Introduction to SoC+FPGA

Marek Vašut <marek.vasut@gmail.com>

March 3, 2017

Marek Vasut

- Software engineer at DENX S.E. since 2011
- Versatile Linux kernel hacker
- Custodian at U-Boot bootloader
- Yocto (oe-core) contributor
- FPGA enthusiast

Structure of the talk

- What is SoC, FPGA and SoC+FPGA ?
- Available solutions, small and big
- Small bare-metal or RTOS solutions
- Big solutions with U-Boot and Linux
- FPGA manager and DTOs
- Conclusion

SoC? FPGA? SoC+FPGA?

SoC:

- System on Chip
- CPU core + peripherals

FPGA:

- Field-Programmable Gate Array
- Programmable logic device

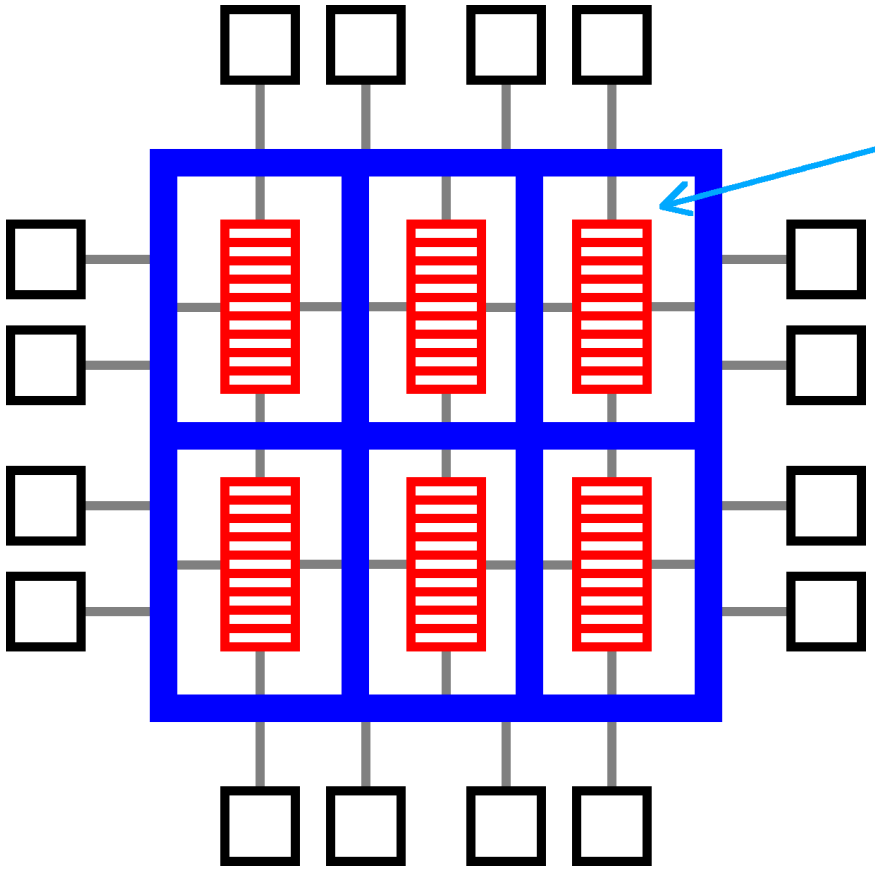
SoC+FPGA:

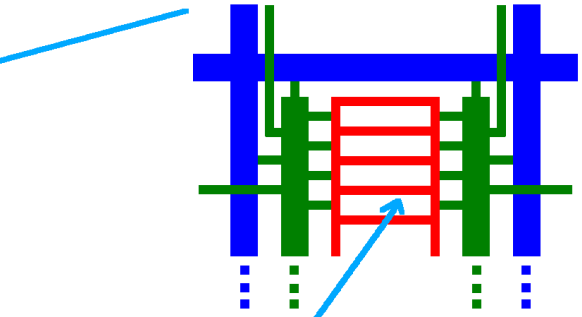
- SoC and FPGA on a single chip
- Connected through on-chip bus

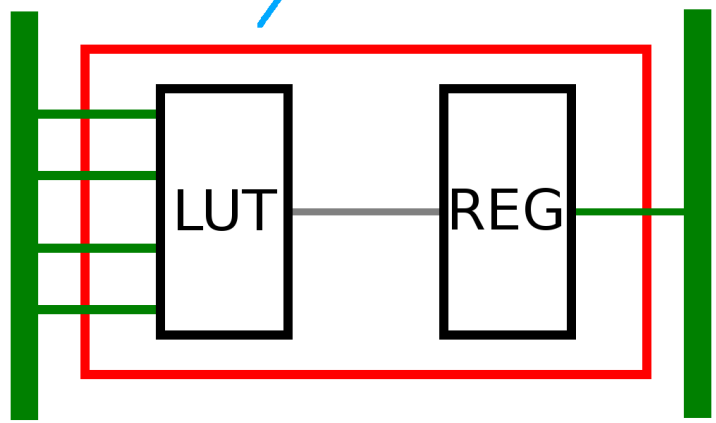
FPGA

- Field Programmable Gate Array
- High-Speed Programmable logic
- Plenty of I/O options
- Extremely parallel architecture
- Usually used for:
  - Digital Signal Processing (DSP)
  - Parallel data processing
  - Custom hardware interfaces
  - ASIC prototyping
  - ...
- Common vendors – Xilinx, Altera, Lattice, Microsemi...

Internal structure







BLUE Global interconnect

GREEN Local interconnect

RED Logic element

Why SoC+FPGA?

- Cost ?
- Need for special bus interface for a CPU
- Need for obscure (amount of) I/O
- Need for extra CPU power for your FPGA

What's available?

A lot !

- Cypress PSoC: 8051/CortexM0/M3 , Flash+SRAM
- Microsemi SF2: CortexM3 , Flash+SRAM+DRAM
- Altera SoCFPGA: CortexA9 SoC + FPGA
- Xilinx Zynq: CortexA9/A53 SoC + FPGA

Cypress PSoC

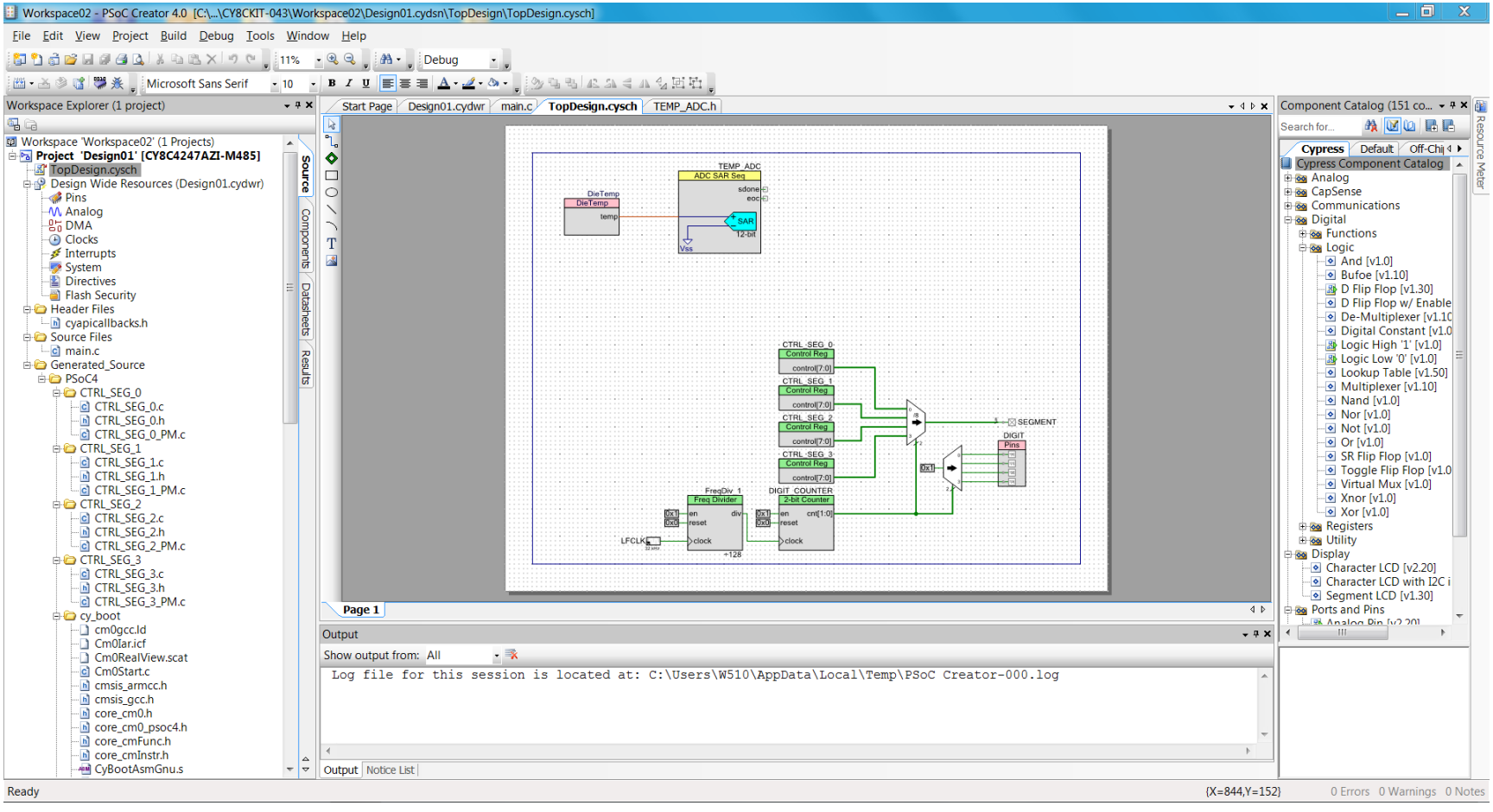
- Originally 8051 + Analog programmable fabric
- Since PSoC4, ARM Cortex M0 + Optional digital blocks
- Since PSoC5, ARM Cortex M3
- All PSoCs are flash-based , so non-volatile
- Targets deeply embedded systems, like smoke detectors
- Kit is $10 with easily accessible pins and programmer

This is awesome! But ...

Cypress PSoC getting started

- PSoC Creator is Windows only (or Wine) :-(
- GreenPAK project is working on fixing this :-)
- Installation is annoying, but doable
- Lot of examples in the design tool :-)
- Most of them don't target cheap kits :-(
- Programable logic design is done via schematic entry
- Click compile - program - done ...

Cypress PSoC Creator



Cypress PSoC software

- PSoC creator has bare-metal code templates
- Each PL component has register interface
- PSoC creator generates templates for PL components
- There are even convenience functions !
- Or export the PL init blob and include it in RTOS
- FreeRTOS and uC/OS2 BSPs are available

Microsemi SmartFusion 2

- Has roots in Actel offerings
- CortexM3 with MPU, Flash/SRAM/DDR DRAM
- Arrow SF2PLUS kit is $125 with programmer
- Usual RTOS offerings – FreeRTOS, uC/OS-III, Keil RTX
- Capable of running Linux *

* uClinux with prehistoric kernel

Microsemi SF2 getting started

It's easy ... no, not really ...

- Register at Microsemi website
- Download Libero SoC design software 11.7
- Download separate service pack 3
- Download license server daemons
- Install the first two (howto kinda works ...)
- Install asortment of 32bit libs
- Unpack the daemons
- Obtain evaluation license from Microsemi
- See next slide for how to launch this monster

Microsemi SF2 getting started

```
1   export LD_LIBRARY_PATH=/lib/i386-linux-gnu/:/usr/lib/i386-linux-gnu/
2   export LIBERO_INSTALLED_DIR=/work/MicroSemi/Libero_v11.7/
3   export PATH=$PATH:$LIBERO_INSTALLED_DIR/Libero/bin/
4   export PATH=$PATH:$LIBERO_INSTALLED_DIR/Synplify/bin/
5   export PATH=$PATH:$LIBERO_INSTALLED_DIR/Model/modeltech/linuxacoem/
6   export PATH=$PATH:$LIBERO_INSTALLED_DIR/../Linux_Licensing_Daemon/
7   export LM_LICENSE_FILE=1702@localhost
8   export SNPSLMD_LICENSE_FILE=1702@localhost
9   cd /work/MicroSemi/Libero_v11.7/Libero
10  /work/MicroSemi/Linux_Licensing_Daemon/lmgrd \
11        -c /work/MicroSemi/License.dat         \
12        -l /tmp/microsemi-lmgrd.log
13  libero
14  killall lmgrd actlmgrd
```

Altera SoCFPGA

- ARM Cortex A9 UP/SMP
- SPI NOR/NAND/SD storage, DDR2/3 DRAM
- Standard peripherals (I2C, SPI, CAN, USB . . .)
- Upcoming Stratix 10 is ARMv8 Cortex A53
- Usually runs U-Boot, Linux
- RTOS offerings exist, uC/OS, FreeRTOS
- Capable of running in AMP configuration

Altera SoCFPGA design software

- Altera Quartus , now intelFPGA
- Proprietary, but runs fine on Linux
- Project Typhoon

Altera SoCFPGA bootloader

U-Boot or MPL:

- U-Boot
  - Altera
    - 2013.01.01
    - Ancient, buggy, obtuse
  - Mainline
    - 2017.xx
    - Actively maintained
    - Altera is contributing
    - Used in production (use it)
- MPL
  - BSD-licensed bootloader
  - Bugs fixed in U-Boot not fixed here
  - Very rudimentary (init hw, start blob)

Altera SoCFPGA Linux support

- Vendorkernel
  - Reasonably recent 4.x
  - Altera is trying to keep it in sync with Linus
  - Still a lot of questionable patches
- Mainline
  - HPS peripherals supported out of the box
  - FPGA part needs a few patches from ML
    - DT overlay support
    - FPGA manager support
    - DT overlay support for FPGA manager

Xilinx Zynq

- ARM Cortex A9 or Cortex A53 (ZynqMP)
- SPI NOR/NAND/SD storage, DDR2/3 DRAM
- Standard peripherals (I2C, SPI, CAN, USB . . .)
- ZynqMP has a lot of multimedia stuff
- ZynqMP has GPU, but it's ARM Mali :-(
- Usually runs U-Boot, Linux
- RTOS offerings exist, uC/OS, FreeRTOS

Xilinx Zynq design software

- Xilinx Vivado
- Proprietary, but runs fine on Linux
- FOSS solution is in the works :-)

Xilinx Zynq bootloader

- U-Boot
  - Mainline U-Boot works, with limitations on ZynqMP
  - ZynqMP ATF loading is in progress
  - Xilinx is active at contributing
- FSBL + U-Boot
  - Xilinx's preloader with extended capabilities
  - Sets up the hardware, loads blobs, starts U-Boot
  - In this setup, U-Boot runs without SPL
  - This configuration is thus far needed on ZynqMP

Xilinx Zynq Linux support

- Vendorkernel
  - Reasonably recent 4.x
  - Xilinx is trying to keep it in sync with Linus
  - Version is usually picked based on Xilinx release cycle
  - Some questionable patches in the tree
- Mainline
  - PS peripherals supported out of the box
  - FPGA part needs patches from ML for Zynq
  - ZynqMP support is work in progress

U-Boot on SoCFPGA and Zynq

Altera SoCFPGA

- In Quartus, build project and generate handoff files
- Use qts-filter.sh in mainline U-Boot to process them
- Build mainline U-Boot to obtain u-boot-with-spl.sfp
- Install u-boot-with-spl.sfp to partition 0xa2 on SD card
- Install u-boot-with-spl.sfp to offset 0x0 on QSPI NOR
- Use `fpga` command to load FPGA RBF bitstream

Xilinx Zynq

- In Vivado, build project and generate HDF file
- Unzip HDF file to obtain ps*_init*.c and ps*_init*.h
- Copy the ps*_init* files to U-Boot source, build U-Boot
- Install BOOT.BIN to FAT partition on SD card
- Use `fpga` command to load FPGA BIT bitstream

Vendorkernel FPGA loading horror

- SoCFPGA: `cat bitstream.rbf > /dev/fpga`
- Zynq: `cat bitstream.rbf > /dev/xdevcfg`
- Enable bridges
- Access hardware via devmem and hope it works
- Bind drivers and enjoy how things work ...

But what if someone reprograms the FPGA while the driver uses it?

- Too bad, **GAME OVER**
- System hangs or misbehaves

Linux with DTOs

DTO - Device Tree Overlays

- Dynamic device tree
- Kernel can load DT fragments at runtime
- The "live" DT is patched by these fragments
- Fragments can be loaded via ie. configfs
- Drivers are bound based on the DT content

Linux DTO demo

```
1  overlaydir=/sys/kernel/config/device-tree/overlays/mydto
2  inputdts=/usr/share/dto/overlay.dts
3
4  # Compile and load DTO
5  mkdir $overlaydir
6  dtc -@ -I dts -O dtb $inputdts > $overlaydir/dtbo
7  #   ^^ this option indicates we're compiling DT fragment
8
9  #
10 # Do your stuff here
11 #
12
13 # Unload DTO
14 rmdir $overlaydir
```

DTO source

```
/dts-v1/;
/plugin/;
/ {
        #address-cells = <1>;
        #size-cells = <0>;
        fragment@0 {
                reg = <0>;
                target-path = "/soc/ethernet@ff700000";
                __overlay__ {
                        #address-cells = <1>;
                        #size-cells = <0>;

                        status = "okay";
                        phy-mode = "rgmii";
                };
        };

        fragment@1 {
                reg = <1>;
                target-path = "/soc/i2c@ffc04000/i2cswitch@70/i2c@1";
                __overlay__ {
                        #address-cells = <1>;
                        #size-cells = <0>;
                        eeprom@51 {
                                compatible = "at,24c01";
                                pagesize = <8>;
                                reg = <0x51>;
                        };
                };
        };
};
```

Linux FPGA manager

- Responsible for handling the FPGA part of the SoC
- Loads the FPGA bitstream
- Manages the bridges between SoC and FPGA
- Uses Linux firmware facility to obtain bitstream from FS
- Well integrated into Linux DM, unlike vendorkernel stuff
- Supports Altera SoCFPGA, Xilinx Zynq and Lattice iCE40 (more are coming)
- Supports partial reconfiguration too (here be dragons)

FPGA manager with DTOs

How it works:

- Describe FPGA content in DTO
- DTO must also point to a matching bitstream
- Load DTO into the kernel
- Kernel programs the FPGA (using FPGA manager)
- Kernel enables bridges (using FPGA manager)
- Kernel binds drivers based on the DTO content
- User is happy!

DTO can be removed:

- Kernel unbinds drivers
- Kernel disables bridges (using FPGA manager)
- FPGA remains programmed and running

FPGA manager DTO

```
/dts-v1/;
/plugin/;
/ {
        #address-cells = <1>;
        #size-cells = <0>;
        fragment@0 {
                reg = <0>;
                /* controlling bridge */
                target-path = "/soc/fpgamgr@ff706000/bridge@0";
                __overlay__ {
                        #address-cells = <1>;
                        #size-cells = <1>;
                        area@0 {
                                compatible = "fpga-area";
                                #address-cells = <2>;
                                #size-cells = <1>;
                                /* We use one bridge, so one range */
                                ranges = <0 0x00000000 0xff200000 0x00080000>;

                                firmware-name = "fpga/default/output_file.rbf";

                                a_16550_uart_0: serial@01000 {
                                        compatible = "altr,16550-FIFO128", "ns16550a";
                                        reg = <0 0x001000 0x00000200>;
                                        interrupt-parent = <&intc>;
                                        interrupts = <0 40 0>;
                                        clock-frequency = <32000000>;
                                        fifo-size = <128>;
                                        reg-io-width = <4>;
                                        reg-shift = <2>;
                                };
```

Conclusion

- All sorts of PL devices available
- Using SoC with FPGA in Linux today is becoming easy
- FPGA manager is great (already) !

The End

Thank you for your attention!

Contact: Marek Vasut <marek.vasut@gmail.com>