NBUILD User Guide

ABSTRACT:
This is the NBUILD user guide document applicable for AMCU_SW_S32V234

KEYWORDS:
Makefile, AFC, APU, ISP, ARM

APPROVAL TABLE

AUTHOR	SIGN-OFF SIGNATURE #1 APPROVAL DATE	SIGN-OFF SIGNATURE #2 & APPROVAL DATE
Mihail Nistor		

REVISION HISTORY

VERSION	DATE	AUTHOR	COMMENT
0.1	2017-03-16	Mihail Nistor	First draft
1.0	2017-08-27	Mihail Nistor	Updated for VSDK 1.2 RTM release

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Introduction

NBUILD is a collection of make-packages that provide dedicated functionalities and can be included by Makefiles to build the Vision SDK.

1.1 Scope

The scope of this document is to describe the NBUILD user interface as well as the important design details of the Vision SDK build system. It is intended to serve as a reference source for future NBUILD development and maintenance.

1.2 Audience Description

The audience for this document consists of Vison SDK developers familiar with GNU-MAKE, APU-2 C, ACF in scope of [1],[2],[3]

1.3 References

The documents referred across the plan can be found in the following table:

NAME	VERSION	LOCATION
[1] GNU-MAKE	4.2	HTTPS://WWW.GNU.ORG/SOFTWARE/MAKE/MANUAL/MAKE.HTML
[2] APU-2 C PROGRAMMING GUIDE	UG-10301-00-06	S32V234 SDK
[3] AFC USER GUIDE	UG-10267-03-11	S32V234 SDK
[4] RELEASE CONFIGURATION		S32V234 SDK

Table 1 References Table

1.4 Definitions, Acronyms, and Abbreviations

TERM/ACRONYM	DEFINITION
MAKE	It is a toolchain that automatically determines which pieces of a large program need to be recompiled, and issues commands to recompile them.
APU-2	Array Processor Unit
ACF	APEX Core Framework
S32V234	S32V234 SoC
SDK	System Development Kit
ARM	Family of RISC architectures
ISP	Image Signal Processor subsystem of the S32V234 SoC.

IPU	Image processing unit, as there are IPUS (scalar) and IPUV (vector)

Table 2 Acronyms Table

1.5 Document Location

The document is located <VISION_SDK >*/s32v234_sdk/docs/nbuild/*

1.6 Problem Reporting Instructions

Problems or corrections to this document should be reported to the following email address: Mihail.Nistor@nxp.com

2 Functional description

NBUILD is a collection of make-packages that provide dedicated functionalities. A sub-set of make-packages can be included by Makefiles from project as needed. We call them entry point for platform definition. Most packages require some parameters that must be passed in variables. The variables are set in the BUILD.mk at the project level.

The Project template that includes an APEX graphs (the sub-folder graphs contains the specific APEX graphs structure), each Makefile from Project should include the corresponding entry platform package. The coordinator.mk entry platform package coordinates the build process for build-apu-[nxp|tct]-sa-d by using the build-v234ce-gnu-[linux|sa]-[d|o] platform. A BUILD.mk can be shared by more than one platform (The first BUILD.mk at project root is shared by build-v234ce-gnu-linux-[d|o] and build-v234ce-gnu-sa-[d|o] platform; the second BUILD.mk, in graphs folder, is shared by build-apu-nxp-sa-d and build-apu-tct-sa-d platforms.)

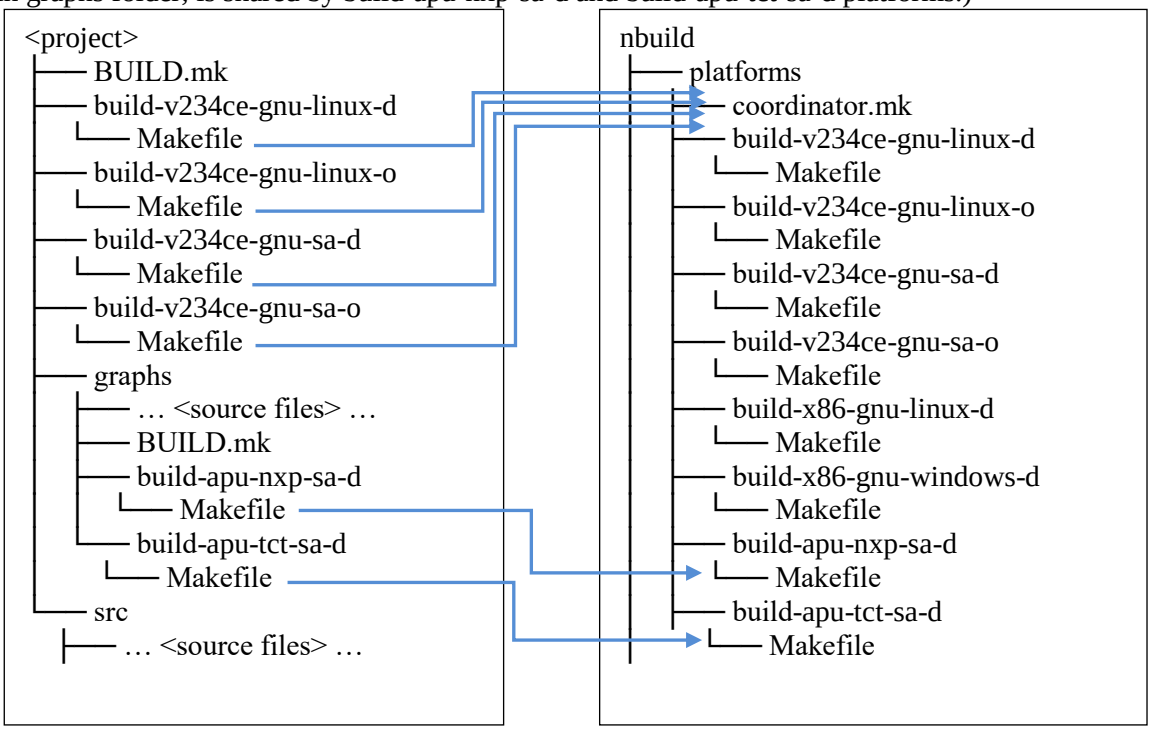

Figure 1 Project template + NBUILD entry platforms

The NBUILD Makefile system provides support to build program for the following architectures:

- Cortex-A53 – AArch64 target
- Cortex-M0 – M0+ target
- IPUx – ISPUS/ISPUV targets
- APU2 – apu target
- X86 – development PC

The build-[target]-[compiler]-[OS type]-[configuration type] pattern is used to define a platform.

The supported target list:

- v234ce for Treerunner

- apu for APU2
- X86 – development PC

The supported compiler list:
- tct for APU2 (Synopsys vendor)
- nxp for APU2 (NXP vendor)
- gnu for AArch64/M0 (Linaro vendor)
- gnu for X86 (GNU vendor)

The supported OS type list:
- linux for Linux
- sa for standalone - OSLess

The supported configuration type list:
- d for **d**ebug configuration
- o for **o**ptimized (release) configuration
- d for **d**efault configuration; this is used in case the platform supports only one configuration.

Variables for AArch64 target that can be used in BUILD.mk:
- ARM_APP – name of the application to generate
- ARM_APP_SRCS – source files of the application
- ARM_APP_LIBS – full paths of libraries linked to the application
- ARM_LIB – name of the library to generate
- ARM_LIB_SRCS – source files of the library
- ARM_LIB_LIBS – full paths of libraries that are dependencies for library or application. The developer can set the partial or full build order for all dependencies. This list will be added in the build order list (SUB_MAKES variable). The build order list will be used for recursive make.
- ARM_DEFS – additional precompile defines
- ARM_INCS – additional include paths
- ARM_CCOPTS – additional compiler options C
- ARM_CXOPTS – additional compiler options C++
- ARM_LDOPTS – additional linker options

Variables for APU target that can be used in BUILD.mk:
- APU_APP – name of the application to generate
- APU_APP_SRCS – source files of the application
- APU_APP_LIBS – full paths of libraries linked to the application
- APU_LIB – name of the kernel library to generate
- APU_LIB_SRCS – source files of the kernel library
- APU_LIB_LIBS – full paths of libraries that are dependencies for library or application. The developer can set the partial or full build order for all dependencies. This list will be added in the build order list (SUB_MAKES variable). The build order list will be used for recursive make.
- APU_DEFS – additional precompile defines
- APU_INCS – additional include paths
- APU_CCOPTS – additional compiler options C and C++
- APU_LDOPTS – additional linker options
- APU_GRAPH_LIBS – full paths of kernel libraries linked to the APU graph application
- APU_GRAPH_INCS – additional include paths
- APU_GRAPHS – list of process description header files (e.g. <file_id>_apu_process.hpp)

Variables for M0+ target that can be used in BUILD.mk:
- SEQ_APP – name of the application to generate

- SEQ_APP_SRCS – source files of the application
- SEQ_APP_LIBS – full paths of libraries linked to the application
- SEQ_LIB – name of the kernel library to generate
- SEQ_LIB_SRCS – source files of the kernel library
- SEQ_LIB_LIBS – full paths of libraries that are dependencies for library or application. The developer can set the partial or full build order for all dependencies. This list will be added in the build order list (SUB_MAKES variable). The build order list will be used for recursive make.
- SEQ_DEFS – additional precompile defines
- SEQ_INCS – additional include paths
- SEQ_CCOPTS – additional compiler options C
- SEQ_LDOPTS – additional linker options
- SEQ_LDSCRIPT - linker script

Variables for ISPUS target that can be used in BUILD.mk:

- IPUS_APP – name of the application to generate
- IPUS_APP_SRCS – source files of the application
- IPUS_DEFS – additional precompile defines
- IPUS_INCS – additional include paths

Variables for ISPUV target that can be used in BUILD.mk:

- IPUV_APP – name of the application to generate
- IPUV_APP_SRCS – source files of the application
- IPUV_DEFS – additional precompile defines
- IPUV_INCS – additional include paths

Variables for X86 target that can be used in BUILD.mk:

- X86_APP – name of the application to generate
- X86_APP_SRCS – source files of the application
- X86_APP_LIBS – full paths of libraries linked to the application
- X86_LIB – name of the kernel library to generate
- X86_LIB_SRCS – source files of the kernel library
- X86_LIB_LIBS – full paths of libraries that are dependencies for library or application. The developer can set the partial or full build order for all dependencies. This list will be added in the build order list (SUB_MAKES variable). The build order list will be used for recursive make.
- X86_DEFS – additional precompile defines
- X86_INCS – additional include paths
- X86_CCOPTS – additional compiler options C
- X86_CXOPTS – additional compiler options C++

The NBUILD support to build more than one targets in the same directory (platform folder name). The AArch64, IPUS, IPUV, M0+ targets may share the build-v234ce-[compiler]-[OS type]-[configuration type] folder name pattern; this means the NBUILD can build more targets inside a BUILD.mk.

Find below an example of BUILD.mk from s32v234_sdk/isp/graphs/isp_generic/dynamic_isp_generic project.

```
##############################################################################
SDK_ROOT := $(call path_relative_to,$(CURDIR),$(CURR_SDK_ROOT))
SEQ_MODE = dynamic
include ../../graph.mk
```


Find below an example of graph.mk from above project.

```
################################################################################
...
GRAPH_NAME = isp_generic

################################################################################
# Sequencer Firmware
################################################################################
SEQ_APP =sequencer

VPATH +=$(SDK_ROOT)/isp/graphs/$(GRAPH_NAME)/src

SEQ_APP_SRCS =$(SEQ_MODE)_$(GRAPH_NAME).c

SEQ_DEFS +=
SEQ_INCS +=
SEQ_CCOPTS +=

SEQ_APP_LIBS = $(SDK_ROOT)/isp/firmware/$(ODIR)/libsequencer.a

...

################################################################################
# ISP Graph
################################################################################
ARM_LIB = lib$(SEQ_MODE)_$(GRAPH_NAME).a

ARM_LIB_SRCS = \
          $(GRAPH_NAME).c \
          sequencer_srec.c

ARM_INCS = \
          -I$(SDK_ROOT)/include                                                 \
          -I$(SDK_ROOT)/isp/graphs/$(GRAPH_NAME)/inc                            \
          -I$(SDK_ROOT)/isp/inc                                                 \
          -I$(SDK_ROOT)/platform/s32_v234
...

################################################################################
# IPUx kernels
################################################################################
IPUS_APP = ipus.elf
IPUS_APP_SRCS = \
  $(SDK_ROOT)/isp/kernels/generic/src/all_in_one_isp.ipus   \
  $(SDK_ROOT)/isp/kernels/generic/src/copy_1to1_ipus.ipus \
  $(SDK_ROOT)/isp/kernels/generic/src/fir3x3.ipus \
  $(SDK_ROOT)/isp/kernels/generic/src/gauss3x3.ipus \
...

IPUV_APP = ipuv.elf
IPUV_APP_SRCS = \
  $(SDK_ROOT)/isp/kernels/generic/src/fir5x5.ipuv \
  $(SDK_ROOT)/isp/kernels/generic/src/rgb2yuv_planar.ipuv \
  $(SDK_ROOT)/isp/kernels/generic/src/yuvplanar2bgrinterleaved.ipuv \
...
```

The apu and X86 targets may share the build-apu-[nxp|tct]-sa-d folder name pattern.
Find below an example of BUILD.mk from s32v234_sdk/kernels/apu/apexcv_base_arithmetic project.

```
################################################################################

SDK_ROOT := ../../../..

APU_LIB = apexcv_base_arithmetic.a

################################################################################
# APU Kernel library built (code for APU - kernel implementation)
################################################################################

APU_INCS +=                                                                     \
    -I./                                                                        \
    -I$(SDK_ROOT)/include

APU_LIB_SRCS +=                                                                 \
    arithmetic_acf.cpp                                                          \
    arithmetic_apu.cpp


APU_DEFS += -DACF_KERNEL_IMPLEMENTATION
```

```
ifeq (,$(findstring -nxp,$(ODIR)))
  APU_CCOPTS += +W noodle,-Wnone
  APU_CXOPTS += +W noodle,-Wnone
endif

X86_INCS += -I$(SDK_ROOT)/include

################################################################################
# Kernel db generator - builds an exe application and generates .h header with
#                       kernel description
# Do not change following code
################################################################################
X86_LIB    =  $(addsuffix acf.a,$(basename $(APU_LIB)))
X86_LIB_SRCS =  $(APU_LIB_SRCS)

X86_KERNEL_DB = "on"
################################################################################
```

2.1 Directory structure

nbuild [dir]

- platform.mk – global variables definition; these variables can be used by user in BUILD.mk
- gmsl.mk – functions from GNU Make Standard Library; these can be used in BUILD.mk
- utilities.mk – utilities functions that can be used in BUILD.mk
- helpers.mk – pattern rules to generate object and dependency header files and other primitives; these cannot be used in BUILD.mk
- sub.mk – provide recursive makes in project; the build order is given by SUB_MAKES.
- .C0.mk – template for AARM64 target by using GNU compiler type
- .C1.mk – template for APU2 target by using Synopsys/NXP compiler
- .C2.mk – template for X86 target by using gnu native compiler
- .C4.mk – template for IPUS target by using gnu assembly type
- .C5.mk – template for IPUV target by using gnu assembly type
- .C6.mk – template for M0+ target (ARM32) by using GNU compiler type
- apex_tools [dir] – specific tool for APEX
    - apu-kernel-database.mk – generates kernel database header for APEX library
    - apu-kernel-getgraphs.mk – AFC offline resolution for APEX graphs
- isp_tools [dir] – specific tool for ISP
    - isp-graphgen.mk – generates ISP graph; puts IPUS/IPSV kernels into C array from srecord file that will be compiled by using the AArch64 target. The result is a library for AArch64 target.
- scripts [dir] – scripts files
    - parser to extract INSTANCE_ID and CLASS_ID for ACF offline resolution.
- toolchains [dir] – it links the architecture, the OS type and the compiler
    - AArch64 target on Linux OS by using the GNU Linaro compiler:
        - aarch64-linux-gnu-debug.mk – the debug configuration

        - aarch64-linux-gnu-optimized.mk – the optimized configuration
        - aarch64-linux-gnu.mk – the common part
    - AArch64 target on OSLess (standalone) by using the GNU Linaro compiler
        - aarch64-sa-gnu-debug.mk – the debug configuration
        - aarch64-sa-gnu-optimized.mk – the optimized configuration
        - aarch64-sa-gnu.mk – the common part
    - APU2 target on OSLess (standalone) by using the NXP compiler:
        - apu-acf-sa-nxp.mk – the optimized configuration
    - APU2 target on OSLess (standalone) by using the Synopsys compiler:
        - apu-acf-sa-tct.mk – the optimized configuration
    - M0+ target on OSLess (standalone) by using the GNU Linaro compiler:
        - arm-sa-gnu-m0.mk the default configuration, the debug or optimized options are set in BUILD.mk at application level.
    - IPUS target on OSLess (standalone) by using the GNU Freescale assembler:
        - ipus-sa-gnu.mk – the default configuration.
    - IPUV target on OSLess (standalone) by using the GNU Freescale assembler:
        - ipuv-sa-gnu.mk – the default configuration.
    - X86 target on Linux desktop by using the GNU compiler
        - x86-linux-gnu.mk – the default configuration.
    - X86 target on Windows desktop by using the GNU compiler
        - x86-windows-gnu.mk – the default configuration.
- platforms [dir] – build-[target]-[compiler]-[OS type]-[configuration type] pattern to define a platform
    - build-apu-nxp-sa-d [dir] – APU2 standalone application or library by using NXP compiler
        - Makefile – the entry point for build-apu-nxp-sa-d platform, this Makefile should be included in the Makefile at the project level.
    - build-apu-tct-sa-d [dir] – APU2 standalone application/library by using Synopsys compiler
        - Makefile – the entry point for build-tct-nxp-sa-d platform, this Makefile should be included in the Makefile at the project level.
    - apu.mk - the common part for build-apu-tct-sa-d and build-apu-tct-sa-d.
    - build-v234ce-gnu-linux-d [dir] – v234ce Linux OS application or library for debug configuration
        - Makefile the entry point for build-v234ce-gnu-linux-d platform, this Makefile should be included in the Makefile at the project level.
    - build-v234ce-gnu-linux-o [dir] – v234ce Linux OS application or library for optimized configuration

        - Makefile the entry point for build-v234ce-gnu-linux-o platform, this Makefile should be included in the Makefile at the project level.
    - build-v234ce-gnu-sa-d [dir] – v234ce OSLess application or library for debug configuration
        - Makefile the entry point for build-v234ce-gnu-sa-d platform, this Makefile should be included in the Makefile at the project level.
    - build-v234ce-gnu-sa-o [dir] – v234ce OSLess application or library for optimized configuration
        - Makefile the entry point for build-v234ce-gnu-sa-o platform, this Makefile should be included in the Makefile at the project level.
    - build-v234ce.mk – the common part that defines the build-v234ce-[compile]-[OS type]-[configuration type] platforms
    - isp.mk – the ISP firmware and graph, IPU used by build-v234ce.mk.
    - coordinator.mk – coordinates the build process for build-apu-[nxp|tct]-sa-d by using the build-v234ce-[compiler]-[OS type]-[configuration type] platforms; this should be included in the Makefile for build-v234ce-[compiler]-[OS type]-[configuration type] at the project level when the project has APU2 graphs as direct dependencies (***graphs*** is a sub-folder of project folder).

3 In a nutshell

The most important make targets are:

- make allsub – recursively build all libraries then build all in the current directory
- make cleansub - clean libraries and the current directory
- make all – build all in current directory; the dependency libraries are not compiled; it reports an error if a dependency library does not exit.
- make clean - clean current directory

3.1 Feature overview

3.1.1 Verbose mode

The verbose mode can be enabled by setting the V variable to 1.

Examples:

- make V=1 allsub – be verbose when all dependencies and current directories are built
- make V=1 cleansub – be verbose when all dependencies and current directories are cleaned

The verbose mode is disabled by default.

3.1.2 Parallel compilation

The NBUILD supports parallel compilation and the recommended options to enable it are below:

- for Windows desktop: -j 2 * number of CPU
- for Linux desktop: -j -l 2 * number of CPU

Example: Let's assume that the user PC has 8 CPUs.

- for Windows desktop:
  - make -j 16 allsub
- for Linux desktop:
  - make -j -l 16 allsub

3.1.3 Incremental build

The NBUILD has header dependency support for all compilers supported by Vision SDK. The user can disable the option "generate header dependency" by setting the HEADDEP to 0.

Also, to support incremental build, the Makefile and ../BUILD.mk are added as dependencies when the intermediate file (i.e. object file) is generated.

3.1.4 Multiple build directories

Most often, not all source files for an application are located in the current directory. Two possible cases can be distinguished:

3.1.4.1 Outlying sources

If sources from another directory should be built as part of the current application, they can be referenced by their full relative path name.

ARM_APP_SRCS = main.cpp ../otherdir/otherfile.cpp ...

In this case, the object file is generated in ../otherdir. This can be a problem, if the file is built as

part of other applications too, that might have used different options for compiling. In this case, it would be desirable to have the objects separated.

The VPATH variable must be used to put the object files in the current build directory.

VPATH += ../otherdir

ARM_APP_SRCS = main.c otherfile.c ...

3.1.4.2 Recursive builds

If there is a Makefile in the other directory, a build must be performed there before the local build. This is called a sub-build in the following. A typical use case of such sub-builds are libraries, that must be created or updated before they can be linked to the local application. But, also other use cases exist, e.g. the generation of a source file of one application from binary data of another application.

All cases of sub-builds (and recursively the sub-builds of sub-builds) are uniformly handled by the package sub.mk

3.1.4.2.1 Build Order for recursive make

The build order can be explicitly set by using the <XXX>_LIB_LIBS variable in BUILD.mk where XXX is one of them: ARM, APU, SEQ and X86.

The link order of library is explicitly set by using the <XXX>_ APP_LIBS variable in BUILD.mk where XXX is one of them: ARM, APU, SEQ and X86.

The link order of library will be used for recursive make if build order is not explicitly set.

The build order list (MAKE_SUB) is created by concatenating the <XXX>_LIB_LIBS and <XXX>_APP_LIBS list and keep the unique elements from the list.

The build order should be in reverse order of the link order if the link order is correctly set.

An example to set the build order for ARM_APP is below:

```
ARM_LIB_LIBS := $(call reverse,$(ARM_APP_LIBS))
```

The above line should be inserted after the last line that updates the ARM_APP_LIBS variable.

3.1.5 APU compiler selection

There are two compilers that can be used to build apu target (APEX graphs): the Synopsys compiler or NXP compiler.

The APU_COMP variable should be set to nxp to select the NXP compiler.

The APU_COMP variable should be set to tct to select the Synopsys compiler. The default compiler is Synopsys.

The APU_COMP variable can be set as shell environment variable or can be passed as argument in the command line when the make is invoked.

Examples:

1) let's build the application for Linux or OSLess environment by using NXP compiler:
   - cd <application_path>/build-v234ce-gnu-[linux/sa]-[d/o]
   - make APU_COMP=nxp allsub
2) let's build the application for Linux or OSLess environment by using Synopsys compiler:
   - cd <application_path>/build-v234ce-gnu-[linux/sa]-[d/o]
   - make allsub **OR** make APU_COMP=tct allsub

NOTE: after you install APU compiler you need to do one of the following operations:

- put all binaries for Synopsys compiler into the path (see more information in installer manual)

- define APU_TOOLS pointing to NXP APU compiler installation path

  e.g. "export APU_TOOLS=/home/user/NXP/APU_Compiler_v1.0"

3.1.6 SKIP compilation for APU target

The Vision SDK installer doesn't contain the Synopsys APU compiler. The APU compiler should be installed individually. The pre-compiled code for APEX graphs is put into header files. The header files will be included into ARM application. The Vision SDK contains the pre-compiled code for APEX graphs that were built with Synopsys APU2 compiler, so the user does not need to have the APU compiler installed if user does not need to change the APEX graphs code.

The user can skip the compilation for APU target by setting the APU_PRECOMP to 1.

Examples:

1) let's build the application for Linux or OSLess environment by using pre-compiled code generated with Synopsys APU compiler:
   - cd <application_path>/build-v234ce-gnu-[linux/sa]-[d/o]

- make APU_PRECOMP=1 allsub

NOTE: this is default supported by Vision SDK installer.

2) let's build the application for Linux or OSLess environment by using NXP APU compiler, but the project was previously compiled by using the pre-compiled code generated with Synopsys APU compiler:
   - cd <application_path>/build-v234ce-gnu-[linux/sa]-[d/o]
   - You need to clean all dependences except the pre-compiled code for APEX graphs by using the following command:
     - make APU_PRECOMP=1 cleansub
   - make APU_COMP=nxp allsub

3.1.7 Build a demo project outside the Vision SDK directory tree

All demos from the Vision SDK can be moved/copied to another build location.

The user should correctly set either S32V234_SDK_ROOT or CURR_SDK_ROOT variables before starting the build process.

The Makefile from demo will set the current sdk root (CURR_SDK_ROOT) variable if it is not defined yet in current SHELL environment in the following way:

1. it tries to find the */s32v234_sdk folder (Vision SDK root) in current tree directory and sets it.

2. it sets to S32V234_SDK_ROOT environment variable if the above fails.

3. an error will be reported if the above fails too.

NOTE:

- S32V234_SDK_ROOT variable points to the last Vision SDK installed. It supports the OS-style path.

- CURR_SDK_ROOT supports only Unix-style path.

3.2 List of toolchains/coreutils

Please see ReleaseConfiguration.txt [4].