./kernel concepts

LINUX@CHIP Yocto BSP

Developers Manual

Project title	LINUX@CHIP Yocto BSP
Author	Florian Boor
Project	LINUX@CHIP
Customer	Beck IPC GmbH
Project URL	https://github.com/beck-ipc
Created	2017-03-14
Last modified	2017-10-21
Version	1.3

Contents

1. About

This manual covers the LINUX@CHIP Yocto board support package. It currently supports the following hardware:

- Beck IPC SC145 module / DB150 Evaluation Board
- Beck IPC SC165 module / DB150 Evaluation Board
- Beck IPC FB150 display

If you are experienced developer and have a Linux workstation with an OpenEmbedded or Yocto build system already, just check out the brief guide in chapter 2, otherwise continue with chapter 3. If you want to make use of the ready to run virtual machine continue with chapter 4.

2. Quick start

A short introduction how to get started with the Yocto SDK building for a plain SC145. The target audience are experienced developers which are familiar with Yocto.

2.1 Using the BSP

Download the latest BSP script from here: https://raw.githubusercontent.com/beck-ipc/linux-at-chip/master/scripts/init-bsp.sh

Make sure the script is executable an run it like this:

```
./init-bsp.sh
```

It will prepare a directory *yocto-bsp* checking out all components using Git. If you intend to rename the directory do it now.

Change to the directory and source the environment script:

```
cd yocto-bsp
. oe-init-build-env build-sc145
```

(Please note the leading dot and space.)

Now run bitbake in order to build software for your target device:

```
bitbake at-chip-console-image
```

Running bitbake an internet connection is required in order to fetch the required sources. Once downloaded the sources are usually stored locally to avoid duplicate downloads.

2.2 Using the toolchain

Open a terminal and source the environment script:

```
. /opt/toolchain/environment-setup-cortexa7hf-neon-poky-linux-gnueabi
```

(Please note the leading dot and space.)

The script appends PATH and sets some additional environment variables that enables you to use the cross compiler in this terminal easily.

The prefix of the cross compiler is *arm-poky-linux-gnueabi.*

For example a software tree using autotools can be configured for crosscompiling like this:

```
./configure –host=arm-poky-linux-gnueabi
```

3. Installation and Set-Up

3.1 Requirements

We recommend a powerful Linux X86/64 Bit workstation (>4 cores, >3GHz, 16GB of RAM, 80GB of free disk space) with a Linux distribution such as Debian, Ubuntu or Mint installed.

For using the Yocto / Openembedded BSP components it is required to install some additional software packages. Using Debian/Ubuntu you can install these like this:

```
sudo apt-get install build-essential diffstat docbook-utils gawk git-core
texinfo chrpath gcc-multilib python-pysqlite2 help2man make wget sed
socat libsdl1.2-dev mtd-utils texi2html unzip xterm lzop curl u-boot-
tools
```

3.2 The Yocto BSP

Installation

The recommended way to set up the BSP initially is to use the initialisation script you can obtain from here:

https://raw.githubusercontent.com/beck-ipc/linux-at-chip/master/scripts/init-bsp.sh

Run the script in your desired location. The path to the location must not contain any space and you need to have permissions to execute files in it. The file system needs to provide enough of free

disk space to hold the BSP. The script does not take any arguments and creates a directory *yocto-bsp*. Its okay to rename it, but once you have started building, you have to rebuild everything whenever you move or rename this directory.

This script checks out all required Git repositories and sets up the BSP directory.
Alternatively you can clone the whole Git repository with all utility scripts like this:

```
git clone https://github.com/beck-ipc/linux-at-chip.git
```

Usage

The Yocto BSP can be used to build all software for the target device automatically and assemble it to installable packages as well as flashable file system images and cross development toolchains. It enables flexible and and easy to use adaption to new requirements (such as changing target devices and new libraries) and makes it easy to maintain individual filesystems for multiple needs.

The easiest way to build a complete set of binary files is to run the build script from the BSPs scripts directory. It is required to run after *init-bsp.sh* from the same directory (in which the subdir *yocto-bsp* is located).

```
./build-bsp.sh
```

Depending on the performance of your PC this process takes quite a while running the first time. Expect a duration between 15 minutes and several hours.

The resulting files get deployed in the subdirectory *yocto-bsp/yocto-bsp/build-sc145/tmp/deploy/*.

Interactive use

In order to build single targets, you can use Yocto interactively in a terminal window. For this purpose some environment variables need to be set. This is done by sourcing an environment script.

```
cd yocto-bsp
. oe-init-build-env build-sc145
```

(Please note the leading dot and space. The leading „. “ is important to make sure the settings take place in the current shell.)

The default target device is the SC145, for additional target devices see below.

This command build the example target filesystem image and all of its dependencies:

```
bitbake at-chip-console-image
```

Building of kernel images and modules:

```
bitbake virtual/kernel
```

Apart from these Targets there are several useful target images defined that can be built. For example:

- core-image-sato: A small image with X11 and a Gtk+ user interface.
- core-image-x11: A very basic X11 image. A good starting point for embedded applications with X11.
- core-image-weston: Image with basic Wayland UI server.
- core-image-lsb: Image that comes with all requirements for the Linux Standard Base.

Build of a cross toolchain matching the target systems file system contents:

```
bitbake at-chip-console-image -c populate_sdk
```

Building for other target devices

The default target device is the SC145 board. In the case you have a different target device or intend to create your own one you need to change the MACHINE variable in conf/local.conf in your build diretory. The Yocto BSP supports the following targets:

Yocto machine	Device
sc145-db150	Beck IPC SC145 module / DB150 Evaluation Board
sc165-db150	Beck IPC SC165 module / DB150 Evaluation Board

Yocto supports sharing a single build directory over multiple target devices. All target device specific artifacts will be placed in target specific output directories/files under a shared tmp directory.

3.3 The cross toolchain

The cross compiling toolchain includes a script that sets some very useful environment settings for cross compiling. The toolchains are provided as an executable self-extracting script. They are relocatable and query for a desired installation location.

We assume an installation path of */opt/toolchain* like used for the pre-installed toolchain in the VM.

Sourcing of the environment script:

```
. /opt/toolchain/environment-setup-cortexa7hf-neon-poky-linux-gnueabi
```

(Note the leading dot which tells the shell not to open a subshell for running the script.)

The script appends PATH and sets some additional environment variabes that enables you to use the cross compiler in this terminal easily.

The prefix of the cross compiler is *arm-poky-linux-gnueabi.*

For example a software tree using autotools can be configured for crosscompiling like this:

```
./configure –host=arm-poky-linux-gnueabi –prefix=/usr
```

3.4 Deploying files to the target

The target device comes with a an installed U-Boot bootloader in the internal flash. It can be controlled interactively using a terminal application on the USB debug port connected to a PC. In order to enter the interactive U-Boot console you need to interrupt the boot process by pressing Ctrl+C in the terminal window after reset.

Caution: The bootloader console has complete control over the boot process and offers unlimited access to the flash. It is possible to destroy all data on the device or render the device unbootable.

The Yocto BSP creates UBI filesystem images containing a root file system as well as Linux kernel and Devicetree file. The default U-Boot environment contains all required settings to boot from this UBI filesystem.

For some common tasks there are pre-defined commands available. For example you can replace the filesystem with a new one using DHCP/TFTP with this command:

```
run update_ubi
```

This command retrieves an IP address through DHCP and installs a file rootfs-sc145.ubifs from the TFTP server to the flash.

Writing files to the flash can be done manually as well if you need more control over the flash process. It works like this:

1. Initialise the QSPI flash access

```
sf probe
```

2. Erase the flash – in this example we erase the whole flash not occupied by bootloader and environment. Do not touch the flash from 0x00000000 to 0x00090000 unless you really know what are you doing. This section is usually occupied by bootloader and environment.

```
sf erase 0x00090000 0x03f70000
```

3. Create UBI container and partition

```
ubi part rootfs
ubi create rootfs
```

4. Get an IP address and transfer a filesystem from the TFTP server (In this case 192.168.1.10). In this example we use the Yocto BSP example filesystem image.

```
dhcp 192.168.1.10:at-chip-console-image-sc145-db150.ubifs
```

5. Write the file to the UBI partition:

```
ubi write ${fileaddr} rootfs ${filesize}
```

There are two useful commands to restore defaults in U-Boot:

To restore the default partition layout:

```
mtdparts default
```

To restore the default U-Boot environment use this command:

```
env default -f
```

Use the command *saveenv* to write any change to the environment to the flash.

The Virtual Machine comes with an easy to use tool to restore the initial state of the bards and install a default set of software to it. It is completely independent from the installed software.

4. Using the Virtual Machine

The virtual machine provides a ready to run development environment for the target device. It contains a Linux installation with IDEs, cross compile toolchain and Yocto BSP. For compatibility reasons the VM hardware resources are quite limited. For the Yocto BSP this reduces performance quite a bit.

The virtual machine is provided as an OVF (*.ova, Open Virtualisation Format v. 2.0) which can be imported by common virtualisation software such an VMware or VirtualBox. We recommend to use VirtualBox since the integration support for it is installed into the virtual machine already.

The Virtual Machine needs an Internet connection to download the required sources. In case you have a network using DHCP and have set up networking in your VM software the appliance will retrieve network settings via DHCP automatically. You can set up networking manually with the network applet in the top right corner of the VM desktop. A network proxy can be set up in the menu: System -> Preferences -> Network Proxy. The required tools for using a proxy with Yocto are installed already so that a proxy configuration becomes active for any new terminal window once it has been set up in the proxy settings GUI.

The ready to run cross development toolchain is installed to */opt/toolchain* while the pre-installed Yocto BSP is located in */home/user/yocto-bsp.*

Both can be used like described in the sections above.

4.1 Installing imx_usb

If you are using the Beck IPC virtual machine you can skip this step since the tool is available already.

1. Install required development packges

You need to install libusb 1.0 including development files. In Debian and similar distribution these are the packages *libusb-1.0-0* and *libusb-dev*.

2. Use git to get the latest sources

```
git clone https://github.com/boundarydevices/imx_usb_loader.git
```

3. Build and run the tool

```
$ cd imx_usb_loader
$ make
$ ./imx_usb /path/to/your/images/at-chip<someimagetype>.imx
```

4. 2 Installing flash images through USB

The VM includes a set of easy to install images that contain a complete default software stack including bootloader, kernel and filesystem. These can be used for restoring bootloader and a basic Linux filesystem on the boards. They can be easily installed using the *imx_usb*[1] tool. The tool is available in the VM – on other Linux systems it is required to install it.

There is no dependency into anything on the board. It installs regardless from the current flash contents.

Note: All data in the board will be overwritten!

The flashing procedure is pretty simple:

1. Set jumper at position BM0 on the DB150 baseboard.

2. Connect a USB cable to the USB client/OTG port and make sure device is accessible from the VM.

3. Power-cycle the board.

3. Run the install command: sudo imx_usb <image>.imx

1 Available from here: https://github.com/boundarydevices/imx_usb_loader

You need to select an image matching your hardware (sc145 or sc165).

The installation takes a while and has not finished when the command rerturns! You can follow the progress usin the USB serial debug console. Once the installation process has finished you see such a message on the debug console:

```
SF: 67108864 bytes @ 0x0 Written: OK
=>
```

Now remove BM0 and reset the board.

5. Additional Information

Yocto Project Quick Start:

http://www.yoctoproject.org/docs/1.8/yocto-project-qs/yocto-project-qs.html

Yocto Reference Manual:

http://www.yoctoproject.org/docs/latest/ref-manual/ref-manual.html

If you have questions, ideas or require assistance please drop us a mail to support@beck-ipc.com or visit your helpdesk at:

https://helpdesk.beck-ipc.com