Anacleto

A framework for the integration of Linux FPGA SoC Devices

1A. Rigoni Garola, 1G. Manduchi, 1A. Luchetta,
  1C. Taliercio, 2T. Schröder

1Consorzio RFX, Padova, Italy
2Max-Planck-Institut für Plasmaphysik, Greifswald, Germany

Outline:

- Why SoC FPGA devices
- FPGA development in the market
- Anacleto
  - The project workflow
  - A working example
- Conclusions
Why SoC FPGA devices (1)

Why choosing FPGA System on Chip (SoC)?

- FPGAs recent growth in speed/complexity/volume designs.
- SoC solutions integrate a processor and a user configurable FPGA.
- Very cheap solutions in SoC boards.
- The most important aspect of using FPGAs is the interaction with the processing unit. With a flexible logic design it becomes easy to bridge the data with high-level tools for archiving or making runtime analysis. This is exploited by SoC boards able to embed a fully fledged OS like GNU Linux.

The interconnect between CPU and FPGA uses the recently developed, FPGA-friendly AMBA 4.0 protocol (AXI4), developed by ARM with input from Xilinx.

Red Pitaya  Parallella  MicroZed
Why SoC FPGA devices (2)

Data acquisition
The integrated of low-level logic components with a high-level OS layer loaded in a full featured ARM processor represent an ideal environment to build a fast data acquisition peripheral. Xilinux 28nm Zynq family is able to DMA transfer 64-bit AXI data at a rate of about 300 MBps.

Realtime processing
All the synthesized algorithms in the FPGA are real time by definition, a precise value of the execution time and the clock latency are given by the implementation tools. The standard C,C++ algorithm originally developed for processor units can be in principle ported to RTL logic.

Configurable pre-elaboration on sensors devices
Unlike a soft-cpu deployed on the FPGA, SoC devices usually can perform the FPGA reconfiguration on the fly while maintaining the system up and running. A partial or complete reconfiguration in possible in fractions of second giving a chance to store different firmware for different experimental setup.

Computational speed-up
FPGA can be used to carry out critical computation profiting of the possibility to customize instructions in logic. The DMA controller inside SoC devices can be used to relieve the cpu of the overall computation. Some tools (Xilinux SDAccel, ) are available to export OpenCL kernels directly in RTL computational IPs.
FPGA development in the market

FPGAs led the overall semiconductor market in 2016 (6% vs. 1.5%).

Although Xilinx has been at the top of the rankings for over a decade, many producers are facing the market covering many different technology aspects.

From the designer perspective the FPGA development involves a very heterogeneous set of tools to operate: **write code in different HDLs, integrate code from external vendors, simulate, synthesize, write timing closures and handle the software interface**.

Almost all vendors provide a design suite to cover the same tasks.
FPGA development in the market

- Most of the FPGA vendors tend to promote their **proprietary software tools** to handle the design flow from the synthesis to the bitstream generation. **The code can not be easily exchanged among different SoC platforms.**

- Tcl is widely adopted to write project scripts, but with custom interface commands with different syntax.

- Many vendors support GNU Linux but with opaque systems to handle drivers and software layers.

<table>
<thead>
<tr>
<th>FPGA Vendor</th>
<th>Design</th>
<th>Synthesis</th>
<th>Simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xilinx</td>
<td>ISE, Vivado® HLx suite</td>
<td>Vivado®</td>
<td>Vivado®</td>
</tr>
<tr>
<td>Intel Altera</td>
<td>Quartus® Prime</td>
<td>Quartus® Prime</td>
<td>Quartus® Prime</td>
</tr>
<tr>
<td>Microsemi</td>
<td>Libero® SoC</td>
<td>Synplify™</td>
<td>ModelSim®</td>
</tr>
<tr>
<td>Lattice</td>
<td>ispLEVER Classic</td>
<td>Synplify™</td>
<td>Active-HDL™</td>
</tr>
</tbody>
</table>
Anacleto aims at providing a tool to automate the build and integration of FPGA logic and software programs for common SoC devices.

- It handles the integration of several tools and packages that are required to develop a mixed FPGA logic and Linux software application in a single project flow.
- In this way the development time is drastically reduced and repository organized.
### Project workflow example

#### STEP 1: preparation of the source directory
- Automated components download and source directory tree creation
- Generation of the configure scripts

#### STEP 2: setup the Anacleto project specific build directory
- Set options to configure the current build on the desired board.
  - FPGA version and path
  - Linux kernel version
  - Toolchain version
- Automated verification of the correctness of given options in the current system
- Configure data deployment to the target board.
- Automatic retrieval of external sources and compilation of linux-kernel using board specific configuration.

```
./bootstrap
```

All sources and build script have been setup and all required submodules updated.

```
./configure --with-soc-board=red_pitaya
```

Build directory is configured with system environment and available tools. The Makefiles populate project directories.

```
make
```

Now Linux kernel is downloaded and compiled with a suitable toolchain with parameters coming from the board setup or user configurations.
Project workflow example

From the specific project directory

- make new_project
- make open_project

**STEP 3) edit project**

Automatic creation of a set of tcl scripts that setup the proper configuration of the xilinx IDE (Vivado)

- vivado_src
- vivado_project

**STEP 4) write project back to sources**

The configuration created using graphical tools can be stored back to the corresponding set of scripts that are stored in repository.

- vivado_src
- vivado_project
Project workflow example

```vhdl
entity add_test is
  port (
    a : IN STD_LOGIC_VECTOR (31 downto 0);
    b : IN STD_LOGIC_VECTOR (31 downto 0);
    ap_return : OUT STD_LOGIC_VECTOR (31 downto 0)
  );
end;

architecture behav of add_test is
  begin
    ap_return <= std_logic_vector(unsigned(b) + unsigned(a));
  end
```

STEP 5) Peripheral design

- IP package build from user provided HDL.
- It is now necessary to connect the IP with the CPU.
- Three levels of integration: logic, devicetree, driver
- Each level handled by different tools all integrated in Anacleto.

A precise timing report can be obtained on the basis of the selected SoC board for the specific project.
STEP 6) implementation

- The logic description is compiled as a proper bitstream for the FPGA.
- A *devicetree* file is generated describing the hardware to the kernel.
- A set of Linux driver files is created matching the defined hardware.
Project workflow example

STEP 7) deploy files and run

- all files are copied to target board via sftp

Now it is possible to:
- load bitstream file into the FPGA
- load peripheral driver
- enjoy the new device working

```
root@rp01# cat rfx_adder.bit > /dev/xdevcfg
root@rp01# insmod rfx_adder.ko
root@rp01# ls /dev/rfx_*
rfx_adder0
root@rp01# ./adder_test /dev/rfx_adder0
1 + 1 = 2
TEST OK!
root@rp01# _
```
A working example

- We recently developed on a cheap board (RedPitaya) a configurable timing module that is used in some diagnostics in W7X.
- **MDSplus** has been used to interface the device to the external world.

Anacleto successfully supervised the development of the internal logic, the generation of the linux drivers and the compilation of software modules.
Conclusions

- SoC devices promise to be a valuable choice for new cheap and flexible devices in fusion experiments control and acquisition. In particular they are mostly effective where a complex communication between a running application and a FPGA programmed logic is required.

- The applicability of SoC architecture can be improved using the Anacleto framework that hides to the developer some of the intermediate steps and most of the system configuration.

- Anacleto has been already successfully exploited in the development of a real case project where a highly customizable timing board was foreseen.
Thanks for your kind attention
already implemented features

```
configure --enable-kconfig
```

```
make reconfigure
```

```
./configure --enable-kconfig=nconf
```

The configuration step relies on **kconfig**: a configuration file parser and an interface gui that was made for the linux kernel. The kconfig has been adapted to be launched by the Anacleto during configure step and it helps to manage options saving them in a `config` file. Those config files will be also used to handle boards automatic configurations just like what the kernel does with defconfig targets.

Example of ncurses kconfig gui

```
Select SOC board (red_pitaya)  --->
General build options  --->
[*] Enable PS cross compiler toolchain  --->
[*] Enable FPGA development toolchain  --->
LINUX_BUILD  --->
YOCTO  --->
[*] Docker Build support  --->
TARGET_DEVICE  --->
```

CONSORZIO RFX
Ricerca Formazione Innovazione

11th IAEA TM 2017 GREIFSWALD
A framework for the integration of Linux SoC Devices
<andrea.rigoni@igi.cnr.it>
already implemented features

- The configure step is able to run itself in a docker container.
- The use of preconfigured containers dramatically reduce the installation time that is necessary to fulfill all software requirements needed by FPGA tools.
- Some tools (for example Vivado) have to be started from predefined distributions

Docker can be setup during configure step in three flavors:

```
configure --with-docker-image=rigoni/linux-socdev:latest
configure --with-docker-url=.  
configure --with-docker-container=[existing container name]
```

GUI of the Kconfig version configuring the Docker options
Features coming soon

- Integrate HLS compilation
- Integrate Yocto for layered ecosystem development
- DMA driver template based on selected AXI interface
- Development of DMA driver able to handle the scatter gather capability
- Provide drivers with the netlink kernel-user bridge
- Extend support for other boards: