UDP Interface for hls4ml

JUSTUS-LIEBIG-UNIVERSITÄT GIESSEN
JENNIFER 2 EU grant n.822070
ERUM DATA HUB
Bundesministerium für Bildung und Forschung
ErUM-FSP T09 Belle II

Inhalt

Part I: Introduction

- Setup
- Things to Consider
- Goal

Part II: HLS - High-Level Synthesis

- Overview
- Comparison to VHDL

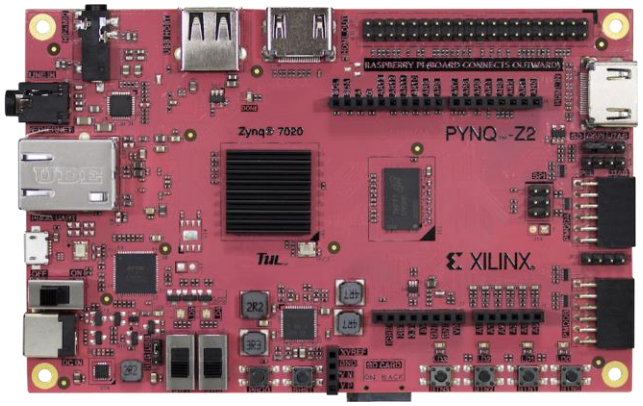
Part III: Ethernet + UDP/IP

- Ethernet PHY
- Networking Workflow with UDP/IP
- Ethernet Frame
- IPv4 Packet
- UDP Packet
- Implementation

Part I: Introduction

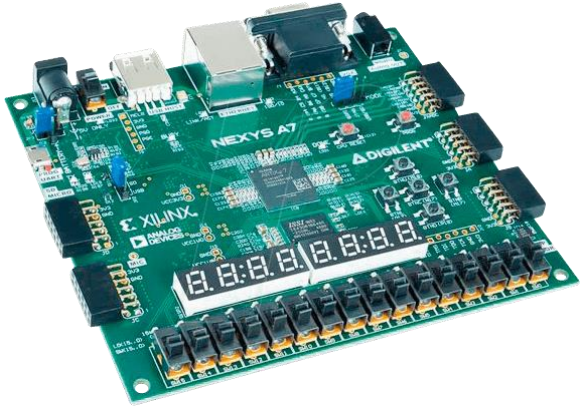
Setup

Working with



Source: https://www.realdigital.org/img/16ebb6e013abdd79d4a33733c8ff0fc9.png

Xilinx Z7000 series (PYNQ Z2)
SoC (System on Chip)



https://cdn-reichelt.de/bilder/web/xxl_ws/A300/DIGIL_410-292_1.png

Nexys A7
„Pure“ FPGA

Setup

- hls4ml
  - Developed at CERN
  - Mostly uses TensorFlow
- FINN
  - Developed at Xilinx / AMD
  - Uses PyTorch

Things to Consider

- Prerequisite for Neural Networks on FPGAs: Getting data in and out of the board
- A simple standard interface might be useful: UDP
- Complex logic can be developed faster with HLS – high-level synthesis
- Note: Xilinx SoC chips add capability to load data into the programmable logic via CPU/RAM

Goal

- Being able to send and receive data via ethernet to neural network applications on FPGAs without relying on SoC-capabilities
- Computer-to-board or board-to-board communication

Part II: HLS – High-Level Synthesis

Overview

- Abstraction level above common hardware description languages (VHDL, Verilog)
- Uses C++, C, or SystemC compiled down to VHDL or Verilog
- Enables faster development of complex hardware modules
- Enables automatic implementations of industry-standard module interfaces (AXI4 and other handshake protocols)

Comparison to VHDL

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity toggler is
    Generic (
        FREQ_IN: integer := 50000000
    );
    Port (
        CLK: in std_logic;
        O: out std_logic
    );
end toggler;

architecture Behavioral of toggler is
    signal cnt: unsigned(31 downto 0) := (others => '0');
    signal toggle: std_logic := '0';
begin
    counter_proc: process(CLK)
    begin
        if rising_edge(CLK) then
            if cnt = FREQ_IN then
                cnt <= (others => '0');
                toggle <= not toggle;
            else
                cnt <= cnt + 1;
            end if;
        end if;
    end process;

    O <= toggle;
end architecture;
```

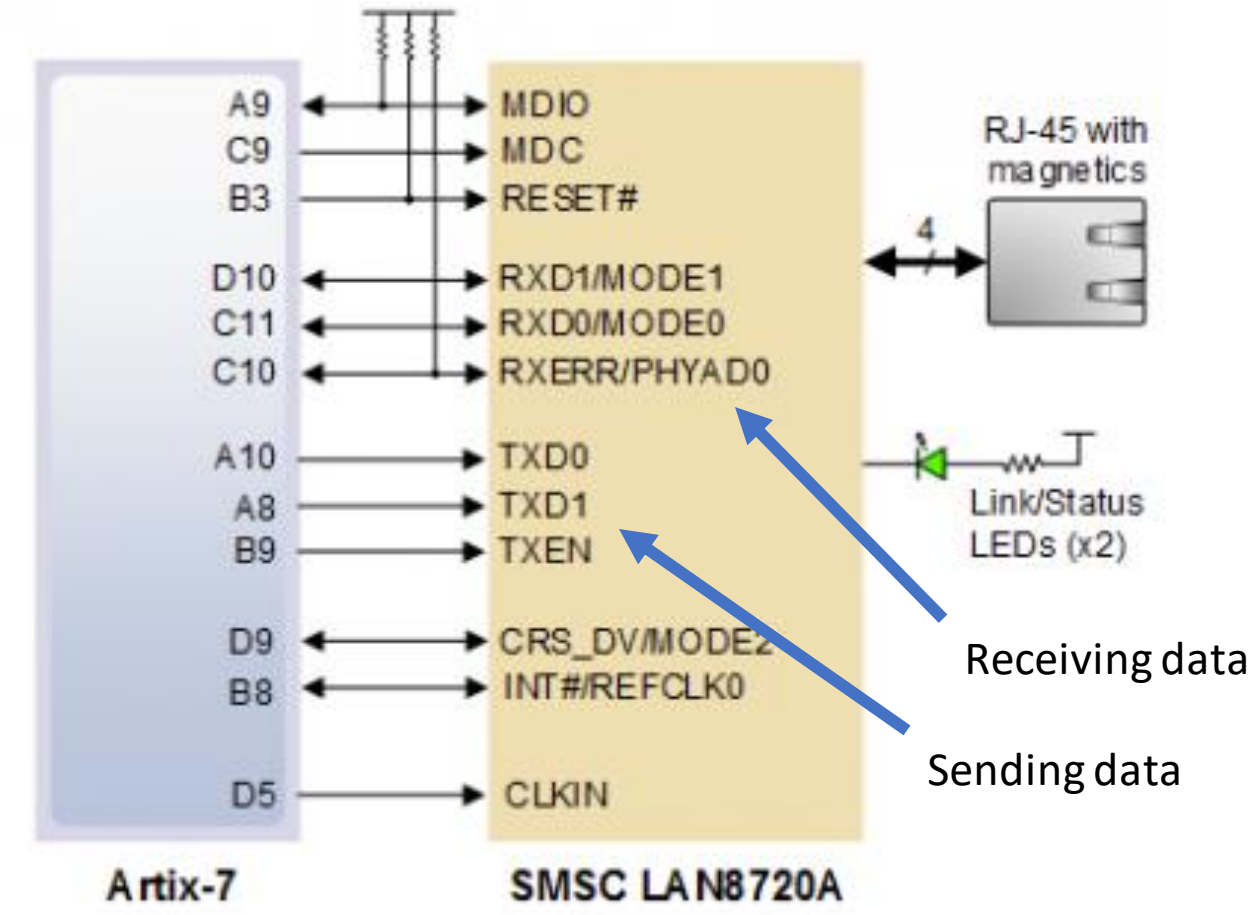
```cpp
// toggler.hpp
#include <ap_int.h>

void toggler(ap_uint<1> &o);


// toggler.cpp
#include "toggler.hpp"

void toggler(ap_uint<1> &o) {
  static ap_uint<32> cnt = 0;

  if (cnt == 50000000) {
    cnt = 0;
    o = !o;
  } else {
    cnt++;
  }
}
```

Part III: Ethernet + UDP/IP

Ethernet PHY – MAC Interface

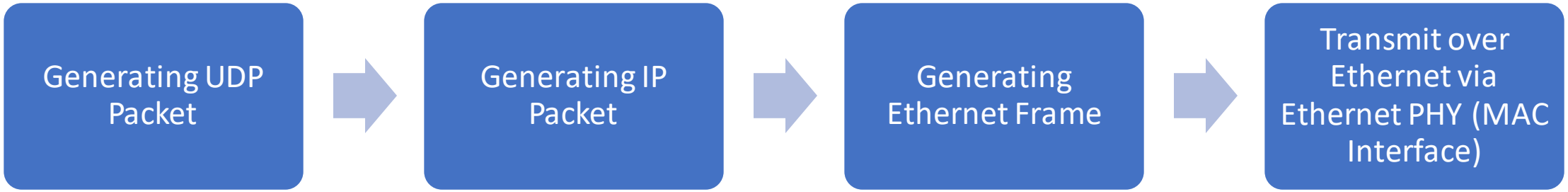
Ethernet PHY is responsible for

- Encoding/Decoding
- Synchronization of clock signal
- Negotiation of speed and other settings with the communication partner

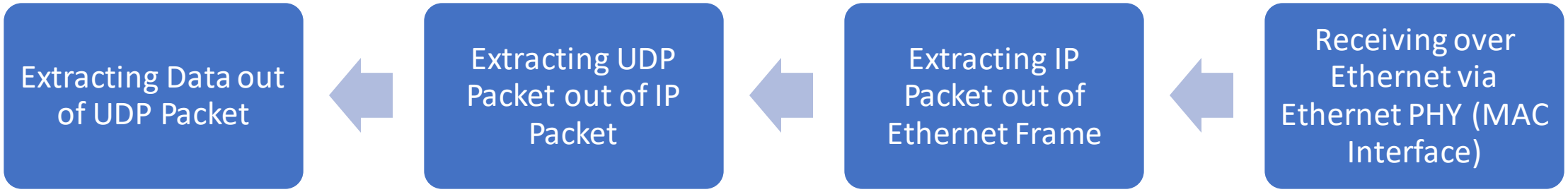


Networking Workflow with UDP/IP

Sending



Receving



Ethernet Frame

Offset	Byte	0								1								2								3							
Byte	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Destination MAC Address																															
4	32	Destination MAC Address																Source MAC Address															
8	64	Source MAC Address																															
12	96	Ethertype																Payload (46 – 1500 bytes)															
16	128	Payload (46 – 1500 bytes)																															
?	?	Frame Check Sequence																															

Ethernet Frame

- Uses 48-bit MAC addresses
- Ethertype: Protocol used within payload data (IPv4, IPv6, ARP, ...)
- Frame Check Sequence: Checksum appended to the end of the frame; CRC32 algorithm (polynomial division of data by predefined polynomial)
- Additionally, each Ethernet Frame transmission is encapsulated with a preamble (0xAA 0xAA 0xAA 0xAA 0xAA 0xAA 0xAA) and the start of frame delimiter (0xAB) at the front and the interpacket gap (idle time between frames) at the back.

IPv4 Packet

Offset	Byte	0								1								2								3							
Byte	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				IHL				DSCP						ECN		Total Length															
4	32	Identification																Flags			Fragment Offset											
8	64	Time to Live								Protocol								Header Checksum															
12	96	Source IP Address																															
16	128	Destination IP Address																															
20	160	Options (Used if IHL > 5)																															
24	192																																
28	224																																
32	256																																
36	288	Payload (26 – 1480 bytes)																															

IPv4 Packet

- Uses 32-bit IP addresses
- Protocol: Protocol used within payload data (UDP, TCP, ICMP, …)
- Header Checksum: Checksum of the header data; simple ones complement addition of 16-bit words

UDP Packet

Offset	Byte	0								1								2								3							
Byte	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source Port																Destination Port															
4	32	Length																Checksum															
8	64	Payload (18 – 1472 bytes)																															

UDP Packet

- Uses 16-bit ports
- Checksum: Checksum of the packet data; simple ones complement addition of 16-bit words
- May contain packets of other subprotocols in its payload (DHCP, DNS, …)

Implementation

- Simple UDP echo server
- Connected directly via ethernet to a computer
- Code available at https://github.com/lp247/FPGA-Networking
- Limitations:
  - General network connections not possible, more protocol implementations like ARP necessary

Thank you for your attention!