KNOWLEDGE RESOURCES
Switzerland GmbH

KRM-4ZURF RevB Example Designs

User Guide RevB

Knowledge Resources GmbH
Ackerstrasse 30
CH – 4057 Basel
Switzerland

www.knowres.com

Table of Contents

Revision History

Date	Document revision	Changes
June 24th 2020	1.0	Initial document for RevA-EA (Early Access)
August 14th 2020	2.0	Update for hardware RevB
September 8th 2020	2.1	Added reference to 8GB DDR4 PL Memory patch
October 6th 2020	2.2.	Add new Appendix to flash the eMMC

Disclaimer

All provided data is for information purposes only and not guaranteed for legal purposes.

Information has been carefully checked and is believed to be accurate;

However, no responsibility is assumed for inaccuracies.

Referenced brand and product names are trademarks or registered trademarks of their respective owners.

Specifications are subject to change without notice.

Trademark Acknowledgement:

Brand and product names are trademarks or registered trademarks of their respective owners.

Assumptions

The reader is familiar with Xilinx FPGA and SoC components and the related terminology in common use.

Acronyms

FOM:	FPGA on Module
FU:	Future Use
KR:	Knowledge Resources GmbH
MIG:	Memory Interface Generator, a tool of Xilinx to easily implement a DDR3 controller
NA:	Not Applicable
PL:	Programmable Logic
PS:	Processing Subsystem
SoC:	System on Chip

Reference documents

ZYNQ all programmable SoC, Xilinx, www.xilinx.com
Zynq UltraScale+ RFSoC, Xilinx, www.xilinx.com
DDR4 SDRAM, Micron, www.micron.com
UltraScale GTY Transceivers, Xilinx, PG196
RF Data Converter Interface, Xilinx, UG1309

Support

KR will provide free of charge to qualified customers:

-Schematic and PCB libraries with Module and carrier board design components (Altium)
-3D STEP models of the module and heat spreader plate
-LINUX BSP and LINUX port (Plug and Boot ready)
-Reference schematics of the evaluation boards (Altium native and PDF)
-Reference designs for on-board PL memory use
-Constraints files (pinning) to accelerate design starts

Further support to aid in customer specific design in's is available at competitive rates, please contact KR for details: + 41 61 545 2080 or mail to office@knowres.com

I. Introduction

The KRM-4ZURF Example Designs are a collection of simple projects which facilitate the evaluation of critical components of KRM-4ZU27DR RevB module and KRC4700 carrier boards. They also serve as building blocks to compose more complex designs.
Each design includes at least a top-level HDL file and a complete XDC file for IO-standard and location constraints. Where is appropriate, a correctly configured processing system and Linux configuration are also provided.
Currently there are three independent example designs for DDR4 RAMs, GTY Transceivers and RF Data Converters. For each component there is a dedicated chapter in this document with detailed information.

Compatibility

These example designs have been tested on KRM-4ZU27DR RevB module with Vivado 2019.1. They may work with newer and older versions of Vivado if they use a compatible IP Block revisions.

On Windows

The make.bat script expects the Xilinx Tools to be installed in C:\Xilinx\Vivado. If they are installed in a different location the path needs to be modified in the make.bat script.

On Linux

The make.sh script expects the Xilinx Tools to be installed in /opt/Xilinx/Vivado. If they are installed in a different location the path needs to be modified in the make.sh script.

II. System overview

The RFSoC evaluation kit consists of the KRM-4ZU27DR RevB module and the KRC4700 carrier board with RF, GTY and PS interfaces. It is provided fully assembled as shown on Figure 1. The principal system interfaces are identified from different perspectives in Figure 1 and Figure 2.

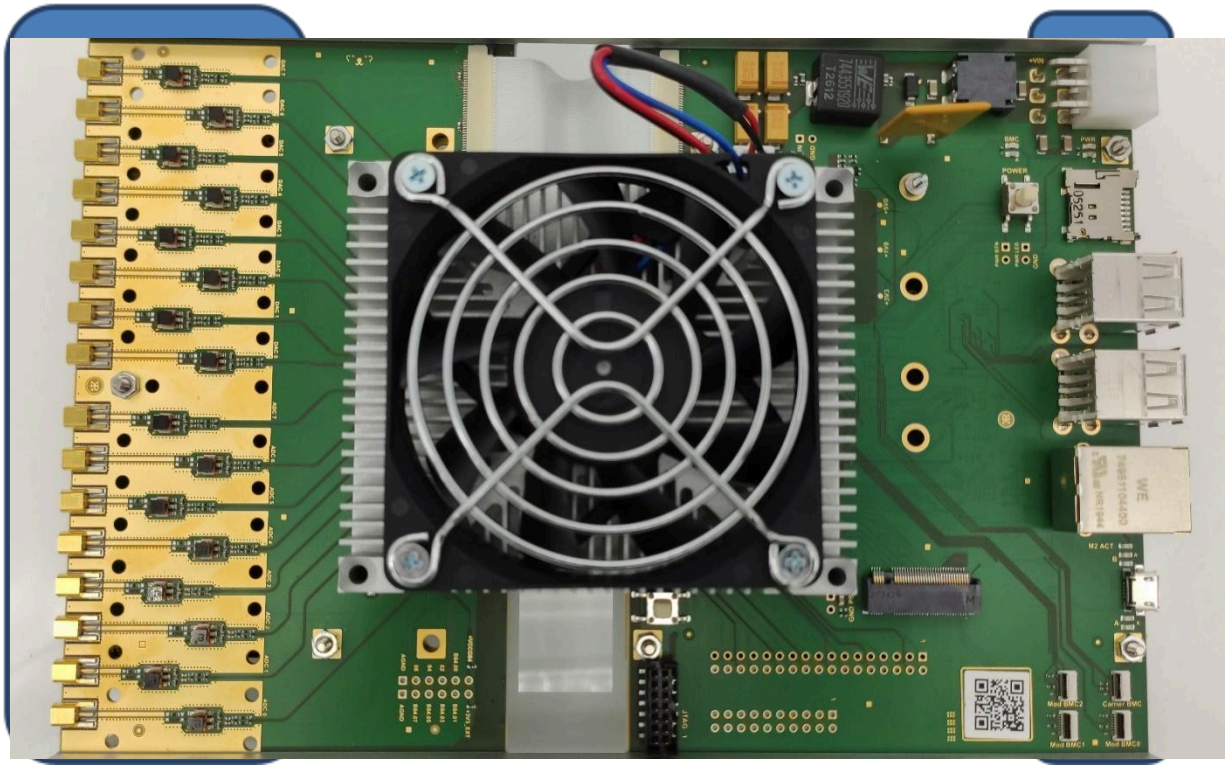

Figure 1. KRC4700 interfaces overview

Figure 2. KRC4700 front interface

In the complete package the following cables and adapters are also included:

- KRM-4ZURF heatsink, with assembly instructions
- USB to JTAG adapter
- Micro USB cables
- Micro ZIF to USB adapter (for BMC firmware updates)
- QSFP loopback cable
- MMCX loopback cables
- SMA to MMCX cables
- Power supply

Power Supply

The KRC4700 RevB system is powered by a +12V DC 6pin connector as described on Figure 3. Its location can be seen on Figure 2. A compatible power supply is provided with the system.

Please verify the correct polarity before using another power supply. Do not use PCI Express power connectors or a KRC4700 RevA power supply since their pinning is not compatible.

6| +12V	5| +12V	4| +12V
3| GND	2| GND	1| GND

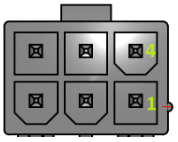

Figure 3. Power supply interface for RevB

Boot Mode

The system can boot from JTAG, eMMC or SDCard interfaces. The BMC user guide describes the process in detail and how to change the boot mode.

Board clocks

There are two main sources of clocks on the system from the KRM-4ZU27DR RevB module and the carrier. The on-module clock generator is configured on boot by the BMC with the frequencies listed on Table 1. A different configuration can be provided on request.

Table 1. On-module clocks configuration

Clock	Default frequency
MGT_CLK.CLK_0	100MHz
MGT_CLK.CLK_1	100MHz
MGT_CLK.CLK_2	100MHz
MGT_CLK.CLK_3	100MHz
RAM_CLK1	200MHz
RAM_CLK2	200MHz

The RF clocks are generated on the dedicated Clock PCB. They are configured during the ZynqMP Linux boot as described on Chapter V. Additional information how to change them is also provided on the same chapter. The current configuration is listed on Table 2.

Table 2. RF clocks default value

Clock	Default frequency
SYSREF_CLK	7.68MHz
CLOCK_PCB_CLK_OUT	7.68MHz
LMX2594_REFIN	122.88MHz
ADC224_CLK	409.6MHz
ADC225_CLK	409.6MHz
ADC226_CLK	409.6MHz
ADC227_CLK	409.6MHz
DAC228_CLK	409.6MHz
DAC229_CLK	409.6MHz

III. mig: DDR4 memories

Note: The example project supports up to 8GB for the PS DDR. The PL DDR by default is limited to 4GB. The process to use the full range of the 8GB configuration requires a patch for Xilinx Vivado. Please refer to the file mig/notes/ddr4_8GB.md for instructions to enable the full PL memory range.

The project *mig* instantiates 2 DDR4 controllers on PL and the PS DDR4 connected directly to the Zynq PS, as can be seen on Figure 4. The three DDR4 SDRAM are configured to use the default address of their interface.

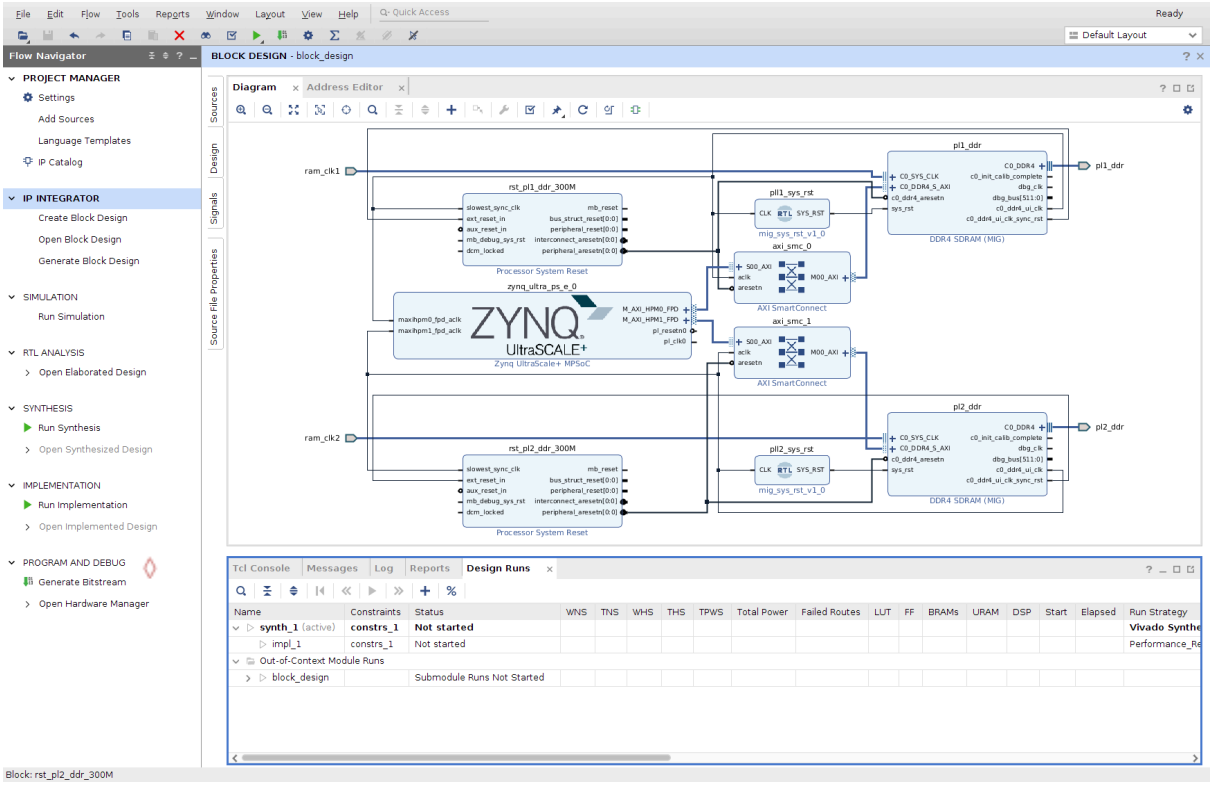

Figure 4. MIG project and block design

The files used in this project are organized as described on Table 3. Folders *prj* and *ip_cache* are generated upon Vivado project creation. In the folder *images* a compiled version equivalent to the output of section Build Bitstream (Vivado).

Table 3. *mig* project files and layout

Name	Contents
make.bat / make.sh	Top level script to create a new Vivado 2019.1 project
scripts/	Project helper scripts (ex: vivado_make_project.tcl)
src/bd	Vivado IP integrator block diagrams in TCL script form
src/xdc	Xilinx constraint files for pin assignment, setting IO standards etc.
src/hdl	HDL modules
prj (generated)	Vivado project folder created by make.bat / make.sh
ip_cache (generated)	Vivado repository of cached IP synthesis results
ip_repo (generated)	Vivado local IP repository (unused)
images	Pre-compiled project binaries

Build Bitstream (Vivado)

1. Create the Xilinx Vivado project
   a. On Windows: double click on make.bat
   b. On Linux: open a terminal and enter ./make.sh
2. Follow the on screen instructions and wait for a few moments.
3. Open the new Vivado project “prj/krm-zurf-mig.xpr”.
4. “Generate Bitstream” (see ◊ mark on Figure 4) and wait for process to finish.
5. Export the new design with the bitstream under “File>Export>Export Hardware” (Figure 5).
6. To move to the next section launch SDK under “File>Launch SDK” (Figure 6).

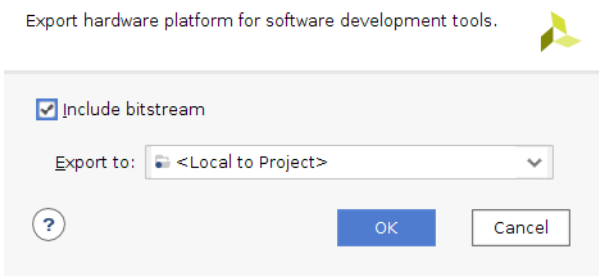

Figure 5. Export hardware

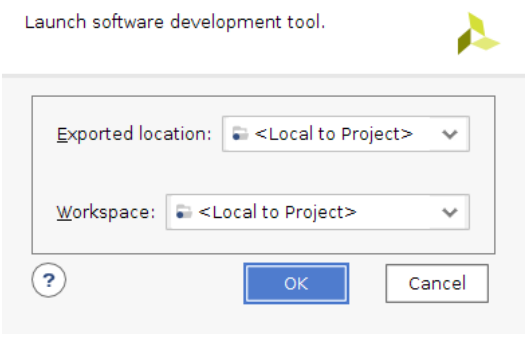

Figure 6. Launch SDK

Compile Memory Tests Software (Xilinx SDK)

1. In Xilinx SDK create a new application under “File>New>Application Project”.
2. Use default settings as shown in Figure 7.
3. Select the “Memory Test” template like Figure 8.

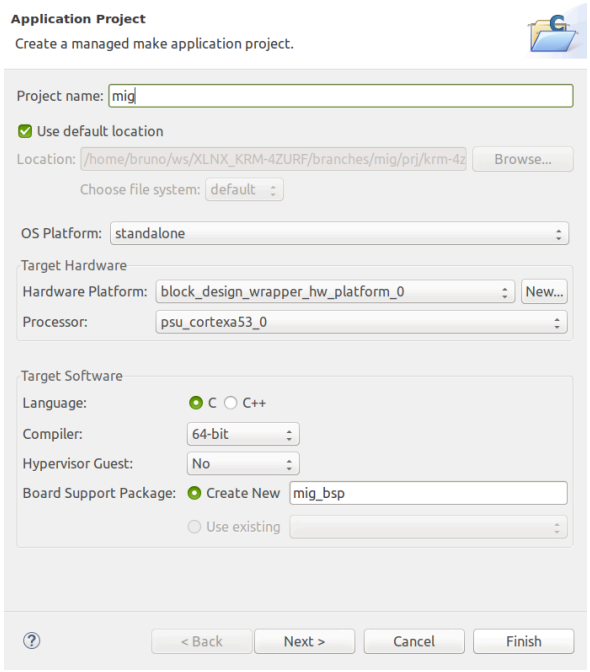

Figure 7. Xilinx SDK new Application Project

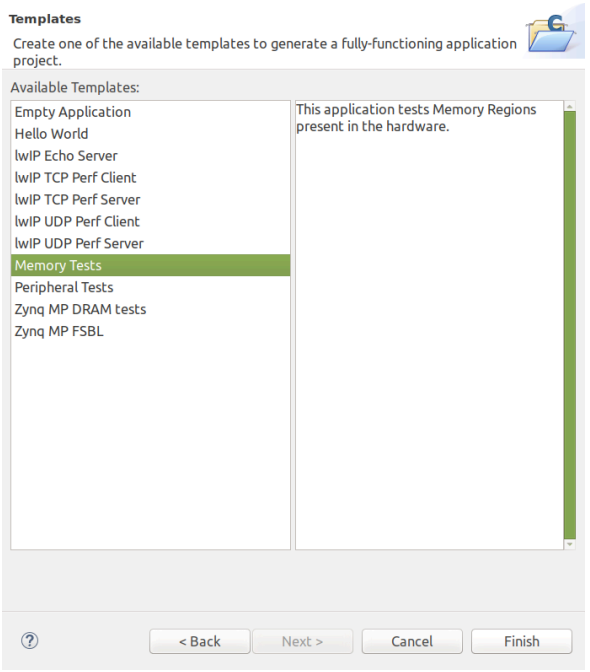

Figure 8. Xilinx SDK Project Templates

4. Run the design as described in the next section.

Run the design

1. Establish a serial connection (115200 baud, 8N1) between the KRC4000 carrier board and the host computer (Figure 2).
2. Run the example design directly from Xilinx SDK
    a. Power up the system without the SDCard
    b. Load the Bitstream by selecting “Xilinx>Program FPGA” in SDK as described in Figure 9
    c. “Run>Run” using the option selected in Figure 10.

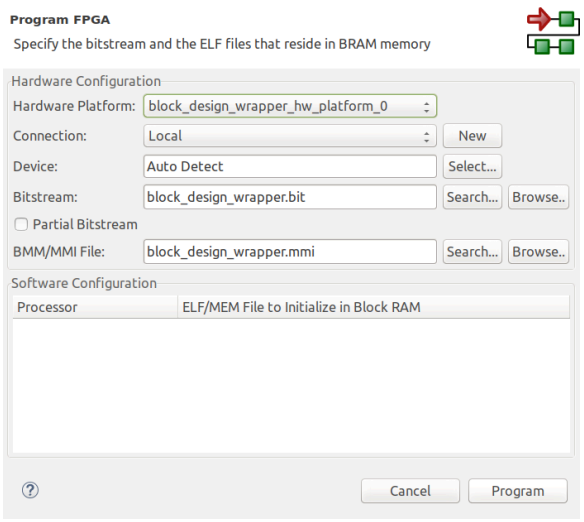

Figure 9. Load the FPGA bitstream on Xilinx SDK

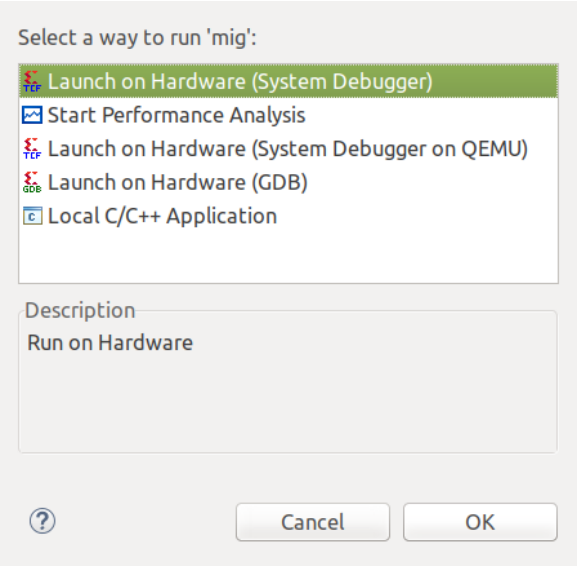

Figure 10. Run bare metal application from Xilinx SDK

3. The result of the test on the serial interface should be the same as described in Figure 11.

```
--Starting Memory Test Application--
NOTE: This application runs with D-Cache disabled.As a result, cacheline requests will not be generated
Testing memory region: pl1_ddr_C0_DDR4_ADDRESS_BLOCK
    Memory Controller: pl1_ddr
          Base Address: 0x400000000
                  Size: 0x80000000 bytes
           32-bit test: PASSED!
           16-bit test: PASSED!
            8-bit test: PASSED!
Testing memory region: pl2_ddr_C0_DDR4_ADDRESS_BLOCK
    Memory Controller: pl2_ddr
          Base Address: 0x500000000
                  Size: 0x80000000 bytes
           32-bit test: PASSED!
           16-bit test: PASSED!
            8-bit test: PASSED!
Testing memory region: psu_ddr_0_MEM_0
    Memory Controller: psu_ddr_0
          Base Address: 0x50
                  Size: 0x7FEFFFB0 bytes
           32-bit test: PASSED!
           16-bit test: PASSED!
            8-bit test: PASSED!
--Memory Test Application Complete--
```

Figure 11. Output of Memory Test Application

IV. qsfp-bert: GTY Transceivers

The project qsfp-bert is used to verify the GTY interfaces. It implements the Xilinx reference design for IBERT for UltraScale GTY Transceivers. Please refer to PG196 for additional information.

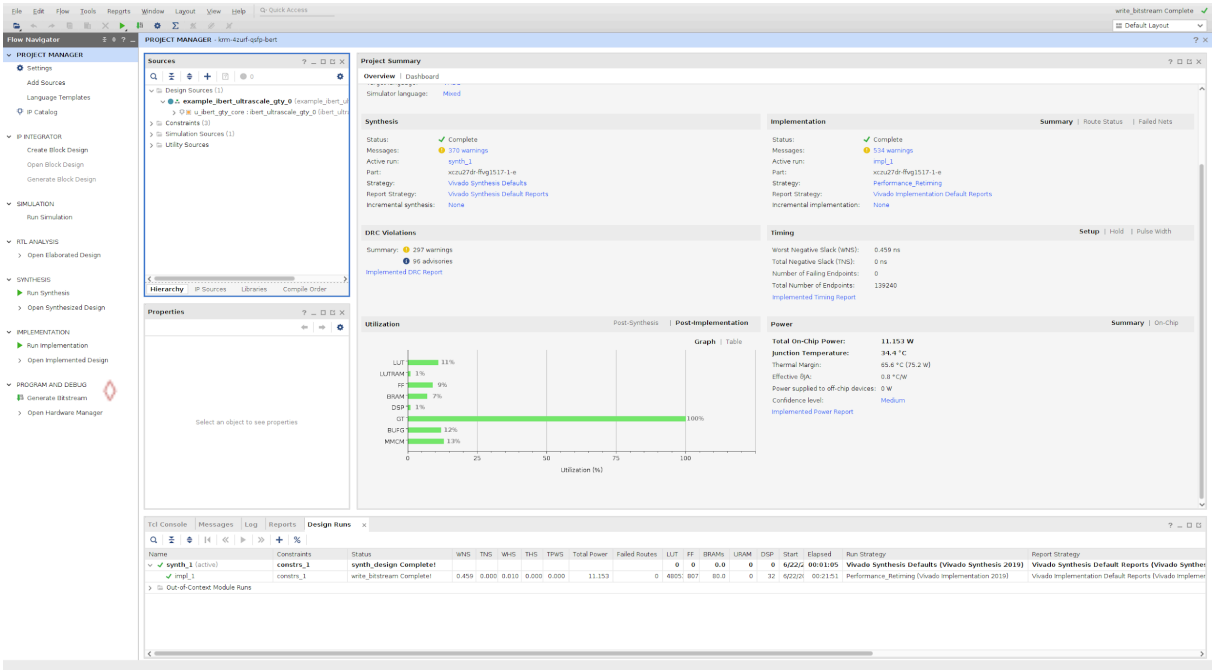

Figure 12. qsfp-bert Vivado project

The files used in this project are organized as described on Table 4. Folders *prj* and *ip_cache* are generated upon Vivado project creation. In the folder *images* a compiled version equivalent to the output of the sections Build Bitstream (Vivado) can be found.

Table 4. *qsfp-bert* project files and layout

Name	Contents
make.bat / make.sh	Top level script to create a new Vivado 2019.1 project
scripts/	Project helper scripts (ex: vivado_make_project.tcl)
src/bd	Vivado IP integrator block diagrams in TCL script form
src/xdc	Xilinx constraint files for pin assignment, setting IO standards etc.
src/hdl	HDL modules
prj (generated)	Vivado project folder created by make.bat / make.sh
ip_cache (generated)	Vivado repository of cached IP synthesis results
ip_repo (generated)	Vivado local IP repository (unused)
images	Pre-compiled project binaries

Build Bitstream (Vivado)

1. Create the Xilinx Vivado project
   a. On Windows: double click on make.bat
   b. On Linux: open a terminal and enter ./make.sh
2. Follow the on screen instructions and wait for a few moments.
3. Open the new Vivado project “prj/krm-zurf-qsfp-bert.xpr”.
4. “Generate Bitstream” (see ◊ mark on Figure 12)
5. Run the design as described in the next section.

Run the design

1. Connect the QSFP interfaces with a predefined order or use the auto-discovery function later, as can be seen on Figure 13.

Figure 13. QSFP loopback

2. Connect the JTAG interface and the power supply (see Figure 1).
3. Open “Hardware Manager” and “Auto Connect” to the FPGA (see Figure 14).
4. Load the FPGA bitstream and debug probes following the steps on Figure 15 and Figure 16.

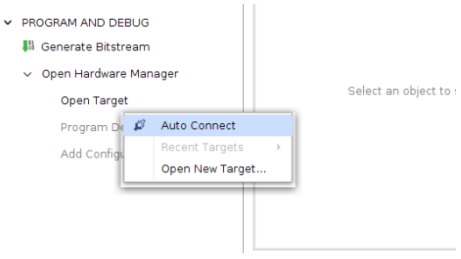

Figure 14. Vivado auto connect to hardware

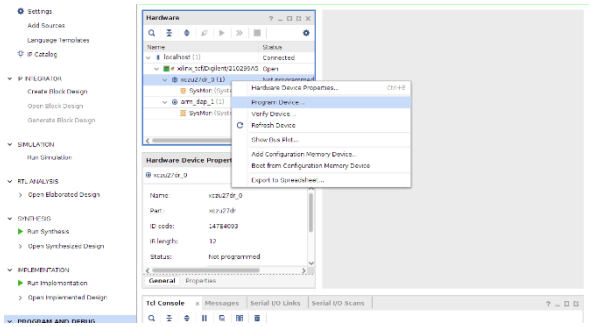

Figure 15. Vivado Program Device

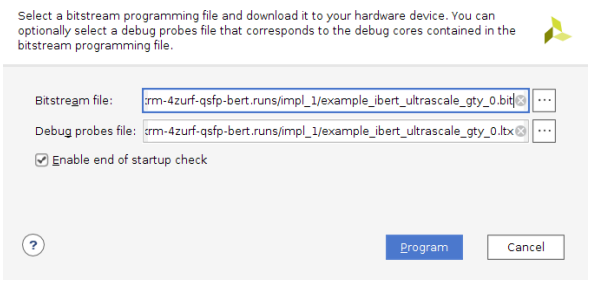

Figure 16. Load bitstream with debug probes

5. On the tab “Serial I/O Links” manually define links connections or use the auto-detect option and the result is demonstrated on Figure 17.

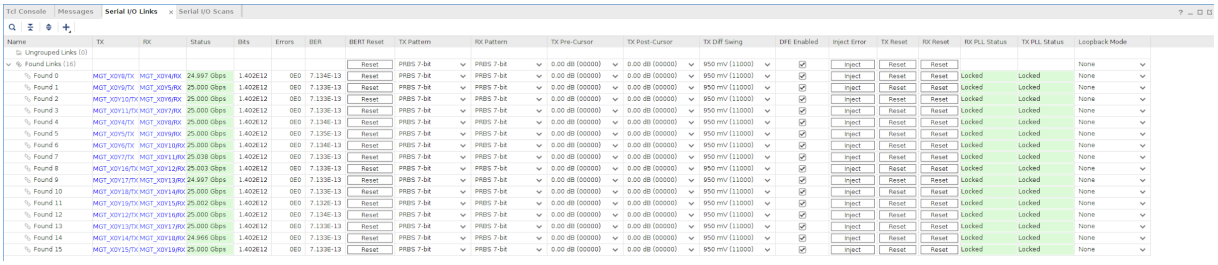

Figure 17. QSFP links

6. Additional measurements are available on “Serial I/O Scans” interface.

V. rf-analyzer: RF Data Converters

The project *rf-analyzer* is a port of the Xilinx Reference Design for Zynq UltraScale+ RFSoC RFData Converter to KRM-4ZU27DR RevB. Please refer to PG269 for additional information on the original design.
The design differences are related to the configuration of the high precision clocks trough SPI interfaces instead of using an indirect I2C access. The project block design is presented on Figure 18 and is divided in two main blocks.
The first is the RF interfaces that are controlled by a MicroBlaze processor and the JTAG interface. Xilinx provides a Windows Application to control this interface as describes on Chapter 5 of RF Data Converter Interface User Guide UG1309.
The second main block is the Zynq PS and SPI interface to control the high precision clocks. A Linux application was developed to provide the means to configure a predefined set of clocks frequencies.

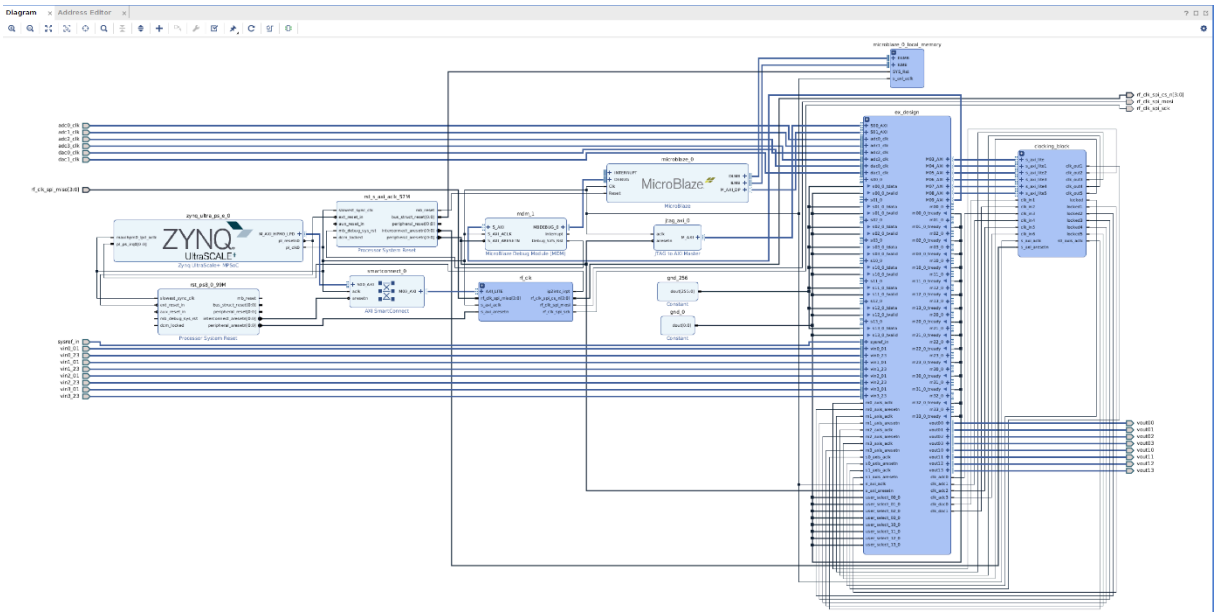

Figure 18. rf-analyzer block diagram

The files used in this project are organized as described on Table 5. Folders *prj* and *ip_cache* are generated upon Vivado project creation. In the folder *images* a compiled version equivalent to the output of the sections Build Bitstream (Vivado) and Generate PetaLinux image can be found.

Table 5. rf-analyzer project files and layout

Name	Contents
make.bat / make.sh	Top level script to create a new Vivado 2019.1 project
scripts/	Project helper scripts (ex: vivado_make_project.tcl)
src/bd	Vivado IP integrator block diagrams in TCL script form
src/xdc	Xilinx constraint files for pin assignment, setting IO standards etc.
src/hdl	HDL modules
src/data	Microblaze application
prj (generated)	Vivado project folder created by make.bat / make.sh
ip_cache (generated)	Vivado repository of cached IP synthesis results
ip_repo	Vivado local IP repository
petalinux	PetaLinux project folder
petalinux/project-spec/ meta-user/recipers-app/ spiclk	Linux applications to control the high precision clocks
images	Pre-compiled project binaries

Build Bitstream (Vivado)

1. Create the Xilinx Vivado project
   a. On Windows: double click on make.bat
   b. On Linux: open a terminal and enter ./make.sh
2. Follow the on screen instructions and wait for a few moments.
3. Open the new Vivado project “prj/krm-zurf-rf-analyzer.xpr”.
4. “Generate Bitstream” (see ◊ mark on Figure 12)
5. Wait until the process ends and export the Hardware using “File>Export>Export Hardware…” to get the menu described on Figure 19. Select the folder “images” as destination and “Include Bitstream”.

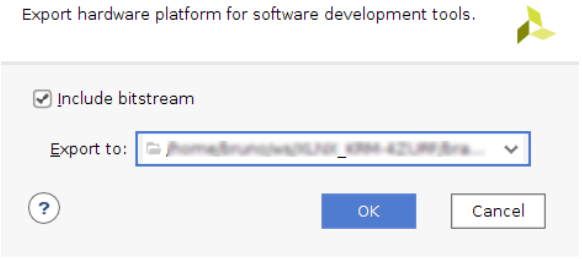

Figure 19. Export rf-analyzer hardware

6. Continue on the next section.

Generate PetaLinux image

Folder *petalinux* contains a project based on “zynqMP” 2019.1 template and the *hdf* file from the previous section with the following changes:

- `petalinux-config`
  - Fixed ip to “192.168.2.3”
  - Primary sd card is “psu_sd_1”
  - Changed hostname to “krm-4zurf”
- `petalinux-config -c kernel`
  - SPIDEV enabled under Device Drivers>SPI Support as described on Figure 20

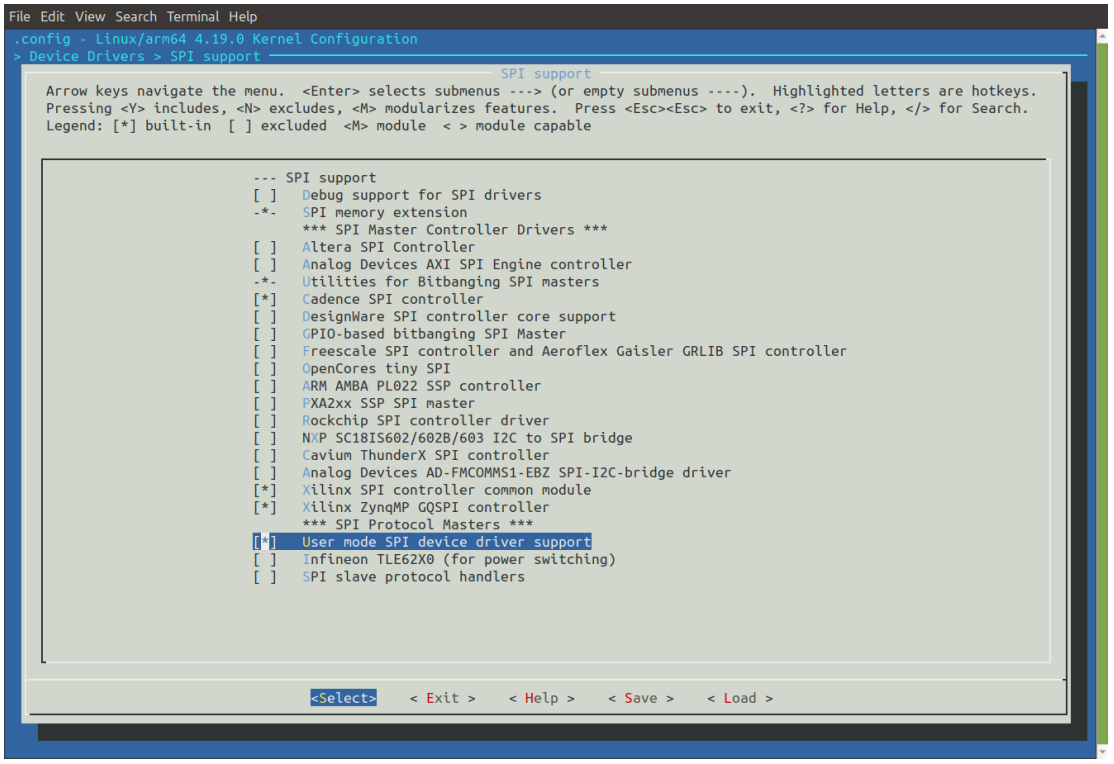

Figure 20. Enable User mode SPI device driver

- `petalinux-config -c rootfs`
  - Include app "spiclk". The source code of this application can be found on: "`petalinux/project-spec/meta-user/recipes-app/spiclk`"
- `petalinux-config -c u-boot`
  - Enable HS200 in "Device Driver" > "MMC Host Controller Support".
- Customizations on the automatic device-tree devices. Changes are done in: "`petalinux/project-spec/meta-user/recipers-bsp/device-tree/files/system-user.dtsi`"
- Root ssh is enabled.

To build a PetaLinux open a terminal on the *petalinux* folder and follow the steps described below.

1. Update the hardware implementation with:

```
$ petalinux-config --get-hw-description=../images/ --silent
INFO: Getting hardware description...
INFO: Rename block_design_wrapper.hdf to system.hdf
[INFO] generating Kconfig for project
[INFO] silentconfig project
[INFO] sourcing bitbake
[INFO] generating plnxtool conf
[INFO] generating meta-plnx-generated layer
[INFO] generating user layers
[INFO] generating machine configuration
[INFO] generating bbappends for project . This may take time !
[INFO] generating u-boot configuration files
[INFO] generating kernel configuration files
[INFO] generating kconfig for Rootfs
[INFO] silentconfig rootfs
[INFO] generating petalinux-user-image.bb
```

2. Build the image with command:

```
$ petalinux-build
[INFO] building project
[INFO] sourcing bitbake
[INFO] generating user layers
INFO: bitbake petalinux-user-image
Loading                                          cache:                                                  100%
|##########################################################################################| Time: 0:00:00
Loaded 3811 entries from dependency cache.
Parsing                                          recipes:                                                100%
|##########################################################################################| Time: 0:00:03
```

```
Parsing of 2778 .bb files complete (2768 cached, 10 parsed). 3813 targets, 146 skipped, 0 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
Initialising                                          tasks:                                                100%
|#########################################################################################################| Time: 0:00:02
Checking            sstate            mirror            object            availability:                   100%
|#########################################################################################################| Time: 0:00:05
Sstate summary: Wanted 907 Found 676 Missed 462 Current 0 (74% match, 0% complete)
NOTE: Executing SetScene Tasks
NOTE: Executing RunQueue Tasks
NOTE: Tasks Summary: Attempted 3250 tasks of which 2223 didn't need to be rerun and all succeeded.
INFO: Copying Images from deploy to images
INFO: Creating images/linux directory
NOTE: copy to TFTP-boot directory is not enabled !!
[INFO] successfully built project
```

3. Generate a new BOOT.bin with command:

```
$ petalinux-package --force --boot --fsbl --pmufw --u-boot --fpga
INFO: File in BOOT BIN: "/.../rf-analyzer/petalinux/images/linux/zynqmp_fsbl.elf"
INFO: File in BOOT BIN: "/.../rf-analyzer/petalinux/images/linux/pmufw.elf"
INFO: File in BOOT BIN: "/.../rf-analyzer/petalinux/project-spec/hw-description/block_design_wrapper.bit"
INFO: File in BOOT BIN: "/.../rf-analyzer/petalinux/images/linux/bl31.elf"
INFO: File in BOOT BIN: "/.../rf-analyzer/petalinux/images/linux/u-boot.elf"
INFO: Generating zynqmp binary package BOOT.BIN...


****** Xilinx Bootgen v2019.1
  **** Build date : May 11 2019-11:15:10
    ** Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.

INFO: Binary is ready.
```

4. The resulting files *BOOT.BIN* and *image.ub* are generated on folder “petalinux/images/linux”.

Run the design

1. Download and install Xilinx RF Analyzer. The application and user guide can be found at https://www.xilinx.com/products/silicon-devices/soc/rfsoc.html#resources
2. Copy the files *BOOT.BIN* and *image.ub* generated on the previous section to the root of a SD card formatted as FAT.
3. Install the SD card on the system and power up the system, as can be seen on Figure 2.
4. Connect the serial port interface (115200 baud, 8N1), identified on Figure 2.
5. Verify the clocks status.
    a. The default user and password of the systems are root:root.
    b. Run the command "`spiclk -s`" and verify that all PLLs are locked. Additional options are listed on Table 6.
    c. The RF clocks are configured to run at 409.6MHz in the boot process. If desired other frequencies described on Table 7 can be used.

Table 6. PLL configuration

Command	Description
`spiclk -h`	List application options
`spiclk -s`	PLL status
`spiclk -i`	Initialize clocks
`spiclk -c /dev/spide1.# <INT>`	Configure PLL # with preset <INT>. # maps to: 1. PLL A, ADC Tile 224 and 225 2. PLL B, ADC Tile 226 and 227 3. PLL C, DAC Tile 228 and 229

Table 7. PLL frequencies presets

Preset ID	Frequency (MHz)	Additional information
0	102.40	Use with internal PLL
1	204.80	Use with internal PLL
2	245.76	Use with internal PLL
3 (default)	409.60	Use with internal PLL
4	491.52	Use with internal PLL
5	737.28	Use with internal PLL
6	1474.56	Use with internal PLL
7	1966.08	Use with external PLL
8	2048.00	Use with external PLL
9	2457.60	Use with external PLL
10	2949.12	Use with external PLL
11	3072.00	Use with external PLL
12	3194.88	Use with external PLL
13	3276.80	Use with external PLL
14	3686.40	Use with external PLL
15	3923.16	Use with external PLL
16	4096.00	Use with external PLL
17	4423.68	Use with external PLL, DAC only
18	4669.44	Use with external PLL, DAC only
19	4915.20	Use with external PLL, DAC only
20	5734.40	Use with external PLL, DAC only
21	5898.24	Use with external PLL, DAC only
22	6144.00	Use with external PLL, DAC only
23	6389.76	Use with external PLL, DAC only
24	6400.00	Use with external PLL, DAC only
25	6553.60	Use with external PLL, DAC only

6. Connect the JTAG interface to the host computer, as can be seen on Figure 1.
7. On the host computer open the application *RF Analyzer*.
8. Connect to the local server and select “MicroBlaze #0” as illustrated on Figure 21. The bitstream can not be loaded on the application due to conflicts with the PLL configuration on Linux.

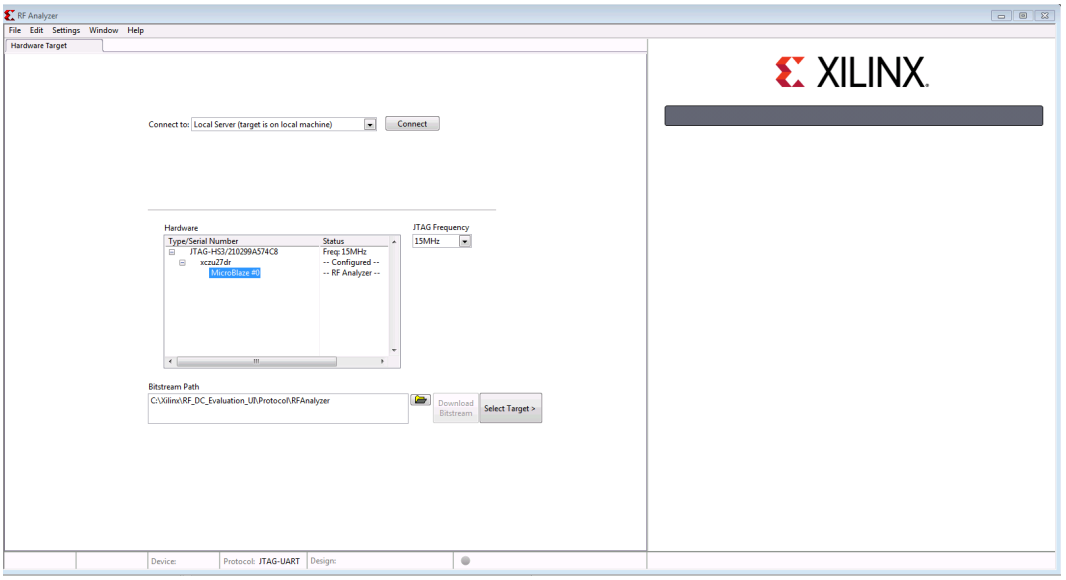

Figure 21. RF Analyzer select target

9. Press “Select Target” and wait for the board information retrieve. After a couple of minutes, the result should be similar to Figure 22 with all status a green.

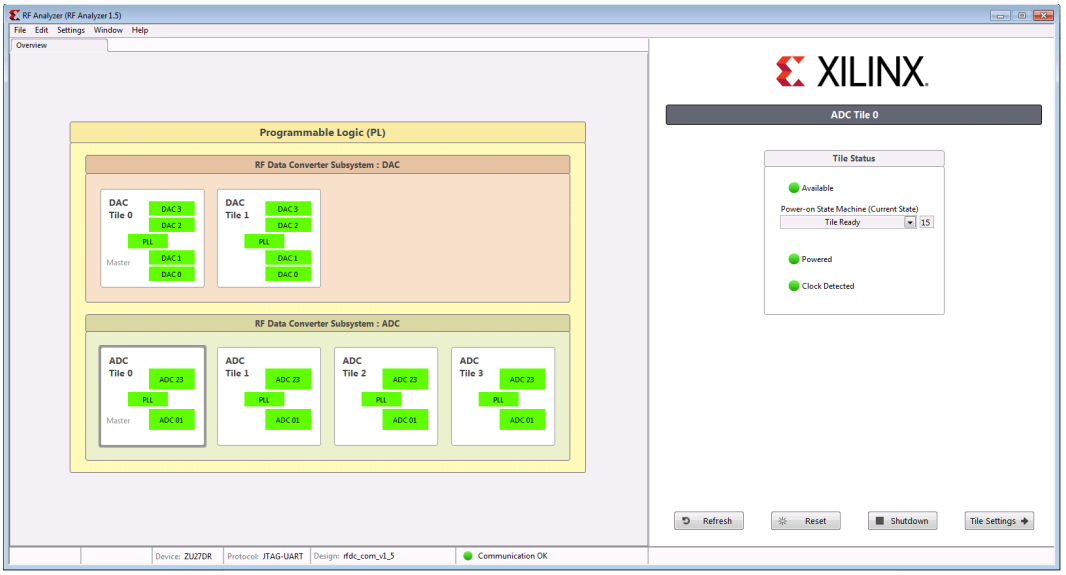

Figure 22. RF Analyzer main interface with ADC Tile 0 selected

10. Configure an external MMCX loop between a DAC and an ADC interfaces as can be seen on Figure 23.
    a. The interfaces are named following Vivado IP cores. The mapping between the interfaces and the RF Analyzer application is detailed on Table 8.

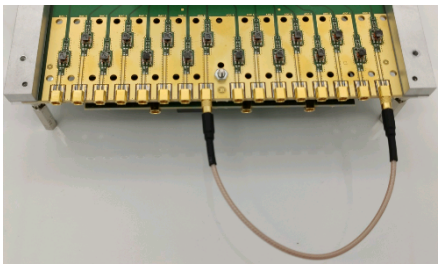

Figure 23. MMCX loopback

Vivado	RF Analyzer
ADC Tile 224	ADC Tile 0
ADC Tile 225	ADC Tile 1
ADC Tile 226	ADC Tile 2
ADC Tile 227	ADC Tile 3
DAC Tile 228	DAC Tile 0
DAC Tile 229	DAC Tile 1

Table 8. RF Interfaces Identifiers

11. Select the desired DAC tile on the RF Analyzer main interface e press “Tile Settings”.
12. As represented on Figure 24 Select one interface a press “Generation” to open a new tab.

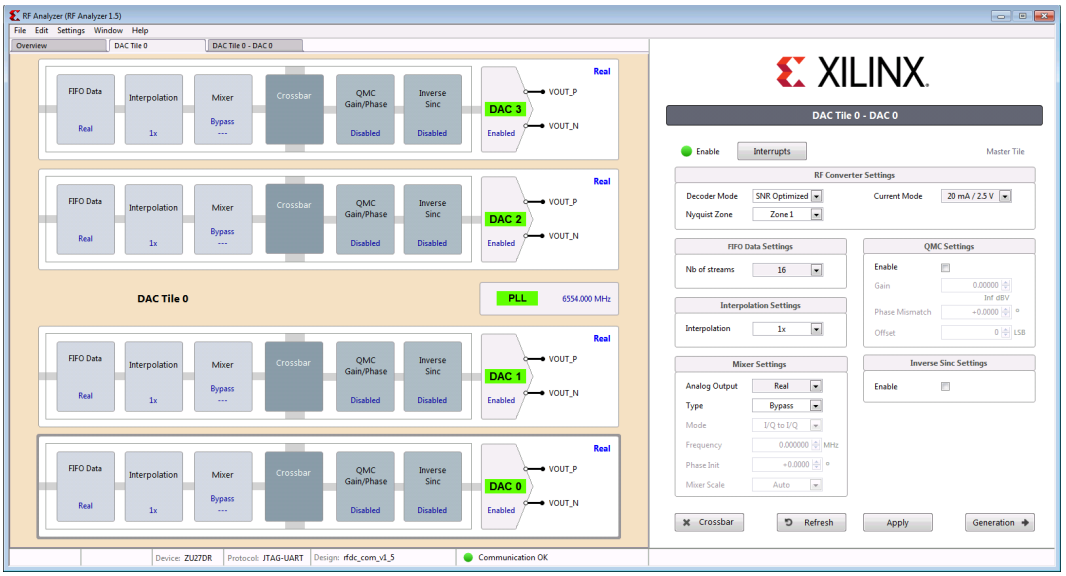

Figure 24. DAC Tile interface

13. Configure one frequency on the interface represented on Figure 25 and press “Generate” to start transmitting the signal.

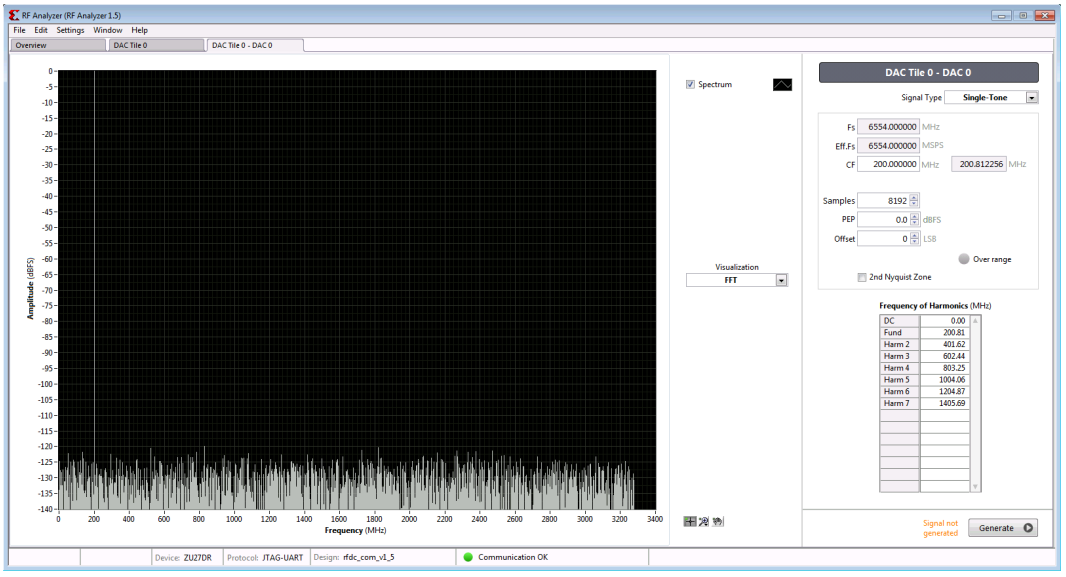

Figure 25. DAC signal generate

14. Return to the main interface and select an ADC tile.
15. On the new tab, illustrated on Figure 26, select one ADC and press “Acquisition“.

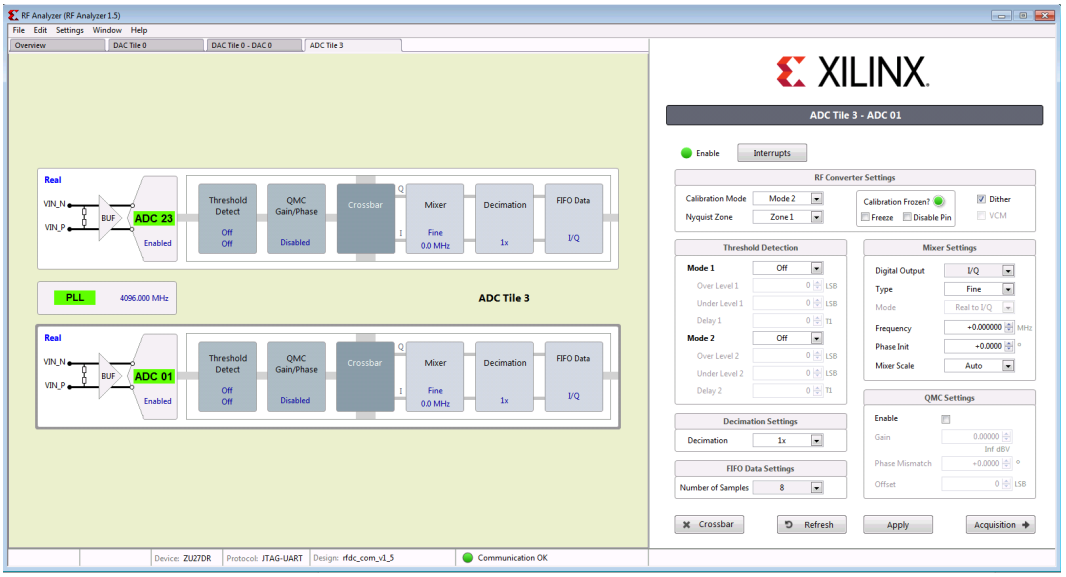

Figure 26. ADC Tile interface

16. On the new interface, represented on Figure 27, enable “loop” and press “Acquire” to start the signal acquisition.

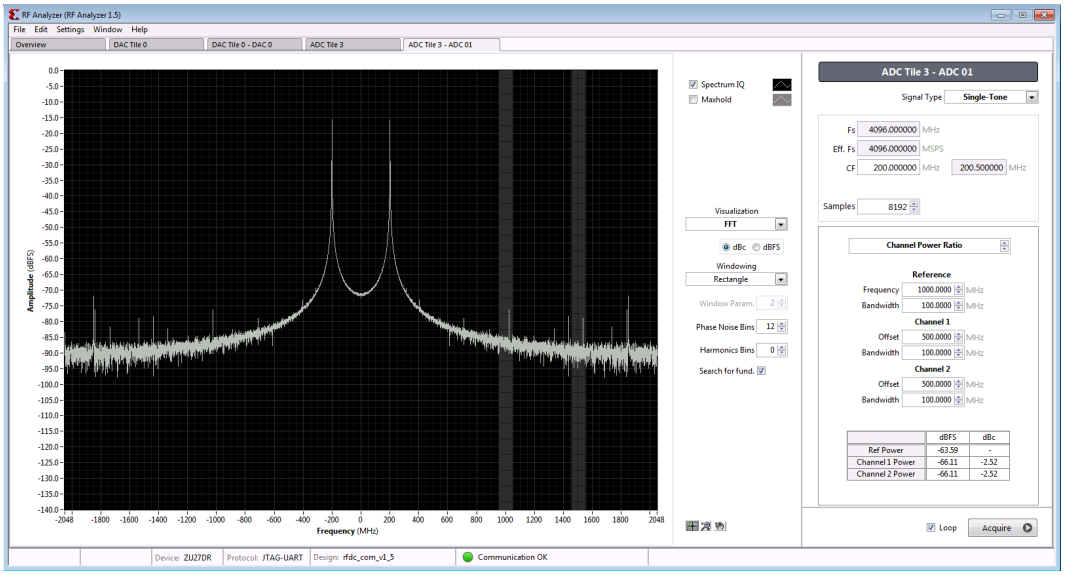

Figure 27. ADC signal acquisition

17. A more detailed description of the application capabilities is available on Xilinx UG1309.

RF design known issues or limitations

a. Ghost frequencies on ADC acquisition

Sometimes there are unexpected frequencies detected on the ADC. For now, the workaround is to “shutdown” and “startup” the tile on the RF main interface represented on Figure 22.

b. Usage of external clock reference

The current LMK04208 configuration only uses the PCB OCXO as reference. The external input is not used.

The PLL registers configurations are loaded from the configuration file “krm4zurf_presets.h” in the *spiclk* PetaLinux application. A different configuration can be generated using Texas Instruments Clocks and Synthesizers (TICS) Pro Software (www.ti.com/tool/TICSPRO-SW).

Appendix

A. Boot PetaLinux from eMMC

This appendix details two methods to flash the onboard eMMC memory with a PetaLinux boot image. The first, described in section "A2. Flashing eMMC from Vivado", only uses Vivado and a JTAG connector but is slow. A faster method, using an initial PetaLinux boot from SDCard, is detailed in section "A3. Flashing eMMC from PetaLinux".

A1. Generating a eMMC boot image

Using a previous PetaLinux project like Chapter V or creating a new one, check the following configuration:

1. The Primary SD/SDIO is set to psu_sd_0:
```
$ petalinux-config
  > Subsystem AUTO Hardware Settings  --->
    > SD/SDIO Settings  --->
      > Primary SD/SDIO (psu_sd_0)  --->
        > (X) psu_sd_0
```

2. The eMMC High Speed support is enabled in in U-Boot:
```
$ petalinux-config -c u-boot
  > Device Drivers  --->
    > MMC Host controller Support  --->
      > [*] enable HS200 support
      > [*] enable HS200 support in SPL
```

3. The rootfs has some basic utilities:
```
$ petalinux-config -c rootfs
  > Filesystem Packages   --->
    > util-linux   --->
      > [*] util-linux
      > [*] util-linux-fsck
      > [*] util-linux-uuidgen
      > [*] util-linux-umount
      > [*] util-linux-mount
      > [*] util-linux-blkid
      > [*] util-linux-mkfs
      > [*] util-linux-mountpoint
      > [*] util-linux-fdisk
      > [*] util-linux-uuidd
```

Build the images with:
```
$ petalinux-build
$ petalinux-package --force --boot --fsbl --pmufw --u-boot --fpga
```

The resulting files *BOOT.BIN* and *image.ub* are generated in the folder "petalinux/images/linux".

A2. Flashing eMMC from Vivado

After creating a boot image:

1. Boot the module in JTAG Mode (*krm4:b0*)
2. Open Vivado Hardware Manager
3. Right-click the device “xczu27_dr(1)” and “Add Configuration Memory Device…” as illustrated in Figure 28.
4. Select “jedec4.51-8gb-emmc” as can be seen on Figure 29.

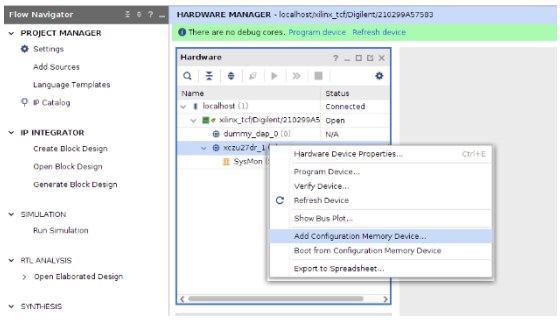

Figure 28. Add Configuration Memory Device…

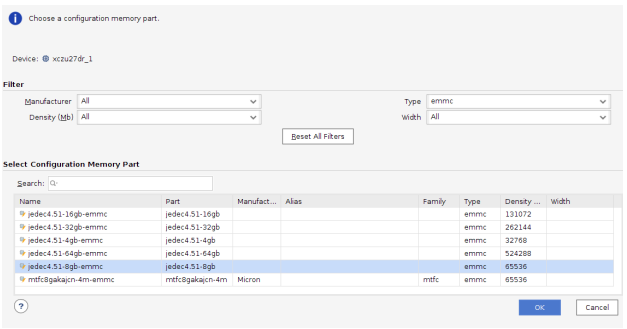

Figure 29. Select eMMC

5. Add the files to be programmed using a Large Partition Size, as represented on Figure 30. The files *BOOT.BIN*, *image.ub* and *zynqmp_fsbl.elf* were previously generated by PetaLinux.

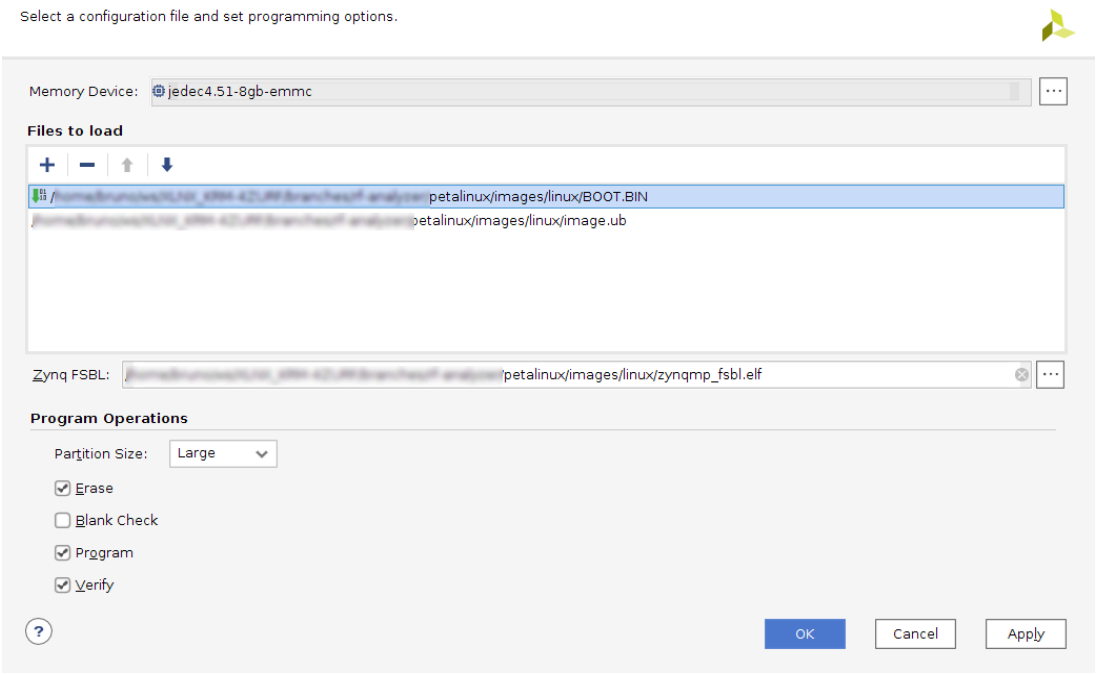

Figure 30. Program Configuration Memory Device

6. Wait until the programing is done (approx. 20min).
7. Reboot the module in eMMC boot mode (*krm4:b6)*.

A3. Flashing eMMC from PetaLinux

After creating a boot image:

1. Copy files *BOOT.BIN* and *image.ub to an SDCard* formatted as FAT32.
2. Insert the SDCard in the carrier and boot the module in SD1 Mode (*krm4:b5*)
3. After U-Boot fails to read file *image.ub* form the eMMC, change the boot device to the SD Card:
   ```
   ZynqMP> env set sdbootdev 1
   ZynqMP> boot
   ```

4. Verify if the eMMC (/dev/mmcblk0p1) is already mounted.
   ```
   $ df -h
   Filesystem                Size      Used Available Use% Mounted on
   devtmpfs                  1.8G      4.0K      1.8G   0% /dev
   tmpfs                     1.9G    100.0K      1.9G   0% /run
   tmpfs                     1.9G    108.0K      1.9G   0% /var/volatile
   /dev/mmcblk0p1            7.3G      4.0K      7.3G   0% /run/media/mmcblk0p1
   /dev/mmcblk1p1           29.7G    238.9M     29.5G   1% /run/media/mmcblk1p1
   ```

5. Format the eMMC if /dev/mmcblk0p1 is not mounted or to start with a clean device.
   a. Unmount the partition, if needed.
      ```
      $ umount /run/media/mmcblk0p1/
      ```

   b. Clear the eMMC.
      ```
      $ sfdisk --delete /dev/mmcblk0
      ```

   c. Create a new boot partition. This example uses all the space in the device but other configurations are also possible.
      ```
      $ echo " , , c, *" | sfdisk -a /dev/mmcblk0
      ```

   d. Format the partition as FAT32.
      ```
      $ mkfs.vfat -F 32 -n boot /dev/mmcblk0p1
      ```

   e. Trigger an automatic detection that will mount the device.
      ```
      $ echo ff160000.mmc > /sys/bus/platform/drivers/sdhci-arasan/unbind;
      $ sleep 1;
      $ echo ff160000.mmc > /sys/bus/platform/drivers/sdhci-arasan/bind
      ```

6. Copy files from the SDCard(/dev/mmcblk1p1) to the eMMC (/dev/mmcblk0p1):
   ```
   $ cp /run/media/mmcblk1p1/BOOT.BIN /run/media/mmcblk0p1/
   $ cp /run/media/mmcblk1p1/image.ub /run/media/mmcblk0p1/
   ```

7. Reboot the module in eMMC boot mode (*krm4:b6)*.