Topic Linux Distribution

Release 2020.2

Niek van Agt, Marc Brakels, Gijs Van Esch, Mike Looijmans

Jun 16, 2021

CONTENTS:

This is the documentation for the TLD (TOPIC Linux Distribution). It describes what it is and how to work with it.

CHAPTER

ONE

WHAT IS THE TLD?

TLD stands for TOPIC Linux Distribution. This embedded Linux distribution can be used to build Linux images for TOPIC boards and development kits. The TLD uses Xilinx' PetaLinux as a base and adds BSP's and supporting scripts for these kits.

The TLD is developed for customers to be able to create prototyping images for TOPIC boards. These images can be used to run proof-of-concepts and other tests on our boards. A simple 'hello world' example is added to the TLD to make it easy for a customer to add a custom application. When prototyping is done these images can be fine-tuned in terms of size/functionality and security to make it a perfect fit for the eventual application.

In the *Get Started* chapter the different aspects of creating a (custom) image and developing/debugging it will be treated and explained in more detail with examples.

1.1 TOPIC boards and development kits

TOPIC developed a complete family of System on Modules (SoM) and carrier boards for them. The SoM's are called Miami's and the carrier boards are called Florida's. To create prototyping platforms, TOPIC created development kits by combining SoM's and carrier boards. These development kits are all supported by the TLD. The BSP's for these kits are added to the distribution through the meta-TOPIC Yocto layer. See the table below for the available boards/kits.

Abbr.	Description	Miami / Florida types
TDKZL	TOPIC Development Kit Zynq-7000 Lite	Miami Lite SoM on a Florida GEN carrier
TDKZ	TOPIC Development Kit Zynq-7000	Miami SoM on a Florida GEN carrier
TDKZU	TOPIC Development Kit Zynq-Ultrascale	Miami MPSoC SoM on a Florida GEN carrier
TDPZU	TOPIC Development kit Plus Zynq-Ultrascale	Miami MPSoC Plus SoM on a Florida Plus carrier
URP	UAV and Robotics Platform	Zynq-Ultrascale 7EV based board suitable for Robotics applications.

For more (detailed) information about the boards and the meta-TOPIC layer, see *Appendix A - Topic boards*.

1.1.1 Support

If something is missing in the documentation or issues pop up, feel free to contact TOPIC: support@topicproducts.com

1.2 Release notes

1.2.1 2020.2.1 (2020-06-03)

Changes:

- Added support for all TOPIC boards
- Fixed QSPI flash driver of Zynq-7000 devices
- Add part about 'version control' to documentation

1.2.2 2020.2.0 (2020-04-01)

This is the first release of the TLD based on petalinux 2020.2. It only supported the TDPZU board.

1.2.3 Known issues

CHAPTER

TWO

GET STARTED

2.1 Set-up development machine

The TLD is based on Xilinx' Petalinux. Therefore, first of all Petalinux 2020.2 must be installed on the development PC. For guidance on installing Petalinux, we refer to chapter 2 of the Petalinux user guide ug1144. Please note we only support 2020.2.

Note: The development PC must be installed with Ubuntu 18.04 LTS.

Note: Make sure you have at least 50 GB free disk space.

Note: Don't forget to configure the terminal to not use dash by running: `sudo dpkg-reconfigure dash`. Choose 'No' when asked for.

The TLD comes with several .bsp files. These are the Board Support Package files that can be loaded into Petalinux to have a kick-start for creating an image for this board. These BSP files can be downloaded from downloads.topic.nl

Now Petalinux is installed and the right BSP file(s) are downloaded, the development PC is correctly configured and ready to use.

2.1.1 Setup static state cache (optional)

Static state cache holds results from previous compile jobs. This way it prevents the need to rebuild everthing, everytime. By default some static state cache is provide by Xilinx. But you can setup your own.

The build will look for `${HOME}/.bitbake-site.conf`. You will need to create this file. A content example is given below. Configuring at least `SSTATE_DIR` and `DL_DIR` already saves a lot of time.

- `SSTATE_DIR` is path to your local static state cache.
- `DL_DIR` is path to your local download cache.

```
# Re-use the build server's hard labour
SSTATE_MIRRORS = "\
        file://.* http://static-state-cache-server.local/share/sstate/PATH \n \
        "
SOURCE_MIRROR_URL = "http://static-state-cache-server.local/sources/"

## Cache on local machine
SSTATE_DIR = "${HOME}/workspace/bitbake_cache/sstate-cache"
DL_DIR = "${HOME}/workspace/bitbake_cache/downloads"
```

For more info on static state cache check yocto manual.

2.2 Build an image

The following steps describe how to build an image for one of the TOPIC development kits.

Note: The assumption is that the PC is configured correctly during the *Set-up development machine* step.

1. **Source petalinux:**

```
source /tools/petalinux/2020.2/settings.sh
```

2. **Create petalinux project/workspace from BSP (.bsp file):**

```
petalinux-create -t project -s BSP_FILE -n PROJECTNAME
```

3. **Build the image:**

```
petalinux-build -c "petalinux-image-minimal-swu-sd"
```

This can take upto a couple of hours to build. Having a machine with many cores speeds up the build a lot. For your reference an AMD Ryzen 9 3900X with 100 MBps internet connection takes about 30 minutes.

When the build is finished the build results will be available in `build/tmp/deploy/images/${TDK}` where ${TDK} is the development kit name (ex: tdkzu)

Note: Setup a static state cache can speed up the build a lot. See *Setup static state cache (optional)*

Note: Sometimes the build fails on a failed download. This can be fixed/worked around by re-running the petalinux-build command.

2.3 Create a bootable SD card

There are several ways to copy the build results to an SD card. Below, 2 ways are described

2.3.1 Using a wic image

The easiest way to create a bootable SD card is to take a WIC image and write it to the SD card using the `dd` command. A tool like bmap-tool or balena-etcher can also be used. For `dd`, first find out what the cardreader device is (plugin the device and run `dmesg` command). Then run the following command, replacing /dev/sdX with the actual device name and TDK with the development kit name (ex: tdkzu):

```
sudo dd if=build/tmp/deploy/images/${TDK}/petalinux-image-minimal-${TDK}.wic of=/
↪dev/sdX bs=1M
```

Note: The SD card needs not be formatted for this. When the board boots, it will automatically resize and create the partitions.

2.3.2 Manually partitioning

Alternatively, you can manually partition the SD card. For example you could use a graphical tool like `gnome-disks`.

1. Partition and format the SD card as follows:

   - `boot`: 128 MB FAT
   - `sd-rootfs-a`: 40% ext4
   - `sd-rootfs-b`: 40% ext4
   - `data`: remaning ext4

   Also mark sd-rootfs-a as bootable
2. From the results directory, copy `boot.bin` to the `boot` partition
3. **Extract the rootfs archive to the `sd-rootfs-a` partition by running:** `tar -xf petalinux-image-minimal-${TDK}.tar.gz --directory=/media/${USER}/sd-rootfs-a/`
4. Safely unmount the SD card (using the umount command or via the filemanager) to make sure everthing is written to the SD card before removing it.

2.4 Boot an image

The several boards can boot from different sources. Supported boot media (per board) are:

- SD (all boards)
- QSPI (all boards)
- eMMC (not present on Zynq-7000 devices)

Boot switches on the board select the boot source. See section *Appendix A - Topic boards*. for the right boot switch setings. That page also describes where the UART interface can be found.

Open a serial connection to the board at baudrate **115200**, no parity, no flow control. For example using `picocom`: `picocom -b 115200 /dev/ttyUSB0`

Power the board, the serial output will look like this:

```
Xilinx Zynq MP First Stage Boot Loader
Release 2020.2   Mar 24 2021  -  11:21:49
PMU Firmware 2020.2     Mar 24 2021   11:21:38
PMU_ROM Version: xpbr-v8.1.0-0
NOTICE:  ATF running on XCZU9EG/silicon v4/RTL5.1 at 0xfffea000
NOTICE:  BL31: v2.2(release):xilinx_rebase_v2.2_2020.2
NOTICE:  BL31: Built : 11:19:30, Mar 24 2021

U-Boot 2020.01 (Mar 24 2021 - 11:21:11 +0000)Topic Miami MPSoC Plus
```

Within a minute the terminal should give a prompt (hostname varies based on ${TDK}:

```
Connected to SWUpdate via /run/swupdateprog
random: crng init done
random: 4 urandom warning(s) missed due to ratelimiting
OK

root@tld-peta-tdpzu9:~#
```

To check that the board is operational type a command like `uptime`:

```
root@tld-peta-tdpzu9:~# uptime
        12:51:24 up 2 min,  load average: 0.00, 0.00, 0.00
```

CHAPTER

THREE

HOW TO . . .

3.1 Use peripherals

3.1.1 LEDs

The TDK's all have 1 or more general purpose LEDs. They are registered (via the devicetree) to the Linux OS as LED devices. Device files for them are created at `/sys/devices/platform/<group-label>/leds/<label>` For example to turn on a LED run the following command (replace the labels with the actual values):

```
echo 1 > /sys/class/leds/<label>/brightness
```

3.1.2 EEPROM

The TDK's all have a I2C EEPROM memory device that can (partly) be used by the user. The first part of this memory is used by TOPIC for production data as serial numbers and MAC address

To find the EEPROM device run the following 'search' command on target:

```
find /sys/bus/nvmem -name nvmem
```

The search result is the path to the EEPROM device

To view its content:

```
hexdump -C <path_to_eeprom_device>
```

To write something to it:

```
echo "Hello World" > <path_to_eeprom_device>
```

3.1.3 RTC (Real Time Clock)

To configure the RTC, first sync time with internet once using `ntpd`

```
echo "server 2.nl.pool.ntp.org" >> /etc/ntp.conf
echo "server 1.us.pool.ntp.org" >> /etc/ntp.conf
ntpd -d -n -q
```

The output should be something like:

```
ntpd: '2.nl.pool.ntp.org' is 149.210.142.45
ntpd: '1.us.pool.ntp.org' is 64.142.54.12
ntpd: sending query to 64.142.54.12
ntpd: sending query to 149.210.142.45
ntpd: reply from 149.210.142.45: offset:+54013.704436 delay:0.007460 status:0x24␣
↪strat:2 refid:0xca4f43c1 rootdelay:0.001831 reach:0x01
```

(continues on next page)

(continued from previous page)

```
ntpd: reply from 64.142.54.12: offset:+54013.700510 delay:0.151523 status:0x24␣
↪strat:3 refid:0x4c4037ce rootdelay:0.089906 reach:0x01
ntpd: sending query to 149.210.142.45
ntpd: reply from 149.210.142.45: offset:+54013.704544 delay:0.008119 status:0x24␣
↪strat:2 refid:0xca4f43c1 rootdelay:0.001831 reach:0x03
ntpd: setting time to 2021-03-30 09:08:45.282092 (offset +54013.704544s)
```

To sync the HW clock with the system clock run the following:

```
hwclock -w
```

After executing the above commands, the system clock should stay intact after a reboot.

3.1.4 Ethernet

Wired ethernet is supported on the TDKZ(U) and TDPZU boards. This interface is registered as the eth0 interface. This can be displayed/configured using the ifconfig command:

```
ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 02:12:78:AB:95:12
          inet addr:192.168.80.81  Bcast:192.168.80.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1311 errors:0 dropped:0 overruns:0 frame:0
          TX packets:593 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:127745 (124.7 KiB)  TX bytes:627085 (612.3 KiB)
          Interrupt:13
```

A command like `ping` can be used to test the connection:

```
ping -c 1 google.com
```

3.1.5 Display

On most of the TDKs a display interface is present as a HDMI output. This display is present as a linux framebuffer device in `/dev/fbX` where X is a sequential number starting at 0.

When running the desktop-image, the framebuffer device is used by that desktop. When running with the minimal-image, the framebuffer can be 'tested' by running:

```
cat /dev/urandom > /dev/fbX
```

3.1.6 WiFi

WiFi is supported on the XDPZU board. To check/configure it run:

```
ifconfig wlan0
wlan0   Link encap:Ethernet  HWaddr C0:EE:40:61:9F:58
        inet6 addr: fe80::c2ee:40ff:fe61:9f58/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:54 errors:0 dropped:0 overruns:0 frame:0
        TX packets:17 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:6164 (6.0 KiB)  TX bytes:3501 (3.4 KiB)
```

When running the desktop-image the GUI can configure the WiFi network.

When running with the minimal-image a connection to a WiFi network can be made using a wpa supplicant file. This file is located (or should be created) at `/etc/wpa_supplicant.conf` and can be 'filled' by using the following command:

```
wpa_passkey MY-SSID MY-PASSWORD >> /etc/wpa_supplicant.conf
```

Change the contents of the file to look like the following:

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
update_config=1
network={
        ssid="SSID-of-network"
        psk="Password-of-network"
}
```

Replacing `SSID-of-network` and `Password-of-network` with the SSID and password of the network you are connecting to.

3.1.7 Bluetooth

Bluetooth is supported on the XDPZU board.

Scan for bluetooth devices using `bluetoothctl`. Running `bluetoothctl` will open a bluetooth console.

In that console, run the following commands:

```
power on
scan on
discoverable on
```

Now the device should show up, type `devices` in bluetooth console to show a list

```
devices
Device 7D:51:A8:F6:3B:9A 7D-51-A8-F6-3B-9A
Device 0C:43:10:1D:B5:43 0C-43-10-1D-B5-43
Device 1D:46:AA:40:F6:DC 1D-46-AA-40-F6-DC
Device 47:A6:EB:66:2E:D3 47-A6-EB-66-2E-D3
Device 00:D1:EB:ED:86:CF 00-D1-EB-ED-86-CF
```

3.2 Use SWUpdate

SWUpdate is a software package for embedded Linux systems to update the contents of non-volatile memories holding the software components like a bootloader, kernel and rootfs.

The SWUpdate service can be reached in several ways but the most convenient way is via a network connection. The board runs a webserver hosting a webpage. Use a browser or program like `curl` to upload a software update package file (.swu) through this webpage. This .swu file contains the new software and also the location of where it should go and other meta-data.

There are several ways to update a board using SWUpdate:

- **SWUpdate webpage:**

  Can be reached by typing its hostname/IP-address + a port number into a browser. The port-number defaults to 8080. This requires a network connection (ethernet, USB or WiFi). Example URL: http://192.168.100.1:8080

- **USB (memory device):** On the desktop images, an USB stick with a .swu file on it can be inserterd. This will then be automatically installed.
- **Command line:** Execute the following command on the target device: `swupdate -v -i <path_to_swu_file>`. This is particularly useful to debug issues with an SWU file.
- **SWUpdate API:** SWUpdate also comes with an API to control it.

For more information check the SWUpdate website.

3.2.1 Update scheme

Most setups use an A/B update scheme. There are two copies of the whole filesystem present, called A and B. When the system is running from A, the update process will write the new image to the B partition. On success, the system will mark B as bootable and reboot. This makes the upgrade almost atomic, a failure during the upgrade will have no effects on the running system.

By default, for SD/eMMC we use A/B scheme for all devices using 4 partitions:

- `boot` Hold the FSBL/u-boot
- **`x-rootfs-a` and `x-rootfs-b` Where *x* is either `sd` or `emmc`** These partitions are the ones being updated by SWUpdate. Either a or b is the 'active' partition.
- `data` General data partition. Can be used to hold persistent data, SWUpdate will not touch this partition. By default it is empty.

The following will happen when providing a new SWUpdate package for eMMC.

1. The non-active will be mounted (lets say B)
2. SWUpdate writes the Update to non-active partition
3. Copy peristent files from A to B (see *Persistent files*)
4. SWUpdate moves the bootable flag from A to B
5. SWUpdate triggers a reboot
6. The boot loader will detect bootable flag is now on B, and will use that one to boot from.

When there is not enough space on the media for two copies, a single update scheme can be used. This allows an atomic update only by booting from another device, for example one can upgrade the QSPI while running from eMMC. When running from QSPI, it is still possible to upgrade the image in QSPI but failure during the upgrade will result in the system being unable to boot. In this case the user will have to manually repair this, for example by booting from an SD card instead.

The following configs use a single scheme instead:

- XDP with QSPI
- TDKZ with QSPI

3.2.2 Persistent files

Some files can be left untouched when updating the board's software with SWUpdate, like for example the WiFi configuration.

See for full list: `meta-topic-platform/recipes-support/swupdate/swupdate/swu-transfer-list`

After writing the new copy of the filesystem, the upgrade process will copy the files in this list (if they exist) to the newly installed copy. At the moment, this is not available on QSPI using ubifs.

3.3 Create your own Vitis app

Xilinx' Vitis SDK can be used to create applications for the TOPIC boards. To create a platform project, import the .XSA file generated by Vivado. Check *FPGA technical reference designs (TRDs)* for more information about this .XSA file.

3.3.1 Standalone (bare-metal) application

For the steps on how to create a standalone application with the Vitis SDK we refer to the
Vitis documentation - Standalone

3.3.2 Linux application

For the steps on how to create a Linux application with the Vitis SDK we refer to the
Vitis documentation - Linux

3.4 Create and use an SDK

In order to generate the SDK in the petalinux project folder run

```
petalinux-build --sdk
```

This command will create a sdk.sh script in the folder images/linux.

To create the SDK, run the petalinux-package command:

```
petalinux-package --sysroot -s images/linux/sdk.sh
```

This will create an sdk folder in images/linux which contains the SDK. This directory can also be changed by passing the -d or –dir flag to the petalinux-package command.

For more information on how to use the SDK, see the UG1144 Petalinx Reference Guide.

3.5 Use a custom FPGA image

By default the TLD will fetch a prebuild FPGA image from downloads.topic.nl The following steps describe how to use a custom build FPGA image.

1. Create your Vivado project (it is recommended to use the TRD as a starting point, see *FPGA technical reference designs (TRDs)*.)

   Note: The TLD 2020.2 release only supports the use of Vivado 2020.2.

2. Build your FPGA image
3. **Export the build results: choose `File -> Export -> Export hardware` (Use default settings in wizard)**
   This exports a .xsa file which is needed in the next step.
4. Import the results into the petalinux project by running the following command in the petalinux workspace:

   ```
   petalinux-config --silentconfig --get-hw-description ${XSA_FILE}
   ```

5. **Generate new device tree based on new FPGA image (by default the TLD does not do this, as it is not reliable)**

Enable the devicetree generation by running the following command: `petalinux-config`
In the menu change the following:

- DTG Settings -> DEselect "Remove PL from devicetree"
- Save the configuration -> default location
- Exit the menu

6. **Rebuild the design** Clean the project (sometimes old build results are not cleaned automaticaly) and then rebuild:

```
rm -rf components/ build/
petalinux-build -c device-tree -x do_compile
```

Now petalinux has generated a device tree based on the new FPGA image. Unfortunately this can not be used as is but it can be used as a starting point / reference.

1. **Create a new device tree file at `project-spec/meta-user/recipes-bsp/device-tree/dtb-example-cus`**
An example of the contents of this file is shown below:

```
SUMMARY = "Devicetree overlay for FPGA image"
require ../../../topic-platform/meta-topic/recipes-bsp/device-tree/dtb-
↪overlay.inc

COMPATIBLE_MACHINE = ".*"

BITSTREAM = "fpga-image-example-custom"
```

2. **Create a new folder to hold the new devicetree files:** `project-spec/meta-user/recipes-bsp/device-tree/dtb-example-custom/`
3. **Copy the device tree from the reference design to**

`project-spec/meta-user/recipes-bsp/device-tree/dtb-example-custom/pl.dts`
Check *Appendix A - Topic boards* for the location of the reference device tree file of your board.

4. **Update the reference device tree with the new components.**

The auto generated version is located here:
`components/plnx_workspace/device-tree/device-tree/pl.dtsi`
It is recommended to only copy/overwrite the changed parts into the new device tree
`project-spec/meta-user/recipes-bsp/device-tree/dtb-example-custom/pl.dts`

5. **Disable the device tree generation again by running the following command: `petalinux-config`**

In the menu do the following:

- DTG Settings -> select "Remove PL from devicetree"
- Save the configuration -> default location
- Exit the menu

The device tree part is done now.
The next steps show how to generate a recipe for the new FPGA image. This to include the FPGA image into the build.

6. **Create a new folder `project-spec/meta-user/recipes-bsp/fpga` and create a new file in there: `fpga-ima`**
An example of the content is shown below. Change the `${FPGA_BITFILE}` variable with the

correct filename.

```
SUMMARY = "FPGA image"
require recipes-bsp/fpga/fpga-image.inc
LICENSE = "CLOSED"

COMPATIBLE_MACHINE = ".*"

FILESEXTRAPATHS_prepend := "${THISDIR}/${PN}:"

PV = "1"
FPGA_BITFILE = "fpga-image-example-custom.bit"

BOARD_DESIGN_NAME = "fpga-image-example-custom"
BOARD_DESIGN_URI = "file://${FPGA_BITFILE}"

PKGV = "${PV}"
S = "${WORKDIR}/${BOARD_DESIGN_PATH}"
B = "${S}"

# Nothing to build
do_compile() {
        cp ${FPGA_BITFILE} fpga.bit
}
```

7. **Copy the .bit file to `project-spec/meta-user/recipes-bsp/fpga/fpga-image-example-custom/`**
   Make sure to use the same filename as provided in the recipe `${FPGA_BITFILE}`. The recipe created in the previous step will pick up the .bit file and copy it into the build.

8. **Update the `project-spec/meta-user/recipes-core/images/user-config.inc` to include new FPGA im**
   Do this by adding the name of the new recipe to the `IMAGE_INSTALL_append` list and adding the original dtb recipe to the `IMAGE_INSTALL_remove` list. See example below:

```
IMAGE_INSTALL_append = " \
   dtb-example-custom \
"

IMAGE_INSTALL_remove = "dtb-miami-florida-gen-reference"

# login by default
inherit autologin
```

9. Now we can build again and the new FPGA image will be built and included into the image:

```
petalinux-build
```

CHAPTER

FOUR

FPGA TECHNICAL REFERENCE DESIGNS (TRDS)

Topic provides FPGA technical reference designs (TRD) for its boards. These reference designs can be found here

Note: When building the XDPZU design, additional licenses are required to build the design.

4.1 Generate and build Vivado project

The TRDs are provided as a .zip file. To build the TRD first unzip it. The .zip file contains a shell file (.sh) and a batch file (.bat) so it can be run on either a Linux or a Windows machine. The instructions on how to build for both OSses are below.

4.1.1 Instructions for Ubuntu

1. Open a terminal window
2. Step into the unzipped TRD directory

```
cd ${TRD_DIR}
```

3. Source Vivado's settings script so that Vivado can be run from the terminal (path to installation directory might vary)

```
source /opt/Xilinx/Vivado/2020.2/settings64.sh
```

4. **Set the FPGA_FAMILY environment variable to match the right FPGA (ex: xczu9eg).** Tip: Run the next step (generate_bitstream.sh) without setting the FPGA_FAMILY variable to see the value options.

```
export FPGA_FAMILY=xczu9eg
```

5. Run the shell script to generate and build the TRD

```
./generate_bitstream.sh
```

This will generate a directory with the FPGA name and the TRD will be created in that directory.

4.1.2 Instructions for Windows 10

1. Open a command prompt
2. Step into the unzipped TRD directory

```
cd %TRD_DIR%
```

3. Execute Vivado's settings64.bat so that Vivado can be run from the command prompt (path to installation directory might vary).

```
C:\Xilinx\Vivado\2020.2\.settings64-Vivado.bat
```

4. **Set the FPGA_FAMILY environment variable to match the right FPGA family (ex: xczu9eg).** Tip: Run the next step (generate_bitstream.bat) without setting the FPGA_FAMILY variable to see the value options.

```
set FPGA_FAMILY=xczu9eg
```

5. Generate the bitstream

```
generate_bitstream.bat
```

This will generate a directory with the FPGA name and the TRD will be created in that directory.

4.2 Export Hardware (for SW development)

After the FPGA build is finished, an .xsa file has been automatically generated in the TRD directory. This file can be imported into Vitis to *Create your own Vitis app* for the board.

CHAPTER

FIVE

VERSION CONTROL

This chapter describes the preferred way to do version control with the TLD, using a GIT repository. This allows for a future update to a new version of the TLD and facilitates support (from TOPIC).

The flow below downloads the BSP file from a remote source. This removes the need to commit the BSP file into the repository.

Note: When `${company_name}` is shown below, replace it with your company name, so for exapmle `topic`. The same goes for `${product_name}`.

Follow the steps below to create a new petalinux project from a BSP file.

1. **Create an empty directory with the following name `${company_name}-${product_name}-platform` and step**

```
mkdir ${company_name}-${product_name}-platform
cd ${company_name}-${product_name}-platform
```

2. **Add the following folders: `config`, `meta-${product_name}`, `scripts`.**

```
mkdir config meta-${product_name} scripts
```

Note: In this example we only create one meta-layer, within the git repo. Meta-layers are often added as git submodules from external sources, like meta-qt5 for example.

3. **Initialize a GIT repository by running:**

```
git init
```

4. **Add a `.gitignore` file and as 'starting point' fill it with the lines below:**

```
${product_name}
data
```

5. **Create the config for a new meta layer in `meta-${product_name}/conf/layer.conf`. Again a 'starting point'**

```
# Add the layer to the ``BBPATH``
# Prepend (=.) to make sure it looks for .inc files in this layer first
BBPATH =. "${LAYERDIR}:"

# Add the directories that contain recipes to ``BBFILES``
BBFILES += "${LAYERDIR}/recipes-*/*/*.bb \
        ${LAYERDIR}/recipes-*/*/*.bbappend"

BBFILE_COLLECTIONS += "meta-${product_name}"
```

(continues on next page)

(continued from previous page)

```
BBFILE_PATTERN_meta-${product_name} = "^${LAYERDIR}/"
BBFILE_PRIORITY_meta-${product_name} = "7"
LAYERSERIES_COMPAT_meta-${product_name} = "zeus"
LAYERDEPENDS_meta-${product_name} = "topic-platform-layer"
```

6. Create the user config file in `meta-${product_name}/recipes-core/images/user-config.inc`

   Add packages that need to be installed into the image. See the example below which adds python and enables the autologin feature:

```
IMAGE_INSTALL_append = " \
        python3 \
"

# login by default
inherit autologin
```

7. **Create a shell script to set up the environment to start the build in.** Create `scripts/setup.sh` with the example content below. Update the paths depending on your setup and make sure you add execution permissions to the script (chmod +x scripts/setup.sh).

```
#!/bin/sh -e
DOWNLOAD_DIR='data'
BSP_NAME="minimal-tdkzu9-default-2020.2-35c45d4.bsp"
REMOTE_BSP_URI="http://${DATA_SERVER}/downloads/tld/${BSP_NAME}"
REMOTE_XSA_URI='http://${DATA_SERVER}/downloads/fpga/fpga-image-${company_
↪name}-${product_name}/fpga-image-${company_name}-${product_name}-
↪9dacc9776efe8c0e3ea5114b3157ddbe51a4f30c.tar.xz'
LOCAL_BSP_PATH="${DOWNLOAD_DIR}/${BSP_NAME}"
LOCAL_XSA_PATH="${DOWNLOAD_DIR}/rel/${product_name}.xsa"
PETA_PROJECT_NAME='${product_name}'

note ()
{
        echo -e "\033[32mNOTE: $1\033[39m"
}

warning ()
{
        echo -e "\033[33mWARNING: $1\033[39m"
}

error ()
{
        echo -e "\033[31mERROR: $1\033[39m"
}

if [ ! -e "${LOCAL_BSP_PATH}" ]; then
        warning "Downloading bsp from ${REMOTE_BSP_URI}"
        wget -q -O "${LOCAL_BSP_PATH}" ${REMOTE_BSP_URI}
fi

if [ ! -e "${LOCAL_XSA_PATH}" ]; then
        warning "Downloading xsa from ${REMOTE_XSA_URI}"
        wget -q -O "/tmp/downloaded.xsa.xz" ${REMOTE_XSA_URI}
        tar -xf "/tmp/downloaded.xsa.xz" --directory=${DOWNLOAD_DIR}

fi

note "Removing old petalinux project if exists in ${PETA_PROJECT_NAME}/"
```

(continues on next page)

(continued from previous page)

```
rm -rf ${PETA_PROJECT_NAME}

if [ ! -e "${LOCAL_BSP_PATH}" ]; then
        error "BSP doesn't exist ${LOCAL_BSP_PATH}"
        return
fi
if [ ! -e "${LOCAL_XSA_PATH}" ]; then
        error "xsa doesn't exist ${LOCAL_XSA_PATH}"
        return
fi

note "Create petalinux project in ${PETA_PROJECT_NAME}/"
petalinux-create \
        --type project \
        --name ${PETA_PROJECT_NAME} \
        --source ${LOCAL_BSP_PATH}

cd ${PETA_PROJECT_NAME}
note "Update with latest XSA file"
petalinux-config --silentconfig --get-hw-description ../${LOCAL_XSA_PATH}

note "Update with user config"
if [ -e "../config/config" ]; then
        cp ../config/config project-spec/configs/
else
        warning "File config/config doesn't exist, will use config from␣
↪BSP file."
fi

if [ -e "../config/rootfs_config" ]; then
        cp ../config/rootfs_config project-spec/configs/
else
        warning "File config/rootfs_config doesn't exist, will use config␣
↪from BSP file."
fi

petalinux-config --silentconfig
note "Setup of petalinux project completed."
```

8. **Create a shell script that will automate the build.** Create `scripts/autobuild.sh` and fill it with the example content below. Also make sure to add execution permissions to the script (chmod +X scripts/setup.sh).

```
#!/bin/sh -e
./scripts/setup.sh
cd ${product_name}
petalinux-build -c "petalinux-image-minimal-swu-emmc"
mkdir -p ../results
cp build/tmp/deploy/images/*/petalinux-image-minimal-*.wic.xz ../results/.
cp build/tmp/deploy/images/*/petalinux-image-minimal-swu-emmc-*.swu ../
↪results/.
```

9. **Run the setup script (making sure petalinux is sourced).**

```
source <petalinux_install_dir>/settings.sh
./scripts/setup.sh
```

   This creates the petalinux project. The config files from this project need to be checked in into the GIT repository.

10. Copy the config files to the GIT repository:

```
cp ${product_name}/project-spec/configs/config config/
cp ${product_name}/project-spec/configs/rootfs_config config/
```

11. **Open `config/config` and scroll to the end of the file.** Add the new user (meta) layer to the `CONFIG_USER_LAYER_x` variables. It should look like the example contents below:

```
#
# User Layers
#
CONFIG_USER_LAYER_0="${proot}/../meta-${product_name}"
CONFIG_USER_LAYER_1="${proot}/project-spec/meta-user"
CONFIG_USER_LAYER_2="${proot}/project-spec/meta-swupdate"
CONFIG_USER_LAYER_3="${proot}/project-spec/meta-topic"
CONFIG_USER_LAYER_4="${proot}/project-spec/meta-topic-platform"
CONFIG_USER_LAYER_5="${proot}/project-spec/meta-xilinx-standalone"
CONFIG_USER_LAYER_6="${proot}/project-spec/meta-rust"
CONFIG_USER_LAYER_7="${proot}/project-spec/meta-dyplo"
CONFIG_USER_LAYER_8=""
```

12. **Now the project is ready to be built, by running:**

```
./scripts/autobuild.sh
```

CHAPTER

SIX

APPENDIX A - TOPIC BOARDS

6.1 TDKZ

The TDKZ is the Topic Development Kit Zynq-7000 which consists of a Miami Zynq on a Florida GEN board. The next table shows the available peripherals/interfaces and their default configuration.

6.1.1 Memory

Type	Description	Size	Chip
DDR	Micron Technology DDR3L DRAM	1 GB	MT41K256M16TW-107 XIT
QSPI	Micron Technology NOR flash	32 MB	N25Q256A11E1240F
NAND	Micron Technology 2Gb NAND (Optional)	256 MB	MT29F2G16
EEPROM	On Semiconductor secured eeprom	4Kb	CAT24C04TDI-GT3

6.1.2 Boot switches:

Name	S1 bootswitch positions							
	1	2	3	4	5	6	7	8
QSPI	on	off	off	on	off	on	on	off
SD card	off	on	off	on	off	on	on	off
JTAG	on	off	on	off	off	on	on	off

6.1.3 Devicetree file

The reference devicetree file of this board is located in the petalinux workspace (.bsp file): `project-spec/meta-topic/recipes-bsp/device-tree/dtb-miami-florida-gen-reference/topic-miami/pl.dts`

6.2 TDKZU

The TDKZU is the Topic Development Kit Zynq MPSoC which consists of a Miami MPSoC on a Florida GEN board. The next table shows the available peripherals/interfaces and their default configuration.

6.2.1 Memory

Type	Description	Size	Chip
DDR	Micron LPDDR4	2GB	MT53E512M32D2NP-046 WT:E
QSPI	Micron Technology NOR flash	64 MB	MT25QU256ABA8E12-1SIT
eMMC	SanDisk eMMC 8GB	8 GB	SDINBDG4-8G-XI1
EEPROM	On Semiconductor secured eeprom	4Kb	CAT24C04TDI-GT3

6.2.2 Boot switches

Name	S1 bootswitch positions					
	1	2	3	4	5	6
QSPI	off	on	on	off	off	on
SD-Card	on	off	off	on	on	off
eMMC	on	off	on	off	off	on

6.2.3 Devicetree file

The reference devicetree file of this board is located in the petalinux workspace (.bsp file): `project-spec/meta-topic/recipes-bsp/device-tree/dtb-miami-florida-gen-reference/topic-miamimp/pl.dts`

6.3 TDPZU

The TDPZU is the Topic Development Kit Zynq MPSoC Plus which consists of a Miami MPSoC Plus on a Florida Plus board. The next table shows the available peripherals/interfaces and their default configuration.

6.3.1 Memory

Type	Description	Size	Chip
DDR	Micron Technology DDR4 DRAM	2-8 GB	MT40A512M16JY-083E IT:B
QSPI	Micron Technology NOR flash	64-256 MB	MT25QU01GBBB8E12-0SIT
eMMC	SanDisk eMMC 8GB	8 GB	SDINBDG4-8G-XI1
EEPROM	ST 32-Kbit I2C eeprom	32 Kb	M24C32S-FCU

6.3.2 Boot switches

Name	S1 bootswitch positions					
	1	2	3	4	5	6
QSPI	on	off	on	on	x	x
SD-Card	off	off	on	on	x	x
eMMC	on	off	off	on	x	x

6.3.3 Devicetree file

The reference devicetree file of this board is located in the petalinux workspace (.bsp file): `project-spec/meta-topic/recipes-bsp/device-tree/dtb-tdpzu9-reference/pl.dts`

6.4 URP

The URP is a Zynq-Ultrascale 7EV based board suitable for Robotics applications. The next table shows the available peripherals/interfaces and their default configuration.

6.4.1 Memory

Type	Description	Size	Chip
DDR	Micron Technology DDR4 DRAM	4 GB	MT40A512M16JY-083E IT:B
QSPI	Micron Technology NOR flash	64 MB	MT25QU256ABA8E12-1SIT
eMMC	SanDisk eMMC 8GB	8 GB	SDINBDG4-8G
EEPROM	ST 32-Kbit I2C eeprom	32 Kb	M24C32S-FCU

6.4.2 Boot switches

Name	S1 bootswitch positions		
	1	2	3
QSPI	on	off	on
SD card	off	on	off
eMMC	off	off	on

Only the first 3 switches are used to select the boot mode.

6.4.3 Devicetree file

The reference devicetree file of this board is located in the petalinux workspace (.bsp file): `project-spec/meta-topic/recipes-bsp/device-tree/dtb-xdp-reference/pl.dts`

CHAPTER

SEVEN

APPENDIX B - META LAYERS

7.1 meta-topic layer

7.2 meta-topic-platform layer

7.3 meta-topic-desktop layer

Last Updated on 2021-06-16