AXI Ethernet Reference Designs

Jeff Johnson

Jun 02, 2021

USER GUIDE

This is the documentation for the AXI Ethernet reference designs for the Ethernet FMC.

CHAPTER

ONE

DESCRIPTION

In this reference design, each port of the Ethernet FMC is connected to an AXI Ethernet Subsystem IP which is connected to the system memory via an AXI DMA IP.

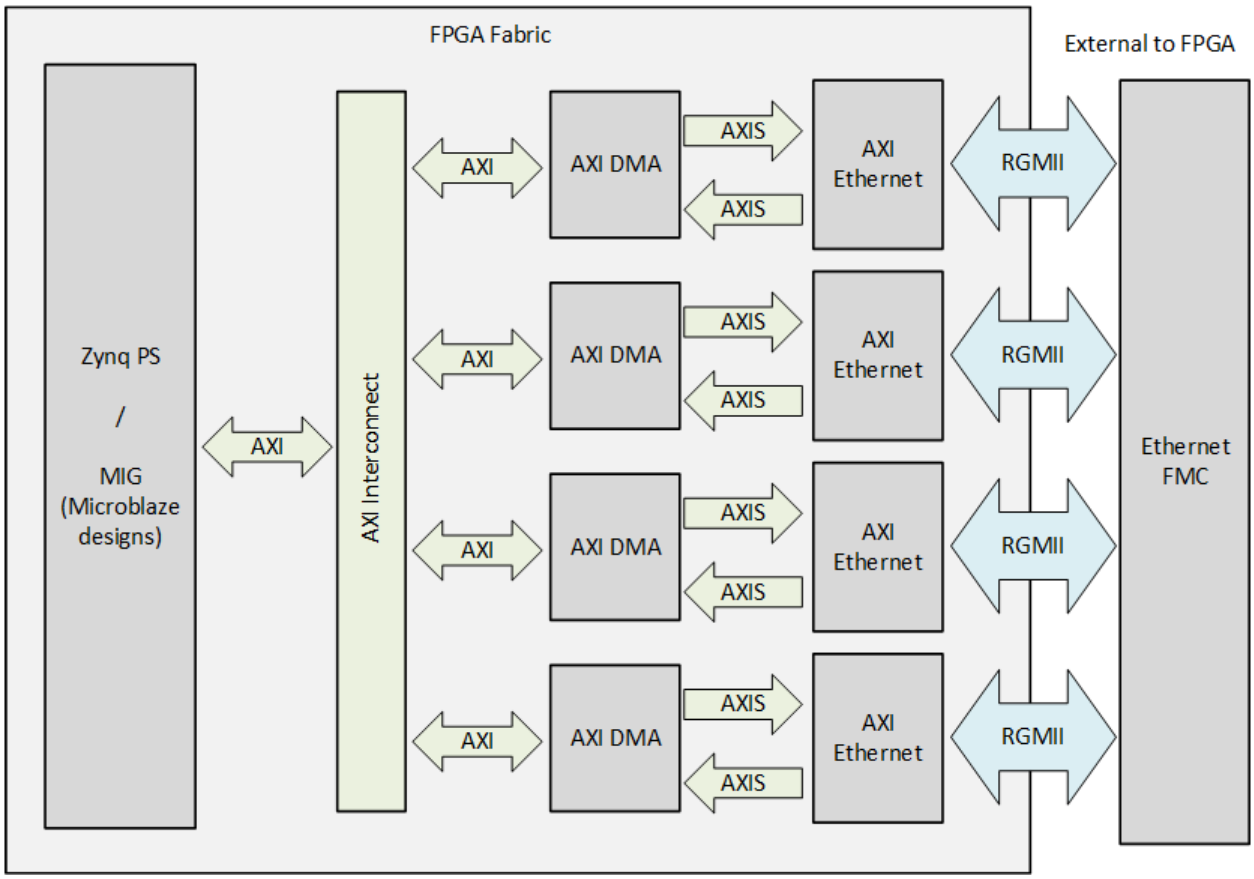

Fig. 1.1: AXI Ethernet design block diagram

CHAPTER

TWO

REQUIREMENTS

In order to test this design on hardware, you will need the following:

- Vivado 2020.2
- Vitis 2020.2
- PetaLinux Tools 2020.2
- Ethernet FMC
- Xilinx Soft TEMAC license

CHAPTER

THREE

SUPPORTED CARRIER BOARDS

3.1 List of supported boards

Carrier board	FMC connector
Zynq-7000 ZedBoard	LPC
Zynq-7000 MicroZed FMC Carrier with MicroZed 7Z020.	LPC
Zynq-7000 PicoZed FMC Carrier Card V2 with PicoZed 7015/20/30	LPC
Artix-7 AC701 Evaluation board	HPC
Kintex-7 KC705 Evaluation board	LPC and HPC
Kintex UltraScale KCU105 Evaluation board	LPC and HPC
Virtex-7 VC707 Evaluation board	HPC1 and HPC2
Virtex-7 VC709 Evaluation board	HPC
Zynq-7000 ZC702 Evaluation board	LPC1 and LPC2
Zynq-7000 ZC706 Evaluation board	LPC
Zynq UltraScale+ ZCU102 Evaluation board	HPC0 and HPC1 (HPC1 limited to 2 ports)
Zynq UltraScale+ UltraZed EV Carrier Card	HPC
Virtex Ultrascale+ VCU108 Evaluation board	HPC0 and HPC1
Virtex Ultrascale+ VCU118 Evaluation board	HPC1

3.2 Unlisted boards

If you need more information on whether the Ethernet FMC is compatible with a carrier that is not listed above, please first check the compatibility list. If the carrier is not listed there, please contact Opsero, provide us with the pinout of your carrier and we'll be happy to check compatibility and generate a Vivado constraints file for you.

3.3 Using 2x Ethernet FMCs for 8-ports

The only evaluation boards that can support two Ethernet FMCs simultaneously are:

- KC705 Evaluation board
- KCU105 Evaluation board
- ZC702 Evaluation board and
- VC707 Evaluation board.

This repository contains example designs for using 2 x Ethernet FMCs on the same carrier. They all use 8 Xilinx AXI Ethernet Subsystem IPs that are configured with DMAs, except for the ZC702 design, which is configured with FIFOs. The reason for this is a lack of FPGA resources as using 8 MACs configured with DMAs requires more resources than is contained in the Zynq device of that board.

These notes provide more details on 8-port support:

- The KC705 and VC707 each have two FMC connectors that support the Ethernet FMC (use kc705-lpc-hpc.xdc and vc707-hpc2-hpc1.xdc respectively).
- The KCU105 can support two Ethernet FMCs however the LPC only supports 3 ports so the dual design contains only 7 ports total.
- The ZC702 has two FMC connectors that can support the Ethernet FMC, however note that the Zynq device on this board has limited FPGA resources for supporting 8 x Xilinx AXI Ethernet IPs (ie. the MACs). The device has enough resources when the 8 MACs are configured with FIFOs, however there are insufficient resources to configure them with DMAs. Alternatively, you could use a MAC that requires less resources. (use zc702-lpc2-lpc1.xdc)
- The ZC706 has two FMC connectors, but only one (the LPC) can support the Ethernet FMC (see detail in board specific notes below).

3.4 Board specific notes

3.4.1 AC701

- The AC701's on-board Ethernet port is not connected in this design.
- This design includes a reset GPIO so that the MicroBlaze can reset itself from PetaLinux.

3.4.2 KC705

- The KC705's on-board Ethernet port is connected to AXI EthernetLite IP in these designs.
- This design includes a reset GPIO so that the MicroBlaze can reset itself from PetaLinux.

3.4.3 VC707 & VC709

- These boards can only support the 1.8V version Ethernet FMC. The device on these boards have only HP (high-performance) I/Os which do not support 2.5V levels.

3.4.4 ZC706

- Zynq-7000 ZC706 Evaluation board (HPC)
  - HPC connector: Pins LA18_CC and LA17_CC of the HPC connector are routed to non-clock-capable pins so they cannot properly receive the RGMII receive clocks for ports 2 and 3 of the Ethernet FMC. The constraints file zc706-hpc.xdc is provided for reference, however it will not pass compilation with the Xilinx tools due to this problem.

3.4.5 KCU105

- This board can only support the 1.8V version Ethernet FMC. The device on this board has only HP (high-performance) I/Os which do not support 2.5V levels.
- KCU105 board design for the LPC connector is configured for only 3 ports as there is a strange placement error which occurs when trying to build a design with 4 ports. The placement error has to do with IDELAYs and I have not reached a solution for this yet. There is no such problem with the HPC for this board.

3.4.6 ZCU102

- These designs support the ZCU102 Rev 1.0 and newer boards. Use a commit before 2016-02-13 for the older Rev-D board design. Note that the FMC pinouts differ between Rev 1.0 and Rev D: https://www.xilinx.com/support/answers/68050.html
- This board can only support the 1.8V version Ethernet FMC. The device on this board has only HP (high-performance) I/Os which do not support 2.5V levels.
- The HPC1 design only supports 2 ports due to the HPC1 pin assignment to the Zynq US+ (see constraints file for more details).

3.4.7 PicoZed

This repository contains a Vivado design for these PicoZed versions: 7Z020, 7Z015 and 7Z030. The main differences between the designs are described below:

- 7Z020: We use 4x AXI Ethernet IPs. The constraints file uses the 2.5V IO standards.
- 7Z015: We use 4x AXI Ethernet IPs. The constraints file uses the 2.5V IO standards.
- 7Z030: We use 4x AXI Ethernet IPs. The constraints file uses the 1.8V IO standards because this device has HP I/Os.

3.4.8 Installation of MicroZed, PicoZed and UltraZed board definition files

To use the projects for the MicroZed, PicoZed and UltraZed, you must first install the board definition files for those boards into your Vivado and Vitis installation.

The following folders contain the board definition files and can be found in this project repository at this location:

https://github.com/Avnet/bdf

- `microzed_7010`
- `microzed_7020`
- `picozed_7010_fmc2`
- `picozed_7015_fmc2`
- `picozed_7020_fmc2`
- `picozed_7030_fmc2`
- `ultrazed_7ev_cc`

Copy those folders and their contents into the `C:\Xilinx\Vivado\2020.2\data\boards\board_files` folder (this may be different on your machine, depending on your Vivado installation directory). You also need to make a copy into the Vitis installation at this location: `C:\Xilinx\Vitis\2020.2\data\boards\board_files`.

3.4.9 Microblaze design differences

The designs for AC701, KC705, VC707, VC709, KCU105, VCU108 & VCU118 all use the Microblaze soft processor. These designs have some specific differences when compared to the Zynq based designs:

- MIG - the MIG is required to exploit the DDR3/4 memory of the eval boards.
- AXI Timer - the lwIP echo server application requires a timer (Microblaze does not have one inherently).
- AXI UART16550 - the lwIP echo server application requires a UART for console output.

CHAPTER

FOUR

BUILD INSTRUCTIONS

4.1 Source code

The source code for the reference designs is managed on this Github repository: https://github.com/fpgadeveloper/ethernet-fmc-axi-eth

4.2 Windows users

1. Download the repo as a zip file and extract the files to a directory on your hard drive –OR– Git users: clone the repo to your hard drive
2. Open Windows Explorer, browse to the repo files on your hard drive.
3. In the Vivado directory, you will find multiple batch files (.bat). Double click on the batch file that is appropriate to your hardware, for example, double-click `build-zedboard.bat` if you are using the ZedBoard. This will generate a Vivado project for your hardware platform.
4. Run Vivado and open the project that was just created.
5. Click Generate bitstream.
6. When the bitstream is successfully generated, select *File->Export->Export Hardware*. In the window that opens, tick "Include bitstream" and "Local to project".
7. Return to Windows Explorer and browse to the Vitis directory in the repo.
8. Double click the `build-vitis.bat` batch file. The batch file will run the `build-vitis.tcl` script and build the Vitis workspace containing the hardware design and the software application.
9. Run Xilinx Vitis and select the workspace to be the Vitis directory of the repo.
10. Connect and power up the hardware.
11. Open a Putty terminal to view the UART output.
12. In Vitis, select *Xilinx Tools->Program FPGA*.
13. Right-click on the application and select *Run As->Launch on Hardware (Single Application Debug)*

4.3 Linux users

1. Download the repo as a zip file and extract the files to a directory on your hard drive –OR– Git users: clone the repo to your hard drive
2. Launch the Vivado GUI.
3. Open the Tcl console from the Vivado welcome page. In the console, `cd` to the repo files on your hard drive and into the Vivado subdirectory. For example: `cd /media/projects/ethernet-fmc-axi-eth/Vivado`.
4. In the Vivado subdirectory, you will find multiple Tcl files. To list them, type `exec ls {*}[glob *.tcl]`. Determine the Tcl script for the example project that you would like to generate (for example: `build-zedboard.tcl`), then `source` the script in the Tcl console: For example: `source build-zedboard.tcl`
5. Vivado will run the script and generate the project. When it's finished, click Generate bitstream.
6. When the bitstream is successfully generated, select *File->Export->Export Hardware*. In the window that opens, tick "Include bitstream" and "Local to project".
7. To build the Vitis workspace, open a Linux command terminal and `cd` to the Vitis directory in the repo.
8. The Vitis directory contains the `build-vitis.tcl` script that will build the Vitis workspace containing the hardware design and the software application. Run the build script by typing the following command: `<path-of-xilinx-vitis>/bin/xsct build-vitis.tcl`. Note that you must replace `<path-of-xilinx-vitis>` with the actual path to your Xilinx Vitis installation.
9. Run Xilinx Vitis and select the workspace to be the Vitis subdirectory of the repo.
10. Connect and power up the hardware.
11. Open a Putty terminal to view the UART output.
12. In Vitis, select *Xilinx Tools->Program FPGA*.
13. Right-click on the application and select *Run As->Launch on Hardware (Single Application Debug)*

CHAPTER

FIVE

STAND-ALONE LWIP ECHO SERVER

These reference designs can be used with the stand-alone lwIP echo server application template that is part of Vitis; however, some modifications are required. The lwIP library needs some modifications to be able to properly configure the Marvell PHYs (88E1510) that are on the Ethernet FMC. The `Vitis` directory of the source repository contains a script that can be used to setup a Vitis workspace containing the echo server application and the modified lwIP library.

The build script does the following:

1. Creates a Vitis workspace in the `Vitis` directory of the source repository.
2. Creates a subdirectory called `embeddedsw` to be used as a local software repository containing the modified lwIP library.
3. Copies the sources from the `EmbeddedSw` directory of the repository to the local software repository (`embeddedsw`), then copies any remaining/unmodified sources from the Vitis installation directory into the local software repository.
4. Generates a lwIP Echo Server example application for each exported Vivado design that is found in the `Vivado` directory. Most users will only have one exported Vivado design.

5.1 Building the Vitis workspace

To build the Vitis workspace and echo server application, you must first generate the Vivado project hardware design (the bitstream) and export the hardware. Once the bitstream is generated and exported, then you can build the Vitis workspace using the provided `Vitis/build-vitis.tcl` script.

5.1.1 Windows users

To build the Vitis workspace, Windows users can run the `build-vitis.bat` file which launches the Tcl script.

5.1.2 Linux users

Linux users must use the following commands to run the build script:

```
cd <path-to-repo>/Vitis
/<path-to-xilinx-tools>/Vitis/2020.2/bin/xsct build-vitis.tcl
```

5.2 Run the application

1. Open Xilinx Vitis.
2. Power up your hardware platform and ensure that the JTAG is connected properly.
3. In the Vitis Explorer panel, double-click on the System project that you want to run - this will reveal the applications contained in the project. The System project will have the postfix "_system".
4. Now click on the application that you want to run. It should have the postfix "_echo_server".
5. Select the option "Run Configurations" from the drop-down menu contained under the Run button on the toolbar (play symbol).
6. Double-click on "Single Application Debug" to create a run configuration for this application. Then click "Run".

The run configuration will first program the FPGA with the bitstream, then load and run the application. You can view the UART output of the application in a console window.

5.3 UART settings

To receive the UART output of this standalone application, you will need to connect the USB-UART of the development board to your PC and run a console program such as Putty. The following UART settings must be used:

- Microblaze designs: 9600 baud
- Zynq and ZynqMP designs: 115200 baud

5.4 IP address

By default, the echo server attempts to obtain an IP address from a DHCP server. This is useful if the echo server is connected to a network. Once the IP address is obtained, it is printed out in the UART console output.

If instead the echo server is connected directly to a PC, the DHCP attempt will fail and the echo server's IP address will default to 192.168.1.10. To be able to communicate with the echo server from the PC, the PC should be configured with a fixed IP address on the same subnet, for example: 192.168.1.20.

5.5 Change the targetted port

The echo server example design currently can only target one Ethernet port at a time. Selection of the Ethernet port can be changed by modifying the defines contained in the `platform_config.h` file in the application sources. Set `PLATFORM_EMAC_BASEADDR` to one of the following values:

- Ethernet FMC Port 0: `XPAR_AXIETHERNET_0_BASEADDR`
- Ethernet FMC Port 1: `XPAR_AXIETHERNET_1_BASEADDR`
- Ethernet FMC Port 2: `XPAR_AXIETHERNET_2_BASEADDR`
- Ethernet FMC Port 3: `XPAR_AXIETHERNET_3_BASEADDR`

5.6 Example usage

5.6.1 Ping the port

The echo server can be "pinged" from a connected PC, or if connected to a network, from another device on the network. The UART console output will tell you what the IP address of the echo server is. To ping the echo server, use the `ping` command from a command console of a PC that is connected to the echo server (either directly or via network).

Example command: `ping 192.168.1.10`

5.6.2 Connect with telnet

We can also connect to the echo server using telnet and confirm that it is sending back (echoing) the data that we are sending it. From the command prompt of a PC on the same network as the echo server, run the following command:

Example command: `telnet 192.168.1.10 7`

The first argument of the telnet command specifies the IP address of the device to connect to (in our case the echo server). The last argument in the command specifies the port number, which should be 7 for the echo server.

In the blank screen that opens after running the command, you can type letters and they will be sent to the echo server and be echoed back.

CHAPTER

SIX

PETALINUX

PetaLinux can be built for these reference designs by using the script in the `PetaLinux` directory of the repository.

6.1 How to build

6.1.1 Requirements

- Windows or Linux PC with Vivado installed
- Linux PC or virtual machine with PetaLinux installed

6.1.2 Instructions

1. First generate the Vivado project hardware design(s) (the bitstream) and export the design(s).
2. Launch PetaLinux by sourcing the `settings.sh` bash script, eg: `source <path-to-installed-petalinux>/settings.sh`
3. Build the PetaLinux project(s) by executing the `build-petalinux` script in Linux.

The script will generate a separate PetaLinux project for all of the generated and exported Vivado projects that it finds in the Vivado directory of this repo.

6.2 UNIX line endings

The scripts and files in the PetaLinux directory of this repository must have UNIX line endings when they are executed or used under Linux. The best way to ensure UNIX line endings, is to clone the repo directly onto your Linux machine. If instead you have copied the repo from a Windows machine, the files will have DOS line endings and you must use the `dos2unix` tool to convert the line endings for UNIX.

1. Copy the cloned repository from your Windows machine to your Linux machine.
2. Use the `cd` command to navigate to the copied repository on your Linux machine.
3. Type `find . -type f -exec dos2unix --keepdate {} +` to convert all of the files to the Unix format.

6.3 How the script works

The PetaLinux directory contains a `build-petalinux` shell script which can be run in Linux to automatically generate a PetaLinux project for each of the generated/exported Vivado projects in the Vivado directory.

When executed, the build script searches the Vivado directory for all projects containing a `.xsa` exported hardware design file. Then for every exported project, the script does the following:

1. Verifies that the `.bit` file exists.
2. Determines the CPU type: Zynq, ZynqMP or Microblaze. It does this by reading the Vivado project file.
3. Creates a PetaLinux project, referencing the exported hardware design (.xsa).
4. Copies the relevant configuration files from the `src` directory into the created PetaLinux project.
5. Builds the PetaLinux project.
6. Generates a BOOT.BIN, boot.scr and image.ub file for the Zynq and ZynqMP projects.

6.4 Launch PetaLinux on hardware

6.4.1 Via JTAG

To launch the PetaLinux project on hardware via JTAG, connect and power up your hardware and then use the following commands in a Linux command terminal:

1. Change current directory to the PetaLinux project directory: `cd <petalinux-project-dir>`
2. Download bitstream to the FPGA: `petalinux-boot --jtag --fpga` Note that you don't have to specify the bitstream because this command will use the one that it finds in the `./images/linux` directory.
3. Download the PetaLinux kernel to the FPGA: `petalinux-boot --jtag --kernel`

6.4.2 Via SD card (Zynq/ZynqMP)

To launch the PetaLinux project on hardware via SD card, copy the following files to the root of the SD card:

- `/<petalinux-project>/images/linux/BOOT.bin`
- `/<petalinux-project>/images/linux/image.ub`

Then connect and power your hardware.

6.5 Configuration files

The configuration files contained in the *src* directory include:

- Device tree
- Rootfs configuration (to include ethtool)
- Interface initializations (sets eth0-3 interfaces to DHCP)
- Kernel configuration

6.6 Port configurations

All designs will try to automatically configure the eth0 device on boot, so it can be useful to connect the eth0 device to a DHCP router before the hardware is powered-up. Note that on Zynq and ZynqMP designs, the eth0 device is connected to the development board's Ethernet port and not the Ethernet FMC.

6.6.1 AC701, KC705

- eth0: Ethernet port of the dev board
- eth1: Ethernet FMC Port 0
- eth2: Ethernet FMC Port 1
- eth3: Ethernet FMC Port 2
- eth4: Ethernet FMC Port 3

6.6.2 KCU105 HPC, VC707, VC709

- eth0: Ethernet FMC Port 0
- eth1: Ethernet FMC Port 1
- eth2: Ethernet FMC Port 2
- eth3: Ethernet FMC Port 3

6.6.3 KCU105 LPC

- eth0: Ethernet FMC Port 0
- eth1: Ethernet FMC Port 1
- eth2: Ethernet FMC Port 3

Ethernet FMC Port 2 is unusable in this design.

6.6.4 MicroZed, PicoZed, ZC702, ZC706, ZedBoard, ZCU102, UltraZed-EV

- eth0: GEM0 to Ethernet port of the dev board
- eth1: Ethernet FMC Port 0
- eth2: Ethernet FMC Port 1
- eth3: Ethernet FMC Port 2
- eth4: Ethernet FMC Port 3

6.6.5 KCU105 Dual design

- eth0: HPC Ethernet FMC Port 0 (AXI Ethernet)
- eth1: HPC Ethernet FMC Port 1 (AXI Ethernet)
- eth2: HPC Ethernet FMC Port 2 (AXI Ethernet)
- eth3: HPC Ethernet FMC Port 3 (AXI Ethernet)
- eth4: LPC Ethernet FMC Port 0 (AXI Ethernet)
- eth5: LPC Ethernet FMC Port 1 (AXI Ethernet)
- eth6: LPC Ethernet FMC Port 3 (AXI Ethernet)

Ethernet FMC Port 2 on the LPC is unusable in this design.

6.6.6 VC707 Dual design

- eth0: HPC2 Ethernet FMC Port 0 (AXI Ethernet)
- eth1: HPC2 Ethernet FMC Port 1 (AXI Ethernet)
- eth2: HPC2 Ethernet FMC Port 2 (AXI Ethernet)
- eth3: HPC2 Ethernet FMC Port 3 (AXI Ethernet)
- eth4: HPC1 Ethernet FMC Port 0 (AXI Ethernet)
- eth5: HPC1 Ethernet FMC Port 1 (AXI Ethernet)
- eth6: HPC1 Ethernet FMC Port 2 (AXI Ethernet)
- eth7: HPC1 Ethernet FMC Port 3 (AXI Ethernet)

6.6.7 ZC702 Dual design

Note that the ZC702 dual design will not produce a working PetaLinux project because it's Ethernet MACs are connected to FIFOs and not AXI DMAs. We are working on a solution to this.

6.7 Example Usage

6.7.1 Enable port

This example will bring up a port.

```
root@axieth:~# ifconfig eth1 up
[  228.274146] xilinx_axienet a0000000.ethernet eth1: Link is Up - 1Gbps/Full - flow␣
↪control off
[  228.282753] IPv6: ADDRCONF(NETDEV_CHANGE): eth1: link becomes ready
```

6.7.2 Enable port with fixed IP address

This example sets a fixed IP address to a port.

```
root@axieth:~# ifconfig eth1 192.168.2.30 up
[  390.080498] net eth1: Promiscuous mode disabled.
[  390.085406] net eth1: Promiscuous mode disabled.
[  390.091089] xilinx_axienet a0000000.ethernet eth1: Link is Down
[  394.175238] xilinx_axienet a0000000.ethernet eth1: Link is Up - 1Gbps/Full - flow␣
↪control off
[  394.183769] IPv6: ADDRCONF(NETDEV_CHANGE): eth1: link becomes ready
```

6.7.3 Enable port using DHCP

This example enables a port and obtains an IP address for the port via DHCP. Note that the port must be connected to a DHCP enabled router.

```
root@axieth:~# udhcpc -i eth1
udhcpc: started, v1.31.0
[   68.814013] xilinx_axienet a0000000.ethernet eth1: Link is Up - 1Gbps/Full - flow␣
↪control off
[   68.822670] IPv6: ADDRCONF(NETDEV_CHANGE): eth1: link becomes ready
udhcpc: sending discover
udhcpc: sending select for 192.168.2.23
udhcpc: lease of 192.168.2.23 obtained, lease time 259200
/etc/udhcpc.d/50default: Adding DNS 192.168.2.1
```

6.7.4 Check port status

In this example, we use the `ifconfig` command with no arguments to check the port status. The first interface (eth0) shown below is connected to the on-board Ethernet port and it has not been enabled, whereas the second interface (eth1) is connected to the Ethernet FMC port 0 and it has been enabled and configured with IP address 192.168.2.30.

```
root@axieth:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0A:35:00:22:01
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:30

eth1      Link encap:Ethernet  HWaddr 00:0A:35:00:01:22
          inet addr:192.168.2.30  Bcast:192.168.2.255  Mask:255.255.255.0
          inet6 addr: fe80::20a:35ff:fe00:122/64 Scope:Link
          UP BROADCAST RUNNING  MTU:1500  Metric:1
          RX packets:38 errors:0 dropped:0 overruns:0 frame:0
          TX packets:26 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:6033 (5.8 KiB)  TX bytes:3302 (3.2 KiB)
```

(continues on next page)

(continued from previous page)

```
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

We can also use `ethtool` to check the port status as follows.

```
root@axieth:~# ethtool eth1
Settings for eth1:
        Supported ports: [ TP MII FIBRE ]
        Supported link modes:   10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Half 1000baseT/Full
        Supported pause frame use: Symmetric Receive-only
        Supports auto-negotiation: Yes
        Supported FEC modes: Not reported
        Advertised link modes:  10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Half 1000baseT/Full
        Advertised pause frame use: No
        Advertised auto-negotiation: Yes
        Advertised FEC modes: Not reported
        Link partner advertised link modes:  10baseT/Half 10baseT/Full
                                             100baseT/Half 100baseT/Full
                                             1000baseT/Full
        Link partner advertised pause frame use: No
        Link partner advertised auto-negotiation: Yes
        Link partner advertised FEC modes: Not reported
        Speed: 1000Mb/s
        Duplex: Full
        Port: MII
        PHYAD: 0
        Transceiver: internal
        Auto-negotiation: on
        Link detected: yes
```

6.7.5 Ping link partner using specific port

In this example we ping the link partner at IP address 192.168.2.10 from interface eth1.

```
root@axieth:~# ping -I eth1 192.168.2.10
PING 192.168.2.10 (192.168.2.10): 56 data bytes
64 bytes from 192.168.2.10: seq=0 ttl=128 time=0.545 ms
64 bytes from 192.168.2.10: seq=1 ttl=128 time=0.455 ms
64 bytes from 192.168.2.10: seq=2 ttl=128 time=0.380 ms
64 bytes from 192.168.2.10: seq=3 ttl=128 time=0.356 ms
```

6.8 Known Issues

6.8.1 AXI Ethernet issue on Zynq designs 2020.2

There is an issue in the PetaLinux 2020.2 release that affects the **AXI Ethernet** connected ports on **Zynq** based designs. On these ports, it seems to be necessary to use the following procedure to bring up a port. Note that the interface and IP address were chosen as examples, but this procedure applies to all AXI Ethernet connected ports (eth0, eth1, eth2 and eth3) on the Zynq based designs (MicroZed, PicoZed, ZedBoard, ZC702 and ZC706).

```
ifconfig eth0 up
ifconfig eth0 down
ifconfig eth0 192.168.1.10 up
```

In earlier releases, it was only necessary to run the last command to bring up a port. This issue does not affect the Zynq Ultrascale+ based designs. This issue does not seem to affect the stand-alone echo server operation. We have not yet determined the cause of this issue but if you have any information, please let us know.

CHAPTER

SEVEN

UPDATING THE PROJECTS

This section contains instructions for updating the reference designs. It is intended as a guide for anyone wanting to attempt updating the designs for a tools release that we do not yet support. Note that the update process is not always straight-forward and sometimes requires dealing with new issues or significant changes to the functionality of the tools and/or specific IP. Unfortunately, we cannot always provide support if you have trouble updating the designs.

7.1 Vivado projects

1. Download and install the Vivado release that you intend to use.
2. If you are using one of the following boards, you will have to download and install the latest board files for that target platform. Other boards are already built into Vivado and require no extra installation.
   - MicroZed board files can be downloaded here
   - PicoZed board files can be downloaded here
   - UltraZed EV board files can be downloaded here
3. In a text editor, open the `Vivado/build-<target>.bat` file for the design that you wish to update, and perform the following changes:
   - Update the tools version number to the one you are using (eg. 2020.2)
4. In a text editor, open the `Vivado/build-<target>.tcl` file for the design that you wish to update, and perform the following changes:
   - Update the `version_required` variable value to the tools version number that you are using.
   - Update the year in all references to `Vivado Synthesis <year>` to the tools version number that you are using. For example, if you are using tools version 2020.2, then the `<year>` should be 2020.
   - Update the year in all references to `Vivado Implementation <year>` to the tools version number that you are using. For example, if you are using tools version 2020.2, then the `<year>` should be 2020.
   - If the version of the board files for your target platform has changed, update the `board_part` parameter value to the new version.

After following the above steps, you can now run the build script. If there were no significant changes to the tools and/or IP, the build script should succeed and you will be able to open and generate a bitstream for the Vivado project.

7.2 PetaLinux

The main procedure for updating the PetaLinux project is to update the BSP for the target platform. The BSP files for each supported target platform are contained in the `PetaLinux/src` directory. For example, the BSP files for the ZedBoard are located in `PetaLinux/src/zedboard`.

1. Download and install the PetaLinux release that you intend to use.
2. Download and install the BSP for the target platform for the release that you intend to use.
   - For AC701, KC705, KCU105, VCU118, ZC702, ZC706, ZCU102 and ZedBoard, download the BSP from the PetaLinux download page
   - For MicroZed and PicoZed, download the BSP for the **ZedBoard** from the PetaLinux download page
   - For the UltraZed EV, download the BSP for the **ZCU102** from the PetaLinux download page
   - For the VC707 and VC709, download the BSP for the **KC705** from the PetaLinux download page
   - For the VCU108, download the BSP for the **VCU118** from the PetaLinux download page
3. Update the BSP files for the target platform in the `PetaLinux/src/<platform>` directory. These are the specific directories to update:
   - `<platform>/project-spec/configs/*`
   - `<platform>/project-spec/meta-user/*`

   The simple way to update the files is to delete those in the repository and copy in those from the BSP that you just downloaded.
4. Apply the required modifications to the updated BSP files. The modifications are described for each target platform in the following sections.

7.2.1 Change project name

This BSP modification applies to all target platforms.

1. Append the following lines to `project-spec/configs/config`:

```
# Set project name
CONFIG_SUBSYSTEM_HOSTNAME="axieth"
CONFIG_SUBSYSTEM_PRODUCT="axieth"
```

Note that this will set the project name to "axieth" but you can use a more descriptive name, for example one that includes the target platform name and the tools version.

7.2.2 Add tools to root filesystem

This BSP modification applies to all target platforms.

1. Append the following lines to `project-spec/configs/rootfs_config`:

```
# Useful tools for Ethernet FMC
CONFIG_ethtool=y
CONFIG_ethtool-dev=y
CONFIG_ethtool-dbg=y
CONFIG_iperf3=y
```

2. Append the following lines to `project-spec/meta-user/conf/user-rootfsconfig`:

```
CONFIG_iperf3
CONFIG_ethtool
```

7.2.3 Include port config in device tree

This BSP modification applies to all target platforms.

1. Append the following line after `/include/ "system-conf.dtsi"` in `project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi`:

```
/include/ "port-config.dtsi"
```

2. Append the following line after `SRC_URI += "file://system-user.dtsi"` in `project-spec/meta-user/recipes-bsp/device-tree/device-tree.bbappend`:

```
SRC_URI += "file://port-config.dtsi"
```

7.2.4 Add kernel configs

This BSP modification applies to all target platforms.

1. Add the following lines to the top of file `project-spec/meta-user/recipes-kernel/linux/linux-xlnx/bsp.cfg`:

```
# Required by all designs
CONFIG_XILINX_GMII2RGMII=y
CONFIG_MVMDIO=y
CONFIG_MARVELL_PHY=y

# Required by BSP
```

7.2.5 Kernel configs for ZynqMP designs

This BSP modification must be applied to all ZynqMP designs (ie. ZCU102 and UltraZed EV) in addition to the previous one.

1. Add the following lines to the top of file `project-spec/meta-user/recipes-kernel/linux/linux-xlnx/bsp.cfg`:

```
# All zynqMP designs need these kernel configs for AXI Ethernet designs
CONFIG_XILINX_DMA_ENGINES=y
CONFIG_XILINX_DPDMA=y
CONFIG_XILINX_ZYNQMP_DMA=y
```

7.2.6 Mods for AC701

These modifications are specific to the AC701 BSP.

1. Append the following lines to `project-spec/configs/config`:

```
# Use lite template
CONFIG_SUBSYSTEM_MACHINE_NAME="ac701-lite"
```

2. Append the following lines to file `project-spec/meta-user/recipes-bsp/u-boot/files/platform-top.h`:

```
/* BOOTCOMMAND */
#define CONFIG_USE_BOOTCOMMAND 1
#define CONFIG_BOOTCOMMAND    "sf probe 0 && sf read ${netstartaddr} ${kernelstart} $
↪{kernelsize} && bootm ${netstartaddr}"

/* Extra U-Boot Env settings */
#define CONFIG_EXTRA_ENV_SETTINGS \
  SERIAL_MULTI \
  CONSOLE_ARG \
  ESERIAL0 \
  "nc=setenv stdout nc;setenv stdin nc;\0" \
  "ethaddr=00:0a:35:00:22:01\0" \
  "autoload=no\0" \
  "sdbootdev=0\0" \
  "clobstart=0x80000000\0" \
  "netstart=0x80000000\0" \
  "dtbnetstart=0x81e00000\0" \
  "netstartaddr=0x81000000\0"  "loadaddr=0x80000000\0" \
  "initrd_high=0x0\0" \
  "bootsize=0x180000\0" \
  "bootstart=0xa00000\0" \
  "boot_img=u-boot-s.bin\0" \
  "load_boot=tftpboot ${clobstart} ${boot_img}\0" \
  "update_boot=setenv img boot; setenv psize ${bootsize}; setenv installcmd \"install_
↪boot\"; run load_boot test_img; setenv img; setenv psize; setenv installcmd\0" \
  "install_boot=sf probe 0 && sf erase ${bootstart} ${bootsize} && " \
    "sf write ${clobstart} ${bootstart} ${filesize}\0" \
  "bootenvsize=0x40000\0" \
  "bootenvstart=0xb80000\0" \
  "eraseenv=sf probe 0 && sf erase ${bootenvstart} ${bootenvsize}\0" \
  "kernelsize=0xc00000\0" \
  "kernelstart=0xbc0000\0" \
  "kernel_img=image.ub\0" \
  "load_kernel=tftpboot ${clobstart} ${kernel_img}\0" \
  "update_kernel=setenv img kernel; setenv psize ${kernelsize}; setenv installcmd \
↪"install_kernel\"; run load_kernel test_crc; setenv img; setenv psize; setenv␣
↪installcmd\0" \
  "install_kernel=sf probe 0 && sf erase ${kernelstart} ${kernelsize} && " \
    "sf write ${clobstart} ${kernelstart} ${filesize}\0" \
  "cp_kernel2ram=sf probe 0 && sf read ${netstart} ${kernelstart} ${kernelsize}\0" \
  "fpgasize=0xa00000\0" \
  "fpgastart=0x0\0" \
```

(continues on next page)

(continued from previous page)

```
  "fpga_img=system.bit.bin\0" \
  "load_fpga=tftpboot ${clobstart} ${fpga_img}\0" \
  "update_fpga=setenv img fpga; setenv psize ${fpgasize}; setenv installcmd \"install_
↪fpga\"; run load_fpga test_img; setenv img; setenv psize; setenv installcmd\0" \
  "install_fpga=sf probe 0 && sf erase ${fpgastart} ${fpgasize} && " \
    "sf write ${clobstart} ${fpgastart} ${filesize}\0" \
  "fault=echo ${img} image size is greater than allocated place - partition ${img} is␣
↪NOT UPDATED\0" \
  "test_crc=if imi ${clobstart}; then run test_img; else echo ${img} Bad CRC - ${img} is␣
↪NOT UPDATED; fi\0" \
  "test_img=setenv var \"if test ${filesize} -gt ${psize}\\; then run fault\\; else run $
↪{installcmd}\\; fi\"; run var; setenv var\0" \
  "netboot=tftpboot ${netstartaddr} ${kernel_img} && bootm\0" \
  "default_bootcmd=bootcmd\0" \
""
```

7.2.7 Mods for KC705

These modifications are specific to the KC705 BSP.

1. Append the following lines to `project-spec/configs/config`:

```
# Use lite template
CONFIG_SUBSYSTEM_MACHINE_NAME="kc705-lite"
```

2. Append the following lines to file `project-spec/meta-user/recipes-bsp/u-boot/files/platform-top.h`:

```
/* BOOTCOMMAND */
#define CONFIG_USE_BOOTCOMMAND 1
#define CONFIG_BOOTCOMMAND    "cp.b ${kernelstart} ${netstartaddr} ${kernelsize} &&␣
↪bootm ${netstartaddr}"

/* Extra U-Boot Env settings */
#define CONFIG_EXTRA_ENV_SETTINGS \
  SERIAL_MULTI \
  CONSOLE_ARG \
  ESERIAL0 \
  "nc=setenv stdout nc;setenv stdin nc;\0" \
  "ethaddr=00:0a:35:00:22:01\0" \
  "autoload=no\0" \
  "sdbootdev=0\0" \
  "clobstart=0x80000000\0" \
  "netstart=0x80000000\0" \
  "dtbnetstart=0x81e00000\0" \
  "netstartaddr=0x81000000\0"  "loadaddr=0x80000000\0" \
  "initrd_high=0x0\0" \
  "bootsize=0x180000\0" \
  "bootstart=0x60b00000\0" \
  "boot_img=u-boot-s.bin\0" \
  "load_boot=tftpboot ${clobstart} ${boot_img}\0" \
```

(continues on next page)

(continued from previous page)

```
  "update_boot=setenv img boot; setenv psize ${bootsize}; setenv installcmd \"install_
↪boot\"; run load_boot test_img; setenv img; setenv psize; setenv installcmd\0" \
  "install_boot=protect off ${bootstart} +${bootsize} && erase ${bootstart} +${bootsize}␣
↪&& "  "cp.b ${clobstart} ${bootstart} ${filesize}\0" \
  "bootenvsize=0x20000\0" \
  "bootenvstart=0x60c80000\0" \
  "eraseenv=protect off ${bootenvstart} +${bootenvsize} && erase ${bootenvstart} +$
↪{bootenvsize}\0" \
  "kernelsize=0xc00000\0" \
  "kernelstart=0x60ca0000\0" \
  "kernel_img=image.ub\0" \
  "load_kernel=tftpboot ${clobstart} ${kernel_img}\0" \
  "update_kernel=setenv img kernel; setenv psize ${kernelsize}; setenv installcmd \
↪"install_kernel\"; run load_kernel test_crc; setenv img; setenv psize; setenv␣
↪installcmd\0" \
  "install_kernel=protect off ${kernelstart} +${kernelsize} && erase ${kernelstart} +$
↪{kernelsize} && "  "cp.b ${clobstart} ${kernelstart} ${filesize}\0" \
  "cp_kernel2ram=cp.b ${kernelstart} ${netstart} ${kernelsize}\0" \
  "fpgasize=0xb00000\0" \
  "fpgastart=0x60000000\0" \
  "fpga_img=system.bit.bin\0" \
  "load_fpga=tftpboot ${clobstart} ${fpga_img}\0" \
  "update_fpga=setenv img fpga; setenv psize ${fpgasize}; setenv installcmd \"install_
↪fpga\"; run load_fpga test_img; setenv img; setenv psize; setenv installcmd\0" \
  "install_fpga=protect off ${fpgastart} +${fpgasize} && erase ${fpgastart} +${fpgasize}␣
↪&& "  "cp.b ${clobstart} ${fpgastart} ${filesize}\0" \
  "fault=echo ${img} image size is greater than allocated place - partition ${img} is␣
↪NOT UPDATED\0" \
  "test_crc=if imi ${clobstart}; then run test_img; else echo ${img} Bad CRC - ${img} is␣
↪NOT UPDATED; fi\0" \
  "test_img=setenv var \"if test ${filesize} -gt ${psize}\\; then run fault\\; else run $
↪{installcmd}\\; fi\"; run var; setenv var\0" \
  "netboot=tftpboot ${netstartaddr} ${kernel_img} && bootm\0" \
  "default_bootcmd=bootcmd\0" \
""
```

7.2.8 Mods for KCU105

These modifications are specific to the KCU105 BSP.

1. Append the following lines to `project-spec/configs/config`:

```
# Use general template
CONFIG_SUBSYSTEM_MACHINE_NAME="template"
```

2. Append the following lines to file `project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi`:

```
&iic_main {
  #address-cells = <1>;
  #size-cells = <0>;
```

(continues on next page)

(continued from previous page)

```
  i2c-mux@75 {
    compatible = "nxp,pca9544";
    #address-cells = <1>;
    #size-cells = <0>;
    reg = <0x75>;
    i2c@3 {
      #address-cells = <1>;
      #size-cells = <0>;
      reg = <3>;
      eeprom@54 {
        compatible = "atmel,24c08";
        reg = <0x54>;
      };
    };
  };
};
```

3. Append the following lines to file `project-spec/meta-user/recipes-bsp/u-boot/files/platform-top.h`:

```
// Boot from QSPI flash
#define CONFIG_USE_BOOTCOMMAND 1
#define CONFIG_BOOTCOMMAND    "sf probe 0 && sf read ${netstartaddr} ${kernelstart} $
↪{kernelsize} && bootm ${netstartaddr}"

/* Extra U-Boot Env settings */
#define CONFIG_EXTRA_ENV_SETTINGS \
  SERIAL_MULTI \
  CONSOLE_ARG \
  ESERIAL0 \
  "nc=setenv stdout nc;setenv stdin nc;\0" \
  "ethaddr=00:0a:35:00:22:01\0" \
  "autoload=no\0" \
  "sdbootdev=0\0" \
  "clobstart=0x80000000\0" \
  "netstart=0x80000000\0" \
  "dtbnetstart=0x81e00000\0" \
  "netstartaddr=0x81000000\0"  "loadaddr=0x80000000\0" \
  "initrd_high=0x0\0" \
  "bootsize=0x180000\0" \
  "bootstart=0x1000000\0" \
  "boot_img=u-boot-s.bin\0" \
  "load_boot=tftpboot ${clobstart} ${boot_img}\0" \
  "update_boot=setenv img boot; setenv psize ${bootsize}; setenv installcmd \"install_
↪boot\"; run load_boot test_img; setenv img; setenv psize; setenv installcmd\0" \
  "install_boot=sf probe 0 && sf erase ${bootstart} ${bootsize} && " \
    "sf write ${clobstart} ${bootstart} ${filesize}\0" \
  "bootenvsize=0x40000\0" \
  "bootenvstart=0x1180000\0" \
  "eraseenv=sf probe 0 && sf erase ${bootenvstart} ${bootenvsize}\0" \
  "kernelsize=0xc00000\0" \
  "kernelstart=0x11c0000\0" \
```

(continues on next page)

(continued from previous page)

```
  "kernel_img=image.ub\0" \
  "load_kernel=tftpboot ${clobstart} ${kernel_img}\0" \
  "update_kernel=setenv img kernel; setenv psize ${kernelsize}; setenv installcmd \
↪"install_kernel\"; run load_kernel test_crc; setenv img; setenv psize; setenv␣
↪installcmd\0" \
  "install_kernel=sf probe 0 && sf erase ${kernelstart} ${kernelsize} && " \
    "sf write ${clobstart} ${kernelstart} ${filesize}\0" \
  "cp_kernel2ram=sf probe 0 && sf read ${netstart} ${kernelstart} ${kernelsize}\0" \
  "fpgasize=0x1000000\0" \
  "fpgastart=0x0\0" \
  "fpga_img=system.bit.bin\0" \
  "load_fpga=tftpboot ${clobstart} ${fpga_img}\0" \
  "update_fpga=setenv img fpga; setenv psize ${fpgasize}; setenv installcmd \"install_
↪fpga\"; run load_fpga test_img; setenv img; setenv psize; setenv installcmd\0" \
  "install_fpga=sf probe 0 && sf erase ${fpgastart} ${fpgasize} && " \
    "sf write ${clobstart} ${fpgastart} ${filesize}\0" \
  "fault=echo ${img} image size is greater than allocated place - partition ${img} is␣
↪NOT UPDATED\0" \
  "test_crc=if imi ${clobstart}; then run test_img; else echo ${img} Bad CRC - ${img} is␣
↪NOT UPDATED; fi\0" \
  "test_img=setenv var \"if test ${filesize} -gt ${psize}\\; then run fault\\; else run $
↪{installcmd}\\; fi\"; run var; setenv var\0" \
  "netboot=tftpboot ${netstartaddr} ${kernel_img} && bootm\0" \
  "default_bootcmd=bootcmd\0" \
""
```

7.2.9 Mods for MicroZed

These modifications are specific to the MicroZed BSP.

7.2.10 Mods for PicoZed

These modifications are specific to the PicoZed BSP.

7.2.11 Mods for UltraZed EV

These modifications are specific to the UltraZed EV BSP.

7.2.12 Mods for VC707

These modifications are specific to the VC707 BSP. As Xilinx doesn't provide a BSP for the VC707, we instead use the BSP for the KC705 and modify it to suit the VC707.

1. Replace the line `CONFIG_XILINX_MICROBLAZE0_FAMILY="kintex7"` with the following in `project-spec/configs/linux-xlnx/plnx_kernel.cfg`:

```
CONFIG_XILINX_MICROBLAZE0_FAMILY="virtex7"
```

2. Append the following lines to `project-spec/configs/config`:

```
# Use general template
CONFIG_SUBSYSTEM_MACHINE_NAME="template"

# Larger partition for bitstream
CONFIG_SUBSYSTEM_FLASH_AXI_EMC_0_BANK0_PART0_SIZE=0xD00000
```

3. Append the following lines to file `project-spec/meta-user/recipes-bsp/u-boot/files/platform-top.h`:

```
/* BOOTCOMMAND */
#define CONFIG_USE_BOOTCOMMAND 1
#define CONFIG_BOOTCOMMAND    "cp.b ${kernelstart} ${netstartaddr} ${kernelsize} &&␣
↪bootm ${netstartaddr}"

/* Extra U-Boot Env settings */
#define CONFIG_EXTRA_ENV_SETTINGS \
  SERIAL_MULTI \
  CONSOLE_ARG \
  ESERIAL0 \
  "nc=setenv stdout nc;setenv stdin nc;\0" \
  "ethaddr=00:0a:35:00:22:01\0" \
  "autoload=no\0" \
  "sdbootdev=0\0" \
  "clobstart=0x80000000\0" \
  "netstart=0x80000000\0" \
  "dtbnetstart=0x81e00000\0" \
  "netstartaddr=0x81000000\0"  "loadaddr=0x80000000\0" \
  "initrd_high=0x0\0" \
  "bootsize=0x180000\0" \
  "bootstart=0x60d00000\0" \
  "boot_img=u-boot-s.bin\0" \
  "load_boot=tftpboot ${clobstart} ${boot_img}\0" \
  "update_boot=setenv img boot; setenv psize ${bootsize}; setenv installcmd \"install_
↪boot\"; run load_boot test_img; setenv img; setenv psize; setenv installcmd\0" \
  "install_boot=protect off ${bootstart} +${bootsize} && erase ${bootstart} +${bootsize}␣
↪&& "  "cp.b ${clobstart} ${bootstart} ${filesize}\0" \
  "bootenvsize=0x20000\0" \
  "bootenvstart=0x60e80000\0" \
  "eraseenv=protect off ${bootenvstart} +${bootenvsize} && erase ${bootenvstart} +$
↪{bootenvsize}\0" \
  "kernelsize=0xc00000\0" \
  "kernelstart=0x60ea0000\0" \
  "kernel_img=image.ub\0" \
  "load_kernel=tftpboot ${clobstart} ${kernel_img}\0" \
  "update_kernel=setenv img kernel; setenv psize ${kernelsize}; setenv installcmd \
↪"install_kernel\"; run load_kernel test_crc; setenv img; setenv psize; setenv␣
↪installcmd\0" \
  "install_kernel=protect off ${kernelstart} +${kernelsize} && erase ${kernelstart} +$
↪{kernelsize} && "  "cp.b ${clobstart} ${kernelstart} ${filesize}\0" \
  "cp_kernel2ram=cp.b ${kernelstart} ${netstart} ${kernelsize}\0" \
  "fpgasize=0xd00000\0" \
  "fpgastart=0x60000000\0" \
  "fpga_img=system.bit.bin\0" \
```

(continues on next page)

(continued from previous page)

```
  "load_fpga=tftpboot ${clobstart} ${fpga_img}\0" \
  "update_fpga=setenv img fpga; setenv psize ${fpgasize}; setenv installcmd \"install_
↪fpga\"; run load_fpga test_img; setenv img; setenv psize; setenv installcmd\0" \
  "install_fpga=protect off ${fpgastart} +${fpgasize} && erase ${fpgastart} +${fpgasize}␣
↪&& "  "cp.b ${clobstart} ${fpgastart} ${filesize}\0" \
  "fault=echo ${img} image size is greater than allocated place - partition ${img} is␣
↪NOT UPDATED\0" \
  "test_crc=if imi ${clobstart}; then run test_img; else echo ${img} Bad CRC - ${img} is␣
↪NOT UPDATED; fi\0" \
  "test_img=setenv var \"if test ${filesize} -gt ${psize}\\; then run fault\\; else run $
↪{installcmd}\\; fi\"; run var; setenv var\0" \
  "netboot=tftpboot ${netstartaddr} ${kernel_img} && bootm\0" \
  "default_bootcmd=bootcmd\0" \
""
```

7.2.13 Mods for VC709

These modifications are specific to the VC709 BSP. As Xilinx doesn't provide a BSP for the VC709, we instead use the BSP for the KC705 and modify it to suit the VC709.

1. Replace the line `CONFIG_XILINX_MICROBLAZE0_FAMILY="kintex7"` with the following in `project-spec/configs/linux-xlnx/plnx_kernel.cfg`:

```
CONFIG_XILINX_MICROBLAZE0_FAMILY="virtex7"
```

2. Append the following lines to `project-spec/configs/config`:

```
# Use general template
CONFIG_SUBSYSTEM_MACHINE_NAME="template"
```

3. Append the following lines to file `project-spec/meta-user/recipes-bsp/u-boot/files/platform-top.h`:

```
/* BOOTCOMMAND */
#define CONFIG_USE_BOOTCOMMAND 1
#define CONFIG_BOOTCOMMAND    "cp.b ${kernelstart} ${netstartaddr} ${kernelsize} &&␣
↪bootm ${netstartaddr}"

/* Extra U-Boot Env settings */
#define CONFIG_EXTRA_ENV_SETTINGS \
  SERIAL_MULTI \
  CONSOLE_ARG \
  ESERIAL0 \
  "nc=setenv stdout nc;setenv stdin nc;\0" \
  "ethaddr=00:0a:35:00:22:01\0" \
  "autoload=no\0" \
  "sdbootdev=0\0" \
  "clobstart=0x80000000\0" \
  "netstart=0x80000000\0" \
  "dtbnetstart=0x81e00000\0" \
  "netstartaddr=0x81000000\0"  "loadaddr=0x80000000\0" \
  "initrd_high=0x0\0" \
```

(continues on next page)

(continued from previous page)

```
  "bootsize=0x180000\0" \
  "bootstart=0x60b00000\0" \
  "boot_img=u-boot-s.bin\0" \
  "load_boot=tftpboot ${clobstart} ${boot_img}\0" \
  "update_boot=setenv img boot; setenv psize ${bootsize}; setenv installcmd \"install_
↪boot\"; run load_boot test_img; setenv img; setenv psize; setenv installcmd\0" \
  "install_boot=protect off ${bootstart} +${bootsize} && erase ${bootstart} +${bootsize}␣
↪&& "  "cp.b ${clobstart} ${bootstart} ${filesize}\0" \
  "bootenvsize=0x20000\0" \
  "bootenvstart=0x60c80000\0" \
  "eraseenv=protect off ${bootenvstart} +${bootenvsize} && erase ${bootenvstart} +$
↪{bootenvsize}\0" \
  "kernelsize=0xc00000\0" \
  "kernelstart=0x60ca0000\0" \
  "kernel_img=image.ub\0" \
  "load_kernel=tftpboot ${clobstart} ${kernel_img}\0" \
  "update_kernel=setenv img kernel; setenv psize ${kernelsize}; setenv installcmd \
↪"install_kernel\"; run load_kernel test_crc; setenv img; setenv psize; setenv␣
↪installcmd\0" \
  "install_kernel=protect off ${kernelstart} +${kernelsize} && erase ${kernelstart} +$
↪{kernelsize} && "  "cp.b ${clobstart} ${kernelstart} ${filesize}\0" \
  "cp_kernel2ram=cp.b ${kernelstart} ${netstart} ${kernelsize}\0" \
  "fpgasize=0xb00000\0" \
  "fpgastart=0x60000000\0" \
  "fpga_img=system.bit.bin\0" \
  "load_fpga=tftpboot ${clobstart} ${fpga_img}\0" \
  "update_fpga=setenv img fpga; setenv psize ${fpgasize}; setenv installcmd \"install_
↪fpga\"; run load_fpga test_img; setenv img; setenv psize; setenv installcmd\0" \
  "install_fpga=protect off ${fpgastart} +${fpgasize} && erase ${fpgastart} +${fpgasize}␣
↪&& "  "cp.b ${clobstart} ${fpgastart} ${filesize}\0" \
  "fault=echo ${img} image size is greater than allocated place - partition ${img} is␣
↪NOT UPDATED\0" \
  "test_crc=if imi ${clobstart}; then run test_img; else echo ${img} Bad CRC - ${img} is␣
↪NOT UPDATED; fi\0" \
  "test_img=setenv var \"if test ${filesize} -gt ${psize}\\; then run fault\\; else run $
↪{installcmd}\\; fi\"; run var; setenv var\0" \
  "netboot=tftpboot ${netstartaddr} ${kernel_img} && bootm\0" \
  "default_bootcmd=bootcmd\0" \
""
```

7.2.14 Mods for VCU108

These modifications are specific to the VCU108 BSP.

1. Append the following lines to `project-spec/configs/config`:

```
# Use general template
CONFIG_SUBSYSTEM_MACHINE_NAME="template"

# Flash Settings - use Linear flash instead of QSPI
CONFIG_SUBSYSTEM_FLASH_AXI_EMC_0_BANK0_SELECT=y
```

(continues on next page)

(continued from previous page)

```
CONFIG_SUBSYSTEM_FLASH_AXI_EMC_0_BANK0_PART0_NAME="fpga"
CONFIG_SUBSYSTEM_FLASH_AXI_EMC_0_BANK0_PART0_SIZE=0x1B00000
CONFIG_SUBSYSTEM_FLASH_AXI_EMC_0_BANK0_PART1_NAME="boot"
CONFIG_SUBSYSTEM_FLASH_AXI_EMC_0_BANK0_PART1_SIZE=0x180000
CONFIG_SUBSYSTEM_FLASH_AXI_EMC_0_BANK0_PART2_NAME="bootenv"
CONFIG_SUBSYSTEM_FLASH_AXI_EMC_0_BANK0_PART2_SIZE=0x20000
CONFIG_SUBSYSTEM_FLASH_AXI_EMC_0_BANK0_PART3_NAME="kernel"
CONFIG_SUBSYSTEM_FLASH_AXI_EMC_0_BANK0_PART3_SIZE=0xC00000
CONFIG_SUBSYSTEM_FLASH_AXI_EMC_0_BANK0_PART4_NAME=""
CONFIG_SUBSYSTEM_FLASH_IP_NAME="axi_emc_0"
```

2. Append the following lines to file `project-spec/meta-user/recipes-bsp/u-boot/files/platform-top.h`:

```
/* BOOTCOMMAND */
#define CONFIG_USE_BOOTCOMMAND 1
#define CONFIG_BOOTCOMMAND    "cp.b ${kernelstart} ${netstartaddr} ${kernelsize} &&␣
↪bootm ${netstartaddr}"

/* Extra U-Boot Env settings */
#define CONFIG_EXTRA_ENV_SETTINGS \
  SERIAL_MULTI \
  CONSOLE_ARG \
  ESERIAL0 \
  "nc=setenv stdout nc;setenv stdin nc;\0" \
  "ethaddr=00:0a:35:00:22:01\0" \
  "autoload=no\0" \
  "sdbootdev=0\0" \
  "clobstart=0x80000000\0" \
  "netstart=0x80000000\0" \
  "dtbnetstart=0x81e00000\0" \
  "netstartaddr=0x81000000\0"  "loadaddr=0x80000000\0" \
  "initrd_high=0x0\0" \
  "bootsize=0x180000\0" \
  "bootstart=0x61B00000\0" \
  "boot_img=u-boot-s.bin\0" \
  "load_boot=tftpboot ${clobstart} ${boot_img}\0" \
  "update_boot=setenv img boot; setenv psize ${bootsize}; setenv installcmd \"install_
↪boot\"; run load_boot test_img; setenv img; setenv psize; setenv installcmd\0" \
  "install_boot=protect off ${bootstart} +${bootsize} && erase ${bootstart} +${bootsize}␣
↪&& "  "cp.b ${clobstart} ${bootstart} ${filesize}\0" \
  "bootenvsize=0x20000\0" \
  "bootenvstart=0x61C80000\0" \
  "eraseenv=protect off ${bootenvstart} +${bootenvsize} && erase ${bootenvstart} +$
↪{bootenvsize}\0" \
  "kernelsize=0xC00000\0" \
  "kernelstart=0x61CA0000\0" \
  "kernel_img=image.ub\0" \
  "load_kernel=tftpboot ${clobstart} ${kernel_img}\0" \
  "update_kernel=setenv img kernel; setenv psize ${kernelsize}; setenv installcmd \
↪"install_kernel\"; run load_kernel test_crc; setenv img; setenv psize; setenv␣
↪installcmd\0" \
```

(continues on next page)

(continued from previous page)

```
  "install_kernel=protect off ${kernelstart} +${kernelsize} && erase ${kernelstart} +$
↪{kernelsize} && "  "cp.b ${clobstart} ${kernelstart} ${filesize}\0" \
  "cp_kernel2ram=cp.b ${kernelstart} ${netstart} ${kernelsize}\0" \
  "fpgasize=0x1B00000\0" \
  "fpgastart=0x60000000\0" \
  "fpga_img=system.bit.bin\0" \
  "load_fpga=tftpboot ${clobstart} ${fpga_img}\0" \
  "update_fpga=setenv img fpga; setenv psize ${fpgasize}; setenv installcmd \"install_
↪fpga\"; run load_fpga test_img; setenv img; setenv psize; setenv installcmd\0" \
  "install_fpga=protect off ${fpgastart} +${fpgasize} && erase ${fpgastart} +${fpgasize}␣
↪&& "  "cp.b ${clobstart} ${fpgastart} ${filesize}\0" \
  "fault=echo ${img} image size is greater than allocated place - partition ${img} is␣
↪NOT UPDATED\0" \
  "test_crc=if imi ${clobstart}; then run test_img; else echo ${img} Bad CRC - ${img} is␣
↪NOT UPDATED; fi\0" \
  "test_img=setenv var \"if test ${filesize} -gt ${psize}\\; then run fault\\; else run $
↪{installcmd}\\; fi\"; run var; setenv var\0" \
  "netboot=tftpboot ${netstartaddr} ${kernel_img} && bootm\0" \
  "default_bootcmd=bootcmd\0" \
""
```

7.2.15 Mods for VCU118

These modifications are specific to the VCU118 BSP.

1. Append the following lines to `project-spec/configs/config`:

```
# Use general template
# We use the template because the board dtsi expects axi_ethernet_0 to be
# the on-board Ethernet, and axi_iic_0 to be the I2C. We define the I2C
# device tree for iic_main in the system-user.dtsi in this BSP.
CONFIG_SUBSYSTEM_MACHINE_NAME="template"

# Flash Settings - use Linear flash instead of QSPI
CONFIG_SUBSYSTEM_FLASH_AXI_EMC_0_BANK0_SELECT=y
CONFIG_SUBSYSTEM_FLASH_AXI_EMC_0_BANK0_PART0_NAME="fpga"
CONFIG_SUBSYSTEM_FLASH_AXI_EMC_0_BANK0_PART0_SIZE=0x1C00000
CONFIG_SUBSYSTEM_FLASH_AXI_EMC_0_BANK0_PART1_NAME="boot"
CONFIG_SUBSYSTEM_FLASH_AXI_EMC_0_BANK0_PART1_SIZE=0x180000
CONFIG_SUBSYSTEM_FLASH_AXI_EMC_0_BANK0_PART2_NAME="bootenv"
CONFIG_SUBSYSTEM_FLASH_AXI_EMC_0_BANK0_PART2_SIZE=0x20000
CONFIG_SUBSYSTEM_FLASH_AXI_EMC_0_BANK0_PART3_NAME="kernel"
CONFIG_SUBSYSTEM_FLASH_AXI_EMC_0_BANK0_PART3_SIZE=0xC00000
CONFIG_SUBSYSTEM_FLASH_AXI_EMC_0_BANK0_PART4_NAME=""
CONFIG_SUBSYSTEM_FLASH_IP_NAME="axi_emc_0"
```

2. Append the following lines to file `project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi`:

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

(continues on next page)

(continued from previous page)

```
  #size-cells = <0>;
  i2c-mux@75 {
    compatible = "nxp,pca9548";
    #address-cells = <1>;
    #size-cells = <0>;
    reg = <0x75>;
    i2c@3 {
      #address-cells = <1>;
      #size-cells = <0>;
      reg = <3>;
      eeprom@54 {
        compatible = "atmel,24c08";
        reg = <0x54>;
      };
    };
  };
  i2c-mux@74 {
    compatible = "nxp,pca9548";
    #address-cells = <1>;
    #size-cells = <0>;
    reg = <0x74>;
    i2c@0 {
      #address-cells = <1>;
      #size-cells = <0>;
      reg = <0>;
      si570: clock-generator@5d {
        #clock-cells = <0>;
        compatible = "silabs,si570";
        temperature-stability = <50>;
        reg = <0x5d>;
        factory-fout = <156250000>;
        clock-frequency = <148500000>;
      };
    };
  };
};
```

3. Append the following lines to file `project-spec/meta-user/recipes-bsp/u-boot/files/platform-top.h`:

```
/* BOOTCOMMAND */
#define CONFIG_USE_BOOTCOMMAND 1
#define CONFIG_BOOTCOMMAND    "cp.b ${kernelstart} ${netstartaddr} ${kernelsize} &&␣
↪bootm ${netstartaddr}"

/* Extra U-Boot Env settings */
#define CONFIG_EXTRA_ENV_SETTINGS \
  SERIAL_MULTI \
  CONSOLE_ARG \
  ESERIAL0 \
  "nc=setenv stdout nc;setenv stdin nc;\0" \
  "ethaddr=00:0a:35:00:22:01\0" \
```

(continues on next page)

(continued from previous page)

```
  "autoload=no\0" \
  "sdbootdev=0\0" \
  "clobstart=0x80000000\0" \
  "netstart=0x80000000\0" \
  "dtbnetstart=0x81e00000\0" \
  "netstartaddr=0x81000000\0"  "loadaddr=0x80000000\0" \
  "initrd_high=0x0\0" \
  "bootsize=0x180000\0" \
  "bootstart=0x61C00000\0" \
  "boot_img=u-boot-s.bin\0" \
  "load_boot=tftpboot ${clobstart} ${boot_img}\0" \
  "update_boot=setenv img boot; setenv psize ${bootsize}; setenv installcmd \"install_
↪boot\"; run load_boot test_img; setenv img; setenv psize; setenv installcmd\0" \
  "install_boot=protect off ${bootstart} +${bootsize} && erase ${bootstart} +${bootsize}␣
↪&& "  "cp.b ${clobstart} ${bootstart} ${filesize}\0" \
  "bootenvsize=0x20000\0" \
  "bootenvstart=0x61D80000\0" \
  "eraseenv=protect off ${bootenvstart} +${bootenvsize} && erase ${bootenvstart} +$
↪{bootenvsize}\0" \
  "kernelsize=0xC00000\0" \
  "kernelstart=0x61DA0000\0" \
  "kernel_img=image.ub\0" \
  "load_kernel=tftpboot ${clobstart} ${kernel_img}\0" \
  "update_kernel=setenv img kernel; setenv psize ${kernelsize}; setenv installcmd \
↪"install_kernel\"; run load_kernel test_crc; setenv img; setenv psize; setenv␣
↪installcmd\0" \
  "install_kernel=protect off ${kernelstart} +${kernelsize} && erase ${kernelstart} +$
↪{kernelsize} && "  "cp.b ${clobstart} ${kernelstart} ${filesize}\0" \
  "cp_kernel2ram=cp.b ${kernelstart} ${netstart} ${kernelsize}\0" \
  "fpgasize=0x1C00000\0" \
  "fpgastart=0x60000000\0" \
  "fpga_img=system.bit.bin\0" \
  "load_fpga=tftpboot ${clobstart} ${fpga_img}\0" \
  "update_fpga=setenv img fpga; setenv psize ${fpgasize}; setenv installcmd \"install_
↪fpga\"; run load_fpga test_img; setenv img; setenv psize; setenv installcmd\0" \
  "install_fpga=protect off ${fpgastart} +${fpgasize} && erase ${fpgastart} +${fpgasize}␣
↪&& "  "cp.b ${clobstart} ${fpgastart} ${filesize}\0" \
  "fault=echo ${img} image size is greater than allocated place - partition ${img} is␣
↪NOT UPDATED\0" \
  "test_crc=if imi ${clobstart}; then run test_img; else echo ${img} Bad CRC - ${img} is␣
↪NOT UPDATED; fi\0" \
  "test_img=setenv var \"if test ${filesize} -gt ${psize}\\; then run fault\\; else run $
↪{installcmd}\\; fi\"; run var; setenv var\0" \
  "netboot=tftpboot ${netstartaddr} ${kernel_img} && bootm\0" \
  "default_bootcmd=bootcmd\0" \
""
```

CHAPTER

EIGHT

TROUBLESHOOTING

8.1 Build failures

Check the following if the project fails to build or generate a bitstream:

1. **Are you using the correct version of Vivado for this version of the repository?**
   Check the version specified in the Requirements section of the README.md file. Note that this project is regularly maintained to the latest version of Vivado and you may have to refer to an earlier commit of this repo if you are using an older version of Vivado.
2. **Did you correctly follow the Build instructions in this readme file?**
   All the projects in the repo are built, synthesised and implemented to a bitstream before being committed, so if you follow the instructions, there should not be any build issues.
3. **Did you copy/clone the repo into a short directory structure?**
   Vivado doesn't cope well with long directory structures, so copy/clone the repo into a short directory structure such as `C:\projects\`. When working in long directory structures, you can get errors relating to missing files, particularly files that are normally generated by Vivado (FIFOs, etc).

8.2 PetaLinux issues

8.2.1 Ports not working

Check the following if you are unable to get ports working in PetaLinux.

1. **Check the interface-to-port assignment for your design**
   The assignment of interfaces (eg. eth0, eth1, eth2, etc) to ports (eg. Ethernet FMC port 0, 1, 2 and 3) is specific to the design that you are using. The interface to port assignment is documented here.
2. **Each port must be assigned to a different subnet**
   If you assign interface eth0 to IP address 192.168.1.10, then you must use a different subnet for the IP address of eth1, eth2 and eth3. Multiple ports that are managed under Linux must be assigned to different subnets, or they will not work. An example address assignment would be eth0=192.168.1.10, eth1=192.168.2.10, eth2=192.168.3.10, eth3=192.168.4.10.

8.2.2 Dropped pings/packets

No dropped packets are to be expected with our example designs. If you are experiencing dropped packets of any kind, this can be an indication of one of the following issues:

- Timing in the FPGA design is not optimal (check for timing errors in the Vivado design)
- Timing of the RGMII interface is not optimal (can be due to the FPGA design or the PHY configuration)

We ensure that there are no timing errors or issues on all of our designs before making a release, so typically this problem occurs on custom designs where the timing issues have not yet been optimized. Please contact us for support if you are experiencing dropped packets.

CHAPTER

NINE

REVISION HISTORY

This is the first version of the documentation.