


Application Note	
COMe-cP2020 (S1000, D0481) Com Express Module supporting P1020, P1011, P2010 and P2020 Building BSP Images using the Yocto Layer meta-kbc-ppc	Version 3 2014-09-19 Ralf Kihm

Revision History

Date	Revision	Author	Chapters affected / Remarks
2012-12-21	0.1	Ralf Kihm	Initial Draft
2013-02-27	2	Ralf Kihm	Release for LIN_BSP R20
2014-09-19	3	Ralf Kihm	Disclaimer removed

Table of Content

Revision History.....	1
Introduction.....	3
Additional Resources.....	3
Preparation.....	4
Prerequisites.....	4
Installing Freescale SDK 1.3	4
Basic Setup.....	4
Installing Kontron BSP Layer “meta-kbc-ppc”	4
Build Directory and Sample Configuration Files	4
Adapt the Configuration Files.....	5
Shared State (SSTATE) Mirror Configuration.....	5
Setup the Build Environment.....	5
Building the BSP	6
Building.....	6
Build Results.....	6
Building the Freescale Toolchain	7
Building.....	7
Build Results.....	7
Appendix A: local.conf.sample.....	8
Appendix B: bblayers.conf.sample.....	12

Introduction

The Kontron provided BSP layer “meta-kbc-ppc” is an additional layer that adds support for the Kontron COMe-cP2020 modules on top of the Freescale SDK 1.3. Kontron provides the “meta-kbc-layer” on request. Please contact <support-keu@kontron.com>.

Additional Resources

- Freescale Technical Information Center - <http://www.freescale.com/infocenter/topic/qfamily-sdk/index.html>
- Yocto Project Homepage - <https://www.yoctoproject.org>
- BitBake User Manual - <http://docs.openembedded.org/bitbake/html/>

Preparation

Prerequisites

- Python 2.6 must be available from the default search path

Installing Freescale SDK 1.3

The Freescale SDK 1.3 is not provided by Kontron, but available from Freescale's homepage. Please contact your Freescale FAE for more information.

```
http://www.freescale.com/webapp/sps/site/prod\_summary.jsp?code=SDKLINUX
```

The location of the Freescale SDK 1.3 installation on the local file system is referenced as \${SDKDIR} in the following text.

Basic Setup

The following examples use shell variables that point to the installation directory of the original Freescale SDK and to the directory that holds intermediate and build results. These variables have to be set properly before following the steps in this Application Note.

Shell Variable	Description
SDKBASE	Point to the location of the Freescale SDK installation
BASEDIR	Points to the folder that holds your intermediate and build results

Installing Kontron BSP Layer “meta-kbc-ppc”

Extract the Kontron provided tar archive to a temporary folder this folder is referenced as \${BASEDIR} in the following text.

```
tar xzf meta-kbc-ppc.tgz -C ${BASEDIR}
```

Build Directory and Sample Configuration Files

Kontron provides the sample configuration files “local.conf.sample” and “bblayers.conf.sample” within the meta-kbc-ppc layer directory.

Create your local build directory by copying the sample configuration files into the new build directory.

```
install -D ${BASEDIR}/meta-kbc-ppc/conf/local.conf.sample  
${BASEDIR}/build_d0481_release/conf/local.conf  
install -D ${BASEDIR}/meta-kbc-ppc/conf/bblayers.conf.sample  
${BASEDIR}/build_d0481_release/conf/bblayers.conf
```

Adapt the Configuration Files

The sample configuration files still contain the placeholders “##COREBASE##” and “##KBCBASE##” that have to be replaced with the absolute paths that reflect your build environment.

```
sed -i "s>##COREBASE##>${SDKBASE}>g" ${BASEDIR}/build_d0481_release/conf/local.conf
sed -i "s>##KBCBASE##>${BASEDIR}>g" ${BASEDIR}/build_d0481_release/conf/local.conf
sed -i "s>##COREBASE##>${SDKBASE}>g" ${BASEDIR}/build_d0481_release/conf/bblayers.conf
sed -i "s>##KBCBASE##>${BASEDIR}>g" ${BASEDIR}/build_d0481_release/conf/bblayers.conf
```

Shared State (SSTATE) Mirror Configuration

Optionally you may want to benefit from the Freescale provided prebuilt binaries. In that case append sstate mirror configuration to the end of your “local.conf” file.

```
echo SSTATE_MIRRORS += \"file://.* file://${SDKBASE}/sstate-cache/ \n\ \" >>
${BASEDIR}/build_d0481_release/conf/local.conf
```

Setup the Build Environment

The following command line setups the build environment and changes to the “\${BASEDIR}/build_d0481_release”.

```
source ${SDKBASE}/oe-init-build-env ${BASEDIR}/build_d0481_release
```

Building the BSP

Building

```
bitbake kbc-image
```

Build Results

The build results are available from the directory "\${BASEDIR}/build_d0481_release/deploy".

Building the Freescale Toolchain

Building

```
bitbake fsl-toolchain
```

Build Results

The toolchain is available from "\${BASEDIR}/build_d0481_release/deploy/sdk".

Appendix A: local.conf.sample

```
#
# This file is your local configuration file and is where all local user settings
# are placed. The comments in this file give some guide to the options a new user
# to the system might want to change but pretty much any configuration option can
# be set in this file. More adventurous users can look at local.conf.extended
# which contains other examples of configuration which can be placed in this file
# but new users likely don't need any of them initially.
#
# Lines starting with the '#' character are commented out and in some cases the
# default values are provided as comments to show people example syntax. Enabling
# the option is a question of removing the # character and making any change to the
# variable as required.
#
# Parallelism Options
#
# These two options control how much parallelism BitBake should use. The first
# option determines how many tasks bitbake should run in parallel:
#
#BB_NUMBER_THREADS = "4"
#
# The second option controls how many processes make should run in parallel when
# running compile tasks:
#
# PARALLEL_MAKE = "-j 4"
#
# For a quadcore, BB_NUMBER_THREADS = "4", PARALLEL_MAKE = "-j 4" would
# be appropriate for example.
BB_NUMBER_THREADS = "2"
PARALLEL_MAKE = "-j 2"
#
# Machine Selection
#
# You need to select a specific machine to target the build with. There are a selection
# emulated machines available which can boot and run in the QEMU emulator:
#
#MACHINE ?= "qemuarm"
#MACHINE ?= "qemumips"
#MACHINE ?= "qemuppc"
#MACHINE ?= "qemux86"
#MACHINE ?= "qemux86-64"
#
# There are also the following hardware board target machines included for
# demonstration purposes:
#
#MACHINE ?= "atom-pc"
#MACHINE ?= "beagleboard"
#MACHINE ?= "mpc8315e-rdb"
#MACHINE ?= "routerstationpro"
#
# This sets the default machine to be d0481 if no other machine is selected:
MACHINE = "d0481"
#
# Where to place downloads
#
# During a first build the system will download many differernt source code tarballs
# from various upstream projects. This can take a while, particularly if your network
# connection is slow. These are all stored in DL_DIR. When wiping and rebuilding you
```



```

# can preserve this directory to speed up this part of subsequent builds. This directory
# is safe to share between multiple builds on the same machine too.
#
# The default is a downloads directory under TOPDIR which is the build directory.
#
#DL_DIR ?= "${TOPDIR}/downloads"

#
# Where to place shared-state files
#
# BitBake has the capability to accelerate builds based on previously built output.
# This is done using "shared state" files which can be through of as cache objects
# and this option determines where those files are placed.
#
# You can wipe out TMPDIR leaving this directory intact and the build would regenerate
# from these files if no changes were made to the configuration. If changes were made
# to the configuration, only shared state files where the state was still valid would
# be used (done using checksums).
#
# The default is a sstate-cache directory under TOPDIR.
#
#SSTATE_DIR ?= "${TOPDIR}/sstate-cache"

#
# Where to place the build output
#
# This option specifies where the bulk of the building work should be done and
# where BitBake should place its temporary files and output. Keep in mind that
# this includes the extraction and compilation of many applications and the toolchain
# which can use Gigabytes of hard disk space.
#
# The default is a tmp directory under TOPDIR.
#
#TMPDIR = "${TOPDIR}/tmp"

#
# Default policy config
#
# The distribution setting controls which policy settings are used as defaults.
# The default value is fine for general Yocto project use, at least initially.
# Ultimately when creating custom policy, people will likely end up subclassing
# these defaults.
#
# DISTRO ?= "poky"
# As an example of a subclass there is a "bleeding" egde policy configuration
# where many versions are set to the absolute latest code from the upstream
# source control systems. This is just mentioned here as an example, its not
# useful to most new users.
# DISTRO ?= "poky-bleeding"
DISTRO = "fsl"

#
# Package Management configuration
#
# This variable lists which packaging formats to enable. Multiple package backends
# can be enabled at once and the first item listed in the variable will be used
# to generate the root filesystems.
# Options are:
# - 'package_deb' for debian style deb files
# - 'package_ipk' for ipk files are used by opkg (a debian style embedded package manager)
# - 'package_rpm' for rpm style packages
# E.g.: PACKAGE_CLASSES ?= "package_rpm package_deb package_ipk"

```

```

# We default to rpm:
PACKAGE_CLASSES ?= "package_rpm"

#
# SDK/ADT target architecture
#
# This variable specified the architecture to build SDK/ADT items for and means
# you can build the SDK packages for architectures other than the machine you are
# running the build on (i.e. building i686 packages on an x86_64 host.)
# Supported values are i686 and x86_64
#SDKMACHINE ?= "i686"

#
# Extra image configuration defaults
#
# The EXTRA_IMAGE_FEATURES variable allows extra packages to be added to the generated
# images. Some of these options are added to certain image types automatically. The
# variable can contain the following options:
# "dbg-pkgs"      - add -dbg packages for all installed packages
#                  (adds symbol information for debugging/profiling)
# "dev-pkgs"      - add -dev packages for all installed packages
#                  (useful if you want to develop against libs in the image)
# "tools-sdk"     - add development tools (gcc, make, pkgconfig etc.)
# "tools-debug"   - add debugging tools (gdb, strace)
# "tools-profile" - add profiling tools (oprofile, exmap, lttnng valgrind (x86 only))
# "tools-testapps" - add useful testing tools (ts_print, aplay, arecord etc.)
# "debug-tweaks"  - make an image for suitable of development
#                  e.g. ssh root access has a blank password
# There are other application targets that can be uses here too, see
# meta/classes/core-image.bbclass and meta/recipes-core/tasks/task-core.bb for more details.
# We default to enabling the debugging tweaks.
EXTRA_IMAGE_FEATURES = "debug-tweaks"

#
# Additional image features
#
# The following is a list of additional classes to use when building images which
# enable extra features. Some available options which can be included in this variable
# are:
# - 'image-mklibs' to reduce shared library files size for an image
# - 'image-prelink' in order to prelink the filesystem image
# - 'image-swab' to perform host system intrusion detection
# NOTE: if listing mklibs & prelink both, then make sure mklibs is before prelink
# NOTE: mklibs also needs to be explicitly enabled for a given image, see local.conf.extended
USER_CLASSES ?= "image-mklibs image-prelink"

#
# Runtime testing of images
#
# The build system can test booting virtual machine images under qemu (an emulator)
# after any root filesystems are created and run tests against those images. To
# enable this uncomment this line
#IMAGETEST = "qemu"
#
# This variable controls which tests are run against virtual images if enabled
# above. The following would enable bat, oot test case under sanity suite and
# toolchain tests
#TEST_SCEN = "sanity bat sanity:boot toolchain"
#
# Because of the QEMU booting slowness issue(see bug #646 and #618), autobuilder
# may suffer a timeout issue when running sanity test. We introduce variable
# TEST_SERIALIZE here to reduce the time on sanity test. It is by default set

```

```

# to 1. This will start image and run cases in the same image without reboot
# or kill. If it is set to 0, the image will be copied and tested for each
# case, which will take longer but be more precise.
#TEST_SERIALIZE = "1"

#
# Interactive shell configuration
#
# Under certain circumstances the system may need input from you and to do this it
# can launch an interactive shell. It needs to do this since the build is
# multithreaded and needs to be able to handle the case where more than one parallel
# process may require the user's attention. The default is to use xterm.
#
# Examples of the occasions this may happen are when resolving patches which cannot
# be applied, to use the devshell or the kernel menuconfig
#
# If you do not use (or have installed) xterm you will need to
# uncomment these variables and set them to the terminal you wish to use
#
# Supported shell prefixes for *_TERMCMD and *_TERMCMDRUN are:
# GNOME, SCREEN, XTERM and KONSOLE
# Note: currently, Konsole support only works for KDE 3.x due to the way
# newer Konsole versions behave
#TERMCMD = "${XTERM_TERMCMD}"
#TERMCMDRUN = "${XTERM_TERMCMDRUN}"
# By default disable interactive patch resolution (tasks will just fail instead):
PATCHRESOLVE = "noop"

#
# Shared-state files from other locations
#
# As mentioned above, shared state files are prebuilt cache data objects which can
# used to accelerate build time. This variable can be used to configure the system
# to search other mirror locations for these objects before it builds the data itself.
#
# This can be a filesystem directory, or a remote url such as http or ftp. These
# would contain the sstate-cache results from previous builds (possibly from other
# machines). This variable works like fetcher MIRRORS/PREMIRRORS and points to the
# cache locations to check for the shared objects.
#SSTATE_MIRRORS ?= "\
#file://.* http://someserver.tld/share/sstate/ \n \
#file://.* file:///some/local/dir/sstate/ \n \
#file://.* file:///##COREBASE##/sstate-cache/"

# use xz instead of gzip for sstate-cache
SSTATE_PKG_SUFFIX ?= "txz"
SSTATE_PKG_TARZIPPROG ?= "xz"

#
# Archiving source code configuration
#
# The following variables control which files to archive and the type to archive to generate.
# There are three basic class definitions of common operations that might be desired and these
# can be enabled by uncommenting one of the following lines:
#
# INHERIT += "archive-patched-source"
# INHERIT += "archive-configured-source"
# INHERIT += "archive-original-source"
#
# Type of archive:
# SOURCE_ARCHIVE_PACKAGE_TYPE = 'srpm'
SOURCE_ARCHIVE_PACKAGE_TYPE ?= 'tar'

```

```

#
# Whether to include WORKDIR/temp, .bb and .inc files:
# 'logs_with_scripts' include WORKDIR/temp directory and .bb and .inc files
# 'logs' only include WORKDIR/temp
# SOURCE_ARCHIVE_LOG_WITH_SCRIPTS ?= 'logs'
# SOURCE_ARCHIVE_LOG_WITH_SCRIPTS ?= 'logs_with_scripts'

# CONF_VERSION is increased each time build/conf/ changes incompatibly and is used to
# track the version of this file when it was generated. This can safely be ignored if
# this doesn't mean anything to you.
CONF_VERSION = "1"

# use sources supplied by Freescale SDK
PREMIRRORS_prepend = "\
cvs://.* file:///##COREBASE##/sources/ \n \
svn://.* file:///##COREBASE##/sources/ \n \
git://.* file:///##COREBASE##/sources/ \n \
ftp://.* file:///##COREBASE##/sources/ \n \
http://.* file:///##COREBASE##/sources/ \n \
https://.* file:///##COREBASE##/sources/ \n \
file://.* file:///##KBCBASE##/meta-kbc-ppc/sources/ \n"

# delete sources after build to save disk space
INHERIT += "rm_work"

# set meta-oe layer with lowest priority
BBFILE_PRIORITY_openembedded-layer = "1"

```

Appendix B: bblayers.conf.sample

```

# LAYER_CONF_VERSION is increased each time build/conf/bblayers.conf
# changes incompatibly
LCONF_VERSION = "4"

BBFILES ?= ""
BBLAYERS = " \
    ##COREBASE##/meta \
    ##COREBASE##/meta-yocto \
    ##COREBASE##/meta-fsl-ppc \
    ##COREBASE##/meta-fsl-ppc-private \
    ##COREBASE##/meta-oe/meta-oe \
    ##KBCBASE##/meta-kbc-ppc \
    "

```