-only. Change the
# next parameter to 'no' if you want to be able to write to them.
read only = no
```


It might also be necessary to change the "workgroup" line to match the workgroup for your machine.

To apply the changes, the next step is to restart all Samba-related processes.

```
sudo restart smbd
sudo restart nmbd
```

Lastly, each user needs to have a password enabled to be able to use the Samba server. There are no rules for this password. The simplest method for choosing this password is to make it the same as the UNIX user's password, but it is not a requirement. After typing in the command below, you will be prompted to enter the password for the specified user.

```
sudo smbpasswd -a <user>
```

As mentioned in the configuration file, the samba share can be connected by accessing "\\<host machine ip>\\<user>" by either mounting a network share or using Windows explorer to navigate to it.

Building the BSP from Source

Create a directory which will house your BSP development. In this example the BSP directory is /opt/PHYTEC_BSPs/. This is not a requirement and if another location is preferred (ex. ~/PHYTEC_BSPs) feel free to modify. We recommend using /opt over your HOME directory to avoid errors attributed to ~ syntax as well as the sudo requirement for the root filesystem and automation package building. We also recommend creating a package download directory (yocto_dl) separate from the yocto tree (yocto_ti), as it makes resetting the build environment easier and subsequent build times much faster.

Setup the BSP Directory:

```
sudo mkdir /opt/PHYTEC_BSPs
cd /opt/

# /opt/ directory has root permission, change the permissions so your user account can access this folder. In
the following replace <user> with your specific username
sudo chown -R <user>: PHYTEC_BSPs

cd PHYTEC_BSPs
mkdir yocto_ti
mkdir yocto_dl
cd yocto_ti
export YOCTO_DIR=`pwd`
```

At this point you will now be able to navigate to the Yocto directory using the *$YOCTO_DIR* environment variable.

Install the Linaro Toolchain:

Run the following commands to install the Linaro Toolchain:

```
wget https://releases.linaro.org/15.05/components/toolchain/binaries/arm-linux-gnueabihf/gcc-linaro-4.9-2015.05-
x86_64_arm-linux-gnueabihf.tar.xz
tar -Jxvf gcc-linaro-4.9-2015.05-x86_64_arm-linux-gnueabihf.tar.xz -C /opt/PHYTEC_BSPs
```

Download the BSP Meta Layers

Download the manifest file for the AM57xx PD15.1-rc3 BSP:

```
cd $YOCTO_DIR
repo init -u https://stash.phytec.com/scm/pub/manifests-phytec.git -b am57xx -m am57xx-PD15.1-rc3.xml
```

Download the Yocto meta layers specified in the manifest file:

```
repo sync
```

Start the Build

Run the Yocto build directory setup script. The TEMPLATECONF variable is used to set the source of the local configuration files (conf/bblayers.conf and conf/local.conf), which are located in the meta-phytec layer:

```
cd $YOCTO_DIR
TEMPLATECONF=$YOCTO_DIR/sources/meta-phytec/conf MACHINE=am57xx-phycore-rdk source sources/oe-core/oe-init-
build-env build
```

Add the the new download directory to build/conf/local.conf:

```
DL_DIR ?= "/opt/PHYTEC_BSPs/yocto_dl"
```

Maximize build efficiency by modifying the BB_NUMBER_THREADS variable to suit your host development system. This sets the maximum number of tasks that BitBake should run in parallel. Also set the variable PARALLEL_MAKE to specify the number of threads that make can run. By default, these are set to 4 in build/conf/local.conf:

```
# Parallelism options - based on cpu count
BB_NUMBER_THREADS ?= "4"
PARALLEL_MAKE ?= "-j 4"
```


The source link in the dt recipe is broken and the mirror provides a different source which leads to a checksum mismatch. To fix this issue, open the sources/meta-arago/meta-arago-extras/recipes-devtools/dt/dt_17.07.bb recipe and amend the source uri from

```
SRC_URI = "http://home.comcast.net/~SCSIguy/SCSI_FAQ/RMiller_Tools/ftp/dt/dt-source.tar.gz"
```

to

```
SRC_URI = "http://arago-project.org/files/sources/dt-source.tar.gz"
```

The setup is complete and you now have everything to complete a build. This BSP has been tested with the arago-core-tisdk-image, it is suggested that you start with this image image before building other images. Alternate images are located in various meta layers at *meta*/recipes*/images/*.bb*. They can be found using the command *bitbake-layers show-recipes "*-image*"* in $YOCTO_DIR/build/.

The following will start a build from scratch including installation of the toolchain as well as bootloader, Linux kernel, and root filesystem images.

```
cd $YOCTO_DIR/build
export PATH=/opt/PHYTEC_BSPs/gcc-linaro-4.9-2015.05-x86_64_arm-linux-gnueabihf/bin:$PATH
MACHINE=am57xx-phycore-rdk bitbake arago-core-tisdk-image
```

Built Images

All images generated by bitbake are deployed to *$YOCTO_DIR/build/arago-tmp-external-linaro-toolchain/deploy/images/<machine>*:

- **Bootloader**: u-boot.img, MLO
- **Kernel**: zImage
- **Kernel device tree file**: zImage-am57xx-phycore-rdk.dtb
- **Root Filesystem**: tisdk-rootfs-image-am57xx-phycore-rdk.tar.gz

Source Locations:

- **Kernel:** *$YOCTO_DIR/build/arago-tmp-external-linaro-toolchain/work/am57xx_phycore_rdk-linux-gnueabi/linux-phytec-ti/4.1.6+git_am57xx-PD15.1-rc3-b39-r7d/git/*
  - The device tree file to modify within the linux kernel source is: *am57xx-phycore-rdk.dts* and its dependencies.
- **u-boot:** *$YOCTO_DIR/build/arago-tmp-external-linaro-toolchain/work/am57xx_phycore_rdk-linux-gnueabi/u-boot-phytec/2015.07+git_am57xx-PD15.1-rc3-b20-r0/git/*

Build Time Optimizations

The build time will vary depending on the package selection and Host performance. Beyond the initial build, after making modifications to the BSP, a full build is not required. Use the following as a reference to take advantage of optimized build options and reduce the build time.

To rebuild Barebox:

```
bitbake u-boot-phytec -f -c compile && bitbake u-boot-phytec
```

To rebuild the Linux kernel:

```
bitbake linux-phytec-ti -f -c compile && bitbake linux-phytec-ti
```

The Yocto project's Bitbake User Manual provides useful information regarding build options: **http://www.yoctoproject.org/docs/1.8/bitbake-user-manual/bitbake-user-manual.html**

Customizing the BSP

We recommend you create your own layer and make changes to the existing BSP there. This will make it easier to update the BSP. Instructions and tips on creating your own layer are available here: **http://www.yoctoproject.org/docs/1.8/dev-manual/dev-manual.html#creating-your-own-layer**.

Appending Recipes

To modify an existing recipe in your own layer, use a bbappend file. The following is an example of modifying the u-boot-phytec_2015.07 recipe, u-boot-phytec_2015.07.bb, located at *$YOCTO_DIR/sources/meta-phytec/recipes-bsp/u-boot/u-boot-phytec_2015.07.bb*.

Create a *recipes-bsp/barebox/* directory in your own meta-layer to place the bbappend file in. Make sure that the new file matches the .bb file name exactly. Alternatively, you may use % after the underscore in place of the specific version for portability with future versions of the recipe.

```
mkdir $YOCTO_DIR/sources/<YOUR_META_LAYER>/recipes-bsp/u-boot/
vim $YOCTO_DIR/sources/<YOUR_META_LAYER>/recipes-bsp/u-boot/u-boot-phytec_%.bbappend
```

For information on how to write a recipe, see chapter 5.3 of the Yocto Development Manual: **http://www.yoctoproject.org/docs/current/dev-manual/dev-manual.html#understanding-recipe-syntax**

Adding Packages to the build

There are various ways to add a package to the BSP. For example, packages and package groups can be added to image recipes. See the Yocto Development manual for how to customize an image: **http://www.yoctoproject.org/docs/current/dev-manual/dev-manual.html#usingpoky-extend-customimage-imagefeatures**.

The following instructions demonstrate how to add a package to the local build of the BSP. First, search for the corresponding recipe and which layer the recipe is in. This link is a useful tool for doing so: http**://layers.openembedded.org/layerindex/branch/daisy/layers/**.

If the package is in the meta-openembedded layer, the recipe is already available in your build tree. Add the following line to *$YOCTO_DIR/build/conf/local.conf*:

```
IMAGE_INSTALL_append = " <package>"
```


The leading whitespace between the " and the package name is necessary for the append command.

If you need to add a layer to the BSP, clone or extract it to the $YOCTO_DIR/sources/ directory. Then, modify *$YOCTO_DIR/build/conf/bblayers.conf* to include this new layer in BBLAYERS:

```
BBLAYERS += "${BSPDIR}/sources/<new_layer>"
```

Configuring the Kernel

The kernel configuration menu allows the user to adjust drivers and support included in a Linux Kernel build. run the following command from the build directory:

```
cd $YOCTO_DIR/build
bitbake linux-phytec-ti -c menuconfig
```

Then rebuild the kernel:

```
bitbake linux-phytec-ti -f -c compile && bitbake linux-phytec-ti
```

To rebuild the root filesystem:

```
bitbake -f arago-core-tisdk-image
```

Customizing the Device Tree

The device tree is a data structure for describing hardware, and is a way of separating machine specific information from the kernel. For information on the device tree concept, devicetree.org is a good source: **http://devicetree.org/Device_Tree_Usage**.

Device trees for PHYTEC products consist of a board DTS file, a SOM dtsi and a carrier board dtsi. The SOM dtsi includes the processor dtsi and contains definitions for all devices that are located on the SOM, such as eMMC flash. Peripherals whose signals are routed through the SOM but whose hardware is located on the carrier board are defined in the carrier board dtsi, such as MMC. All of the peripherals in these files are enabled or disabled in am57xx-phycore-rdk.dts.

Example:

To disable a peripheral such as EEPROM, change the status of the i2c_eeprom in *arch/arm/boot/dts/am57xx-phycore-rdk.dts* from "okay" to "disabled":

```
&i2c_eeprom {
         status = "disabled";
}
```

The kernel source directory has very good documentation and examples on what bindings are supported for specific peripherals: *Documentation/devicetree/bindings/.*

Creating a Bootable SD Card

The process requires an SD card reader operational under Linux to format and access the Linux partition of the card. If you do not have SD card access under Linux then copying the bootloader and mounting the root filesystem on SD/MMC card will not be possible.

1. To format the SD card, you may use the script provided by TI, called "create-sdcard.sh". The script is available **here**. The script will also be built with the BSP, and can be found in the tarball processor-sdk-linux-image-am57xx-phycore-rdk.tar.gz (located in the bin/ directory. For more information regarding the script, see**: http://processors.wiki.ti.com/index.php/Processor_SDK_Linux_create_SD_card_script**

If using pre-built images provided by PHYTEC, the exact image names are listed in the instructions below. If you have built your own images, then the images are located in: *$YOCTO_DIR/build/arago-tmp-external-linaro-toolchain/deploy/images/<machine>/.*

Once the SD card has been formatted, you may update the kernel, root filesystem, and barebox individually as well:

Root Filesystem

1. If modifying the root filesystem, remove the existing:

   ```
   sudo rm -rf /media/rootfs/*
   ```

2. Load the new filesystem to the SD Card.

   ```
   sudo tar -zxf tisdk-rootfs-image-am57xx-phycore-rdk-20160128044033.rootfs.tar.gz -C /media/rootfs; sync;
   ```

Kernel

If intending to replace the kernel and root filesystem with images from the same build, this step can be skipped. The root filesystem already contains the kernel and DTB files in its *boot/* directory.

1. If modifying the kernel, remove the existing kernel image and device tree binary files.

```
sudo rm /media/rootfs/boot/zImage
sudo rm /media/rootfs/boot/am57xx-phycore-rdk.dtb
```

2. Load the new Linux kernel and device tree binary to the SD Card. Note that u-boot expects the kernel to be named "zImage" and the DTB file to be named "am57xx-phycore-rdk.dtb":

```
sudo cp zImage--4.1.6+git_am57xx-PD15.1-rc3-b39-r7d-am57xx-phycore-rdk-20160128044033.bin /media/rootfs
/boot/zImage; sync;
sudo cp zImage--4.1.6+git_am57xx-PD15.1-rc3-b39-r7d-am57xx-phycore-rdk-20160128044033.dtb /media/rootfs
/boot/am57xx-phycore-rdk.dtb; sync;
```

Bootloader

1. Remove the existing barebox and MLO images:

```
rm /media/boot/u-boot.img
rm /media/boot/MLO
```

2. Copy the new images to the SD Card:

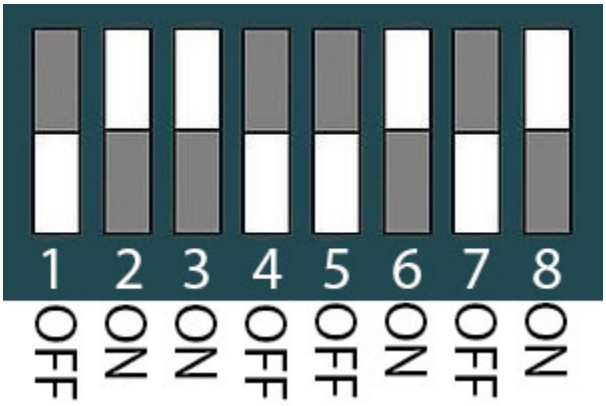
```
cp u-boot-am57xx-phycore-rdk-2015.07+git_am57xx-PD15.1-rc3-b20-r0.img /media/boot/u-boot.img; sync
cp MLO-am57xx-phycore-rdk-2015.07+git_am57xx-PD15.1-rc3-b20-r0 /media/boot/MLO; sync
```

Boot Configurations

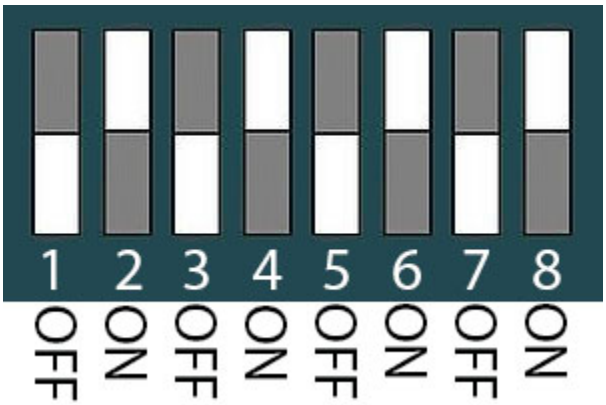
Selecting Boot Modes

The bootloader, one of the key software components included in the BSP, completes the required hardware initializations to download and run operating system images. The boot mode, selected from the S5 dipswitch on the Carrier Board, determines the location of the primary bootloader. Set the S5 dipswitch correspondingly:

SD Card



eMMC



Once the boot switch has been set appropriately, press the power button S2 on the phyCORE-AM57xx carrier board to power on the board.

Basic Settings

After application of power, approximately three seconds are allotted for the user to hit any key which will halt autoboot and enter U-Boot:

```
COM4 - PuTTY

U-Boot SPL 2014.07 (Sep 28 2015 - 13:22:53)
DRA752 ES1.1
reading args
spl_load_image_fat_os: error reading image args, err - -1
reading u-boot.img
reading u-boot.img


U-Boot 2014.07 (Sep 28 2015 - 13:22:53)

CPU  : DRA752 ES1.1
Board: phyCORE-AM57xx
I2C:   ready
DRAM:  2 GiB
MMC:   OMAP SD/MMC: 0, OMAP SD/MMC: 1, OMAP SD/MMC: 2
reading uboot.env

** Unable to read "uboot.env" from mmc0:1 **
Using default environment

SATA link 0 timeout.
AHCI 0001.0300 32 slots 1 ports 3 Gbps 0x1 impl SATA mode
flags: 64bit ncq stag pm led clo only pmp pio slum part ccc apst
scanning bus for devices...
Found 0 device(s).
SCSI:  Net:   <ethaddr> not set. Validating first E-fuse MAC
cpsw
Hit any key to stop autoboot:  0
U-Boot#
```

help *is a useful tool in U-Boot to show available commands and usage.*

Network Settings

You can check the target's default environment settings by running the following:

```
printenv
```

The *ethaddr* variable is the MAC id of the target. This is a pre-programmed value which is read from the E-fuse and matches the sticker on the SOM. Set U-boot's network environment variables to match your required network settings:

```
setenv ipaddr ###.###.###.###
setenv serverip ###.###.###.##.
setenv gatewayip ###.###.###.###
setenv netmask ###.###.###.###
setenv tftploc <TFTP image location>
setenv rootpath /<NFS mount location>
```

- **ipaddr -** A dedicated IP address for the SOM. This is crucial if TFTP will be used for updating the device's images at any point.
- **serverip -** IP address of the host or another machine. serverip corresponds to where the TFTP directory, if it exists, is located.
- **gatewayip -** Gateway IP for the network. This is only necessary if the TFTP directory is located on another network.
- **netmask -** Netmask for the network: typically 255.255.255.0. This is only necessary if the TFTP directory is located on another network.
- **tftploc (required for TFTP) -** Location of the path to the images on the TFTP server on the host system, setup in Section 4.4.1. Set the variable accordingly by referencing the following examples:

File Path	U-Boot Command
/var/lib/tftpboot/PHYTEC/am57xx/PD15.1-rc3	`setenv tftploc PHYTEC`
/var/lib/tftpboot	`setenv tftploc`

- **rootpath (required for NFS) -** Location of the path to the NFS Directory on the host system, set up in Section 4.4.2. Ex: /home/<user>/NFS

Use the following command to verify that all of the environment variables are set as intended:

```
printenv
```

Saving Configurations

After confirming the environment variables are correct, save them and continue on to the next section to set the correct kernel and root filesystem boot location:

```
saveenv
```

Boot Options

The target can be booted from on-board media or from a development host via network. In our standard configuration, this BSP release loads the kernel and root filesystem from eMMC.

For booting via network, the development host is connected to the phyCORE-AM57xx Rapid Development Kit with a serial cable and via Ethernet; the embedded board boots into the bootloader, then issues a TFTP request on the network and boots the kernel and device tree from the TFTP server on the host. Then, after decompressing the kernel into RAM and starting it, the kernel mounts its root filesystem via the NFS server on the host. This method is especially useful for development purposes as it provides a quick turnaround while testing the kernel and root filesystem.

Stand-Alone SD/MMC Card Boot

To configure U-boot to boot the kernel from SD, modify the *boot_mmc* environment variable to search SD before eMMC, and set the *bootcmd* environment variable to make it the default setting:

```
setenv boot_mmc 'run findfdt; setenv mmcdev 0;setenv bootpart 0:2;setenv finduuid 'part uuid mmc 0:2 uuid';run
envboot;run mmcboot;setenv mmcdev 1; setenv bootpart 1:2; setenv finduuid 'part uuid mmc 1:2 uuid'; run
mmcboot;'
setenv bootcmd 'run boot_mmc'
saveenv
```

Remote Boot

To configure U-Boot to boot the kernel from TFTP and mount the root filesystem from NFS, configure the network as described above and then set the *boot cmd* environment variable to make it the default setting:

```
setenv bootcmd 'run boot_net'
saveenv
```

Stand-Alone eMMC Boot

By default, the phyCORE-AM57xx kit is set up to boot the Linux kernel and root filesystem from eMMC. If switching from another boot configuration back to eMMC, set the *bootcmd* environment variable to use the predefined *boot_mmc* environment variable to make it the default setting:

```
setenv bootcmd 'run boot_mmc'
saveenv
```

Custom Boot

Unique boot configurations can be created by defining the desired environment variable settings and setting *bootcmd* to run its contents. The following is an example:

Example

Boot the Linux Kernel via TFTP with Root Filesystem on SD:

```
setenv customboot 'tftp ${loadaddr} ${tftploc}${bootfile}; tftp ${fdtaddr} ${tftploc}${fdtfile}; run
mmcargs; bootz ${loadaddr} - ${fdtaddr}'
setenv bootcmd 'run customboot'
saveenv
```

Flashing Images to eMMC

The board Development Kit is delivered with a pre-flashed bootloader. The following instructions for flashing images from SD card will be useful if you want to:

- Flash images because eMMC is empty
- Upgrade to a new release
- Use custom built images

The images to be flashed will need to be copied to the */boot* or /rootfs/boot/ partition of a properly formatted SD card as described in the Creating a Bootable SD Card section of the Quickstart.

Partition eMMC from U-boot

Write a GPT partition table to eMMC. Create UUIDs for the disk and each partition by executing the following on the **host** machine:

```
uuidgen
<first UUID generated>

uuidgen
<second UUID generated>

uuidgen
<third UUID generated>
```

After making all required connections, power on the board and enter U-Boot. Set the UUIDs for the disk and rootfs to the generated values:

```
U-Boot # setenv partitions 'uuid_disk=${uuid_gpt_disk};name=env,start=1MiB,size=1MiB,uuid=${uuid_gpt_env};
name=rootfs,start=3MiB,size=-,uuid=${uuid_gpt_rootfs}'
U-Boot # setenv uuid_gpt_disk <first UUID>
U-Boot # setenv uuid_gpt_rootfs <second UUID>
U-Boot # setenv uuid_gpt_env <third UUID>
U-Boot # gpt write mmc 1 ${partitions}
U-boot # reset
```

The partition gpt partition will be visible after a reset. (Note that mmc0 corresponds with the SD card slot interface, while mmc1 corresponds with eMMC):

```
U-Boot # mmc dev 1
U-Boot # mmc part
```

U-Boot

Copy the MLO and u-boot.img from the /boot partition of the SD card (connector X2, mmc0 in U-Boot) to eMMC (mmc1 in U-Boot):

```
U-Boot # mmc dev 0
U-Boot # mmc rescan
U-Boot # mmc dev 1

U-Boot # fatload mmc 0 ${loadaddr} MLO
U-Boot # mmc write ${loadaddr} 0x100 0x100
U-Boot # mmc write ${loadaddr} 0x200 0x100

U-Boot # fatload mmc 0 ${loadaddr} u-boot.img
U-Boot # mmc write ${loadaddr} 0x300 0x400
```

Root Filesystem

Copy the root filesystem from the /rootfs partition of the SD card (connector X2, mmc0 in U-Boot) to eMMC (mmc1 in U-boot):

```
U-boot # mmc dev 1
U-boot # ext4load mmc 0:2 ${loadaddr} rootfs.ext4
U-boot # mmc write ${loadaddr} 0x1800 [rootfs.ext4 size in bytes divided by 512, in hex]
```

This assumes the SD card was created with TI's **create-sdcard.sh** script. If the SD card is formatted differently, the *ext4load* command may need to be replaced by *fatload*.

Additionally, the rootfs.ext4 image is not loaded to the card by default.