Pynq and APSoC

Stefano Spellini

Acknowledgements

- **This lesson has been made possible thanks to the courtesy of Xilinx Inc.**
  - Leading company in the programmer logic market
  - Inventor of the FPGA (1985)
  - First semiconductor company with a fabless manufacturing model

- Xilinx Inc. donated to the University of Verona:
  - 5 PYNQ-Z1 FPGAs,
  - 25 SDSoC licenses,
- And gave full access to their training material

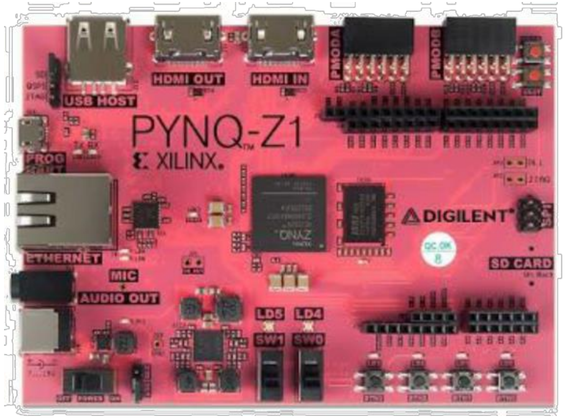
- More information on the **Xilinx University Program**:
  - https://www.xilinx.com/support/university.html

What is PYNQ?

- Open-source project from Xilinx
  - makes it easy to design embedded systems
    - with Xilinx Zynq All Programmable Systems on Chips (APSoCs).
    - Using the Python language and libraries,
    - exploits Zynq programmable logic and microprocessors
- Users can now create high performance embedded applications
  - parallel hardware execution
  - high frame-rate video processing
  - hardware accelerated algorithms
  - real-time signal processing
  - high bandwidth IO
  - low latency control
- The PYNQ-Z1 is the first Zynq board to support PYNQ.

INITIAL SETUP

Prerequisite



- PYNQ-Z1 board
- SD Card with preloaded Image*
- Ethernet Cable
- USB cable and available USB port

*Instructions to download and prepare SD card:
http://pynq.readthedocs.io/en/latest/getting_started.html

Connecting to the board

1. Configure board to boot from SD Card
2. Set Jumper to power from Power Supply
3. Insert SD Card
4. Connect ethernet to the central plug of your desk
5. Power on

- Connection
  - SSH: Secure Shell
    - Textual environment
      - Give commands
      - Edit files (e.g., nano, Vi, Vim) and filesystem
  - SMB: Network service
    - smb://<address to the board>/xilinx
  - Jupyter
    - boardfpga0X.device.univr.it:9090
  - Board names in the DNS
    - boardfpga02.device.univr.it
    - boardfpga03.device.univr.it
    - boardfpga04.device.univr.it
    - boardfpga05.device.univr.it
  - User:Password = **xilinx:xilinx**

Material online

- Unzip the 06_sources.tar.gz
  - `$> tar xzfv 06_sources.tar.gz`
  - Three directories:
    - 1_christmas_lights: first exercise
    - 2_hls_and_SoC: second exercise
    - config_files: constraints file for PYNQ-Z1

How we use to do it…

Classic FPGA development flow

Xilinx Design Constraints (XDC) file

- Take a look at the **PYNQ-Z1_C.xdc** file, in the **config_files** folder

```
#set_property -dict { PACKAGE_PIN H16   IOSTANDARD LVCMOS33 } [get_ports { sysclk }];
#create_clock -add -name sys_clk_pin -period 8.00 -waveform {0 4} [get_ports { sysclk }];

##Switches
#set_property -dict { PACKAGE_PIN M20   IOSTANDARD LVCMOS33 } [get_ports { sw[0] }];
#set_property -dict { PACKAGE_PIN M19   IOSTANDARD LVCMOS33 } [get_ports { sw[1] }];

##RGB LEDs
#set_property -dict { PACKAGE_PIN L15   IOSTANDARD LVCMOS33 } [get_ports { led4_b }];
#set_property -dict { PACKAGE_PIN G17   IOSTANDARD LVCMOS33 } [get_ports { led4_g }];
#set_property -dict { PACKAGE_PIN N15   IOSTANDARD LVCMOS33 } [get_ports { led4_r }];
#set_property -dict { PACKAGE_PIN G14   IOSTANDARD LVCMOS33 } [get_ports { led5_b }];
#set_property -dict { PACKAGE_PIN L14   IOSTANDARD LVCMOS33 } [get_ports { led5_g }];
#set_property -dict { PACKAGE_PIN M15   IOSTANDARD LVCMOS33 } [get_ports { led5_r }];

##LEDs
#set_property -dict { PACKAGE_PIN R14   IOSTANDARD LVCMOS33 } [get_ports { led[0] }];
#set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 } [get_ports { led[1] }];
#set_property -dict { PACKAGE_PIN N16   IOSTANDARD LVCMOS33 } [get_ports { led[2] }];
#set_property -dict { PACKAGE_PIN M14   IOSTANDARD LVCMOS33 } [get_ports { led[3] }];

##Buttons
#set_property -dict { PACKAGE_PIN D19   IOSTANDARD LVCMOS33 } [get_ports { btn[0] }];
#set_property -dict { PACKAGE_PIN D20   IOSTANDARD LVCMOS33 } [get_ports { btn[1] }];
#set_property -dict { PACKAGE_PIN L20   IOSTANDARD LVCMOS33 } [get_ports { btn[2] }];
#set_property -dict { PACKAGE_PIN L19   IOSTANDARD LVCMOS33 } [get_ports { btn[3] }];
```

Hands-on: Christmas Lights

- Implement the following asynchronous functionalities
  1. **The Led number 3 is on whenever you press button 3**
  2. **Consider the Switches (SW1, SW2) and Led 0, 1, 2.**
  - Switches are binary input (SW1 is the first bit, SW2 the second)
  - Leds 0-2 are binary ouput
  - light a number of leds equal to the number represented by switches
    - E.g.,
      SW1 & SW2 (11) => led0 & led1 & led2
      SW1 & not SW2 (01) => led0 & not led1 & not led2

Hands-on: Christmas Lights

- Implement the following synchronous functionalities

1. **Use the two RGB leds to represent a 6 bit number**

- Each 100.000.000 clock cycles add one to an output port
- Represent the value of the output using the RGB leds
  - 3 bit each, to represent R, G, and B.
- Button 0 can be used to reset the functionality
  - Bring back the output port to 0
- Manage the case when the output port is 111111
  - Re-set to 000000 as next change
  - Try to go backward (next change 111110)

Hands-on (continue)

HLS of All Programmable SoC

FPGA overlays – hardware libraries

- Overlays are generic FPGA designs that target multiple users with new design abstractions and tools

- Overlay characteristics
  - Post-bitstream programmable via software APIs
  - Typically optimized for given applicationdomains
  - Encourages the use of open source tools & fast compilation
  - Enables productivity by re-using pre-optimized designs
  - Makes benefits of FPGAs accessible to new users

Anatomy of an overlay IP subsystem

- Designed to be immediately reused by anyone
  - or re-purposed elsewhere by “person skilled in the art” (PSITA)

- Comprises
  - Programmable FPGA IP core
  - FPGA bitstream
  - C code to expose programmable functionality
  - Python-to-C bindings
  - Python library with API
  - Protocol

Hands-on

- Open **Vivado** without creating a new project
- In the **TCL Console**
  - cd to the **2_hls_and_SoC/** folder
  - `$> source build_all.tcl`
  - let Vivado run (it will take a bit of time)
- Find the **addresses** of the registers of the IP elements
  - Take a look at the **sources** of the wrapped design
  - **Note down those addresses**

Python programming environment

- In the **hls_and_SoC/overlays/** folder you will find the bitstream and the .tcl files

- Starting on PYNQ (Note: X is your board number)
  - Copy the bitstream and the TLC file produced by Vivado

```
$> scp design_adder.* xilinx@boardfpga0X.device.univr.it:~/
```

  - Open a ssh connection to the board

```
$> ssh xilinx@boardfpga0X.device.univr.it
```

  - *Check the operating system*

```
$> uname -a
```

  - Open Python

```
$> sudo Python3.6
```

Main PYNQ python functions

- Load the Overlay python functions

```
>>> from pynq import Overlay
```

- Open an Overlay

```
>>> overlay = Overlay('file.bit')
#file.bit: path to the bitstream file
#file.tcl must be present in the same directory
```

- Manage the objects

```
>>> help(overlay)
```

It returns you the informations about the overlay. E.g., list of Ips

```
>>> myip = overlay.myip_instance
>>> help(myip)
```

It returns you the informations about the ip

- Interract with objects

```
>>> myip.write(offset, value)
>>> myip.read(offset)
```

In the offset parameter, use the addresses that you noted down!

- More informations: http://pynq.readthedocs.io/en/latest/