

# phyFLEX-i.MX6 RDK Quickstart imx6-3.14-PL15.1.0

This Quickstart provides you with the tools and know-how to install and work with the Linux Board Support Package (BSP) for the phyFLEX-i.MX6 Rapid Development Kit (RDK). 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 phyFLEX-i.MX6 Hardware Manual for specific information on board-level features such as jumper configuration, memory mapping and pin layout for the phyFLEX-i.MX6 System on Module (SOM) and baseboard. Additionally, gain access to the SOM, mapper board, and baseboard schematics for the phyFLEX-i.MX6 RDK 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 [Bootimg 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 [Download the BSP Meta Layers](#)
  - 5.2 [Setup the Local Build Configuration](#)
  - 5.3 [Start the Build](#)
  - 5.4 [Built Images](#)
  - 5.5 [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 [Kernel](#)
  - 7.2 [Root Filesystem](#)
  - 7.3 [Bootloader](#)
- 8 [Boot Configurations](#)
  - 8.1 [Selecting Boot Modes](#)
    - 8.1.1 [SPI NOR \(Default\)](#)
    - 8.1.2 [SD Card](#)
  - 8.2 [Basic Settings](#)
    - 8.2.1 [Network Settings](#)
    - 8.2.2 [Saving Configurations](#)
  - 8.3 [Boot Options](#)
    - 8.3.1 [Stand-Alone NAND Boot](#)
    - 8.3.2 [Remote Boot](#)
    - 8.3.3 [Stand-Alone SD/MMC Card Boot](#)
    - 8.3.4 [Custom Boot](#)
    - 8.3.5 [Specifiend the Default Boot Mode](#)
- 9 [Flashing Images](#)
  - 9.1 [Barebox](#)
  - 9.2 [Kernel](#)
  - 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

- phyFLEX-i.MX6 System on Module (PFL-A-02)
  - phyFLEX Baseboard (PBA-B-01)
  - phyFLEX-Mapper i.MX6 (FLM-A-XL1) — optional
  - Serial cable (RS-232)
  - Ethernet cable
  - AC adapter supplying 12 VDC / min. 2 A
  - LCD Display — optional
- The following are supported:

- EDT ETM0700G0DH6 TTL (LCD-018-070-KAP) - **default**

## Software

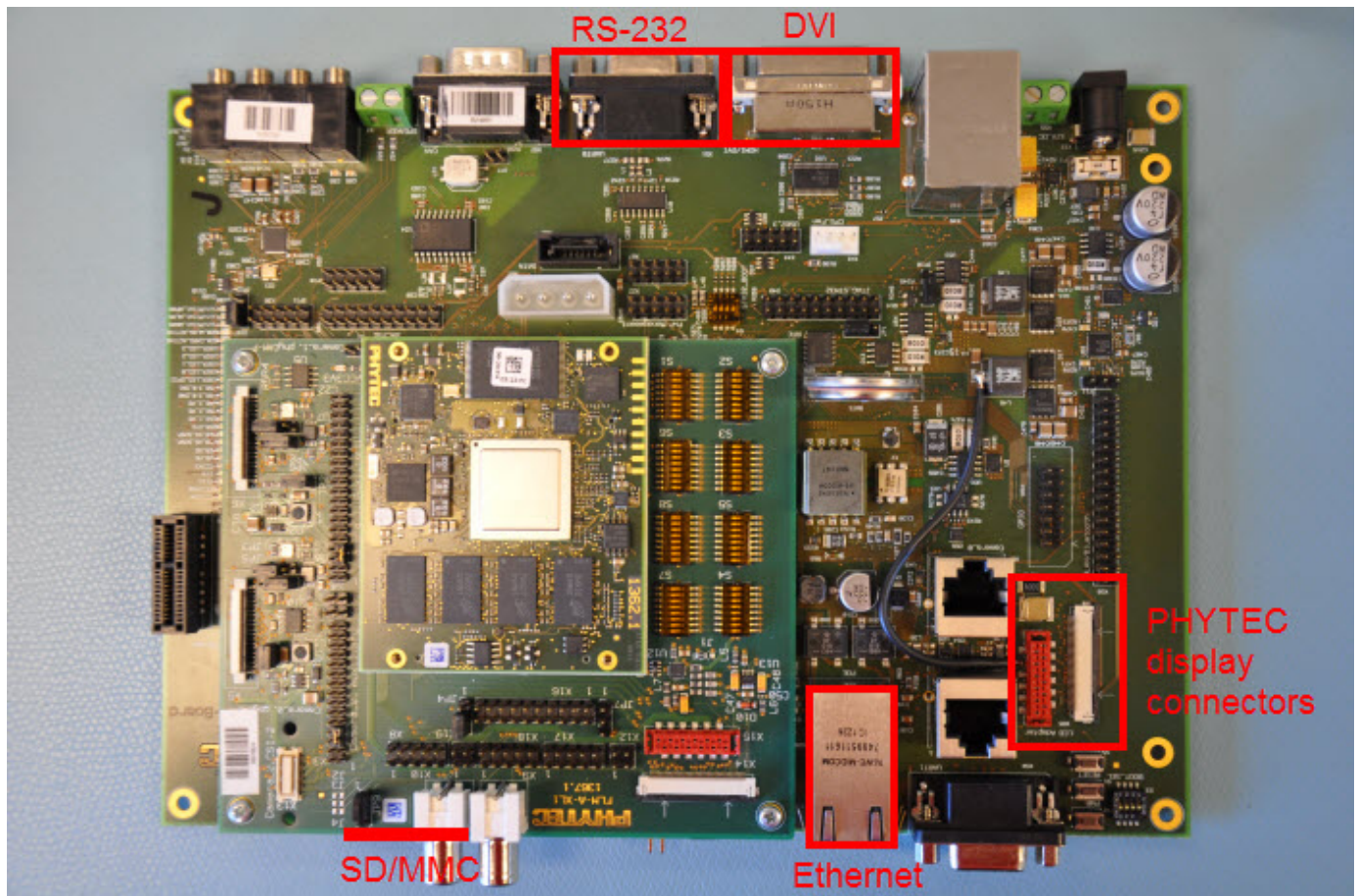
- A modern GNU/Linux Operating host system either natively or via a virtual machine:
  - Ubuntu 12.04 LTS 64-bit recommended. Other distributions will likely work, please note that some setup information as well as OS-specific commands and paths may differ.
  - If using a virtual machine, VMWare Workstation, 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 40-50GB 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 phyFLEX-i.MX6 RDK that will be used in this Quickstart.

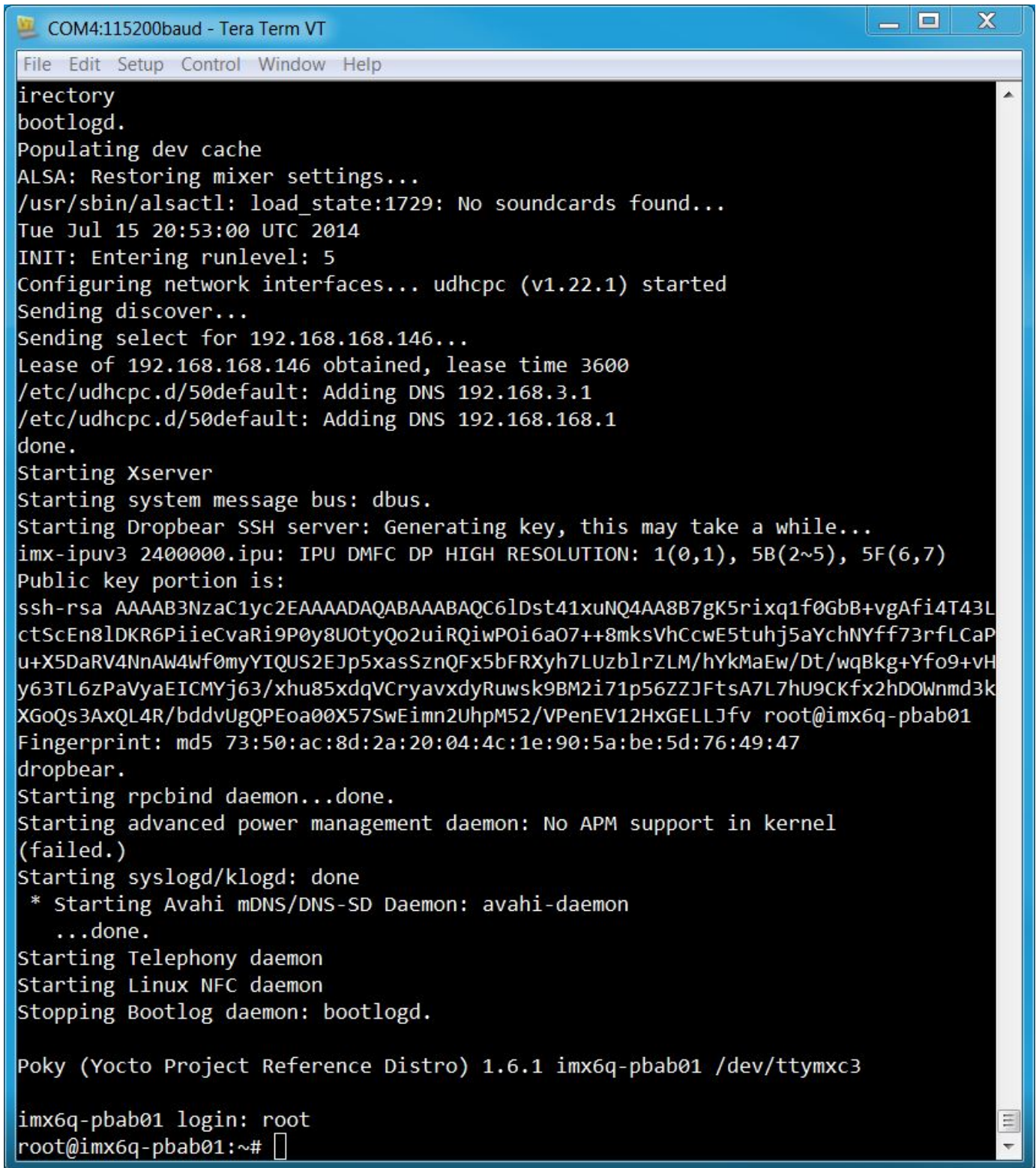


## Booting the Pre-Built Images

The section was designed to show you how to boot the phyFLEX-i.MX6 RDK with the pre-built demo images.

1. Insert a bootable SD card into the SD0 (X57) slot on the baseboard. Since the default boot mode for the imx6-3.14-PL15.1.0 release loads the kernel and root filesystem from SD card, a bootable SD card with *barebox.bin*, *zImage*, *zImage-imx6q-phytec-pbab01.dtb*, and *core-image-directfb-imx6q-pbab01.tar.bz2* extracted on the SD card is required. Instructions for creating an SD card are provided in the [Creating a Bootable SD Card](#) section of the Quickstart.
2. If you ordered a PHYTEC Display such as the LCD-018-070-KAP, plug this into the LCD power and data connectors at X65.

3. Connect the kit supplied serial cable from a free serial port on your host PC to the DB9 connector X51 on the carrier board. This is the UART0 communication channel with the i.MX6 at RS-232 levels.
4. Connect the kit supplied Ethernet cable from the Ethernet connector X28 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.
5. 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.
6. Plug the kit supplied 12 V power adapter into the power connector X12 on the carrier board. You will instantly see power LEDs VCC12IN, VCC12, VCC5\_X, VCC5, and VCC3V3 on the carrier board as well as D1 on the SOM light up 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 *root@imx6q-pbab01* 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.



```

COM4:115200baud - Tera Term VT
File Edit Setup Control Window Help
irectory
bootlogd.
Populating dev cache
ALSA: Restoring mixer settings...
/usr/sbin/alsactl: load_state:1729: No soundcards found...
Tue Jul 15 20:53:00 UTC 2014
INIT: Entering runlevel: 5
Configuring network interfaces... udhcpc (v1.22.1) started
Sending discover...
Sending select for 192.168.168.146...
Lease of 192.168.168.146 obtained, lease time 3600
/etc/udhcpc.d/50default: Adding DNS 192.168.3.1
/etc/udhcpc.d/50default: Adding DNS 192.168.168.1
done.
Starting Xserver
Starting system message bus: dbus.
Starting Dropbear SSH server: Generating key, this may take a while...
imx-ipuv3 2400000.ipu: IPU DMFC DP HIGH RESOLUTION: 1(0,1), 5B(2~5), 5F(6,7)
Public key portion is:
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQAC6lDst41xuNQ4AA8B7gK5rixq1f0Gbb+vgAfi4T43L
ctScEn8lDKR6PiieCvaRi9P0y8U0tyQo2uirQiwP0i6a07++8mksVhCcwE5tuhj5aYchNYff73rfLCaP
u+X5DaRV4NnAW4Wf0myYIQU52EJp5xasSznQFx5bFRXyh7LUzblRZLM/hYkMaEw/Dt/wqBkg+Yfo9+vH
y63TL6zPaVyaEICMYj63/xhu85xdqVCryavxdyRuwsK9BM2i71p56ZZJFtsA7L7hU9CKfx2hDOWNmd3k
XGoQs3AXQL4R/bddvUgQPEoa00X57SwEimn2UhpM52/VPenEV12HxGELLJfv root@imx6q-pbab01
Fingerprint: md5 73:50:ac:8d:2a:20:04:4c:1e:90:5a:be:5d:76:49:47
dropbear.
Starting rpcbind daemon...done.
Starting advanced power management daemon: No APM support in kernel
(failed.)
Starting syslogd/klogd: done
* Starting Avahi mDNS/DNS-SD Daemon: avahi-daemon
...done.
Starting Telephony daemon
Starting Linux NFC daemon
Stopping Bootlog daemon: bootlogd.

Poky (Yocto Project Reference Distro) 1.6.1 imx6q-pbab01 /dev/ttyMXC3

imx6q-pbab01 login: root
root@imx6q-pbab01:~#

```



#### Troubleshooting

Not seeing any output on the console?

- Check that you have setup the terminal software correctly per step 5.
- Flash the release images from the PHYTEC FTP by creating a bootable SD/MMC card ([Creating a Bootable SD Card](#)), then configure the board to boot from SD/MMC ([Selecting Boot Modes](#)).

When finished with the kit demo, from the Linux command line use the **reboot** command to restart, or the **poweroff** command to shutdown the system. It is safe to remove power from the kit when *reboot: System halted* is printed on the console.

## About the Yocto BSP

The [Yocto Project](#) is a Linux embedded development environment which provides layers of meta data and tools. PHYTEC's Yocto BSPs are based on the **Poky** meta layer, which consists of the [bitbake](#) build tool, Linux base recipes, and various scripts to define a rudimentary linux system. 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-fsl-arm layer is part of the [FSL Community BSP](#), a community-driven project to provide support for Freescale ARM reference boards. The **meta-phytec** layer leverages the meta-fsl-arm layer as a base and contains recipes and classes developed by PHYTEC. This layer defines configurations for barebox, the kernel, and software specific to the phyFLEX-i.MX6.

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 gawk wget git-core diffstat unzip texinfo \
    gcc-multilib build-essential chrpath socat \
    libssl1.2-dev xterm
```



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/1.6/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?"
```

### 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"
```

## 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 filesystem. 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_fsl`), as it makes resetting the build environment easier and subsequent build times much faster.

```
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_fsl
mkdir yocto_dl
cd yocto_fsl
export YOCTO_DIR=`pwd`
```

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

## Download the BSP Meta Layers

Download the manifest file for the imx6-3.14-PL15.1.0 BSP:

```
repo init -u git://git.phytec.com/phytec-manifests.git -b imx6 -m imx6-3.14-PL15.1.0.xml
```

Download the Yocto meta layers specified in the manifest file:

```
repo sync
```

## Setup the Local Build Configuration

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:

```
TEMPLATECONF=$YOCTO_DIR/sources/meta-phytec/conf source sources/poky/oe-init-build-env build
```

Add 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 3 in `build/conf/local.conf`:

```
# Parallelism options - based on cpu count
BB_NUMBER_THREADS ?= "3"
PARALLEL_MAKE ?= "-j 3"
```



### Accept Freescale EULA

To use Vivante GPU binaries in package 'gpu-viv-bin-mx6q' you need to accept the Freescale EULA at '/opt/yocto/yocto\_fsl/sources/meta-fsl-arm/EULA'. Please read it and if you accept it, write the following in build/conf/local.conf:

```
ACCEPT_FSL_EULA = "1"
```

## Start the Build

The setup is complete and you now have everything to complete a build. This BSP has been tested with and supports the image core-image-directfb. It is suggested that you start with this image. 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 barebox, Linux kernel, and root filesystem images.

```
cd $YOCTO_DIR/build
bitbake core-image-directfb
```



If the package fetch fails, you will need to manually download the linuxptp IEEE 1588 stack to *\$DL\_DIR* (<http://sourceforge.net/projects/linuxptp/files/>). Once you've downloaded the archive (e.g. linuxptp-1.3.tgz), you will need to create a "done" file so fetch doesn't delete the file and try to refetch it:

```
cd yocto_dl
touch linuxptp-1.3.tgz.done
```

Another issue that may happen is ncurses5.9 won't be built in time for a different package in the build (says [libpanelw.so](#) is missing). Forcing the reinstall of the base ncurses package unsticks this:

```
bitbake ncurses -f -c install
```

## Built Images

All images generated by bitbake are deployed to *\$YOCTO\_DIR/build/ deploy/images/imx6q-pbab01*:

- **Bootloader:** barebox-imx6q-pbab01.imx
- **Kernel:** zImage
- **Kernel device tree file:** zImage-imx6q-phytec-pbab01.dtb
- **Root Filesystem:** core-image-directfb-imx6q-pbab01.tar.bz2
- **Kernel modules:** modules-imx6q-pbab01.tgz



The phyFLEX-i.MX6 SOMs can be populated with various sizes of DDR3 SDRAM. Since the DDR3 initialization is performed by the bootloader, different versions of barebox may be required if using a custom SOM. The available versions of barebox are:

- barebox-phytec-pbab01-1gib.img
- barebox-phytec-pbab01-1gib-1bank.img (tested)
- barebox-phytec-pbab01-2gib.img
- barebox-phytec-pbab01-4gib.img (tested)
- barebox-phytec-pbab01dl-1gib.img
- barebox-phytec-pbab01s-512mb.img

All of these images are built with the BSP, and are located in the barebox source directory: *\$YOCTO\_DIR/build/tmp/work/imx6q\_pbab01-poky-linux-gnueabi/barebox/2014.11.0-r0/git/images/*

To change which of these barebox images is deployed to the image directory, edit the machine file located at: *\$YOCTO\_DIR/sources/meta-phytec/conf/machine/imx6q-pbab01.conf* and modify the variable "BAREBOX\_IMAGE\_IMX " to the appropriate barebox image above.

Source Locations:

- **Kernel:** `$YOCTO_DIR/build/tmp/work/imx6q_pbab01-poky-linux-gnueabi/linux-fslc/3.14+gitAUTOINC+2592e3c6fc-r0/git/`
  - The device tree file to modify within the linux kernel source is: `/arch/arm/boot/dts/imx6q-phytec-pbab01.dts` and its dependencies
- **Barebox:** `$YOCTO_DIR/build/tmp/work/imx6q_pbab01-poky-linux-gnueabi/barebox/2014.11.0-r0/git/`
- **Toolchain:** `$YOCTO_DIR/build/tmp/sysroots/x86_64-linux/usr/bin/cortexa9hf-vfp-neon-poky-linux-gnueabi/arm-poky-linux-gnueabi-`

## 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 barebox -f -c compile && bitbake barebox
```

To rebuild the Linux kernel:

```
bitbake linux-fslc -f -c compile ; bitbake linux-fslc
```

The Yocto project's Bitbake User Manual provides useful information regarding build options: <http://www.yoctoproject.org/docs/1.6/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/1.6/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 barebox\_2014.11 recipe, barebox\_2014.11.0.bb, located at `$YOCTO_DIR/sources/meta-phytec/recipes-bsp/barebox/barebox_2014.11.0.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/barebox/
vim $YOCTO_DIR/sources/<YOUR_META_LAYER>/recipes-bsp/barebox/barebox_%.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/1.6/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-fslc -c menuconfig
```

Then rebuild the kernel:

```
bitbake linux-fslc -f -c compile ; bitbake linux-fslc
```

To rebuild the root filesystem:

```
bitbake -f core-image-directfb
```

## 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](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 NAND 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. The board file then enables optional nodes that were defined in the dtsi files that pertain to this specific board configuration.



To disable a peripheral such as UART3, change the status of the uart3 node in *tmp/work/imx6q\_pbab01-poky-linux-gnueabi/linux-fslc/3.14+gitAUTOINC+e6c7ae7613-r0/git/arch/arm/boot/dts/imx6qdl-phytec-pbab01.dtsi*

from "okay" to "disabled":

```
&uart3 {
    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.

If using a SD/MMC card that has already been formatted skip to the appropriate sections below – [here](#) if updating the kernel, or [here](#) if updating the Root filesystem,

1. Determine the SD card device name
  - a. The SD card device name is of the form /dev/sd[b|c|d|e]. Run the following without and with the SD card connected:

```
ls /dev/sd*
```

- b. The device that appeared on the call to ls with the SD card but not the call without is the SD card device.

2. Unmount all partitions of the SD card, using the SD card device name from Step 1:

```
umount /dev/sd[b|c|d|e]?*
```

3. Make two partitions on the SD card using *fdisk* from the Linux kernel, specifying the SD card device name from Step 1:

```
sudo fdisk /dev/[b|c|d|e]
```

```
Command (m for help): o
Building a new DOS disklabel with disk identifier 0x2fe3ef94.
Changes will remain in memory only, until you decide to write them.
After that, of course, the previous content won't be recoverable.
```

- a. Create a new primary partition (n command) with partition id C, leaving 8 MB of free space at the beginning of the card:

```
Command (m for help): n
Partition type:
   p   primary (0 primary, 0 extended, 4 free)
   e   extended
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-7626751, default 2048): 17432
Last sector, +sectors or +size{K,M,G} (17432-7626751, default 7626751): 1267432

Command (m for help): t
Partition number (1-4): 1
Hex code (type L to list codes): C
Changed system type of partition 1 to c (W95 FAT32 (LBA))
```

- b. Create a new Linux partition (n command) with partition id 83

```
Command (m for help): n
Partition type:
   p   primary (1 primary, 0 extended, 3 free)
   e   extended
Select (default p): p
Partition number (1-4, default 2): 2
First sector (2048-7626751, default 2048): 1267433
Last sector, +sectors or +size{K,M,G} (1267433-7626751, default 7626751):
Using default value 7626751

Command (m for help): t
Partition number (1-4): 2
Hex code (type L to list codes): 83
```

- c. Write the partition table to the card (w command) which will destroy all data on the SD card

```
Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.

WARNING: If you have created or modified any DOS 6.x
partitions, please see the fdisk manual page for additional
information.
Syncing disks.
```

- d. Create a filesystem on the partitions from the Linux command line:

```
sudo mkfs.vfat /dev/sd[b|c|d|e]1 -F 32 -n boot
sudo mkfs.ext3 /dev/sd[b|c|d|e]2 -L rootfs
```

4. Mount all partitions

- a. Remove and reinsert the SD card, automount will mount */media/boot* and */media/rootfs*

## Kernel

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

```
rm /media/boot/zImage
rm /media/boot/zImage-imx6q-phytec-pbab01.dtb
```

2. Load the new Linux kernel and device tree binary to the SD Card. Note that the default bootloader environment is configured to recognize "zImage" and "zImage-imx6q-phytec-pbab01.dtb" as the kernel and dts respectively:

```
cp zImage /media/boot/zImage; sync
cp zImage-imx6q-phytec-pbab01.dtb /media/boot/; sync
```

## 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 -jxf core-image-directfb-imx6q-pbab01.tar.bz2 -C /media/rootfs/; sync
```

3. Extract the kernel modules to the root partition of the SD card:

```
sudo tar -zxf modules-imx6q-pbab01.tgz -C /media/rootfs/; sync
```

4. If you intend to flash images to NAND, also copy the UBIFS filesystem image to the boot partition of the SD card:

```
cp core-image-directfb-imx6q-pbab01.ubifs /media/boot/; sync
```

## Bootloader

1. Umount each partition before copying the bootloader:

```
umount /media/boot /media/rootfs
```

2. Use the dd command to copy the barebox image to the SD card:

```
sudo dd if= barebox-imx6q-pbab01.imx of=/dev/sd[b|c|d|e] bs=512 skip=2 seek=2
```

## Boot Configurations

The default boot mode for the imx6-3.14-PL15.1.0 Yocto BSP is the following:

- **Barebox:** NOR flash or SD card
- **Kernel and device tree file:** SD card
- **Root Filesystem:** SD card

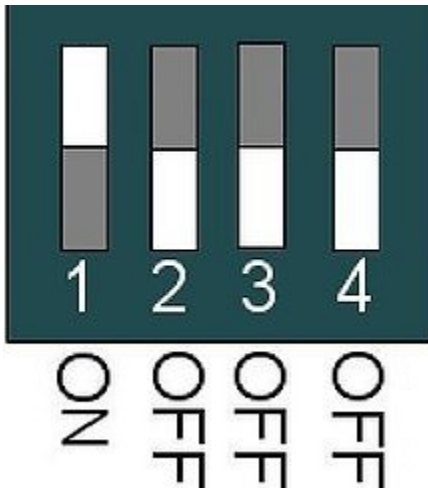
## Selecting Boot Modes

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 dipswitch on the Carrier Board, determines the location of the primary bootloader. Set the dipswitch correspondingly:

### SPI NOR (Default)

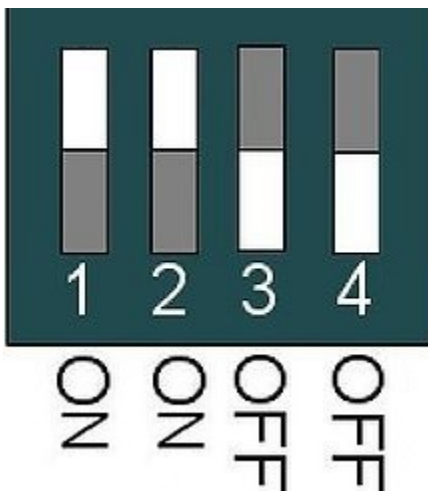
S3-1 ON

S3-2, S3-3, S3-4 OFF

**SD Card**

S3-1, S3-2 ON

S3-3, S3-4 OFF

**Basic Settings**

After application of power, approximately three seconds are allotted for the user to hit any key which will halt autoboot and enter Barebox.


```

COM1:115200baud - Tera Term VT
File Edit Setup Control Window Help

Board: phyFLEX i.MX6
registered netconsole as cs1
imx-esdhc@mci0: registered as mci0
Cannot reset the SD/MMC interface
ehci@ehci1: USB EHCI 1.00
Got valid MAC from hardware device
eth@eth0: MAC address: 50:2D:F4:05:23:FC
phyflex_init_ethernet() mii_open() ok, mdev->address = 0x3
m25p@m25p0: n25q128 (16384 Kbytes)
NAND device: Manufacturer ID: 0xec, Chip ID: 0xd3 (Samsung NAND 1GiB 3,3V 8-bit)
Scanning device for bad blocks
Bad eraseblock 817 at 0x06620000
Bad eraseblock 1216 at 0x09800000
Bad eraseblock 1239 at 0x09ae0000
Bad eraseblock 1671 at 0x0d0e0000
Bad eraseblock 2637 at 0x149a0000
Bad eraseblock 3623 at 0x1c4e0000
Bad eraseblock 6710 at 0x346c0000
Set nor0.barebox as self0 device
Use nor0.bareboxenv for default barebox environment
Malloc space: 0x47c00000 -> 0x4fbfffff (size 128 MB)
Stack space : 0x47bf8000 -> 0x47c00000 (size 32 kB)
running /env/bin/init...

Hit any key to stop autoboot:  3
barebox:/

```

 **help** is a useful tool in Barebox to show available commands and usage.

## Network Settings

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

```
devinfo eth0
```

The `ethaddr` variable is the MAC id of the target. This is a pre-programmed value which is read from the EEPROM and matches the sticker on the SOM. To modify any of the network settings, type:

```
edit /env/network/eth0
```

You should see something similar to the following, modify the variables to specify your network configuration for ETH0:

```

ipaddr=###.###.###.###
netmask=###.###.###.###
gateway=###.###.###.###
serverip=###.###.###.###

```

- **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.

- **netmask**  
Netmask for the network: typically 255.255.255.0. This is only necessary if the TFTP directory is located on another network.
- **gateway**  
Gateway IP for the network. This is only necessary if the TFTP directory is located on another network.
- **serverip**  
IP address of the host or another machine. serverip corresponds to where the TFTP directory, if it exists, is located.

## Saving Configurations

From any of the Barebox scripts, return to the Barebox prompt by pressing **CTL+D** to apply changes or **CTL+C** to cancel. To retain changes, at the Barebox prompt save the environment.

```
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 SD card. This process requires a properly formatted SD card which can be prepared using the instructions in the [Creating a Bootable SD card](#) section of the Quickstart.

For booting via network, the development host is connected to the phyFLEX-i.MX6 RDK 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 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 NAND Boot

To use the target stand-alone, the kernel and root filesystem have to be made persistent in the on-board media of the device. This is done by default in the kit, however if the kernel and root filesystem are not available in the on-board media or you would like to update them, refer to the [Flashing Images](#) section for information on flashing images.

The nand boot entry can be run from the Barebox shell:

```
boot nand
```

### Remote Boot

The network-remote boot variant is intended to be used during development because of the frequent need to rebuild the Linux kernel and root filesystem. TFTP and NFS accelerate the development process. Reflashing the newest kernel and root file system to the SOM after every new build would be very cumbersome and time consuming. All that is needed is an Ethernet connection and a network aware bootloader which can fetch the kernel from a TFTP server.

Restart the board and stop autoboot to go into the Barebox shell. Run the command:

```
boot net
```

### Stand-Alone SD/MMC Card Boot

The SD/MMC card boot variant is an alternative stand-alone boot option. All that is needed is a properly formatted (see the [Creating a Bootable SD Card](#) section of the Quickstart) SD/MMC card.

Restart the board and stop autoboot to go into the Barebox shell. Run the command:

```
boot mmc
```

### Custom Boot

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.

- In Barebox, create your own boot entry, for example named *custom*:

```
edit /env/boot/custom
```

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

```
#!/bin/sh

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

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

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

- **`<kernel_loc_bootm.image>`**  
Specifies the location of the Linux kernel image
  - `/dev/nand0.kernel.bb` - To boot the Linux kernel from NAND
  - `$(path)/zImage` - To boot the Linux kernel via TFTP
  - `/mnt/mmc/zImage` - To boot the Linux kernel from SD/MMC card
- **`<dtb_loc_bootm.oftree>`**  
Specifies the location of the device tree binary
  - `/dev/nand0.oftree.bb` - To boot the device tree binary from NAND
  - `$(path)/zImage-imx6q-phytec-pbab01.dtb` - To boot the device tree binary via TFTP
  - `/mnt/mmc/zImage-imx6q-phytec-pbab01.dtb` - To boot the device tree binary from SD/MMC card
- **`<rootfs_loc_dyn.root>`**  
Specifies the location of the root filesystem
  - `root=ubi0:root ubi.mtd=root rootfstype=ubifs` - To mount the root filesystem from NAND
  - `root=/dev/nfs nfsroot=$nfsroot,vers=3,udp rw consoleblank=0` - To mount the root filesystem via NFS
  - `root=/dev/mmcblk0p2 rootfstype=ext3 rootwait` - To mount the root filesystem from SD/MMC card
- **`<nfs_root_path>`**  
Only required if mounting the root filesystem from NFS. Replace with the following:
  - `nfsroot="/home/${global.user}/nfsroot/${global.hostname}"`

Once complete with file modifications exit the editor using **CTRL+D**. Save the environment:

```
saveenv
```

To run your custom boot entry from the Barebox shell:

```
boot custom
```

## Specifiend the Default Boot Mode

The device is configured by default to boot the kernel and mount the root filesystem from the bootsource set for Barebox in the dipswitch ([Selecting Boot Modes](#)). This default setting can be changed by adding or modifying the `global.boot.default` environment variable.

```
edit /env/config-board
```

Comment out the if else statements which set the boot sequence depending on the Barebox bootsource, and add the `global.boot.default` variable to set to the desired boot source. This variable can be set to any of the entries in `/env/boot`. For example, to set the default boot configuration to net, `global.boot.default` should be set in the following way:

```
global.boot.default=net
```

Exit the editor by **CTRL+D** and save the environment (saveenv). On a reset or power cycle the new default boot source will take affect. Similarly it will be used in the Barebox shell when executing the following:

```
boot
```

## Flashing Images

The phyFLEX-i.MX6 Rapid Development Kit is delivered with a pre-flashed bootloader. The following instructions for flashing images from TFTP or SD card will be useful if you want to:

- Flash images because NOR/NAND are empty
- Upgrade to a new release
- Use custom built images

The images to be flashed will need to be copied to the exported TFTP directory or the `/boot` partition of a properly formatted SD card as described in the [Creating a Bootable SD Card](#) section of the Quickstart.

After making all required connections, power on the board and enter Barebox:

- If flashing from TFTP, connect an Ethernet cable from the Ethernet connector *ETH0/POE* on the Carrier Board to your network hub, router, or switch.
- If flashing from SD card, insert a correctly formatted SD card into the SD/MMC card slot connector X57 on the Carrier Board.
- Connect a serial RS-232 cable from a free port on your Host PC to the connector X51 on the Carrier Board.
- Start your favorite terminal software (Windows: PuTTY or TeraTerm; Linux: Minicom) on your Host PC and configure it for 115200 baud, 8 data bits, no parity, and 1 stop bit (8n1).
- Power on your board by connecting the 5 V wall adapter to X12 on the Carrier Board.
- After application of power, within approximately three seconds, hit any key to halt autoboot and enter Barebox. If an SD card was inserted, the `/boot` partition will automount to `/mnt/mmc`. If an Ethernet cable is connected and the network has been configured, TFTP will automount to `/mnt/tftp`.

If flashing from TFTP, additional setup to configure the Barebox environment variables to meet your network environment and development host settings is required. The current network settings can be checked in `/env/network/eth0`.

If you need to change you network configuration, type:

```
edit /env/network/eth0
```

Edit the settings as described in the [Network Settings](#) section of the Quickstart. Save the environment and reboot the board, this will automount your tftp server at boot to `/mnt/tftp`.

## Barebox

If you would like to upgrade, have custom Barebox requirements, or are interested in seeing the version you built in action, follow the steps below:

barebox.bin should be copied to your TFTP exported directory or the `/boot` partition of the SD card depending on your chosen flashing procedure.



Be sure that the `barebox-imx6q-pbab01.imx` file has the correct RAM size for your version of the phyFLEX-i.MX6 SOM. This is configured as part of the build settings.

- Copy the new Barebox from your tftp-server or SD card into the module's RAM:

### Method: TFTP

```
cp /mnt/tftp/barebox-imx6q-pbab01.imx .
```

### Method: SD/MMC

```
cp /mnt/mmc/barebox.bin .
```

- Store the Barebox image into SPI NOR Flash:

```
erase /dev/m25p0.barebox
cp barebox-imx6q-pbab01.imx /dev/m25p0.barebox
```

- To restore the barebox environment in SPI NOR Flash to the default environment:

```
erase /dev/m25p0.barebox-environment
```



If something goes wrong and you don't have a bootloader anymore on your module you need to boot from an SD card into Barebox (set the DIP-switch as stated in [Boot Configurations](#)) and then do the flashing. See the [Creating a Bootable SD Card](#) section of this Quickstart for a description of how to create a bootable SD card.

## Kernel

Placing the kernel into NAND Flash allows booting the system without the need for the TFTP hosted kernel image. This is the most common place to put the kernel in a stand-alone application. Normally development is done using a TFTP hosted kernel image until the configuration has become more stable and is unlikely to change frequently. Once stable, the kernel image can be moved to NAND Flash. This section assumes a Linux kernel with file name zImage is available on the exported TFTP directory or /boot partition of the SD card depending on your chosen flashing procedure.

- Erase the area of NAND Flash reserved for the kernel image and device tree binary partitions:

```
erase /dev/nand0.kernel.bb
erase /dev/nand0.oftree.bb
```

- Copy the Linux kernel from your tftp-server or SD card and store it into the NAND Flash:

**Method: TFTP**

```
cp /mnt/tftp/zImage /dev/nand0.kernel.bb
cp /mnt/tftp/zImage-imx6q-phytec-pbab01.dtb /dev/nand0.oftree.bb
```

**Method: SD/MMC**

```
cp /mnt/mmc/zImage /dev/nand0.kernel.bb
cp /mnt/mmc/zImage-imx6q-phytec-pbab01.dtb /dev/nand0.oftree.bb
```

## Root Filesystem

Similar to the Linux kernel, placing the root filesystem into NAND Flash allows booting the system without the need for a remote connection to the NFS server. This section assumes a root filesystem of the form core-image-directfb-imx6q-pbab01.ubifs is available on the exported TFTP directory or /boot partition of the SD card depending on your chosen flashing procedure. Note that you should not flash Linux's root filesystem into NAND the same way as you did with Linux kernel.

- Ubiufs keeps erase counters within the NAND in order to be able to balance write cycles equally over all NAND sectors. So if there's already an ubifs on your module and you want to replace it with a new one, using erase and cp will also erase these erase counters, and this should be avoided. Instead, execute the following:

```
ubiiformat /dev/nand0.root
ubiattach /dev/nand0.root
ubimkvol /dev/ubi0 root 0
```

- Copy the root filesystem from your tftp-server or SD card and store it into the NAND Flash:

**Method: TFTP**

```
cp /mnt/tftp/core-image-directfb-imx6q-pbab01.ubifs /dev/ubi0.root
```

**Method: SD/MMC**

```
cp /mnt/mmc/core-image-directfb-imx6q-pbab01.ubifs /dev/ubi0.root
```



If you experience issues flashing the root filesystem, erase the entire NAND partition before reflashing the images:

```
erase /dev/nand0.bb
```