Up your PetaLinux game with BSPs

Board Support Package creation: explanations and examples

N. Džemaili
CMS DAQ & Trigger group

Acknowledgements: P. Žejdl & M. Dobson

Summary

- What is a BSP and why to create one?
- PetaLinux inner workings
  (yocto layer and recipe introduction)
- Example BSP creation for ZCU102 development board with customization of:
  - PMU firmware
  - FSBL
  - U-Boot
  - Device-tree
  - Kernel

ZCU102 development board
Zynq Ultrascale+ MPSoC

What is a BSP and why to create one?

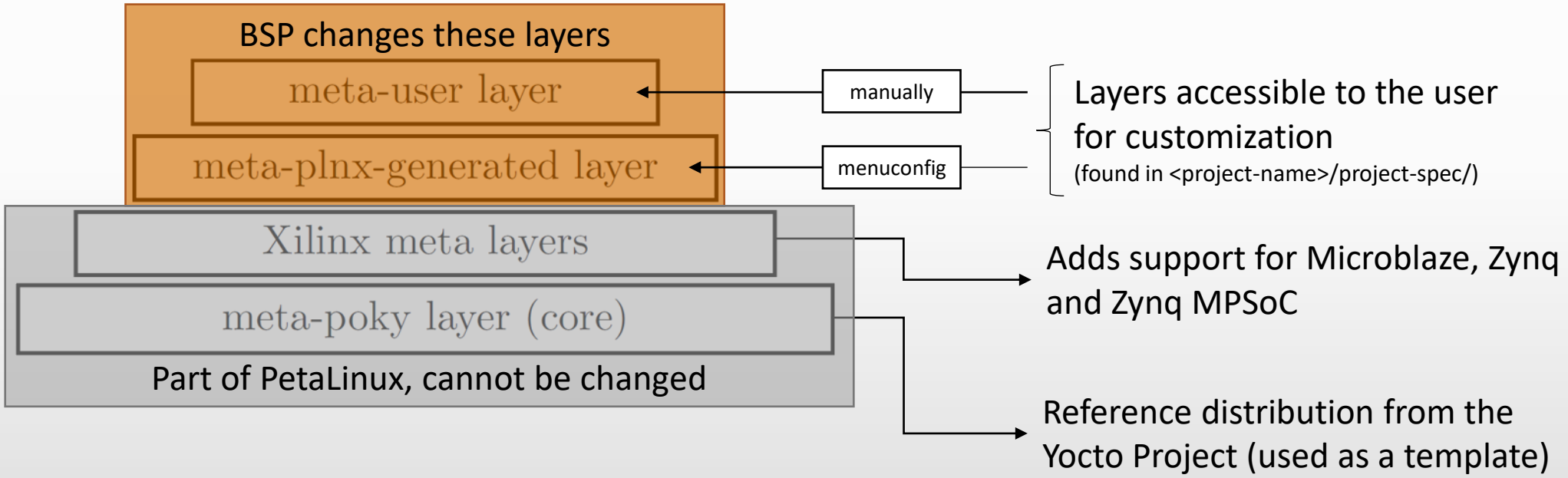
A board support package is a **template** that defines how to support a particular hardware platform

- Why create a BSP?
  - A BSP allows you to define all the features for your board that you want to use
  - Package definitions into reusable format
  - Why not? :D

- Most important parts of a BSP:
  - Device-tree
  - Kernel configuration
  - U-Boot configuration

A PetaLinux project from the inside

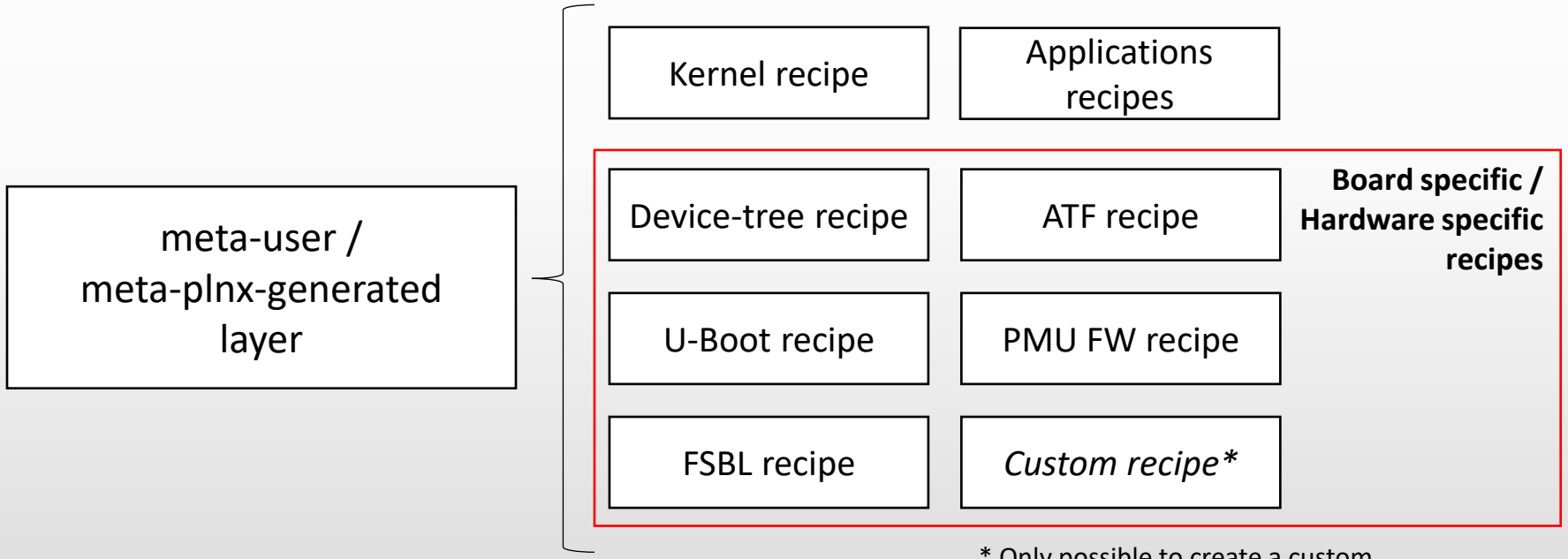
PetaLinux uses **Yocto layers** under its hood:



These Yocto layers contain recipes

A layer consists of recipes

A recipe is a file that provides a *"list of ingredients"* and *"cooking instructions"* for *"baking"* a part of the project → BitBake recipe file (`.bb` or `.bbappend`)



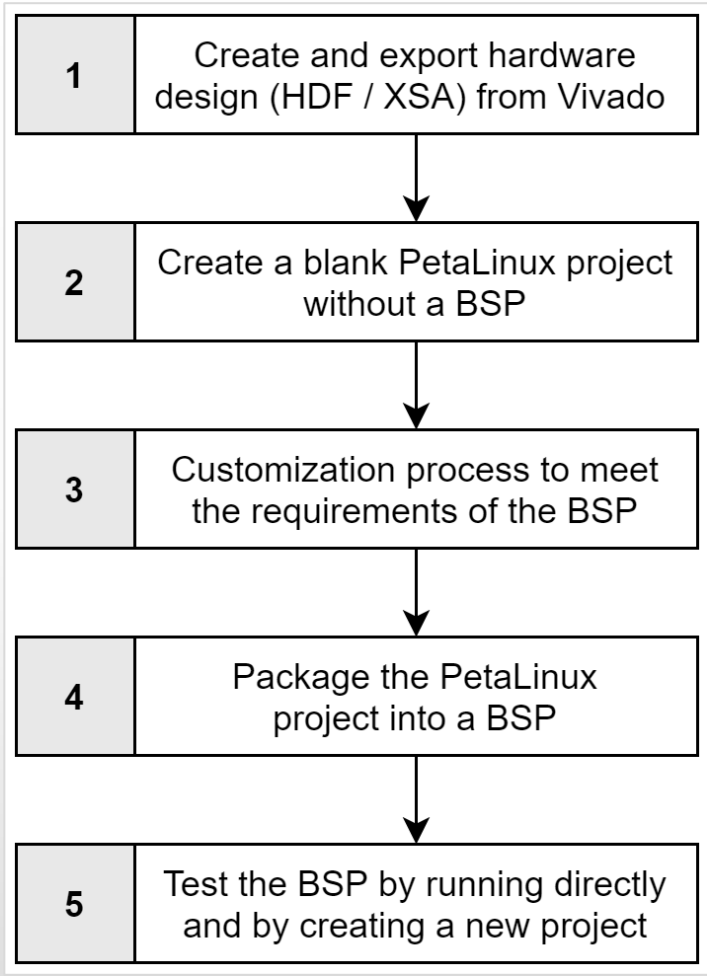
* Only possible to create a custom recipe in the meta-user layer

BSP creation workflow

- 5 steps when creating a Board Support Package
  - HDF = Hardware Description File
    (XSA in newer versions of Vivado)

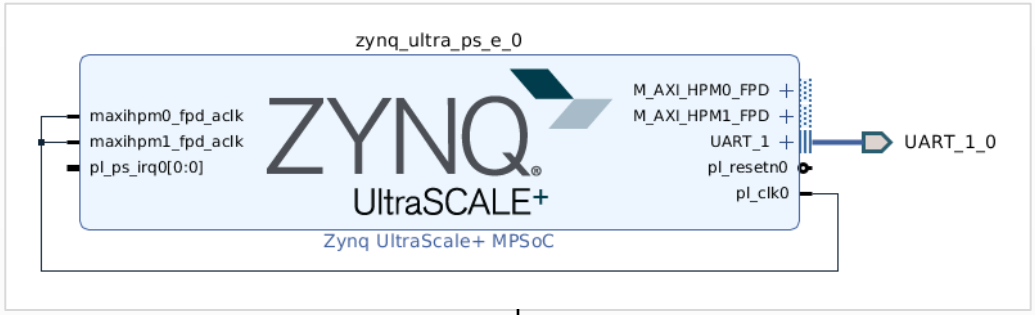
- This workflow shows that a BSP is **actually** a PetaLinux project that is packaged and used as an template

5 steps to creating a BSP



Exporting your hardware design

Block design in Vivado:



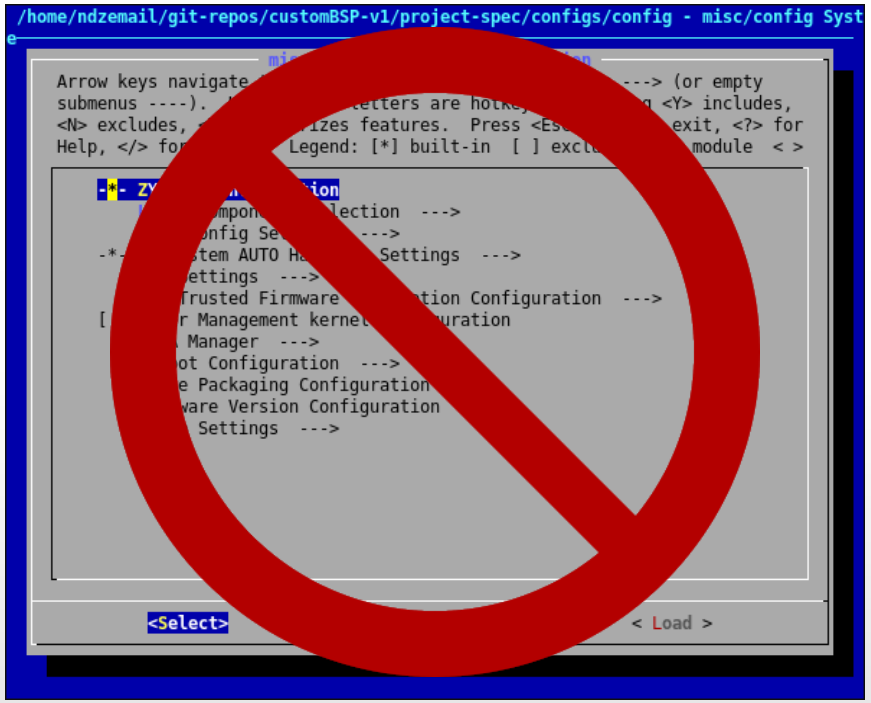
File → Export → Export hardware…

Make sure to include the bitstream as well

CMS-ZCU102-hardware.xsa

Creating a PetaLinux project

```
$ petalinux-create -t project -n CMS-ZCU102-v1 --template zynqMP

$ petalinux-config --get-hw-description=<vivado-project-path>
--silentconfig
```



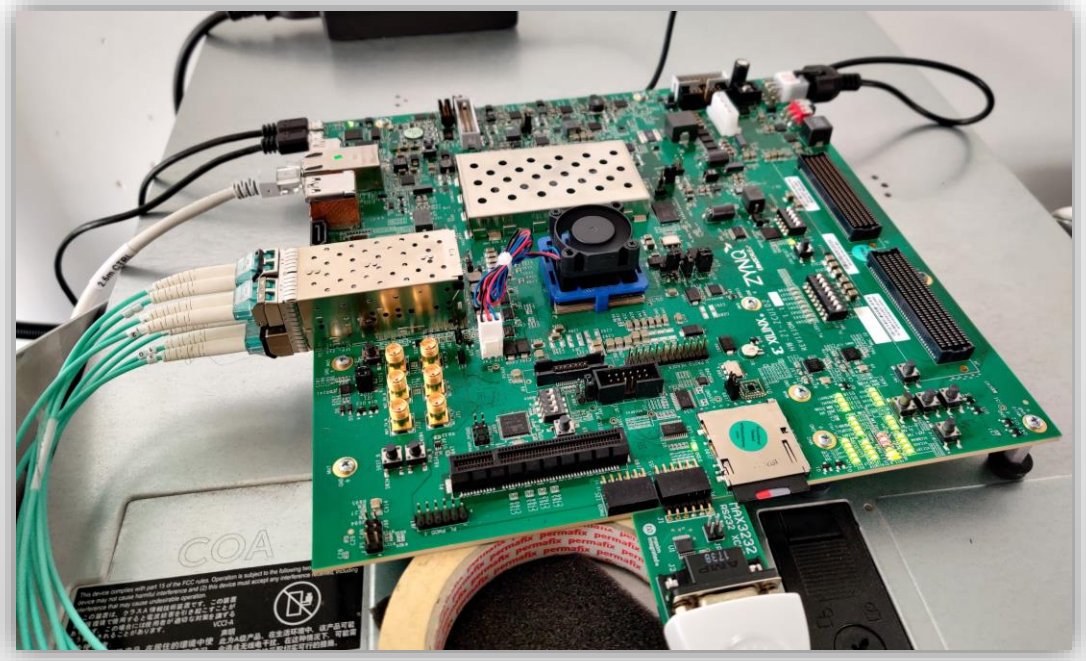
- Create the PetaLinux project using the **zynqMP** template
- Configure the project using the HDF / XSA file

- Use the **silentconfig** option to use default configuration

Example BSP creation for ZCU102

- Hardware used:
  - ZCU102 Development board (Zynq MPSoC)

- Board specific requirements:
  1. PMU watchdog timers enabled
  2. FSBL needs to provide debug info
  3. Use the MAC-address stored in the EEPROM
  4. Network boot using TFTP + NFS
  5. Kernel support for crashkernel (dump-capture kernel)



The ZCU102 board in our lab

PMU FW watchdog configuration

How to enable

- The Zynq MPSoC has 2 main watchdog timers:
  - LPD watchdog timer (Low Power Domain)
  - FPD watchdog timer (Full Power Domain)

  (Zynq MPSoC watchdog timers)

The PMU firmware handles watchdog timers

- Add watchdog hardware to device-tree
- Add these build flags for PMU:

Required by	**ENABLE_RECOVERY**	**Watchdog timer handling by PMU**	
	ENABLE_EM	Error management module	
	ENABLE_SCHEDULER	Scheduler module	**(Enabled by default)**
	ENABLE_PM	Power management module	**(Enabled by default)**

(PMU firmware build flags)

Adding build flags to PMU FW

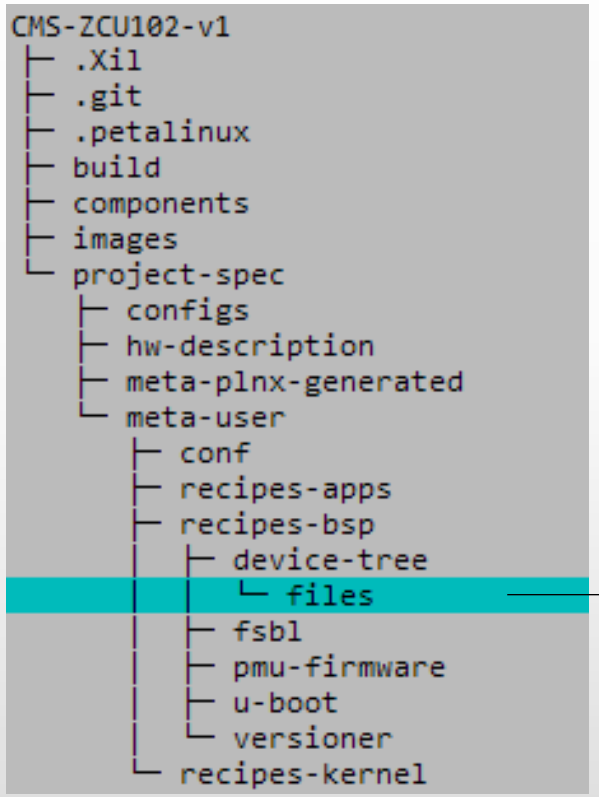
```
CMS-ZCU102-v1
├─ .Xil
├─ .git
├─ .petalinux
├─ build
├─ components
├─ images
└─ project-spec
   ├─ configs
   ├─ hw-description
   ├─ meta-plnx-generated
   └─ meta-user
```

1. Create **`pmu-firmware`** directory
2. Create **`pmu-firmware_%.bbappend`** file
3. File contents:

```
YAML_COMPILER_FLAGS_append = " -DENABLE_EM -DENABLE_RECOVERY “
```

Adding watchdog hardware to device-tree

Accessing the watchdogs from Linux



Edit the **`system-user.dtsi`** file

Initial file contents:

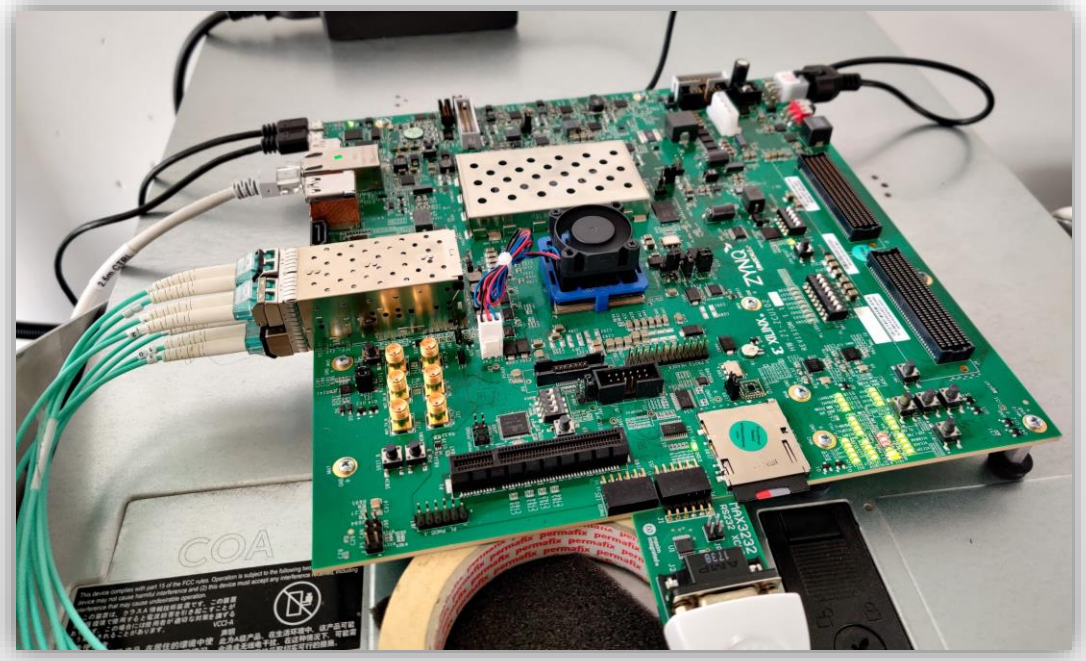
```
/include/ "system-conf.dtsi"

/ {

};
```

After adding watchdog hardware:

```
/include/ "system-conf.dtsi"

/ {
    model = “CMS ZCU102 Devboard";
    compatible = "xlnx,zynqmp";
};

&watchdog0 {
        status = "okay";
        reset-on-timeout;
};

&lpd_watchdog {
        status = "okay";
        reset-on-timeout;
};
```

Example BSP requirements (board specific)

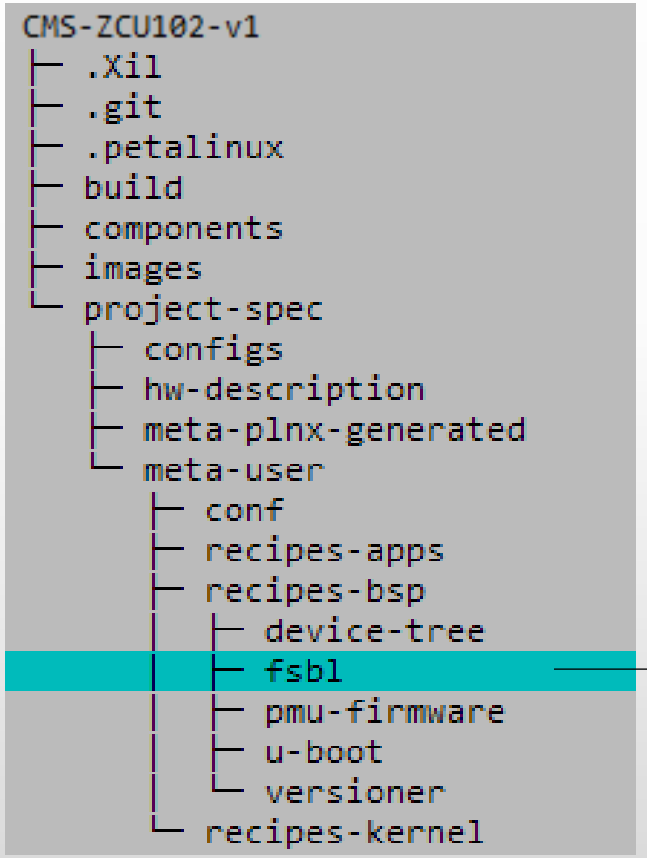
1. ~~PMU watchdog timers enabled~~
2. FSBL needs to provide debug info
3. Use the MAC-address stored in the EEPROM
4. Network boot using TFTP + NFS
5. Kernel support for crashkernel (dump-capture kernel)



The ZCU102 board in our lab

FSBL debug output configuration

Adding build flags

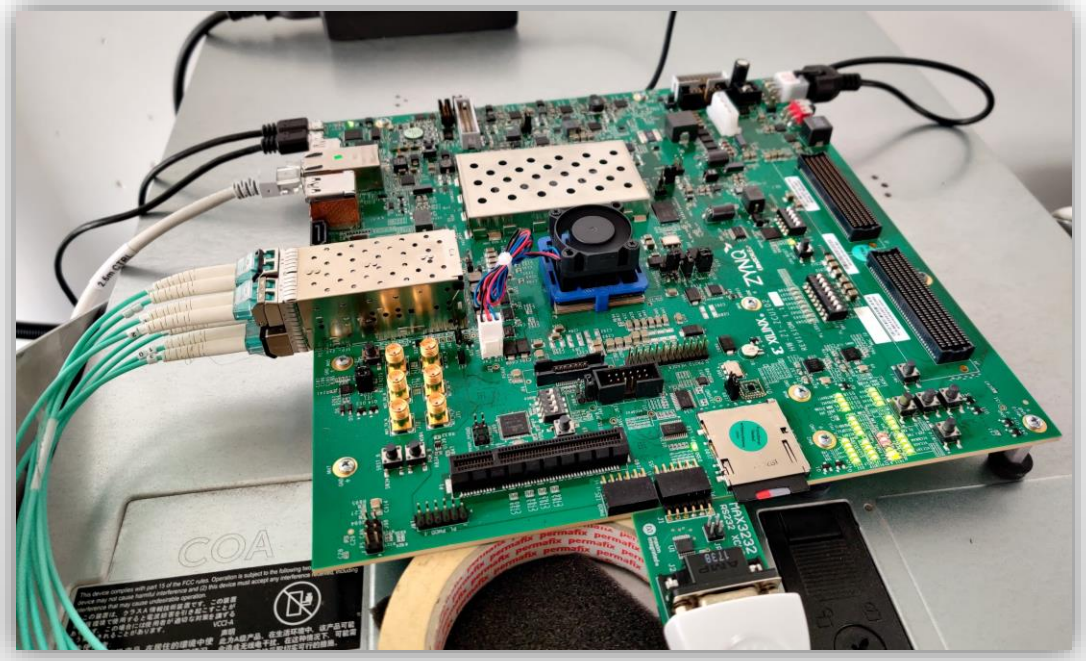


1. Create **`fsbl`** directory
2. Create **`fsbl_%.bbappend`** file
3. File contents:

```
XSCTH_BUILD_DEBUG = "1"
YAML_COMPILER_FLAGS_append = " -DFSBL_DEBUG_INFO"
```

Example BSP requirements (board specific)

1. ~~PMU watchdog timers enabled~~
2. ~~FSBL needs to provide debug info~~
3. Use the MAC-address stored in the EEPROM
4. Network boot using TFTP + NFS
5. Kernel support for crashkernel (dump-capture kernel)



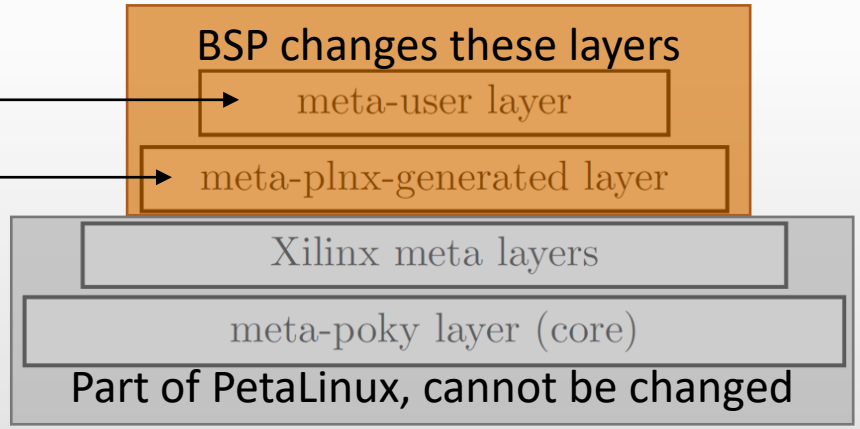
The ZCU102 board in our lab

EEPROM configuration for MAC-address retrieval

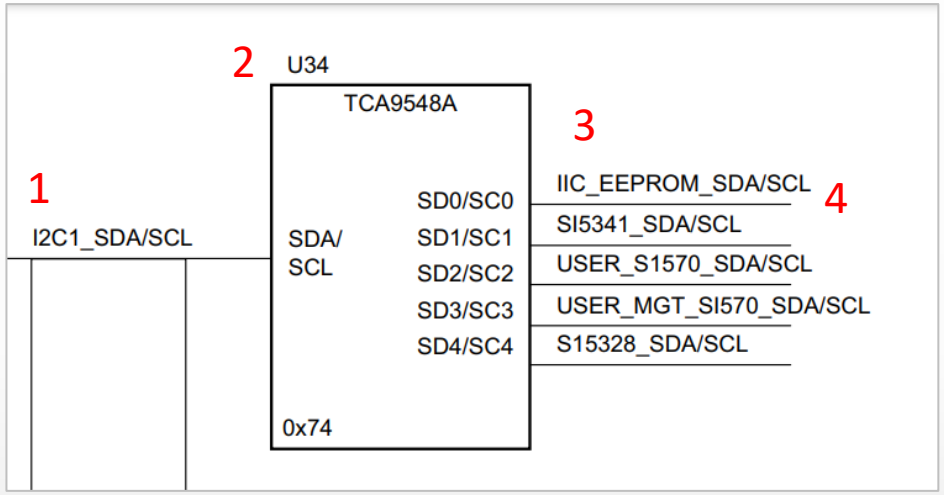
- Consists of 2 parts:
  1. Adding EEPROM and other related hardware to device-tree
  2. Adding U-Boot configuration options for EEPROM

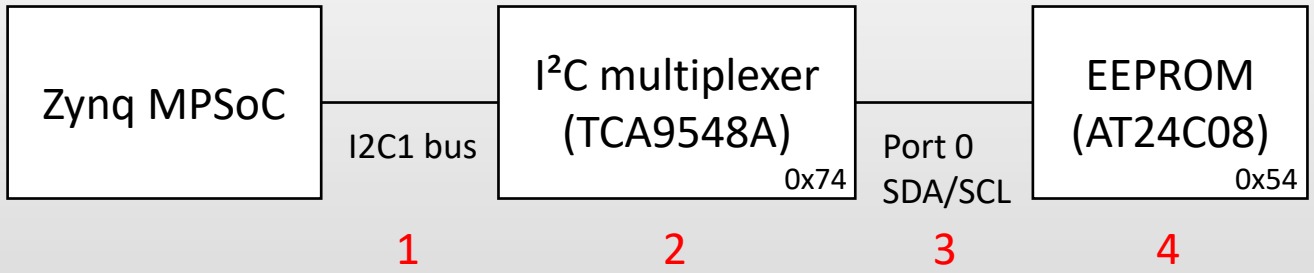
- 2 options if you want to configure U-Boot build:
  1. Manually
  2. Using the U-Boot menuconfig

Lets do it manually ☺



EEPROM connection on the ZCU102



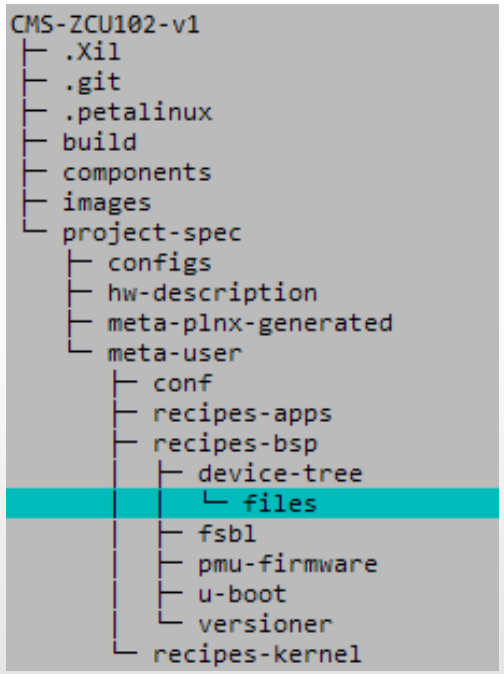


EEPROM connection to Zynq MPSoC on ZCU102:

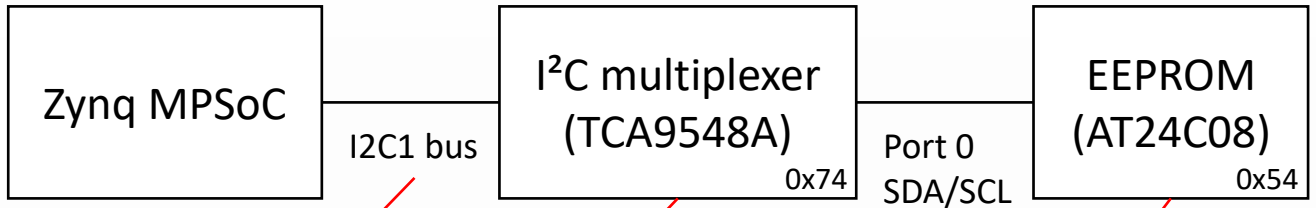
1	I2C1 bus	
2	I²C multiplexer (TCA9548A)	0x74
3	Port 0 SDA/SCL	
4	AT24C08 EEPROM	0x54

(EEPROM on ZCU102 evaluation board)

Add I²C hardware to device-tree



Edit the **system-user.dtsi** file




```
&i2c1 {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_i2c1_default>;

    i2c-mux@74 { /* u34 */
        compatible = "nxp,pca9548";
        #address-cells = <1>;
        #size-cells = <0>;
        reg = <0x74>;

        i2c@0 { /* i2c mw 74 0 1 */
            #address-cells = <1>;
            #size-cells = <0>;
            reg = <0>;
            eeprom: eeprom@54 { /* u23 */
                compatible = "atmel,24c08";
                reg = <0x54>;
};  };  };  };
```

Pin controller node:

```
&pinctrl0 {
    status = "okay";

    pinctrl_i2c1_default: i2c1-default {
                mux {
                    groups = "i2c1_4_grp";
                    function = "i2c1";
                };

                conf {
                    groups = "i2c1_4_grp";
                    bias-pull-up;
                    slew-rate = <SLEW_RATE_SLOW>;
                    io-standard = <IO_STANDARD_LVCMOS18>;
                };
    };
};
```

(pin controller documentation)

Add EEPROM to the device-tree

- EEPROM node contains definitions of the values stored in the EEPROM
- EEPROM must be linked using a phandle

EEPROM node:

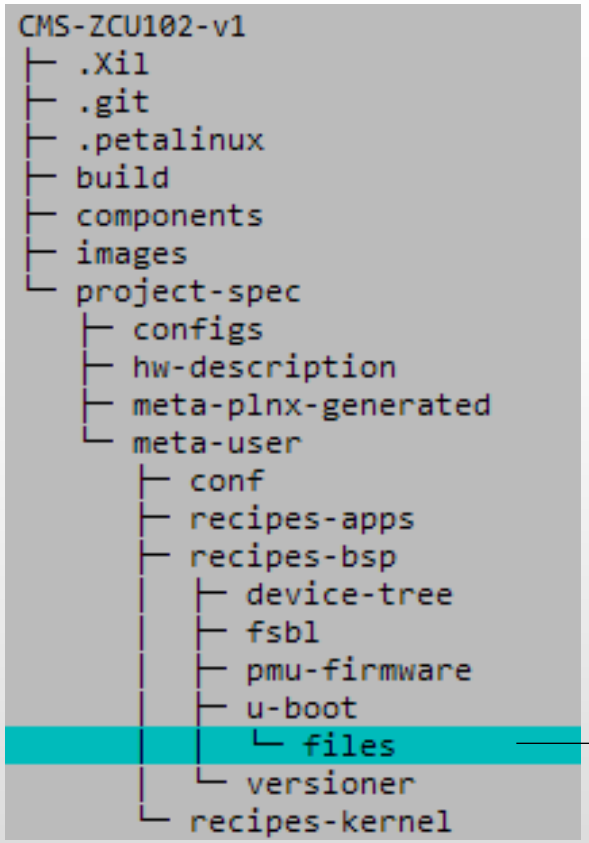
```
&eeprom {
    #address-cells = <1>;
    #size-cells = <1>;

    board_sn: board-sn@0 {
        reg = <0x0 0x14>;
    };

    eth_mac: eth-mac@20 {
        reg = <0x20 0x6>;
    };

    board_name: board-name@d0 {
        reg = <0xd0 0x6>;
    };

    board_revision: board-revision@e0 {
        reg = <0xe0 0x3>;
    };
};
```

Add EEPROM to chosen node:

```
/include/ "system-conf.dtsi"

/ {
    model = "CUSTOM BOARD CERN";
    compatible = "xlnx,zynqmp";

    chosen {
        xlnx,eeprom = &eeprom;
    };
};
```

Adding U-Boot configuration options for EEPROM

Last step of configuration



1. Create **`eeprom.cfg`** file
2. File contents:

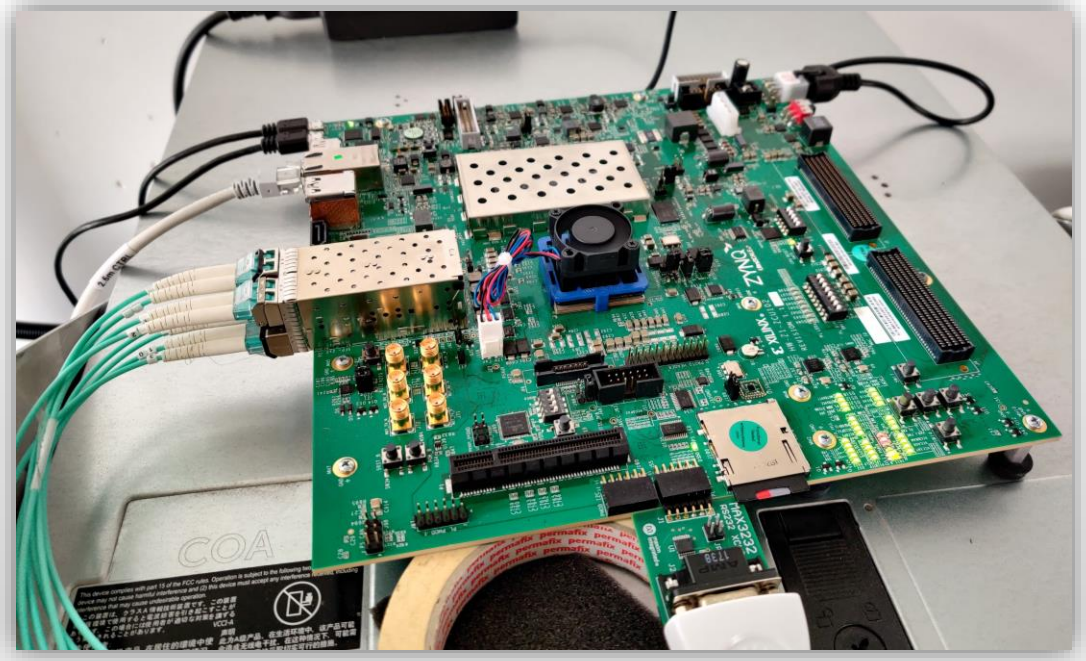
```
CONFIG_I2C_EEPROM=y
CONFIG_SYS_I2C_EEPROM_ADDR=0x54
CONFIG_SYS_I2C_EEPROM_ADDR_OVERFLOW=0x0
CONFIG_SYS_TEXT_BASE=0x10080000
CONFIG_ZYNQ_GEM_I2C_MAC_OFFSET=0x20
```

3. Edit **`u-boot_%.bbappend`** file
4. Add following line:

```
SRC_URI += "file://eeprom.cfg"
```

Example BSP requirements (board specific)

1. ~~PMU watchdog timers enabled~~
2. ~~FSBL needs to provide debug info~~
3. ~~Use the MAC-address stored in the EEPROM~~
4. Network boot using TFTP + NFS
5. Kernel support for crashkernel (dump-capture kernel)



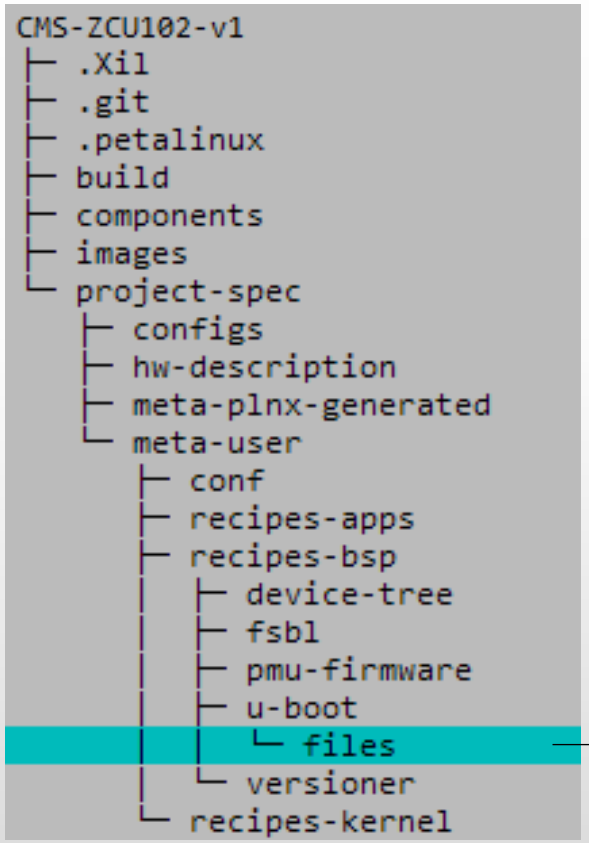
The ZCU102 board in our lab

Network boot configuration

Support for TFTP and NFS

- Consists of 3 parts:
  1. Adding support for **TFTP** in U-Boot → In U-Boot configuration
  2. Adding support for **NFS** through b device-tree → Through boot arguments in the device-tree
  3. Adding **networking hardware** to device-tree → By adding ethernet node and pinctrl node to the device-tree

U-Boot configuration for network boot



1. Find and edit **`platform-top.h`** file

2. Add the following lines:

```
#ifdef  CONFIG_SERVERIP
#undef  CONFIG_SERVERIP
#define CONFIG_SERVERIP
#endif

#ifdef  CONFIG_BOOTP_SERVERIP
#undef  CONFIG_BOOTP_SERVERIP
#endif

#ifdef  CONFIG_PREBOOT
#undef  CONFIG_PREBOOT
#define CONFIG_PREBOOT "echo U-BOOT for CERN CMS; setenv serverip;"
#endif
```

Modifying the device-tree for network boot

```
CMS-ZCU102-v1
├─ .Xil
├─ .git
├─ .petalinux
├─ build
├─ components
├─ images
└─ project-spec
   ├─ configs
   ├─ hw-description
   ├─ meta-plnx-generated
   └─ meta-user
      ├─ conf
      ├─ recipes-apps
      ├─ recipes-bsp
      │  ├─ device-tree
      │  │  └─ files
      │  ├─ fsbl
      │  ├─ pmu-firmware
      │  ├─ u-boot
      │  └─ versioner
      └─ recipes-kernel
```

Edit the **`system-user.dtsi`** file

Add boot arguments:

- NFS root filesystem
- IP from DHCP

```
/ {
    model = "CUSTOM BOARD CERN";
    compatible = "xlnx,zynqmp";

    chosen {
        xlnx,eeprom = &eeprom;
        bootargs = "earlycon console=ttyPS0,115200 clk_ignore_unused" \
                   "earlyprintk cpuidle.off=1 root=/dev/nfs ip=dhcp rw";
    };
};
```

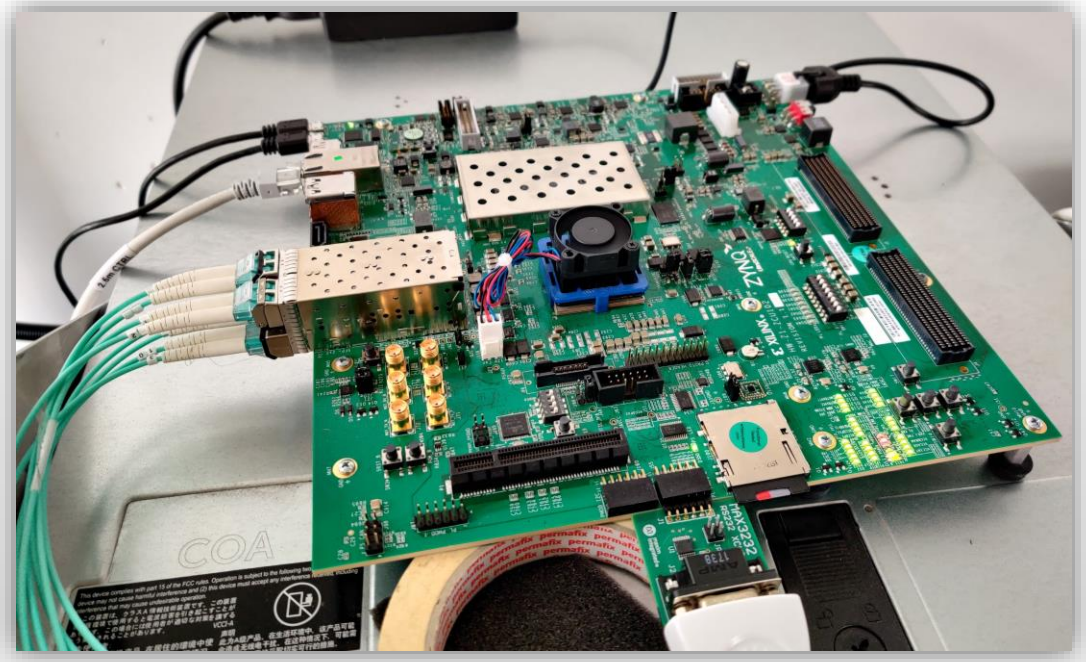
Add networking hardware:

- Ethernet node
- Pinctrl node (not shown)

```
&gem3 {
    phy-handle = <&phyc>;
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_gem3_default>;
    phyc: ethernet-phy@c {
        reg = <0xc>;
        ti,rx-internal-delay = <0x8>;
        ti,tx-internal-delay = <0xa>;
        ti,fifo-depth = <0x1>;
        ti,rxctrl-strap-worka;
    };
};
```

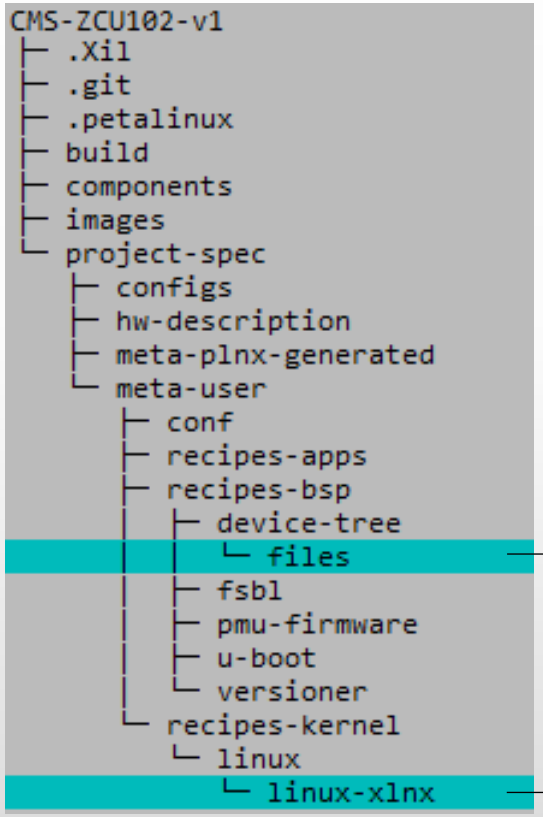
Example BSP requirements (board specific)

1. ~~PMU watchdog timers enabled~~
2. ~~FSBL needs to provide debug info~~
3. ~~Use the MAC-address stored in the EEPROM~~
4. ~~Network boot using TFTP + NFS~~
5. Kernel support for crashkernel (dump-capture kernel)



The ZCU102 board in our lab

Adding crashkernel support



1. Edit the **`system-user.dtsi`** file

```
/ {
    model = "CUSTOM BOARD CERN";
    compatible = "xlnx,zynqmp";

    chosen {
        xlnx,eeprom = &eeprom;
        bootargs = "earlycon console=ttyPS0,115200 clk_ignore_unused" \
                   "earlyprintk cpuidle.off=1 crashkernel=256M root=/dev/nfs ip=dhcp rw";
    };
};
```

2. Edit the **`devtool-fragment.cfg`** file
3. Add the following lines:

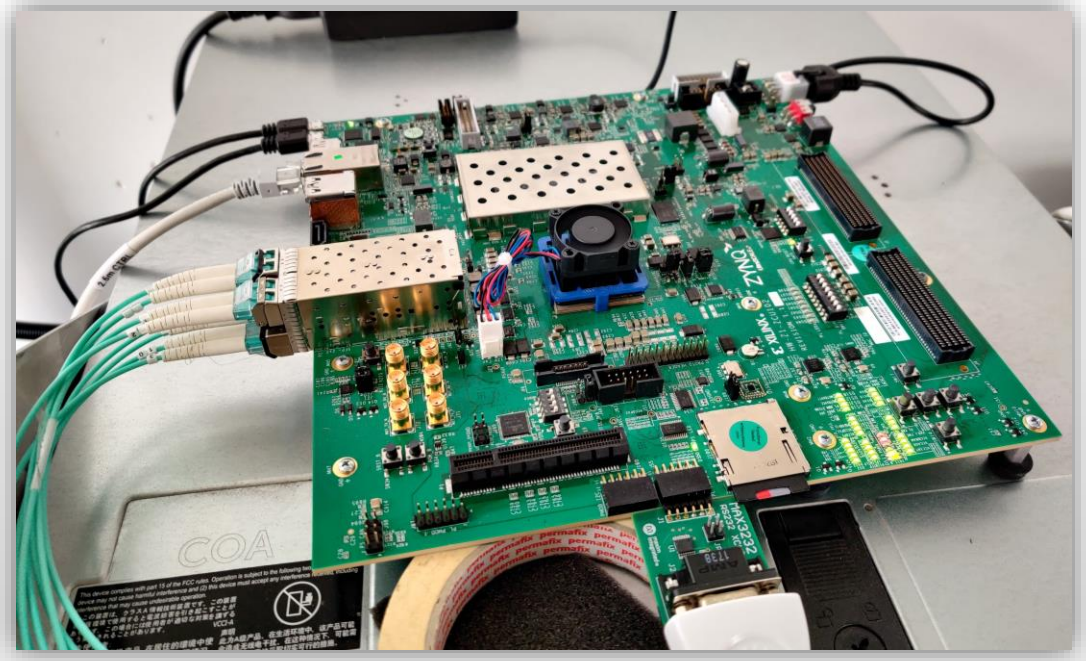
```
CONFIG_KEXEC=y
CONFIG_CRASH_DUMP=y
CONFIG_CRASH_CORE=y
CONFIG_KEXEC_CORE=y
CONFIG_PROC_KCORE=y
CONFIG_PROC_VMCORE=y
```

[Crashkernel GitLab]

[Crashkernel presentation]

Example BSP requirements (board specific)

1. ~~PMU watchdog timers enabled~~
2. ~~FSBL needs to provide debug info~~
3. ~~Use the MAC-address stored in the EEPROM~~
4. ~~Network boot using TFTP + NFS~~
5. ~~Kernel support for crashkernel (dump-capture kernel)~~



The ZCU102 board in our lab

Finally packaging your project into a BSP

Final steps!

1. Build the project:

```
$ petalinux-build
```

2. Package the project into a BSP:

```
$ petalinux-package --bsp -p <petalinux-project-path> --hwsource=<vivado-project-path> --output CMS-ZCU102-v1.BSP --force
```

3. (Optional) Test your new BSP by creating a new project

You have level up your PetaLinux game!!

Thank you for listening
Any questions?