# Why new protocolks were added not to system but to user layer

Original project implements all protocols at the system layer. This is best solution from architecture point of view because user no need think about any system purposes. Everything is hidden in the low level modules. Also that modules are located in the system folders. When any new board is added, it will use that files.

But of course, we must pay for this architect. The payment is using of multiplexers. There are many AXI Multiplexers are present in the project for choose the way of data from layer to layer. That multiplexers are producing many wires. And wires are worst item for Lattice FPGA.

Lattice FPGA has no good solutions for inter-block communication. Most connections are going through trace resources. That is why router has much problems to optimize FMax. AS a result, any change of design can significantly changes FMax. Strange, but sometimes adding of extra wires improves FMax to 10..20 MHz. Sometimes remove of any modules drops FMax. That is wwhy we can say that FMax depends not only from Netlist, but also from router’s quality. And less wires => easier processing for router.

That is why if we can skip multiplexer for Lattice FPGA, better to do it. And that is wy we added all protocols to User Layer.

For now , all support files are also located in the Top Level Folder.Maybe later will be better to move them to any common place because now we must synchronize that files when any changes were added. From other hand, now we are providing many experiments with different boards and sometimes that differences are important. But in the future, all versions of the protocol support modules will be the same.

# Which buses we have at the User Layer

The User Layer has two global buses (x2 directions = 4 totally) for provide the functionality.

**rx\_udp\_.**.. This bus will be activated when detected UDP Datagram is came from the network. The receiver is processing datagrams in the promiscuous mode but it contains all addressing information such as Target IP, Target MAC and Target Port. User layer’s logic can use or ignore it.

**tx\_udp\_**.. This bus can be used for send UDP Datagram from User Layer to the network.

**rx\_ip\_**... All Ethernet Frames which were not served at Low Level and not detected as UDP Datagrams will be routed to this bus. We are using it for provide ICMP Protocol

**tx\_ip\_**... If we need send any non-UDP Packet, we must do it, using this bus

# Project’s items

Now project contains 4 subprojects.

1. Overclocked version for ColorLight\_5A-75B board. Uses Ethernet in Gigabit Mode.
2. 100M version for ColorLight\_5A-75B board. System Bus has CLK= 104.167 MHz. Why not 100M? This value is got from the PLL Generator. We ordered primary clock 125 MHz because this is most important for RGMII. Of course, we ordered Secondary Frequency 100 MHz but got this value. Also Chinese Oscilloscope and Chinese Logic Analyzer shows that duty cycle of PLL Output is not 50%. Maybe this is problem of Chinese equipment at high frequency, maybe not.
3. Overclocked version for ColorLight\_i5 Board (DIMM Version). Uses Ethernet in Gigabit Mode.
4. 100M version for ColorLight\_i5 Board (DIMM Version) with System Clock 104.167 MHz

Why we still using two boards? The most problem of current DIMM solution is motherboard. We don’t see any input pins. Also we don’t know which FPGA pin is connected to the button on that motherboard. But we need Input pins at least for force DHCP or AutoIP processing.

For ColorLight\_5A-75B, we know which pin is connected to the button. That button is single. We cannot test DHCP and AutoIP together. But one is more than nothing! That is why we must use ColorLight\_5A-75B too.

# Project’s top hierarchy



The Top Level Module has name **fpga**. It is located in file **fpga.v**. This tradition is came from the original project. This file contains all FPGA Based items such as PLL, and some non-standard pins, if needed. That pins are needed for debug purposes, but not for real production. Also traditionally, this module contains LED Blinker because it also depends of hardware depended frequency divider. This is last module in ther hierarchy, which contains any physically depended information.

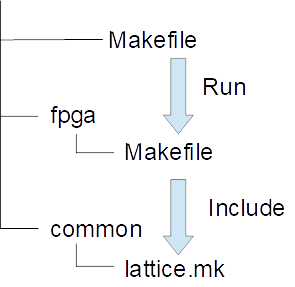
Next layer has name **fpga\_core**. This module is also came from the original project. Exacxtly this module is used for automatic tests in the original project. It has unified interface. This interface is same for Xilinx, Altera, Lattice etc.

The module **fpga\_logic** is added at this project. We separated it from fpga core for simplify simulation process in the ModelSim. This module receives Ethernet data already packed into AXI-Stream format. That is why we can use files, captured by WireShark without translate them into any more low level fromat and without thinking about any low level CRS. The mail FSM is located in this file.

# Project’s metafiles

## Makefile

The project comprises a structure of Make files (see structure below). Main Make file located in a main folder of every example. This Make file comprises general project information and runs more specific Makefile located in **fpga** folder.



fpga/Makefile specifies:

* list of files to be processed by Yosys
* Paramters for Nextpnr
* Trellis file location
* Command downloading firmware to FPGA

The most important settings:

FPGA\_TOP – name of target file.

SYN\_FILES – list of files to be processed by Yosys.

NEXTPNR\_PARAMS – Parameters for Nextpnr, including:

* type of package,
* lpf-file describing the FPGA’s pinout
* log-file, where information on place and route process
* times – this is not Nextpnr parameter, it is intended to be provided to common/place\_and\_route.py script (described below) and defines the number of Nextpnr passes with different seeds.

fpga/Makefile includes common/lattice.mk, which comprises rules for targets all, and clean. Also by target $(FPGA\_TOP).ys it specifies process of creation of script for Yosys. This script comprises commands and list of files to be processed by Yosys.

Among other things, lattice.mk runs common/place\_and\_route.py script, which is described below.

## NextPNR Script

According to our investigations, FMax may differ for different seed parameter of Nextpnr. This difference may be significant (10 to 20 MHz) that is why we have implemented a python script, which runs Nextpnr several times with different seed values and chooses the best one.

The lattice.mk runs common/place\_and\_route.py script and provides it with parameters for Nextpnr and –times parameter, which defines number of times Nextpnr will be run with different seeds (7 times by default).

The script chooses the best seed by analyses of Nextpnr’s log-file (can be set by NEXTPNR\_PARAMS in fpga/Makefile or is equal to Logfile.log by default) based on parsing errors related to FMax values.

# Protocol Based Modules

## Module SpiFlashReader

This module is used for give access to SPI Flash. The logic of this module is unusual for the project. Most modules are waiting in idle state most time. This module drops to reset state when all strobes are inactive. It is made for release SPI Bus immediately.

if (rst || ((read\_strobe == 0) && (write\_strobe==0)&& (erase\_strobe==0)))

begin

state <= idle;

spi\_cs <= 1;

spi\_sck <= 0;

spi\_mosi <= 0;

m\_tvalid <= 0;

s\_tready <= 0;

finished <= 0;

end else

User must set up 23 bit **start\_addr** parameter then activate read, write or erase strobe. Tata will be read into Master AXIS, written from Slave AXIS or erased (4K block) without transferring data. For erase case, **finished** signal will be activated at the end of the process.

## Icmp

This module contains FSM for processing ICMP Functionality. For now ip\_read\_... and ip\_write\_... buses are simply routed into this module and cannot be used at the top level.

## Module DHCPhelper

This module is just helper. It is designed for simplify main module and contains specific functionality for provide DHCP protocol. There two State Machines are present. Forst FSM is using for Generate DHCP Traffic, the second – for analyze incoming DHCP packets.

When we designed second FSM, the FMAx was dropped to value about 80 MHz. Probably it happened because too many compares of wide values are providing at the same clock period. We don’t see any good way for put them to the pipeline that is why we added one more clock domain with Frequency = 52.0833 MHz (also result of auto generated PLL, we ordered for 50 MHZ).

That is why DHCPHepler contains not only two State Machines, but also Dual Clock FIFO. This FIFO is filling at the System CLK (125 MHz for overclocked case or 104.167 MHz for 100M Case). Read speed is 52.0833 MHz. And the second FSM is clocked from the same slow speed.

Also module contains some shift registers fro provide cross clock domain for control and status signals.

## Module mDNShelper

Very simple module which contains some logic for remove it from the main module. It contains comparator for detect Unicast MAC Address 01 00 5E 00 00 FB which is used for mDNS.

All other functionality was easier keep in the main FSM. Any separate could create too much extra wires which could drop FMax.

## Module prng

AutoIP Protocol requires Random Generator. This module is providing this functionality. DHCP also uses it for generate XID Value.

## Module lfsr

This module replaces basic CRC Calculate method because original CRC Generator is using very specific Verilog Language constructions and Yosys producing wrong RTL code for it. We got more traditional version of CRC Generator. Need use it in the project instead original file

# Main FSM Logic

TODO

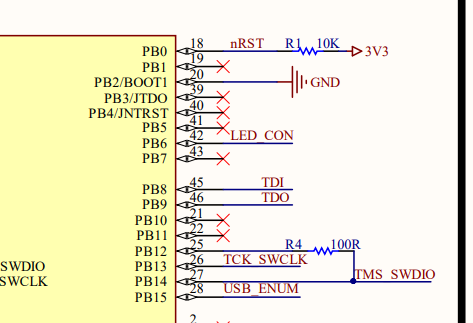
# How to unprotect EEPROM

When Board is came from Ali Express, the EEPROM Area is protected. We need unprotect it. **NB!** I don’t know why, but my ColorLight\_5A-75B board self protects EEPROM some days later after unlocking. Need unlock it again before wirting. “DIMM” version has no this problem. I unlocked it once.

For provide unlock operation, First of all, need use CMSIS DAP adapter. We made this adapter from BluePill Board, using the project

[wuxx/Colorlight-FPGA-Projects: current focus on Colorlight i5 series module (github.com)](https://github.com/wuxx/Colorlight-FPGA-Projects)

The pinout can be got from Schematic folder (TDI – PB8, TDO – PB9, TCK – PB13, TMS – PB14).



The firmware file is: **\firmware\flash\_image\_20210824.bin**

When this adapter is used, we can use **ecpdap** project

Where to get:

[adamgreig/ecpdap: ECPDAP allows you to program ECP5 FPGAs and attached SPI flash using CMSIS-DAP probes in JTAG mode. (github.com)](https://github.com/adamgreig/ecpdap)

How to use:

**D:\LATICE\ecpdap\_windows\_x86\_64.exe flash unprotect**

# How to adjust Network Parameters

Describe Qt program

# How to flash firmware with

Describe how to build Flash Image with parameters and flash it