PROGRAM UPDATE OF ZYNQ-BASED DEVICES

Master Thesis

Bc. Branislav Michálek
Supervisor: Ing. Jan Král

EUROPEAN UNION

14. 6. 2019

Motivation

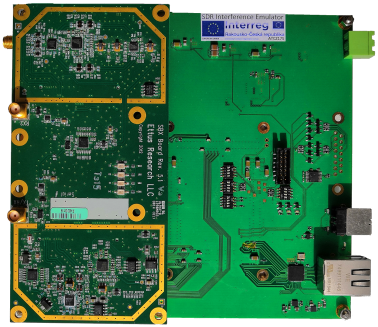
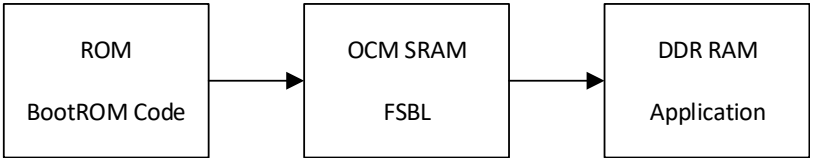


Figure: SDR Interference Emulator.

Figure: Module TE0803.

- Target platform - module TE0803:
  - Zynq UltraScale+ MPSoC.
- Development platform - module TE0720:
  - Zynq-7000 SoC.
  - 2x ARM Cortex-A9 CPU.
  - 1 GB DDR3 RAM.
  - GbE Transciever.
  - 32 MB SPI .
- Possibility to store multiple versions of the firmware.
- Update of firmware controlled by an application from command line.
- Read-in default configuration in case of failure.

Boot process

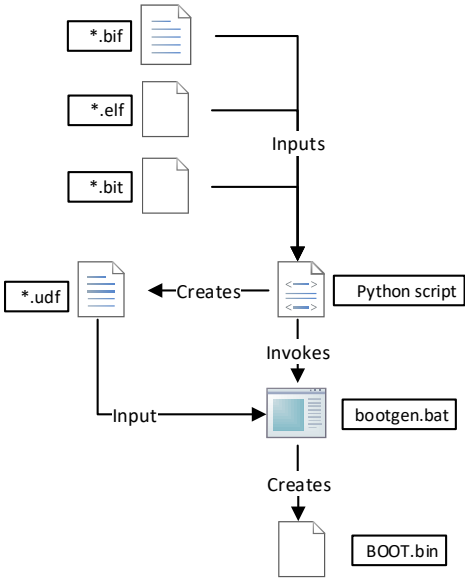


1. BootROM loads First-Stage Bootloader (FSBL) from flash memory into On-Chip SRAM (OCM). Validates only *header* of the boot image.
2. FSBL boots other partitions (bare-metal/FreeRTOS application, bitstream for FPGA, Second-Stage Bootloader - SSBL) into external DDR RAM memory.
3. Optional SSBL boots higher operating system (Linux).
4. Program execution from DDR RAM.

Program update

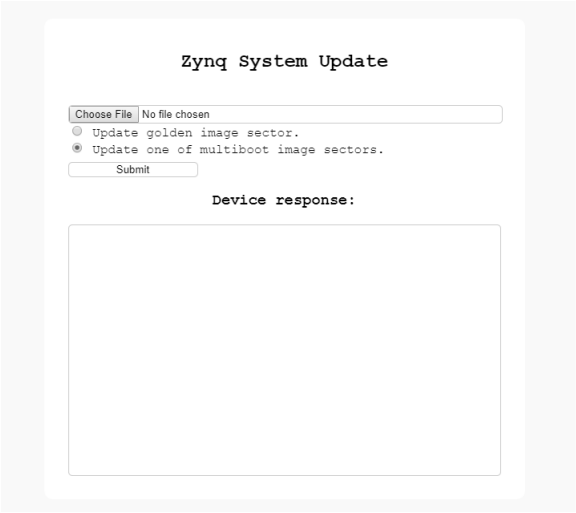
1. Setup of boot image file from input files (bootloader, aplication, bitstream etc.).
2. Transfer of boot image file into the target device over Ethernet with the use of suitable communication protocol.
3. Writing file to flash memory, validation.
4. Automatic loading of the newest boot image at the next reboot.

Setup of boot image



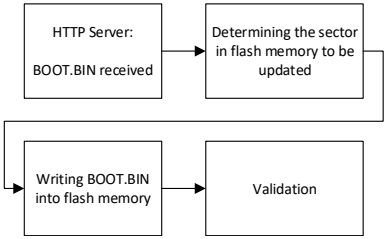
- **In non-secure boot mode, Zynq-7000 cannot validate FSBL partition**.
- Python script:
  - Processes input files for Bootgen (`*.bif`, `FSBL.elf`.)
  - Creates User-Defined Field file (`udf.txt`) containing:
    - FSBL check sum,
    - Time stamp,
    - Image Validity Word - result of the data integrity check.
  - Executes Bootgen (Xilinx tool to generate boot image).
- The output is a binary file `BOOT.BIN`.

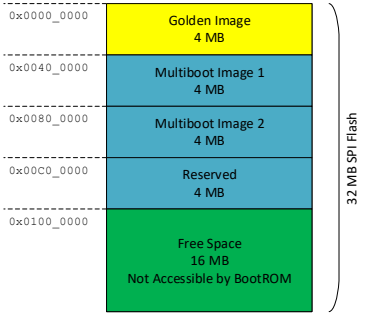
Transfer of the file into the target device



- Simple HTTP server on target device.
- Client HTTP:
  - Standard application executable form command line (cURL, Wget, etc.).
  - Web page hosted on HTTP server.

After receiving the file

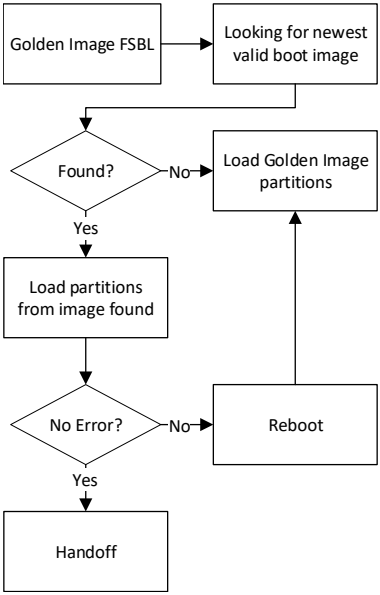




- Flash memory is split to 4MH blocks.

1. Detection of valid image based on Image Validity Word:
   - `0xFFFFFFFF` - invalid boot image.
   - `0xFFFFFFFE` - valid boot image.
2. Determine the version basedon time stamp.
3. Update (i.e. overwriting) of invalid or oldest valid boot image.
4. Write into flash memory.
5. Validation of written boot image, incl FSBL.

Multiboot and backup configuration



- Multiboot functionality id implemented in a FSBL golden image.

1. At startup, FSBL golden image is executed.
2. If a newer valid boot image is found, attempt to boot it.
3. In case of failure, an automatic reset will boot the golden image.

Conclusions

- Python script to create boot image with timestamp, FSBL checksum and Image Validity Word.
- Application for update (HTTP server, read/write support to/from flash memory, data integrity check).
- Web page to transfer the boot image file.
- Modified bootloader to automatically load the latest valid image.