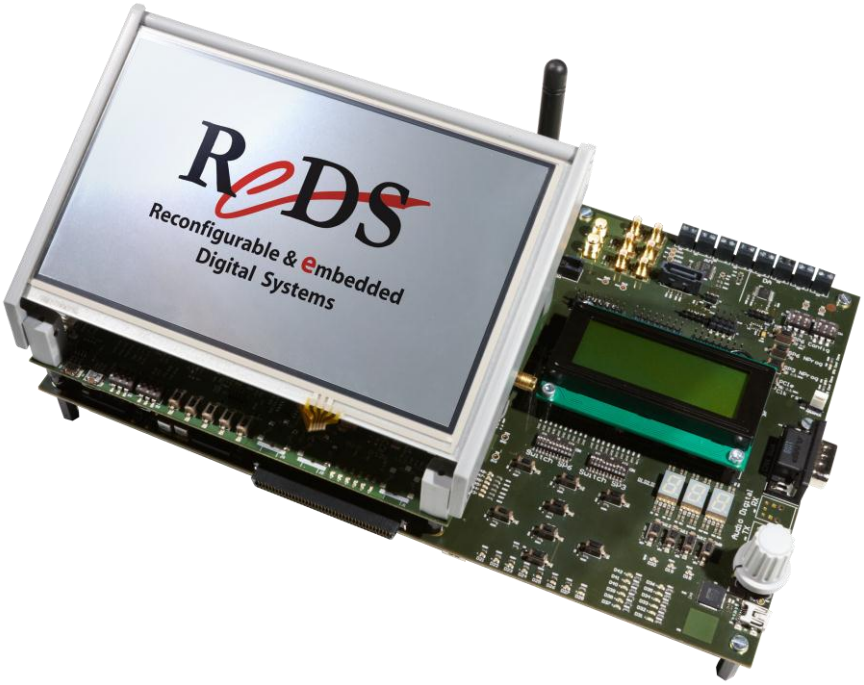
heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

ReDS
Reconfigurable & embedded
Digital Systems

REPTAR

Reconfigurable **E**mbedded **P**latform for **T**raining **A**nd **R**esearch

Quick Start Guide



REDS Institute
March 2013
Ver 1.0

Table of Content

1. INTRODUCTION 5

2. GENERAL DESCRIPTION 6

2.1 FPGA BOARD OVERVIEW 7
2.2 CPU BOARD OVERVIEW 8

3. EQUIPMENT NEEDED 9

3.1 MINIMAL REQUIREMENTS 9
3.2 OPTIONAL REQUIREMENTS 9
3.2.1 For development in the embedded processor 9
3.2.2 For development in the FPGA Spartan6 (without embedded processor usage) 9
3.2.3 For development in the FPGA Spartan3 9

4. SOFTWARE AND FILES NEEDED 10

4.1 FOR DEVELOPMENT IN THE EMBEDDED PROCESSOR 10
4.2 FOR DEVELOPMENT IN THE FPGA SPARTAN6 OR IN THE FPGA SPARTAN3 10

5. DOCUMENTS NEEDED 11

6. MAIN PROCEDURES STEP-BY-STEP 12

6.1 SETUP OF THE BOARD 12
6.1.1 Jumpers 12
6.1.2 DIP switches on FPGA board 14
6.1.3 DIP switches on CPU board 15
6.2 PROCEDURES FOR DEVELOPMENT IN THE EMBEDDED PROCESSOR WITH OS OR RTOS 16
6.2.1 Getting started 16
6.2.2 Using U-boot 16
6.2.3 Preparing a "Master" SD card 21
6.2.4 Booting from SD card 22
6.2.5 reptar_soft compilation 22
6.3 PROCEDURES FOR DEVELOPMENT IN THE EMBEDDED PROCESSOR WITHOUT OS 25
6.4 PROCEDURES FOR DEVELOPMENT IN THE FPGA 25
6.4.1 Using the REPTAR base project 25
6.4.2 Using the REPTAR standard project 26
6.4.3 ISE project compilation 27
6.4.4 Spartan3 programming 27
6.4.5 Spartan6 programming 30
6.4.6 Platform Flash programming 31

6.5 PROCEDURES FOR MIX DEVELOPMENT USING EMBEDDED PROCESSOR AND FPGA 34
6.5.1 Board setup 34
6.5.2 Using the Local Bus 35
6.5.3 Using the REPTAR standard ISE project 36
6.5.4 Programming the SP6 from the CPU 37

7. TROUBLESHOOTING 40

8. ADDITIONAL INFORMATION 41

8.1 REVISION HISTORY 41
8.2 CONTACT 41

Figures Table

FIGURE 1 - FPGA BOARD DETAILS 7
FIGURE 2 - CPU BOARD DETAILS 8
FIGURE 3 – JUMPERS ON CPU BOARD 12
FIGURE 4 – JUMPERS ON FPGA BOARD 13
FIGURE 5 – DIP SWITCHES ON FPGA BOARD 14
FIGURE 6 – DIP SWITCHES ON CPU BOARD 15
FIGURE 7 – REPTAR FPGA, SP6 AND SP3 CONFIGURATION LEDS AND BUTTONS 28
FIGURE 8 – REPTAR FPGA, JTAG HEADERS AND SMT MODULE CONNECTOR 29

1. INTRODUCTION

This document provides the most important information needed to start using the REPTAR board.

The REPTAR datasheet and the reference manual provide a detailed description of the board and it's strongly recommended to read both documents before this guide.

This guide applies only to the Proto II and Series I REPTAR board versions.

2. GENERAL DESCRIPTION

The REPTAR board was designed in the REDS Institute of the HEIG-VD in 2012/2013.

REPTAR combines an OMAP type processor (which itself consists of a cortex-A8 ARM and a DSP) with a programmable logic component (FPGA) Xilinx Spartan 6.

The platform also includes a large number of control, display and communication devices.

The platform may be used in different ways:

- Use of the embedded processor with OS or RTOS without using the FPGA (the peripherals connected to the FPGA are not used)
- Use of the embedded processor without OS and without using the FPGA (the peripherals connected to the FPGA are not used)
- Use of the FPGA without using the embedded processor (the peripherals connected to the embedded processor are not used, and the CPU board may not be present)
- Co-design or Mix development using embedded processor and FPGA (all peripherals on the board are accessible)

REPTAR is a board that offers enough flexibility to customize your development environment via many expansion connectors, I/O and daughter cards.

The REPTAR board is made of two boards: The CPU and the FPGA board

The FPGA board is considered as the mainboard of the REPTAR system. All the power supplies are located on this board. The CPU board is then considered as a daughter card for the system itself.

2.1 FPGA BOARD OVERVIEW

The picture below shows the main features of the REPTAR FPGA board.

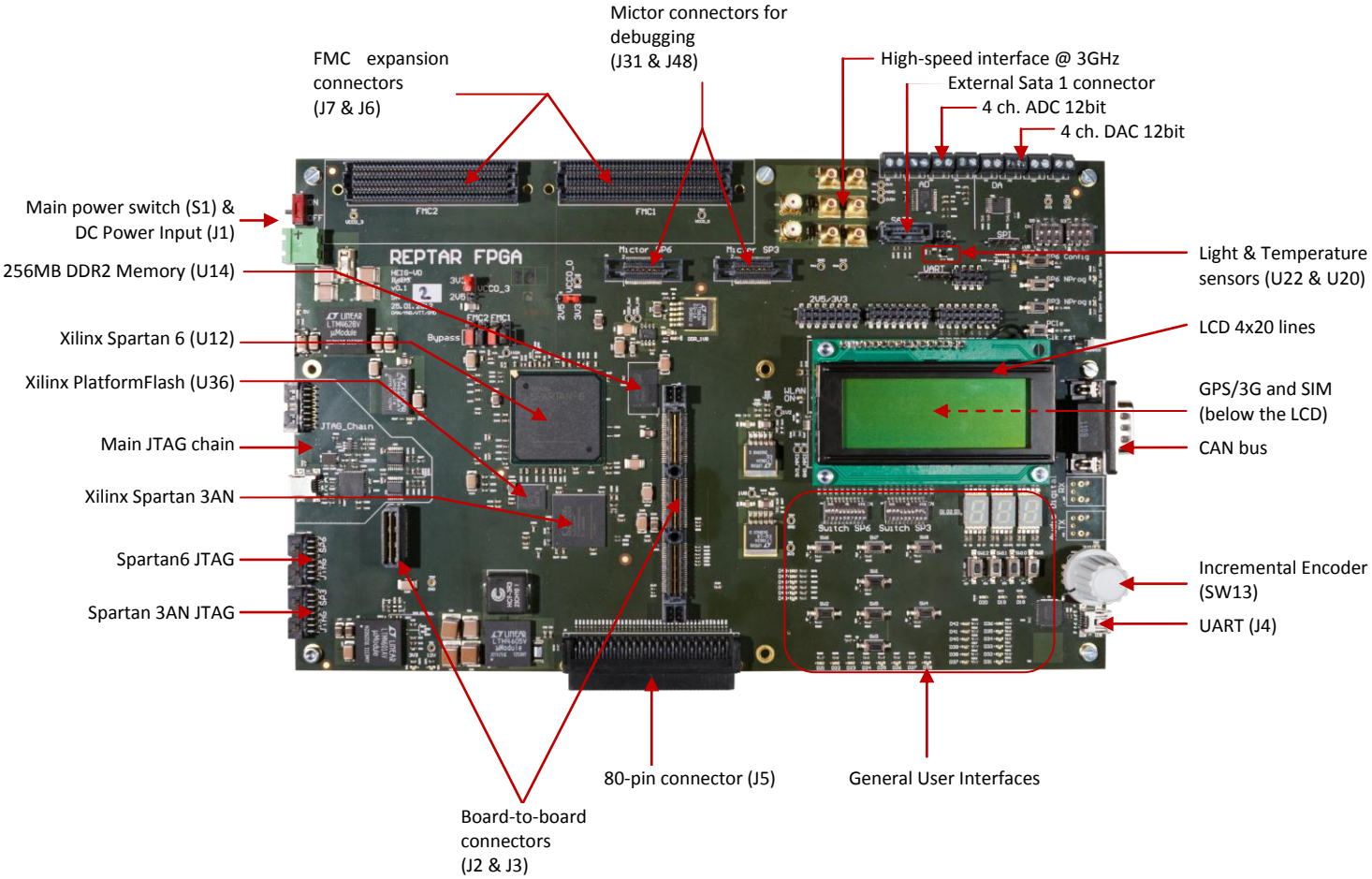


Figure 1 - FPGA Board details

2.2 CPU BOARD OVERVIEW

The picture below shows the main features of the REPTAR CPU board.

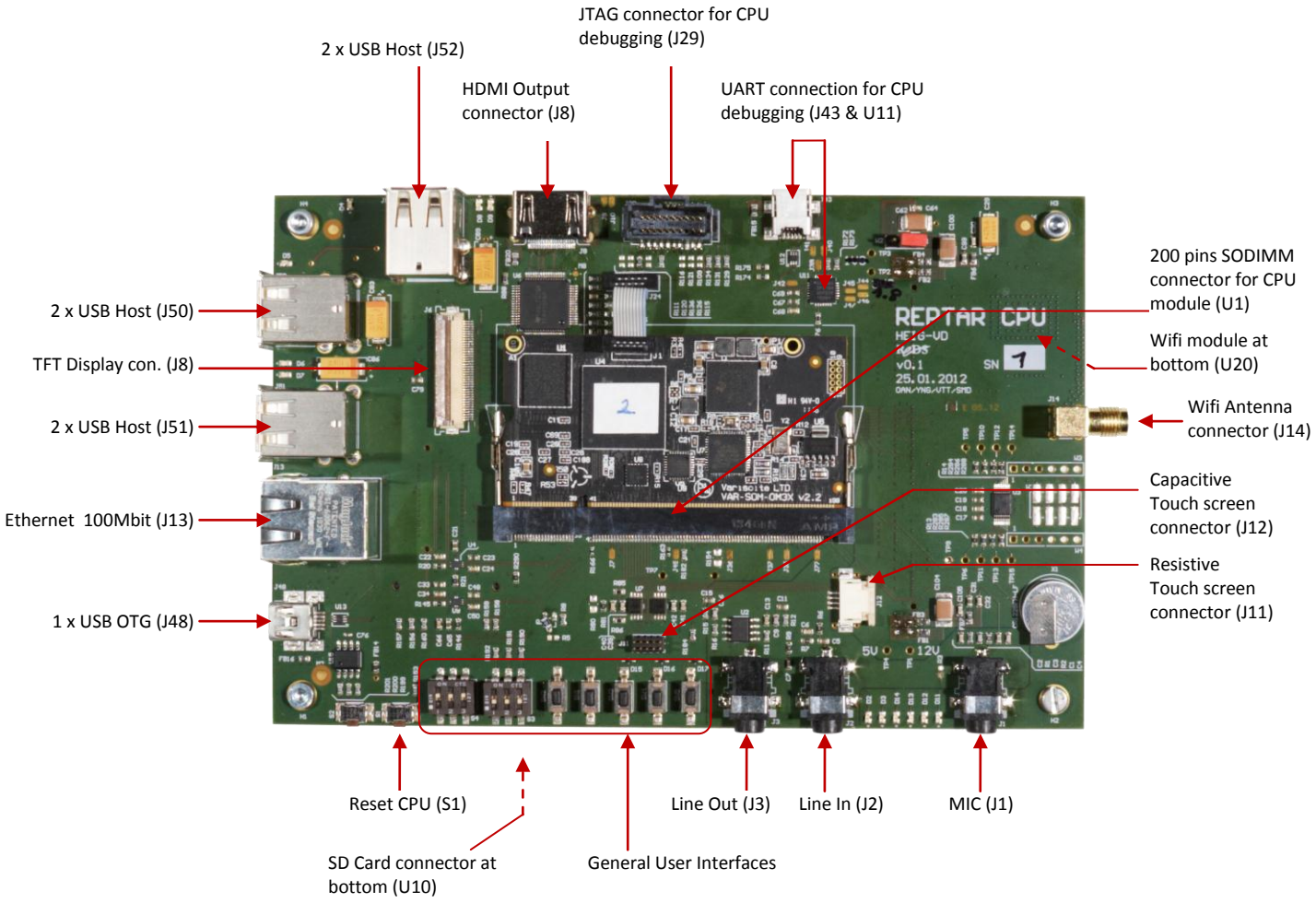


Figure 2 - CPU Board details

3. EQUIPMENT NEEDED

The list below give you a detail of the equipment needed for doing embedded development with REPTAR.

3.1 MINIMAL REQUIREMENTS

- ✓ REPTAR board
- ✓ DC power supply 12V – 3A
- ✓ 2 x banana-plug cables

3.2 OPTIONAL REQUIREMENTS

The optional requirements depend on how you use the board.

3.2.1 For development in the embedded processor

- ✓ PC with USB ports
- ✓ mini USB cable (for UART communication with the processor)
- ✓ Optional: SIM card

With OS

- ✓ SD card (to store the File System and the kernel or only the File System)
- ✓ SD card reader
  Or
- ✓ PC with network cards
- ✓ Ethernet cable (to use NFS, Network File System)

Without OS

- ✓ Blackhawk USB100v2 JTAG emulator (for ARM debug) + mini USB cable

3.2.2 For development in the FPGA Spartan6 (without embedded processor usage)

- ✓ Xilinx Platform Cable USB or USBII
  Or
- ✓ Micro USB cable

3.2.3 For development in the FPGA Spartan3

- ✓ Xilinx Platform Cable USB or USBII

4. SOFTWARE AND FILES NEEDED

The software required depends on how you use the board.

4.1 FOR DEVELOPMENT IN THE EMBEDDED PROCESSOR

With OS

- ✓ PC running Linux
- ✓ For development on x-loader, u-boot or Linux kernel: REPTAR BSP available on the GIT repository *reptar_soft*
- ✓ For SD card image creation: Packages python2, python-parted, uboot-mkimage
- ✓ Rootfs archive (available on http://eigit.heig-vd.ch/public/rootfs/reptar/ or on SD card)
- ✓ Optional: NFS server for remote loading of kernel and/or rootfs
- ✓ Optional: tftp server for remote loading of kernel and/or rootfs
- ✓ Optionnal: Examples, demos and tests programs available on the GIT repository *reptar_usr*

Without OS

- ✓ Code Composer Studio 5 available on the net or at \\eistore0\Softs-REDS\Embedded\CodeComposerStudio

- ✓ Toolchain arm-2011.03-42-arm-none-eabi available on the net or at \\eistore0\Softs-REDS\Board_Support_Package\Toolchains

- ✓ Tests programs available on the GIT repository *reptar_usr* folder *\tests\standalone*

4.2 FOR DEVELOPMENT IN THE FPGA SPARTAN6 OR IN THE FPGA SPARTAN3

- ✓ Xilinx ISE design suite 13.3
- ✓ Standard VHDL descriptions and ISE project for REPTAR

5. DOCUMENTS NEEDED

- Components datasheets, design guidelines, standards, etc. provided by the manufacturers or found on the net.
- Documents created by the REPTAR developers: REPTAR presentation, datasheet, Reference Manual, board schematics, U-boot Tests Details, Linux Tests Details, this guide

You can find additional information on the *wiki* web page at (no public access):
https://eigit.heig-vd.ch/projects/reptar/wiki/Wiki

Specific information about REPTAR software:
https://eigit.heig-vd.ch/projects/reptar/wiki/Software

Specific information about REPTAR hardware:
https://eigit.heig-vd.ch/projects/reptar/wiki/Hardware

6. MAIN PROCEDURES STEP-BY-STEP

6.1 SETUP OF THE BOARD

6.1.1 Jumpers

Before using REPTAR, please verify that the jumpers are placed like on the images bellow (shown by red, blue or black squares):

REPTAR CPU

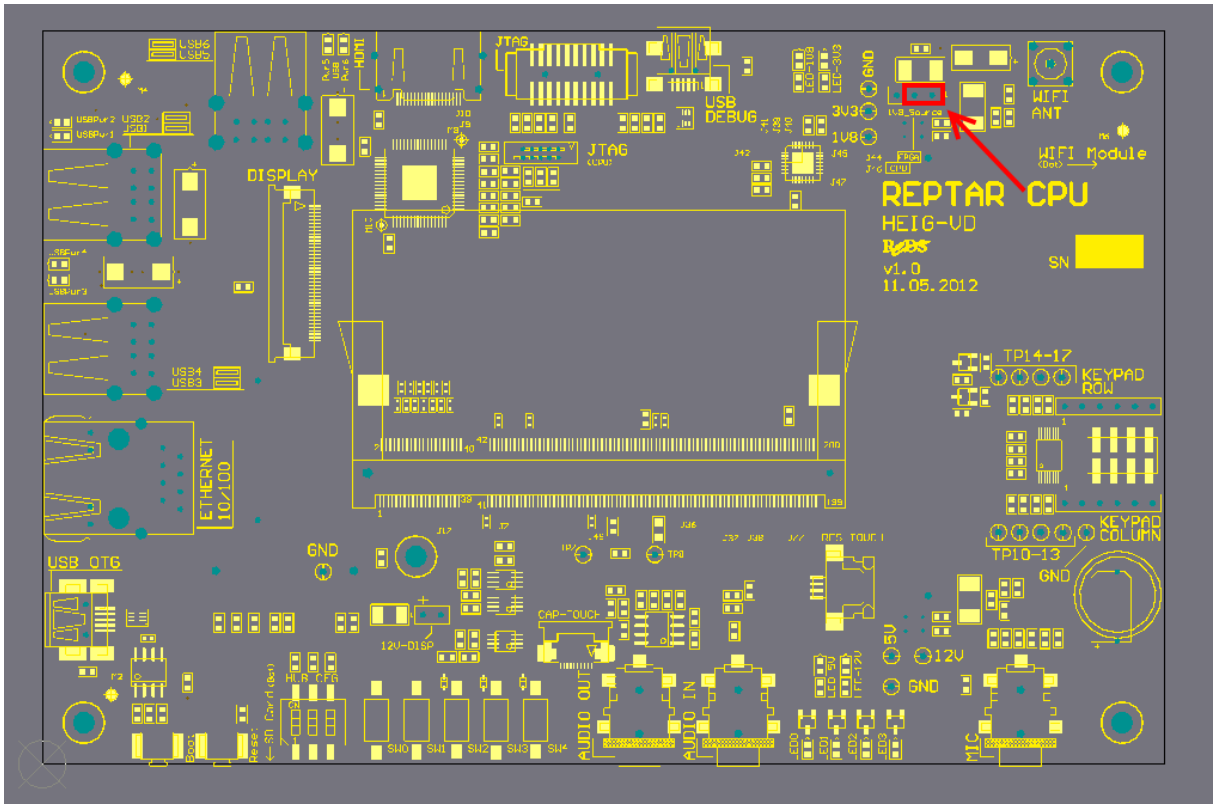


Figure 3 – Jumpers on CPU Board

REPTAR FPGA

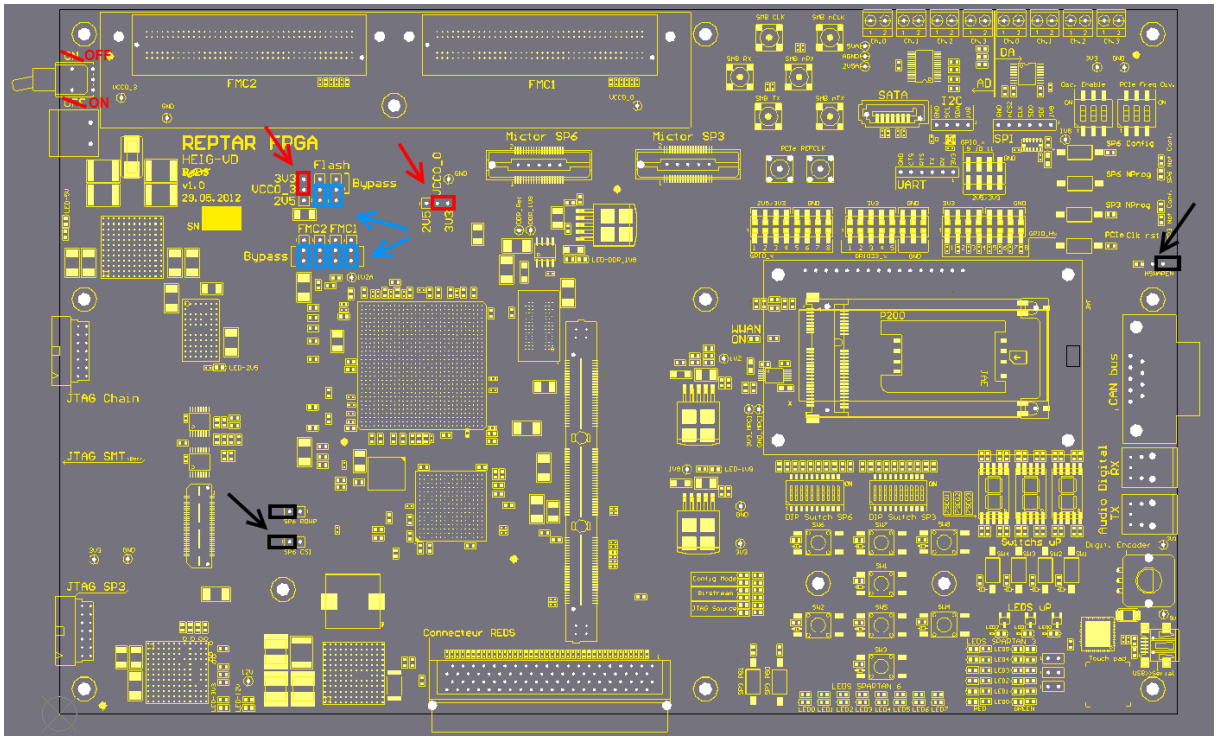


Figure 4 – Jumpers on FPGA Board

6.1.2 DIP switches on FPGA board

Locate the different DIP switches on the image below, and then verify that each one is in the correct position.

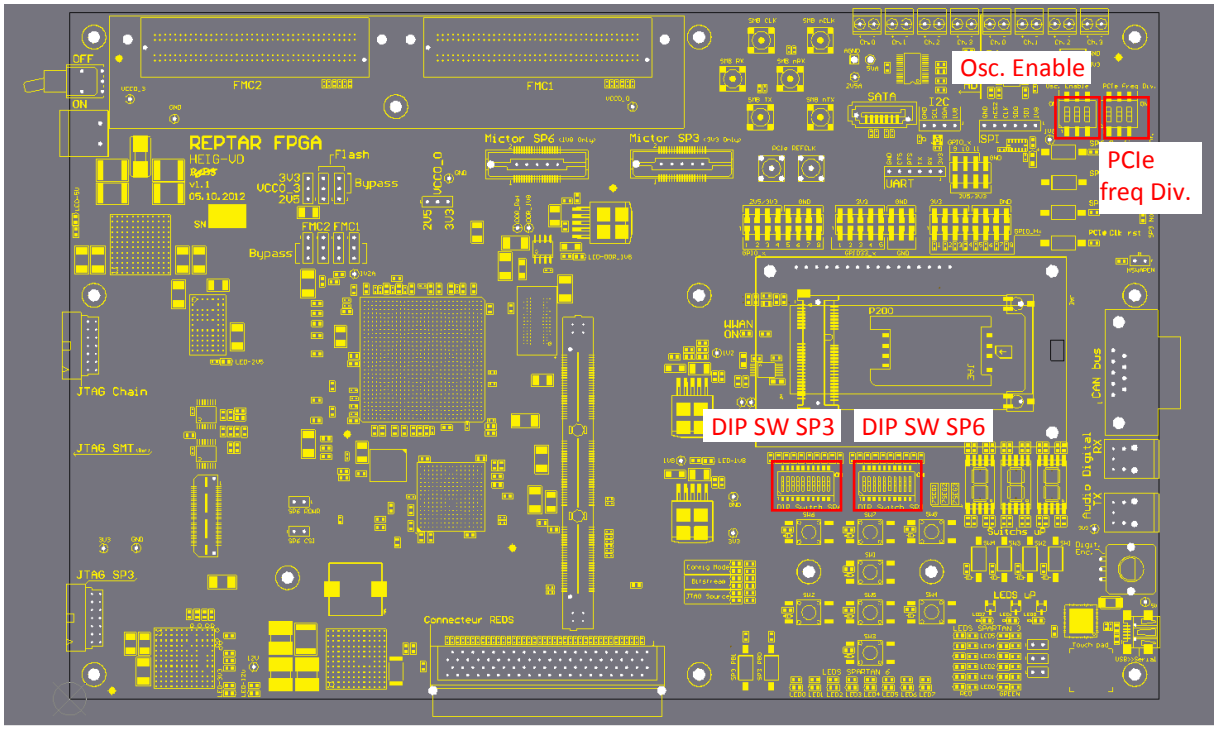


Figure 5 – DIP switches on FPGA Board

You can find an explanation about the DIP switches configuration in the document DIP_Switch_Description.xlsx

Notice that there is a page for FPGA board and another one for the CPU board.

6.1.3 DIP switches on CPU board

Locate the different DIP switches on the image below, and then verify that each one is in the correct position.

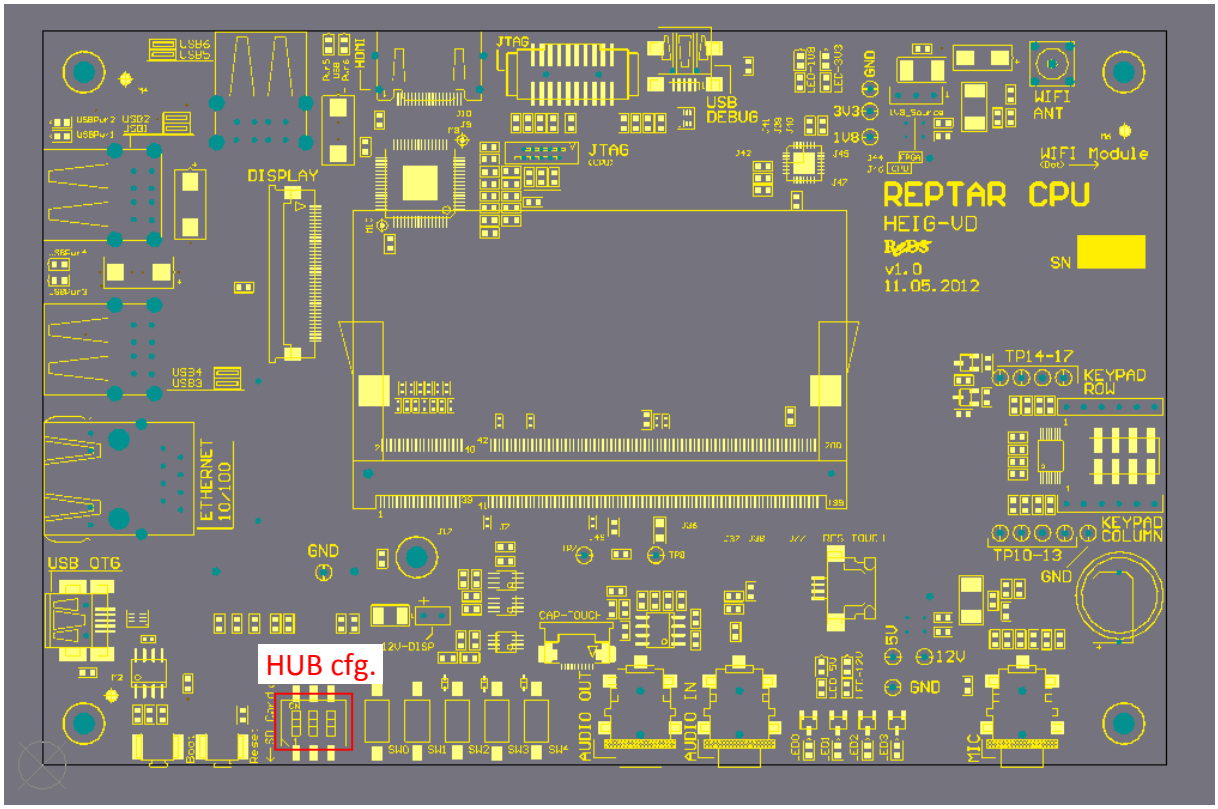


Figure 6 – DIP switches on CPU Board

You can find an explanation about the DIP switches configuration in the document DIP_Switch_Description.xlsx

Notice that there is a page for FPGA board and another one for the CPU board.

6.2 PROCEDURES FOR DEVELOPMENT IN THE EMBEDDED PROCESSOR WITH OS OR RTOS

6.2.1 Getting started

When working in the embedded processor with Linux, the most common way to use REPTAR is loading the *rootfs* from an SD card and the Linux kernel image from NAND or from the same SD card.

1. Connect the mini-USB cable between the REPTAR board and your PC. See the Figure 2 to find the UART connector on the REPTAR CPU board.
2. Insert an SD card containing a kernel image and *rootfs* in the reader of the REPTAR CPU board.
3. Set-up your power supply to 12V/2A and plug the banana wires to the board.
4. Turn on briefly the board by using the ON/OFF switch. The USB port used is detected by the PC.
5. Open a terminal on your PC and run a serial emulator like *picocom* with the parameters: 115200 bauds, 8-bit data, 1 stop bit, no parity, and no flux control. Example with picocom

```
sudo picocom –b 115200 /dev/ttyUSB0
```

6. Turn on the board, U-boot is loaded from NAND and a prompt is shown in the UART terminal

```
Reptar#
```

7. If you want to load the kernel image from the SD card, type simply

```
boot
```

8. Otherwise, to load kernel from the NAND, type

```
run boot_failsafe
```

9. For other boot modes see explanations below.
10. For resetting the CPU module, press the “Reset” button of the REPTAR CPU board. This button is not related to the FPGA board.

6.2.2 Using U-boot

U-boot is a universal boot-loader with full source code under GPL. You can find a lot of information about it on the net.

In the frame of the REPTAR board it’s used principally to load Linux kernel image and rootfs.

It also provides a simple mean to access memory in a debugging goal. You can, for instance, read or write the DM3730 registers or, if you are using the FPGA you can access the Spartan6 and Spartan3 registers.

To show a list of all commands available within U-boot just type

```
help
```

The `itbok` command available in the u-boot environment allows the test of many peripherals connected to the CPU or the FPGA. You can find additional information about the tests under U-boot in the document "*uboot_tests_details*"

The commands concerning the environment variables, the boot modes and the memory access are explained hereafter.

6.2.2.1 Environment variables

The U-Boot environment is a block of memory stored in the flash and copied to RAM when U-Boot starts. It is composed of environment variables which can be used to configure the system and to group several commands together in order to invoke them easily.

Some environment variables have a special meaning for u-boot, but you can also create your own as you like.

Variables having a special meaning for u-boot, and used in the REPTAR standard environment are:
baudrate, bootargs, bootcmd, bootdelay, ethaddr, ipaddr, loadaddr, netmask, nfsroot, root, serverip, setargs, stderr, stdin, stdout

All other variables in the u-boot environment of the REPTAR board were created by the REPTAR developers.

When you modify the standard uboot variables, be aware when you save them, because you can make the board not bootable anymore!

To show the environment variables type:

```
printenv
```

To set an environment variable type:

```
setenv [variable name] [value]
```

To erase an environment variable type:

```
setenv [variable name]
```

To save the changes on the environment (permanently on flash):

```
saveenv
```

6.2.2.2 *Booting from a server*

As explained before, the usual ways to boot the kernel are from the NAND or from the SD card. When debugging your code it may be useful to boot from an NFS or tftp server installed on your machine. This avoids flashing the REPTAR CPU board too many times and extends its NAND lifetime. This method is also less annoying than rewrite an SD card each time that the code is modified.

First, ensure you that tftp and/or NFS servers are installed on your host machine.

For booting from tftp server:

```
tftpboot 0x80000000 uImage
```

For booting from NFS server:

```
nfs 0x80000000 ${serverip}:/export/kernel/uImage
```

If you want to change the way the system boots by default, you must create environments variables for each mode and then modify the *bootcmd* variable.

Example:

```
setenv tftp_kernel "tftpboot 0x80000000 uImage"
setenv bootcmd "run tftp_kernel"
saveenv
```

6.2.2.3 *Loading the file system from a server*

As explained before you can load the file system from the SD card. When debugging your code it may be useful to boot from an NFS server installed on your machine. This method may be faster than rewrite an SD card.

First, ensure you that tftp and/or NFS servers are installed on your host machine. Then copy your *rootfs* in the folder /export/fs and the kernel image uImage in the tftp folder.

For loading the rootfs from NFS server and the kernel from tftp:

```
setenv nfsroot /export/fs
setenv setargs_nfs 'setenv bootargs ${bootargs_common}
                root=/dev/nfs nfsroot=${serverip}:${nfsroot}'
setenv boot_net 'run setmac setip;run tftp_kernel;run
                setargs_nfs addmac addip;bootm ${loadaddr}'
saveenv
```

If you want to change the way the system boots by default, you must create environments variables for each mode and then modify the *bootcmd* variable.

Example:

```
setenv bootcmd "run boot_net"
saveenv
```

If you want manually boot from server, just type:

```
run boot_net
```

6.2.2.4 Useful commands for debugging

Here is an explanation of two useful commands for memory access.

md

```
md [.b, .w, .l] address [ # of objs ]
```

The `md` command displays memory contents both as hexadecimal and ASCII data. The last displayed memory address and the value of the count argument are remembered, so when you enter md again without arguments it will automatically continue at the next address, and use the same count again.
If invoked as `md` or `md.l`, data is displayed as 32-bit long words. If invoked as `md.w` or `md.b` instead, 16-bit words or 8-bit bytes are used, respectively.

Arguments

address
memory location to display
objs
number of objects to display

mw

```
mw [.b, .w, .l] address value [count]
```

The `mw` command writes a `value` to a specific `address`.
When called without a `count` argument, the `value` will be written only to the specified `address`. When used with a `count`, then a memory area beginning at `address` will be initialized with this `value`.
If invoked as `mw` or `mw.l`, data is 32-bit long. If invoked as `md.w` or `md.b` instead, 16-bit words or 8-bit bytes are used, respectively.

Arguments

address
first memory location to fill
value
data to write
count
number of memory locations to write

6.2.3 Preparing a "Master" SD card

This section explains how to create an SD card containing all the binaries (X-loader, U-boot and uImage) and the file system (rootfs) needed for running Linux on REPTAR, as well as the programs and scripts used for tests.

Clone the *reptar_soft* repository in your machine, open a command-line interface terminal and change the directory to *reptar_soft*.

1. Compile the REPTAR project by invoking the `make` command, see the reptar_soft compilation section if needed. Your *root* password will be asked several times.
2. Insert an SD card in the reader of your PC.
3. If the SD card is automatically mounted, demount-it manually with
   ```
   sudo umount /media/<mount_point>
   ```
4. Get the device name associated to the SD card (i.e. /dev/sdb) with `dmesg | tail`. This name is necessary for the next step.
   Example:
   ```
   $ dmesg | tail
   …
   [255069.569920] sd 15:0:0:0: [sdb] 7862272 512-byte logical blocks: (4.02 GB/3.74 GiB)
   [255069.571112] sd 15:0:0:0: [sdb] Assuming drive cache: write through
   [255069.572853] sd 15:0:0:0: [sdb] Assuming drive cache: write through
   [255069.572859]  sdb: sdb1
   ```

   Make sure of use the correct device name, you risk trashing your PC by using the wrong device!
5. Start the copy of the SD image on the SD card with the device name got in the previous step. This operation may last several minutes. Wait for the end of the copy: results are shown on the screen and the shell prompt appears.

   ```
   sudo dd if=build/sd-card.img of=/dev/sdb bs=5M
   ...
   2048+0 records in
   2048+0 records out
   2147483648 bytes (2.1 GB) copied, 270.365 s, 7.9 MB/s
   ```

6. Remove the SD card from the slot and then insert-it again. Two partitions must be mounted automatically:
   /media/boot: boot FAT partition with MLO, u-boot.bin and uImage files
   /media/filesystem: ext3 partition with the *rootfs* tree structure
7. Take the REPTAR bitstream for SP6 from the *reptar* repository at *reptar/public/test/FPGAs_Flashing* and copy-it manually in the partition /media/boot
8. Demount both partitions before removing the SD card from the reader
9. Your SD card is now ready to be used on the REPTAR board

6.2.4 Booting from SD card

1. Connect the mini-USB cable between the REPTAR board and your PC. See the Figure 2 to find the UART connector on the REPTAR CPU board.
2. Insert an SD card containing a kernel image and *rootfs* in the reader of the REPTAR CPU board (REPTAR "Master" SD card).
3. Set-up your power supply to 12V/2A and plug the banana wires to the board.
4. Turn on briefly the board by using the ON/OFF switch. The USB port used is detected by the PC.
5. Open a terminal on your PC and run a serial emulator like *picocom* with the parameters: 115200 bauds, 8-bit data, 1 stop bit, no parity, and no flux control. Example with picocom:

```
sudo picocom –b 115200 /dev/ttyUSB0
```

6. Press the "Boot" button of the REPTAR board.
7. Without release the button, turn-on the board. U-boot is loaded and a prompt is shown in the UART terminal

```
Reptar#
```

10. Release the "Boot" button.
11. Verify that X-loader and U-boot have been loaded from the SD card by looking the messages on the console

```
Texas Instruments X-Loader 1.51 (Jul 30 2012 - 13:19:13)
Starting X-loader on MMC
Reading boot sector
370200 Bytes Read from MMC
Starting OS Bootloader from MMC...
Starting OS Bootloader...


U-Boot 2011.09-00000-g8c918f0-dirty (Jul 30 2012 - 13:19:39)
…
```

6.2.5 reptar_soft compilation

6.2.5.1 Toolchain installation

The toolchain currently used for the cross-compilation is Linaro 2013.01, available at http://eigit.heig-vd.ch/public/toolchains/

Once you have cloned the git repository reptar_soft, you can run a script to automatically download and install the toolchain on your PC. The script can be found at the *reptar_soft/scripts* folder.

To run the script:

```
reptar_soft$./scripts/reptar_toolchain.sh
```

By default, the toolchain is installed in the */opt* folder, if you want to install it to another path you can specify it like this:

```
reptar_soft$ ./scripts/reptar_toolchain.sh /my/path
```

Verify that the toolchain is well installed by taping in a shell:

```
$ arm-none-linux-gnueabi-gcc -v
```

The last line of the output must be:

```
gcc version 4.7.3 (prerelease) (crosstool-NG linaro-1.13.1-4.7-2013.01-20130125 - Linaro GCC 2013.01)
```

If the command is not found, update your PATH environment variable by adding the folder {toolchain_path}/gcc-linaro-arm-linux-gnueabihf-4.7-2013.01-20130125_linux/bin.

6.2.5.2 Make command

When you invoke the `make` command from the *reptar_soft* folder, a complete compilation is done (x-loader, uboot, kernel) and an SD card image is built with two partitions: the boot partition containing all the boot binaries, and a filesystem partition containing the rootfs with the modules installed.

In order to build an SD card image you will need a PC running Linux and the following packages should be installed:

- ✓ Python2 (2.6 or 2.7
- ✓ python-parted
- ✓ u-boot-mkimage

The make program uses the file system by default which is automatically downloaded from http://eigit.heig-vd.ch/public/rootfs/reptar/. This server holds also many other rootfs archives used on the REPTAR board for demos, tests, labs and other purposes.
The default rootfs is set by the variable ROOTFS_ARCHIVE in the *Makefile* found at the top level of the *reptar_soft* repository. Currently, the default rootfs is based on Buildroot, and the archive used is **rootfs_br_2013v1.tar.bz2**

All the boards where fully tested with a rootfs based on ArchLinux its name is **ArchLinux-filesystem-tests.tar.bz2**. This is why the access to all the peripherals and the well working of every module of the board is guaranteed only with this rootfs. All the tests will be soon supported by the new rootfs.

If you want to compile the project with a version of filesystem different from the default version:

- ✓ If the archive is on the *eigit* server, just compile passing the variable ROOTFS_ARCHIVE=<*name_of_your_filesystem*> as argument.

- ✓ If you want to use a locale version, copy manually the archive to reptar_soft/filesystem and then compile passing the variable ROOTFS_ARCHIVE =<*name_of_your_filesystem*> as argument.

Whenever you compile, the rootfs gift by the ROOTFS_ARCHIVE variable is search first locally at the *reptar_soft/filesystem* folder, and if it is not present, then it is search on the *eigit* server.

To compress a rootfs keeping the symbolic links:

- ✓ from the command-line change the directory to the filesystem root and then type

```
sudo tar –cjpf ../<archive_filename>.tar.bz2 *
```

To copy a new file on the server from your Linux PC, open a file explorer and type

```
sftp://eigit.heig-vd.ch/var/www/public/rootfs/reptar
```

login with your *einet* account. If you prefer use the command-line, use *scp*.

6.3 PROCEDURES FOR DEVELOPMENT IN THE EMBEDDED PROCESSOR WITHOUT OS

For development without OS we usually use Code Composer Studio and a JTAG emulator. You can find information about how to use Code Composer Studio in the document Utilisation_CCSv5.pdf

Remember that we use a special Toolchain for standalone projects without Linux: arm-none-eabi.

As an example of project you can see the accelerometer test available on the *reptar_usr* GIT repository or the ASP lab works.

6.4 PROCEDURES FOR DEVELOPMENT IN THE FPGA

6.4.1 Using the REPTAR base project

The REPTAR base project is an ISE project with the VHDL sources and UCF configuration file necessaries for using the Spartan6 FPGA without communication with the CPU. The only components instantiated are the tri-state buffers to select the direction of the GPIO pins and a PLL that generate 300MHz and 100 MHz frequencies.

All the pins of the SP6 physically connected to other peripherals are set to their inactive state ('1' or '0' depending on the component).

The Reset button used is the button named "SP6 Config" on the FPGA board. This button is connected to a Spartan3 pin and not directly relied to the SP6. For that reason, the **Reset of the SP6 in the base project works only if the SP3 is configured with the standard REPTAR bitstream**.

Differential input buffers are instantiated for use of the differential clock inputs from the FMC boards (Mezzanine to Carrier clocks):

Clock0 positive	FMCx_CLK0_M2C_P_i
Clock0 negative	FMCx_CLK0_M2C_N_i
Clock1 positive	FMCx_CLK1_M2C_P_i
Clock1 negative	FMCx_CLK1_M2C_N_i

You can find the REPTAR base project on the folder:
reptar_hard\reptar_series_1\cpld_fpga\fpga_projet_base for series I version or at **reptar_hard\reptar_proto_2\cpld_fpga\fpga_projet_base** for proto II version

- ✓ The subfolder *src* contains the VHDL sources and the UCF constraints file
- ✓ The subfolder *ise_v13_3* contains the ISE project file *.xise.* When you compile the project, all the generated files are written to this folder
- ✓ The subfolder bitstream holds a copy of the last version of the standard bitstream issue from the ISE project compilation
- ✓ The *IP_core* folder, if exists, contains the *.cgp* Core Generator project and a *.xco* configuration file per IP core used
- ✓ The *sim* subfolder contains the test benches and scripts used for simulation

6.4.2 Using the REPTAR standard project

The REPTAR standard project allows running all the tests of the REPTAR board from the CPU (under u-boot and Linux environments).

This bitstream is functional only **if the SP3 is configured with the standard REPTAR bitstream**.

In this project the Local Bus component, that implements the communication between CPU and FPGA, is instantiated. For additional information about the Local Bus see the section Using the Local Bus on the Mix Development Procedures chapter.

The Reset button used is the same as for the base project, named “SP6 Config” on the FPGA board.

The following components useful to manage peripherals are instantiated too: buzzer controller, encoder sense detector, mini-lcd controller, touchpad controller, DDR controller (coming soon).

Other components instantiated are:

- ✓ tri-state buffers to select the direction of the GPIO pins
- ✓ PLL that generate 300MHz and 100 MHz frequencies
- ✓ differential input buffers for FMC clocks

The REPTAR standard ISE project can be found on the folder
reptar_hard\reptar_series_1\cpld_fpga\fpga for series I version or at **reptar_hard\reptar_series_1\cpld_fpga\fpga** for proto II version

- ✓ The subfolder *src* contains the VHDL sources and the UCF constraints file. If IP are used, their VHDL entities and *.ngc* files are in a sub-folder of *src*
- ✓ The subfolder *ise_v13_3* contains the ISE project file *.xise.* When you compile the project, all the generated files are written to this folder

- ✓ The subfolder bitstream holds a copy of the last version of the standard bitstream issue from the ISE project compilation
- ✓ The *IP_core* folder, if exists, contains the *.cgp* Core Generator project and a *.xco* configuration file per IP core used
- ✓ The *sim* subfolder contains the test benches and scripts used for simulation

6.4.3 ISE project compilation

The compilation of the ISE projects (base and standard) issues two kinds of binary files:

- ✓ .bin: used for configuring the FPGA from the CPU, this is a file without header
- ✓ .bit: used for configuring the FPGA with the Impact tool, this is a file with a header

6.4.4 Spartan3 programming

Normally, the SP3 of the REPTAR board must always be configured with the REPTAR standard bitstream.
This bitstream has the following functions:

- ✓ manage the different programming modes for the FPGA Spartan 6 (from the CPU, from the PlatformFlash or JTAG header, or with a micro-USB cable)
- ✓ receive the bitstream from the CPU through dedicated GPIOs and send it to the FPGA SP6
- ✓ implement the Local Bus to allow access from the CPU to the LEDs, switches, push-buttons and GPIOs connected to the SP3
- ✓ transmit the reset signal from the "SP6 Config" push-button to an input of the FPGA SP6

The SP3 has an internal persistent memory that holds a bitstream. This bitstream can be automatically loaded to the SP3 on the power-on reset. In order to use this feature you must configure the DIP switches SP3 8, 9 and 10 to "110" where '1' is ON and '0' is OFF (mode internal master SPI). If you want to deactivate this feature and program the SP3 only from JTAG you must set the DIP switches SP3 8, 9 and 10 to "101".

The "SP3 NProg" button erases the SP3 configuration, and the LED "SP3 Not. Conf" is ON when the SP3 is empty.

The switches "SP3 Configuration mode" (DIP SP3 numbers 8 to 10) allow selecting the configuration mode between "Master SPI mode" (from flash) or "JTAG mode" (bitstream loaded from your PC with the Impact tool and the Platform Cable).

- ✓ DIP SP3 (8..10) = "110" to select the loading from flash

✓ DIP SP3 (8..10) = “101” to select the JTAG mode

Currently, all the REPTAR boards have the standard bitstream on their SP3 flash, nevertheless if you want to update this bitstream with the last version or test your own bitstream, you can follow the procedures explained here-after.

6.4.4.1 Programming the SP3’s flash with the REPTAR standard bitstream

1. Set DIP switches SP3 8, 9 and 10 to “110” where ‘1’ is ON and ‘0’
2. Connect the Xilinx Platform Cable USB to the SP3 header (see Figure 8 – REPTAR FPGA, JTAG headers and SMT module connector)
3. Turn-on the board
4. Run the Impact tool
5. Open the project **reptar\publi\test\FPGAs_Flashing\Serie_I\Impact_Spartan3.ipf**
6. Right-click on the SP3 chip and select "Program Flash and Load FPGA", wait for the message "Program Succeeded"
7. Verify that the LED “SP3 Not. Conf” is OFF
8. Turn-off and then turn-on the board again, re-verify the LED
9. Push on the “SP3 NProg” button to erase the SP3. The bitstream must be automatically reloaded

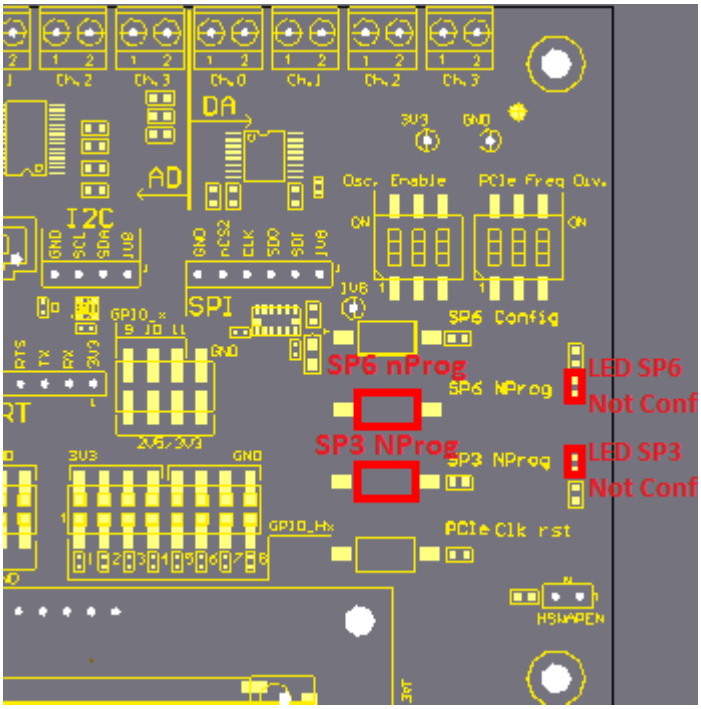


Figure 7 – REPTAR FPGA, SP6 and SP3 configuration LEDs and buttons

6.4.4.2 Loading a different bitstream on the SP3

When debugging your VHDL description or during tests it can be useful to program the FPGA SP3 without using the bitstream stored in flash. In this way you don't overwrite the functional bitstream, which will be still reloaded on power-on reset.

1. Set DIP switches SP3 to "1111111101" where '1' is ON and '0'
2. Connect the Xilinx Platform Cable USB to the SP3 header (see Figure 8 – REPTAR FPGA, JTAG headers and SMT module connector)
3. Turn-on the board
4. Run the Impact tool
5. Don't open any project
6. Double-click on "boundary scan"
7. Right-click on an empty zone on the right window and choice "initialize chain", a chip appears on the right window (xc3s200an)
8. Assign the bitstream to the chip: browse on your PC and open your bitstream file
9. Right-click on the SP3 chip and select "Program", wait for the message "Program Succeeded"
10. Verify that the LED "SP3 Not. Conf" is OFF

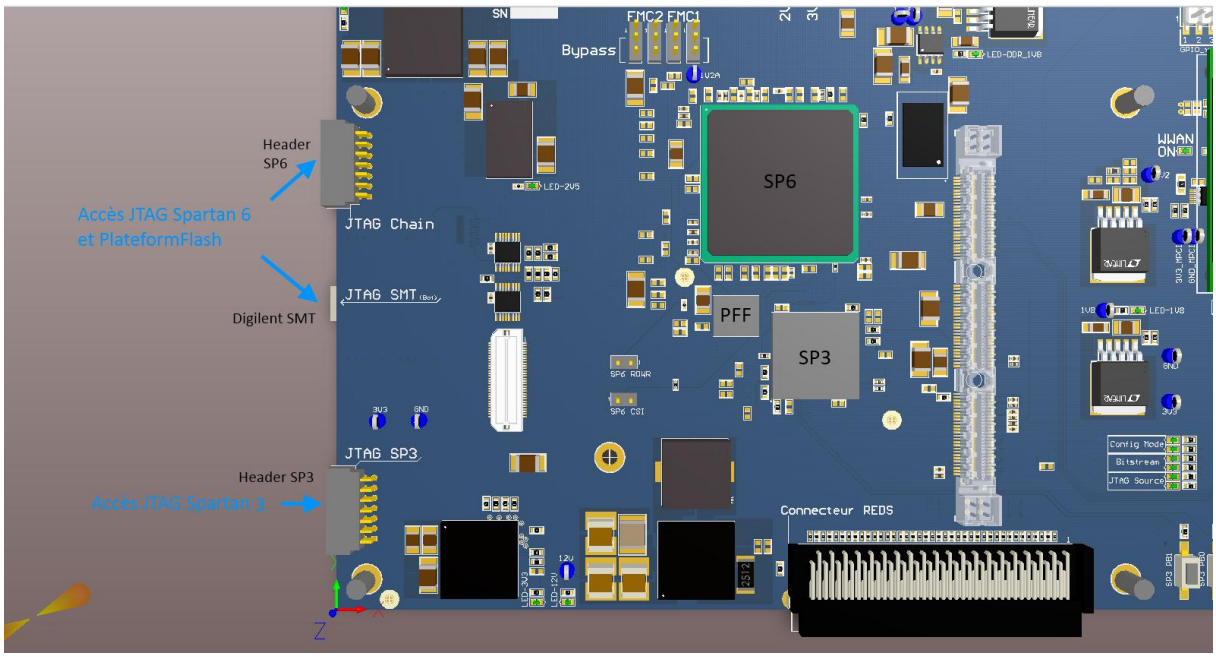


Figure 8 – REPTAR FPGA, JTAG headers and SMT module connector

6.4.5 Spartan6 programming

When debugging your VHDL description or during tests it can be useful to program the FPGA SP6 without using the bitstream stored in flash. In this way you don't overwrite the functional bitstream, which will be still reloaded on power-on reset.

For programming the SP6 you can use a Xilinx Platform Cable or a micro-USB cable. The micro-USB cable uses a Digilent SMT JTAG programming module integrated on the board (on the bottom).

The switches "JTAG mode selection" (DIP SP3 number 2 and 3) allow to select the JTAG programmer for the SP6/Platform Flash /FMC FPGAs between the Xilinx Platform Cable USB or the SMT JTAG micro-USB.

- ✓ DIP SP3 (2..3) = "11" to select the JTAG header with the Platform Cable
- ✓ DIP SP3 (2..3) = "01" to select the SMT JTAG Digilent with the micro-USB cable

6.4.5.1 Using the Xilinx Platform Cable USB

1. Set DIP switches SP3 to "1111111110" where '1' is ON and '0'
2. Connect the Xilinx Platform Cable USB to the SP6 header (see Figure 8 – REPTAR FPGA, JTAG headers and SMT module connector)
3. Turn-on the board
4. Run the Impact tool
5. Don't open any project
6. Double-click on "boundary scan", two chips appear on the right (flash and SP6)
7. Right-click on an empty zone on the right window and choice "initialize chain"
8. Assign bitstreams to the chips of the chain: choose "bypass" for the flash (xcf32p) and browse for your bitstream for the SP6 (xc6slx150t)
9. Right-click on the SP6 chip and select "Program", wait for the message "Program Succeeded"
10. Verify that the LED "SP6 Not. Conf" is OFF

6.4.5.2 Using a micro-USB cable

1. Set DIP switches SP3 to "1011111110" where '1' is ON and '0'
2. Connect the micro-USB cable to the micro-USB connector of the SMT JTAG
3. Turn-on the board
4. Run the Impact tool

5. Don't open any project
6. Double-click on "boundary scan", two chips appear on the right (flash and SP6)
7. Right-click on an empty zone on the right window and choice "initialize chain"
8. Assign bitstreams to the chips of the chain: choose "bypass" for the flash (xcf32p) and browse for your bitstream for the SP6 (xc6slx150t)
9. Right-click on the SP6 chip and select "Program", wait for the message "Program Succeeded"
10. Verify that the LED "SP6 Not. Conf" is OFF

6.4.6 Platform Flash programming

The FPGA Spartan6 loses its configuration when the power is OFF. In order to keep its configuration, the bitstream is hold in a flash memory that can load the SP6 on each power-on.

The Platform Flash (PFF) must be programmed with a *.mcs* file. This file is generated from the *.bit* file using the Impact tool. You will find the steps to follow for generating this kind of file in this section.

Currently, all the REPTAR boards have the standard bitstream on their SP6 Platform Flash, nevertheless if you want to update this bitstream with the last version, you can follow the procedure explained here-after.

6.4.6.1 Loading the standard bitstream

1. Set DIP switches SP3 to "1011111110" for micro-USB or "1111111110" for Xilinx Platform Cable
2. Connect the micro-USB cable to the micro-USB connector of the SMT JTAG or connect the Xilinx Platform Cable USB to the SP6 header (see Figure 8 – REPTAR FPGA, JTAG headers and SMT module connector)
3. Turn-on the board
4. Run the Impact tool
5. Open the project **reptar\publi\test\FPGAs_Flashing\Serie_I\Impact_PFF_Spartan6.ipf**
6. Right-click on the "xcf32p" chip and select "Program", wait for the message "Program Succeeded"
7. Push on the "SP3 NProg" button to reset the SP3 configuration. The bitstream must be automatically reloaded on SP6 from the Platform Flash
8. Verify that the LED "SP6 Not. Conf" is OFF

9. Turn the encoder to the right and verify that the LED7 turns ON, then turn the encoder to the left and verify that the LED6 turns ON

6.4.6.2 Generating your own flash configuration file

1. Compile your ISE project to get the *.bit* output file
2. Open the Impact tool
3. Don't open any project
4. Double-click on "Create PROM File"
5. Step 1. Select: "Xilinx Flash / PROM" and then click on the arrow
6. Step 2. Select: "Platform Flash" and "xcf32p", and then click on "Add Storage Device"
7. Click on the green arrow
8. Step 3. Enter the "Output File Name" and "Output File Location"
9. Choose "No" for "Enable Revisioning" and "No" for "Enable Compression"
10. Push OK
11. On the pop-up message "Start adding…" push OK
12. Browse for your SP6 *.bit* file and click on "open"
13. Answer "No" to the question "Would you like to add another…"
14. Click OK to continue
15. Click on an empty zone on the right window and select "Generate File…"
16. A blue message "Generate Succeeded" is shown
17. Verify that a new *.mcs* file was written to your output folder

6.4.6.3 Programming the flash with your own file

18. Set DIP switches SP3 to "1011111110" for micro-USB or "1111111110" for Xilinx Platform Cable
19. Connect the micro-USB cable to the micro-USB connector of the SMT JTAG or connect the Xilinx Platform Cable USB to the SP6 header
20. Turn-on the board
21. In Impact, double-click on "boundary scan", two chips appear on the right (flash and SP6)
22. Right-click on an empty zone on the right window and choice "initialize chain"
23. Assign bitstreams to the chips of the chain: choose browse for your bitstream *.mcs* file for the flash (xcf32p) and choose "bypass" for the SP6 (xc6slx150t)
24. On the "Device Programming Properties" of the PROM, under "PROM Specific Properties" select "Parallel Mode"

25. On the “Device Programming Properties” of the PROM, under “Advanced PROM Programming Properties” select “During Configuration: PROM is Master”
26. Select the clock source: Internal Clock (40MHz)
27. You can save your Impact project and when you want to program the flash you can follow the procedure Loading the standard bitstream but with your own project

6.5 PROCEDURES FOR MIX DEVELOPMENT USING EMBEDDED PROCESSOR AND FPGA

6.5.1 Board setup

6.5.1.1 SP3 DIP switches

The switch "Local Bus enable" (DIP SP3 number 1) when it's active, allows the Spartan 3 to put data on the Local Bus when the CPU performs a read operation. This switch must be ON when you attempt to read a Spartan3 register from the CPU.

- ✓ DIP SP3 (1) = "1" to enable the Local Bus
- ✓ DIP SP3 (1) = "0" to disable the Local Bus

The switch "Configuration mode SP6" (DIP SP3 number 4) allows to choose if the bitstream of the SP6 will be loaded from the flash or from the CPU.

- ✓ DIP SP3 (4) = "1" to load the bitstream from the flash
- ✓ DIP SP3 (4) = "0" to load the bitstream from the CPU

The switches "SP3 Configuration mode" (DIP SP3 numbers 8 to 10) allow selecting the configuration mode between "Master SPI mode" (from flash) or "JTAG mode" (bitstream loaded from your PC with the Impact tool and the Platform Cable). For mix development ensure that the SP3 loads its bitstream from the flash:

- ✓ DIP SP3 (8..10) = "110" to select the loading from flash

6.5.1.2 Standard bitstreams

For mix development, the SP3 must be configured with the standard bitstream. See the section Programming the SP3's flash with the REPTAR standard bitstream for details in the procedure.

The bitstream of the SP6 must be the standard bitstream or a variation of this bitstream including the instantiation of the Local Bus controller. The bitstream stored in the Platform Flash must be always the standard one, if you need a different one, use the *.bin* file and send-it from the CPU to the SP6 by using the u-boot or Linux drivers. See the section Programming the SP6 from the CPU or the section Loading the standard bitstream

6.5.2 Using the Local Bus

The Local Bus allows the CPU to access the peripherals of the REPTAR FPGA board connected to the Spartan3 or Spartan6 FPGAs.

The Local Bus may also be used to transfer specific data from applications running on the CPU to the FPGA.

From the CPU point of view, the FPGA is a memory-mapped device. All the data transfers between CPU and FPGAs use register addresses.

The registers concerning peripheral access through the Spartan3 are defined in the document Spartan3_Registers_v1.xlsx

The registers concerning peripheral access through the Spartan6 are defined in the document Spartan6_Registers_v1.xlsx

From the CPU side, we use the GPMC (General Purpose Memory Controller) to configure a NOR-like 16-bit device: the FPGAs.
The Spartan3 and the Spartan6 share the memory space mapped by the GPMC chip select CS3 (128 MB max), nevertheless, only 25 IO lines are wired between CPU and FPGA, this allow to address 32Mbytes maximum for an 8-bit device and 64Mbytes for a 16-bit device.

The GPMC configuration is done during the board initialization under the U-boot environment for the CS3.

Currently map is (versions before May 2013):

- ✓ Spartan6 registers are in the memory space going from 0x1800 0000 to 0x18FFFFFF (16MB)
- ✓ Spartan3 registers are in the memory space going from 0x1900 0000 to 0x19FFFFFF (16MB)
- ✓ Unused zone from 0x1A000000 to 0x1FFFFFFF (96 MB)

In a further version, the map will change to:

- ✓ Spartan6 registers in the memory space going from 0x1800 0000 to 0x19FFFFFF (32MB)
- ✓ Spartan3 registers are in the memory space going from 0x1A00 0000 to 0x1BFFFFFF (32MB)

- ✓ Inaccessible zone from 0x1C000000 to 0x1FFFFFFF (64MB)

The Local Bus is 16-bit data and is multiplexed with the low 16 address bits.

Access to the Spartan6 and Spartan3 registers was implemented as asynchronous operations, because in most cases the CPU needs to read or write a single register, so high throughput is not necessary.

From the FPGAs point of view, this means that the Local Bus controller runs at the FPGA clock frequency, and the clock coming from the GPMC is not used. The controller decodes the address and then performs write or read operations to/from the internal registers or the peripherals.

There is another chip select going from CPU to FPGA, the chip select CS4 of the GPMC used to allow the CPU access to the DDR connected to the Spartan6. In this case, the Local Bus is used in synchronous mode, that is, the GPMC clock is used to run the state machine of the Local Bus Controller of the FPGA.

Addresses between 0x20000000 and 0x23FFFFFF (64 MB) access the first quarter of the DDR of the REPTAR FPGA board. In order to be able to access the three other quarters, a paging method can be implemented using Spartan6 registers.

In synchronous mode, the frequency of the GPMC clock used is 62.5MHz. Currently, transfers use bursts of 4 x 16 bits, the maximum being 16 x 16 bits.

6.5.2.1 SP6 DIP switches

There are 10 DIP switches connected to the FPGA Spartan6, only the first is reserved for a particular functionality, the rest are available for the user.

The switch “Local Bus enable” (DIP SP6 number 1) when it’s active, allows the Spartan 6 to put data on the Local Bus when the CPU performs a read operation. This switch must be ON when you attempt to read a Spartan6 register from the CPU.

- ✓ DIP SP6 (1) = “1” to enable the Local Bus
- ✓ DIP SP6 (1) = “0” to disable the Local Bus

6.5.3 Using the REPTAR standard ISE project

The REPTAR standard project allows running all the tests of the REPTAR board from the CPU (under u-boot and Linux environments).

This bitstream is functional only **if the SP3 is configured with the standard REPTAR bitstream**.

In this project the Local Bus component, that implements the communication between CPU and FPGA, is instantiated. For additional information about the Local Bus see the section Using the Local Bus on the Mix Development Procedures chapter.

The Reset button used is the same as for the base project, named "SP6 Config" on the FPGA board.

The following components useful to manage peripherals are instantiated too: buzzer controller, encoder sense detector, mini-lcd controller, touchpad controller, DDR controller (coming soon).

Other components instantiated are:

- ✓ tri-state buffers to select the direction of the GPIO pins
- ✓ PLL that generate 300MHz and 100 MHz frequencies
- ✓ differential input buffers for FMC clocks

The REPTAR standard ISE project can be found on the folder
reptar_hard\reptar_series_1\cpld_fpga\fpga for series I version or at **reptar_hard\reptar_proto_2\cpld_fpga\fpga** for proto II version

- ✓ The subfolder *src* contains the VHDL sources and the UCF constraints file
- ✓ The subfolder *ise_v13_3* contains the ISE project file *.xise.* When you compile the project, all the generated files are written to this folder
- ✓ The subfolder bitstream holds a copy of the last version of the standard bitstream issue from the ISE project compilation
- ✓ The *IP_core* folder, if exists, contains the *.cgp* Core Generator project and a *.xco* configuration file per IP core used
- ✓ The *sim* subfolder contains the test benches and scripts used for simulation

6.5.4 Programming the SP6 from the CPU

6.5.4.1 Board setup

The switch "Configuration mode SP6" (DIP SP3 number 4) allows to choose if the bitstream of the SP6 will be loaded from the flash or from the CPU.

- ✓ DIP SP3 (4) = "1" to load the bitstream from the flash
- ✓ DIP SP3 (4) = "0" to load the bitstream from the CPU

6.5.4.2 U-boot

TODO

6.5.4.3 Linux

The SP6 can be programmed from Linux through the */dev/fpgaloader* driver entry point. When using the Archlinux-tests rootfs, the *load_fpga.sh* script in */root/tests* allows doing this in an easy way.

This file system is available at http://eigit.heig-vd.ch/public/rootfs/reptar/, its name is **ArchLinux-filesystem-tests.tar.bz2**, see the procedure to prepare a "Master" SD card if you don't have an SD card with this rootfs.

Syntax:

```
./load_fpga.sh   <bitstream_file.bin>
```

Example:

```
[root@reptar tests]# ./load_fpga.sh spartan6_bitstream_top.bin
fpgaloader v0.92 ready
Bitstream file is spartan6_bitstream_top.bin !
fpga_init_load :in slave serial case
reptar_fpga_config_slave_serial call success
wait inib low: Current time = 0  initb = 0
spartan_serial_init returned with ret = 0
Starting reptar FPGA download
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
........................3337+1 records in
3337+1 records out
1708550 bytes (1.7 MB) copied, 9.18951 s, 186 kB/s
gpmc_fpga_init:357: registered misc device 58
ioctl cmd: FPGA_RD_REG = c0047400 FPGA_WR_REG=c0047401
reptar-fpga ready
[root@reptar tests]#
```

Procedure:

1. Set up the REPTAR board as explained above
2. Insert a "Master" SD card with the *tests* rootfs into the board
3. Connect a mini-USB cable to your PC and open a serial terminal

4. Turn on the board
5. If initialization stops on uboot, type `boot` to boot Linux and load the rootfs from the SD card
6. Wait for the Linux initialization end and then log in
7. Change the directory to *tests*
8. Verify that the red LED “SP6 NProg” is ON (close to the top-right corner of the board)
9. Call the script with your bitstream file as parameter
10. Wait until the prompt appears again
11. Verify that the red LED “SP6 NProg” is OFF

7. TROUBLESHOOTING

Are the jumpers in the good position?
Are the DIP switches SP3 in the good position?
Are the SP3 and SP6 configured?
Are the SP3 and SP6 bitstreams updated?
Is the SD card inserted?
Does the SD card contain a valid kernel and/or rootfs?

8. ADDITIONAL INFORMATION

8.1 REVISION HISTORY

Chapter	Date	Version	Changes Made
All	March 2013	1.0	▪ First publication.

Table 1 - Revision History

8.2 CONTACT

For the most up-to-date information or remarks about this document contact the REDS institute:

REDS
Heig-vd
Route de Cheseaux 1
CH-1401 Yverdon-les-Bains

reds@heig-vd.ch