ALPHA DATA

ADM-VPX3-9Z2
Board Support Package Guide

Document Revision: 1.0
05 December 2018

	Head Office	US Office
Address:	4 West Silvermills Lane Edinburgh, EH3 5BD, UK	611 Corporate Circle, Suite H Golden, CO 80401
Telephone:	+44 131 558 2600	(303) 954 8768
Fax:	+44 131 558 2700	(866) 820 9956 - toll free
email:	sales@alpha-data.com	sales@alpha-data.com
website:	http://www.alpha-data.com	http://www.alpha-data.com

Table Of Contents

List of Tables

List of Figures

Page Intentionally left blank

1 Introduction

The **ADM-VPX3-9Z2** board support package consists of 4 main components:

- Section 3, Vivado Project
- Section 4, First Stage Bootloader (FSBL)
- Section 5, Second Stage Boot Loader
- Section 6, Linux (Yocto)

Pre-built SD Card images are described in Section 2.

1.1 System Requirements

The FSBL and FPGA projects require Vivado and Xilinx SDK version 2017.4, running on either Windows or Linux. See https://www.xilinx.com/support/answers/54242.html for more details about supported operating systems. Building the Yocto Linux image requires specific Linux versions, described in Section 6.1.

2 Pre-Built SD Card Images

2.1 Booting the Board

Some pre-built SD card images are available on github: https://github.com/adps/fsbl-vivado_admvpx39z2/tree/master/SD_images. The "pcie_x4" image will configure the PS GTR interface as PCIE x4 endpoint. The "sata_usb_displayport" image will configure the PS to enable SATA, USB and DisplayPort.

To boot the **ADM-VPX3-9Z2**:

1. Copy the contents of the sata_usb_displayport directory to an SD card.
2. Insert the SD card into the **ADM-VPX3-9Z2** and set the switch settings to boot from the SD card and enable the SATA/DisplayPort (SW4[7:0] 01001100 and SW3 set to all off).
3. Insert the **ADM-VPX3-9Z2** and **ADM-VPX3-9Z2-RTM** into a VPX chassis.
4. Connect a null-modem adaptor and serial cable to the COM port on the RTM and open a serial terminal at 115.2k.
5. If available, a DisplayPort monitor, Ethernet, USB device, and external SATA drive can be connected to the RTM
6. Power the VPX chassis on and the board should start to boot from the SD card.

3 Vivado Project

The Vivado project contains the processing system (PS) configuration, which is required by the First Stage Bootloader. There is also a simple FPGA design attached to an AXI Master bus which contains a block RAM and a block to reverse the data received from the block RAM during a read operation. i.e. writing 0x12345678 to the design will be read back as 0x87654321.

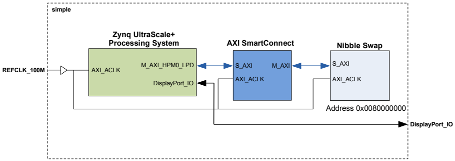

Figure 1 : Simple Design Top Level

3.1 Generating the Vivado project

To generate the Vivado project:

1: Open Vivado 2017.4, and in the tcl console navigate to the fpga/vivado/ directory, e.g:

```
cd C:/ZynqMPSoc/fpga/vivado
```

2: Source the tcl script to generate the project:

```
source ./simple-9z2.tcl
```

This will generate the Vvado project in the fpga/vivado/simple-9z2 directory. A bitstream can then be generated in the usual way.

After bistream generation, the hardware description file can optionally be generated for use with XSDK. A pre-built hdf file is supplied with the BSP, so this is only required if any changes are made to the PS configuration. To export the .hdf file, click File->Export->Export Hardware, then click Ok.

3.1.1 Outputs

After building has completed, the following files can be found:

admvpx39z2_zu15_simple.bit	This file is the FPGA bitstream.
simple_9z2_wrapper.hdf	This is the hardware description file to be loaded by the First Stage Bootloader.

3.1.2 Customising the Processor Configuration

The processor can be configured to us different peripherals, such as the EMMC or SD card, or different configurations of the GTR transceivers. If any modifications are made to the processing system, the block design should be generated by clicking "Generate Block Design" in the left bar, and then exporting the hdf file again. After this, the First Stage Bootloader should be re-generated with the new hdf file.

3.1.3 Constraint files

The pinout for the FMC connector is found in **simple_9z2_fmc.xdc**, and the pinout for the clocks/DisplayPort are found in **9z2.xdc**.

3.1.4 Building the Example Application

There is an application to interact with the Vivado design, located in the apps/simple folder. This application simply writes data to the FPGA design, and then reads it back and prints the results. The FPGA design will swap the written data around, which confirms that it is working properly. After building the cross-compile toolchain (see Section 6.3.2.2) and sourcing the generated environment setup script, run:

```
make all
```

The program can then be run:

```
admvpx39z2> /mnt/9z2_simple
admvpx39z2> Type q to quit or press ctrl-c
admvpx39z2> Enter an 8 digit hex number:
admvpx39z2> 12345678
admvpx39z2> 0x87654321
admvpx39z2>
admvpx39z2> abcdef01
admvpx39z2> 0x10fedcba
admvpx39z2> q
```

4 First Stage Bootloader

The FSBL (First Stage Boot Loader) configures the processor with the settings from Vivado before loading U-Boot.

4.1 Building the FSBL

1: Open a command line interface and change directory to the fsbl folder.

2: Source the XSDK setup script. e.g for Windows:

```
call C:\Xilinx\SDK\2017.4\settings64.bat
```

3: Start XSDK in batch mode, using build_fsbl_2017_4.tcl

```
xsdk -batch -source build_fsbl_2017_4.tcl
```

This will build all the required projects, and generate the BOOT.bin file required to boot.

The BOOT.bin file will include the bitstream file located in the "bit" folder, and the u-boot.elf file located in xsdk/build_bootimg folder.

4.1.1 Outputs

After building has completed, the following files can be found in the build directory:

9z2_fsbl.elf	This file is the first stage bootloader executable.
BOOT.bin	This is the boot file that the Zynq process will first try to load for booting. It contains first stage bootloader
9z2_workspace	This directory contains the workspace and all required projects to build the 9z2 FSBL.

4.1.2 Customising the FSBL and Processor Configuration

The FSBL can be customised after generating the output files. Open Xilinx SDK 2017.4 and set the workspace to the "9z2_workspace" folder that was created when running the fsbl build script. The project called **9z2_fsbl** will contain the first stage bootloader source files, which can be modified as needed.

The project will use a pre-built hdf file that configures the processor to use DisplayPort, USB and the SD Card. This can be changed by right clicking on the 9z2_hw project, and selecting "Change Hardware Platform Specification", click yes to the warning, navigate to the new hdf file and click OK. This will re-generate the first stage bootloader with the updated processor configuration.

After generating the FSBL, a BOOT.bin file must be generated manually:

1. Within Xilinx SDK, Select Xilinx->Create Boot Image.
2. Select Import from existing BIF file
3. In the "Import BIF file path" section, select "browse" and select to "build_bootimg/ mkimg-uboot_bitstream.bif"
4. Click "Create Image"

The updated BOOT.bin file will be genreatred in the xsdk/build_bootimg directory.

5 Second Stage Boot Loader

To make it easier to develop and update Embedded Linux images, the FSBL (First Stage Boot Loader) will load the second stage boot oader, Das U-Boot. Das U-Boot is a second stage boot loader that allows a number of boot options, including booting from QSPI, EMMC, SD card, and TFTP.

5.1 Sources

Acquire the sources to build Das U-Boot using the following command:

```
git clone -b master https://github.com/adps/u-boot-ad-zynqmp.git
```

5.2 Building Das U-Boot

Execute the following commands inside the u-boot-ad-zynqmp directory:

To set up the system for cross-compiling, source the environment setup script that was generated when building the standalone toolchain.

```
source /opt/poky/1.8.2/environment-setup-cortexa9-vpf-neon-poky-linux-gnueabi
```

```
make alphadata_zynqmp_admvpx39z_defconfig
```

```
make all
```

5.2.1 Output files

After building has completed, the following files can be found in the build directory:

u-boot.elf	This file is an ELF version of the built U-Boot binary.

The u-boot.elf file can be used with Xilinx SDK's "bootgen" utility to create the BOOT.bin file required for booting. See https://github.com/adps/fsbl-vivado_admvpx39z2/tree/master/doc/ad-ug_v1_0_fsbl_and_vivado_for_9Z2.pdf for infomration about building the FSBL and BOOT.bin files.

6 Linux Kernel and Root File System

6.1 Required Tools

Note:
To follow this procedure you will require a PC or Virtual Machine running a Desktop Linux distribution. The system must be setup with the correct configuration before starting a build. See the 'The Linux Distribution' and 'The Build Host Packages' sections in http://www.yoctoproject.org for instructions on setting up the required packages on your Linux Desktop system.

6.2 Sources

After setting up a Linux desktop environment for the Yocto build system, **git** can be used to acquire the source code required. Start by creating a new directory for your Embedded Linux project. From inside the new directory use the following commands to acquire a copies of the required code bases.

```
git clone -b admvpx39z2 https://github.com/adps/meta-adlnx.git
git clone -b morty git://git.yoctoproject.org/poky.git
git clone -b rel-v2017.4 https://github.com/Xilinx/meta-xilinx.git
git clone -b morty https://github.com/openembedded/openembedded-core.git
git clone -b master https://github.com/adps/meta-admvpx39z.git
```

The last two code bases, **meta-adlnx** and **meta-admvpx39z**, are Alpha Data's example root files system, and Alpha Data's board support package for the ADM-VPX3-9Z2.

6.3 Building

From the **~/poky** sub-folder in your Embedded Linux project directory, enter the following command to initialise the build environment:

```
source oe-init-build-env
```

This will create a build directory, **build**, which will be switched into to as the active directory after the script completes execution.

6.3.1 Build setup

Before a build can be started, two files must be edited in the **~/poky/build/conf** directory; poky/build/conf/bblayers, and poky/build/conf/local.config

6.3.1.1 poky/build/conf/bblayers

The **bblayers** file must be modified to include the full path of additional layers that need to be added to the Yocto Linux build system. Edit **bblayers** to be similar to the following, including the **meta-adlnx** and **meta-admxrc7z** and layers and the **openembedded-core/meta** layer. Note the path **/home/my_home/yocto_linux** should be changed to the full path of your Embedded Linux project directory.

```
# POKY_BBLAYERS_CONF_VERSION is increased each time build/conf/bblayers.conf
# changes incompatibly
POKY_BBLAYERS_CONF_VERSION = "2"

BBPATH = "${TOPDIR}"
BBFILES ?= ""
```

```
BBLAYERS ?= " \
/home/my_home/yocto_linux/openembedded-core/meta \
/home/my_home/yocto_linux/meta-xilinx \
/home/my_home/yocto_linux/meta-adlnx \
/home/my_home/yocto_linux/meta-admvpx39z \
/home/my_home/yocto_linux/poky/meta \
/home/my_home/yocto_linux/poky/meta-yocto \
/home/my_home/yocto_linux/poky/meta-yocto-bsp \
  "
```

6.3.1.2 poky/build/conf/local.config

The **local.config** file must be modified to specify the target machine. Edit this file to include the following lines to select the target machine and source mirror URL:

```
MACHINE ??= "admvpx39z"
SOURCE_MIRROR_URL ?= "http://petalinux.xilinx.com/sswreleases/rel-v2017.4/downloads"
INHERIT += "own-mirrors"
```

6.3.2 Build

Use the following command in the **~/poky/build directory** to start a build of the Embedded Linux Kernel and root file system. This will take some time to complete.

```
bitbake adlnx-image
```

The output should look similar to the following:

```
Loading cache: 100% |#################################################################
######################################################| ETA:  00:00:00
Loaded 2592 entries from dependency cache.
Parsing recipes: 100% |###############################################################
######################################################| Time: 00:00:01
Parsing of 1787 .bb files complete (1784 cached, 3 parsed). 2591 targets, 128 skip
ped, 0 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies

Build Configuration:
BB_VERSION        = "1.32.0"
BUILD_SYS         = "x86_64-linux"
NATIVELSBSTRING   = "universal-4.8"
TARGET_SYS        = "aarch64-poky-linux"
MACHINE           = "admvpx39z"
DISTRO            = "poky"
DISTRO_VERSION    = "2.2.3"
TUNE_FEATURES     = "aarch64"
TARGET_FPU        = ""
meta              = "morty:1718f0a6c1de9c23660a9bebfd4420e3c4ed37e6"
meta-admvpx39z
meta-adlnx        = "<unknown>:<unknown>"
meta-xilinx       = "rel-v2017.4:cdd0bf1e206922fad1826d818c463e6eae4f8388"
meta
meta-poky
meta-yocto-bsp    = "morty:0e730770a9529f2a0b3219efcc1de7e8842105e8"


NOTE: Preparing runqueue
NOTE: Executing SetScene Tasks
NOTE: Executing RunQueue Tasks
```

6.3.2.1 Output files

After the build is completed, several output files are created in **poky/build/tmp/deploy/images/admvpx39z/**

Image-admvpx39z.bin

The Linux Yocto Poky Morty kernel image. This is the core of the operating system, but requires a device tree to provide details of the hardware configuration.

admvpx39z.dtb

Device tree blob. This provides details of the hardware configuration to the kernel.

adlnx-image-admvpx39z.cpio.gz.u-boot

The root file system as a RAM disk containing a CPIO image prepended with a Das U-Boot header.

6.3.2.2 Building the cross-compile toolchain

In addition to building the root file system and Kernel a standalone toolchain might be needed for developing application outside the Yocto enviroment. This can be build with the following command.

```
bitbake adlnx-image -c populate_sdk
```

This will create the a toolchain installer script **poky/build/tmp/deploy/sdk/**
poky-glibc-x86_64-adlnx-image-aarch64-toolchain-2.2.3.sh. Executing this on a Linux system will allow the user to install cross compiler tools customised to target the adlnx reference image.

After running the toolchain installer script, the environment can be setup for cross-compiling by running

```
source /opt/poky/2.2.3/environment-setup-aarch64-poky-linux
```

After this, the cross-compipler can be called using the environment variabls set by the environment setup script. e.g. gcc can be called with $CC, or g++ can be called with $CXX.

Revision History

Date	Revision	Nature of Change
23/10/18	1.0	Initial Release.
05/12/18	1.1	Updated for 9Z2 board revision 2.