Branch: master ▾ Find file Copy path

Z-turn-examples / README.md

 wzab Update README.md

7c74955 on 16 Aug 2016

1 contributor

Raw Blame History

63 lines (55 sloc) 5.75 KB

Z-turn-examples

The repository with my simple Z-turn examples, to be used as templates for more serious projects.

Please note, that the Buildroot configuration in my designs sets the root password to "test". Setting the password is necessary to allow SSH login to the Z-Turn board. Of course you can easily modify that (either to set a better password or to delete the password at all).

My examples use the VEXTPROJ scripts for building the Vivado projects. Therefore the HDL sources and other project configuration details may be stored in git-friendly form. In each project there is the `hdl` directory with the `build.sh` script, which creates and builds the Vivado project (of course Vivado must be installed and accessible via PATH). After the project is created, you can open it in Vivado GUI, modify and rebuild it interactively. There is also the `software` directory with the `build_soft` script, that downloads the Buildroot environment, configures it and compiles the related software. *Please note, that compilation of software should be done in a shell session without Vivado binaries in PATH. Otherwise strange problems e.g., with `awk` may occur.*

Test configuration

My favorite configuration for working with Z-turn is to download the FPGA bit file, the DTB file and the kernel with initramfs from the TFTP server (172.19.1.1 in my setup). I use the following uEnv.txt file located on the SD card:

```
bootargs=console=ttyPS0,115200 root=/dev/mmcblk0p2 rw earlyprintk
rootfstype=ext4 rootwait devtmpfs.mount=0
ipaddr=172.19.3.33
srvaddr=172.19.1.1
load_fpga=tftpboot ${kernel_load_address} ${srvaddr}:Zturn.bit && fpga loadb 0
${kernel_load_address} 4045676
load_image=tftpboot ${kernel_load_address} ${srvaddr}:Zturn_uImage
load_dtb=tftpboot ${devicetree_load_address} ${srvaddr}:Zturn.dtb
uenvcmd=echo Loading FPGA bitstream... && run load_fpga && echo Loading kernel
image from the server... && run load_image && echo Loading Device Tree... &&
run load_dtb && bootm ${kernel_load_address} - ${devicetree_load_address}
```

Except of that my SD card contains only the BOOT.bin file generated in SDK

How to get working FSBL for Z-turn with Vivado 2016.2

Unfortunately it appeared, that it is not trivial to build the working FSBL in Vivado 2016.2 working with Buildroot. The workflow that finally works for me is the following:

- In Debian/testing I had multiple strange problems with Vivado SDK, before I found the advise to set the SWT_GTK3 environment variable to 0 before starting Vivado. Therefore you should do `export SWT_GTK3=0` before you start Vivado.
- Ensure that your design has configured the following peripherials:
  - ENET0 (connected to MIO16-27) with MDIO (connected to MIO52-53).
    - Remember to connect the ENET0 clock to "IO PLL" (in my design it was connected to "External" by default)
    - Remember to set all ENET0 signals to "fast" (except of tx_clk - MIO 16, which may be set to "slow")
  - SD0 (connected to MIO40-45). Remember to lower the clock frequency to 50 MHz (with default 125 MHz it won't work with most cards!)
  - UART1 (connected to MIO48-49)
- After you compile your design, export the hardware (File -> Export -> Export hardware) locally to the project.
- Then run the SDK (File -> Launch SDK)
- In the SDK add the DT repository (Xiling Tools->Repositories, Local repositories -> New, xilinx-tree-xlnx) available from git://github.com/Xilinx/device-tree-xlnx.git
- In the SDK create:

- The new Board Support Package of type "device_tree". Build it and use the resulting dts and dtsi files in Buildroot (in my design just put them into the `software/dts` directory).
- The new Application Project of type "Zynq FSBL" with name "fsbl". This project requires a small adjustment:
  - In the `fsbl/src/fsbl_debug.h` file add `define FSBL_DEBUG_INFO` before `#define DEBUG_GENERAL 0x00000001`. That ensures that FSBL displays possible error messages. Without that I wouldn't be able to resolve all problems related to the SD booting.
- Create the boot image - BOOT.bin
  - You should have compiled the U-Boot in the ELF format. Remember to switch off SPL and select the ELF format when compiling the Buildroot. I use the configuration "zynq_zed" for the U-Boot as there is no dedicated configuration for Z-turn. Below I assume that the compiled U-Boot has been copied to /tmp/u-boot.elf. (Buildroot compiles the U-Boot to the file `output/images/u-boot` but SDK requires that it has the `.elf` extension, soo you should anyway copy or rename the output file.)
  - Before I learned about the `export SWT_GTK3=0` trick, the "Create Boot Image" option didn't work for me. Therefore I had to create the bootimage directory and fsbl.bif file manually

    ```
    The_ROM_image:
    {
        [bootloader]../Debug/fsbl.elf
        /tmp/u-boot.elf
    }
    ```

    In the SDK start the shell (Xilinx Tools -> Launch Shell) and in the shell do:

    ```
    $ cd fsbl/bootimage/
    $ bootgen -arch zynq -image fsbl.bif -w on -o BOOT.bin
    ```

  - If the "Create Boot Image" option works for you, you should select the Architecture "Zynq", and mark "Create new BIF file". The list of files to be put to the boot image should already contain the "(bootloader) ...fsbl.elf" entry. Add your compiled u-boot.elf to it (add it as "datafile" with checksum set to "none"). You can preview the created `.bif` file. It should look like shown in the previous point (probably with an additional comment line at the begining). After the `.bif` file is ready, you can create the bootimage.