VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV RADIOELEKTRONIKY
DEPARTMENT OF RADIO ELECTRONICS

JEDNODUCHÝ PRŮMYSLOVÝ ETHERNET
INDUSTRIAL LOW COMPLEXITY ETHERNET SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. Vladimír Šustek

VEDOUCÍ PRÁCE
SUPERVISOR
doc. Ing. Tomáš Götthans, Ph.D.

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Elektronika a sdělovací technika**
Ústav radioelektroniky

Student: Bc. Vladimír Šustek ***ID:*** 164421
Ročník: 2 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Jednoduchý průmyslový Ethernet

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s možnostmi implementace lwIP -- IP stacku v embedded systému architektury ARM. Navrhněte embedded systém implementující univerzální vstupně výstupní zařízení komunikující na lwIP TCP/IP vrstvě s důrazem na kompatibilitu zařízení s embedded periferiemi komunikujícími pomocí rozhraní PMOD, Arduino a standardy Analog Devices, Inc. Pro systém použijte mikrokontrolér ADI adUCm4050 a jako linkovou vrstvu neveřejný prototyp ethernetového připojení s integrovaným MAC. Dalším požadavkem tohoto modulu je univerzálnost napájení v standardních průmyslových aplikacích stejně jako napájení v uživatelských podmínkách (USB) a navrhnutí základní ochrany bloku zdroje. Seznamte se s průmyslovými komunikačními protokoly jako například MODBUS, či OPC - UA pro další část práce.

Zajistěte výrobu hardwaru zmíněného embedded systému s dodržením zásad při použití vysokorychlostních komunikačních bloků (Ethernet). Naprogramujte a oživte vytvořený systém, zprovozněte lwIP -- IP stack. Zvolte praktickou a jednoduchou demonstrační aplikaci (například měření teploty , či tlaku vzduchu) a vyberte vhodný průmyslový protokol jako nádstavbu systému (MODBUS, OPC- UA). Daný protokol implementujte a připojte k vámi vytvořené síti založené na průmyslovém protokolu. Navrhněte grafické rozhraní pro interpretaci získaných dat. Zhodnoťte robustnost systému, zvažte, případně navrhněte možná bezpečnostní vylepšení a další rozšíření.

DOPORUČENÁ LITERATURA:

[1] DUNKELS, A. Design and Implementation of the lwIP TCP/IP Stack. Swedish Institute of Computer Science 2001. Online https://www.artila.com/download/RIO/RIO-2010PG/lwip.pdf

Termín zadání: 4.2.2019 ***Termín odevzdání:*** 16.5.2019

Vedoucí práce: doc. Ing. Tomáš Götthans, Ph.D.
Konzultant:

prof. Ing. Tomáš Kratochvíl, Ph.D.
předseda oborové rady

ABSTRACT

The diploma thesis is focused on the building embedded demonstration application of the proprietary Low Complexity Ethernet module for industrial usage further called the LEN/LES 2. At the first, main used technologies such as MCU, or the lightweight IP stack is discussed, Consequently, there is detailed view on system hardware architecture proposed by hardware and software requirements. Then though part describes blocks of embedded system are in term of specific parts and hardware requirements to create universal board. Following chapters expresses first startup and known hardware bugs, LWIP implementation and MODBUS system implementation. The core of the system is the new released microcontroller an ADuCM4050 and the Low Complexity Ethernet MAC-PHY prototype block and much more dependent convenient peripherals of the MCU based application.

KEYWORDS

Low Complexity Ethernet (LES), MAC-PHY, ADuCM4050, LWIP stack, SDP and Arduino interface, universal power supply, MODBUS, OPC – UA

ABSTRAKT

Předmětem diplomové práce je vytvoření vestavěného neboli embedded systému za účelem demonstračního zařízení s využití neveřejného modulu Low Complexity Ethernet pro průmyslové aplikace – tedy jednoduché průmyslového ethernetu. Nejprve se dokument zabývá technologickými bloky jakožto MCU a použitým LWIP, poté se práce zaobírá detailním popisem architektury univerzálního systému podle zadaných požadavků po stránce fyzické (hardwarové) a systémové (softwarové). Další kapitoly se zabývají oživením desky, včetně vysvětlení technických závad, dále implementací LWIP stacku a nadstavěným MODBUS protokolem. Hlavními stavebními bloky jsou zmíněný Low Complexity Ethernet modul a mikrokontroler ADuCM4050 a další důležité hardwarové periferie mikroprocesoru.

KLÍČOVÁ SLOVA

Low Complexity Ethernet (LES), MAC-PHY, ADuCM4050, LWIP stack, SDP and Arduino interface, univerzální napájecí zdroj, MODBUS, OPC – UA

ŠUSTEK, V. *Jednoduchý průmyslový ethernet*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav radioelektroniky, 2019. 63 s., 15 s. příloh. Diplomová práce. Vedoucí práce: doc. Ing. Tomáš Götthans, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svoji diplomovou práci na téma Jednoduchý průmyslový ethernet jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne ....

(podpis autora)

PODĚKOVÁNÍ

Děkuji školnímu vedoucímu diplomové práce doc. Ing. Tomáši Götthansovi, Ph.D. a externímu vedoucímu panu Ing. Michalovi Brychtovi včetně aplikačního týmu ADI, Limerick, zejména Connelu O'Sullivanovi za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

CONTENT

LIST OF PICTURES

LIST OF TABLES

1 INTRODUCTION

Over the last few years, interest in mutually connected computer-based devices has greatly increased. These devices are becoming more available because of prices have dropped and so can be used in various areas such as healthcare, industrial production, safety, transportation etc. Originally the army project named ARPANET emerged in highly developed Internet technology for different purposes with often hard requirements such as safety, high speed, data rate. This document aims into Low Complexity Ethernet module which collect data from essential communication protocol on the one side (SPI, I2C, UART etc.), encapsulates data to convenient ethernet frames and consequently sends for general processing on the other side. This chapter aims into the fundaments of the proposed system.

Recently, Analog Devices (ADI) designed the FPGA based low complexity prototype with two ETH - PHYs and I2C + SPI access interfaces. This block is proprietary, and its content is not point of this diploma thesis. This prototype is called as the LEN/LES (switchable Low complexity Ethernet Node/ Low complexity Ethernet Switch) can be understood yet not released product providing MAC service. In the further text is this block called as the MAC-PHY. This document aims only into to LES mode system implementation. Further implementation (up to application layer) is subject of this document. The main point of this assignment is to create demonstration application board with the MAC-PHY included (further called LEN/LES 2 board). Demonstration board should highlight potential of the project and it is pre – iteration for final intended SoC (System on Chip) implementation of the whole described system. The MAC-PHY requires host MCU for control – intended part of the final SoC. Final system should use one of the convenient industrial protocol such MODBUS, OPC - UA. Note that of the main corporal requirements is to use new released ARM MCU the ADuCM4050 (M-4F Cortex) with enhanced power management and convenient peripherals to prove its performance.

However, there are some constrains during Ethernet implementation in the miniaturized embedded system – available space memory and computing performance. Therefore, a LWIP stack is discussed as the first – special variant of a TCP/IP stack for embedded. The LWIP familiarization is followed by brief introduction to used ARM microcontrollers. Consequently, the fundaments of the system are described as was mentioned in the abstract. Finally, last chapters express hardware debug, LWIP implementation and MODBUS implementation.

2 PRELIMINARY

This chapter is dedicated for brief LWIP stack introduction such as reason of usage in embedded system or light view on the architecture. Secondly, the basic features of the ARM MCUs are discussed.

2.1 Light Weight TCP IP stack

The miniaturized computer-based devices such as sensors must be inexpensive and small, the implemented Internet Protocol must be optimized as well. For these purposes LWP TCP/IP stack by Adam Dunkles was developed (From the Swedish institute of computer science) [1]. The ordinary TCP/IP stack provides and IP, ICMP, UDP and a TCP protocol [3]. The application layer is highly hardware abstract, that entry point of the convenient stack is in file I/O fashion, especially because of fundamental elements implementation in the system kernel. The TCP/IP is designed as a layered system, whereas layers are strictly divided and only entry points are exposed for interfacing. Since the stand-alone layered architecture has advantages such as safety (no other layer can access resources of neighbor layer except interfacing with dedicated entry points), there is a disadvantage of performance and memory requirements. For layer division, it is needed to copy all used data buffers through the stack, therefore significant amount of the memory and computing time is occupied by system.

However, the typical embedded application has very limited memory resources, so compromise is needed especially in the buffering. The main goal of LWIP (light weight TCP/IP stack) is the sharing buffer through the layers, what is violation of strictly divided layers in the original intended stack. The memory sharing can be easily supervised because of native the C – language stack implementation. Except supported protocols (IP, ICMP, UDP, TCP) LWIP needs a few support modules such:

- Emulation layer – only implementation dependent part of stack, includes timers, process synchronization, message passing etc.
- Buffer and management subsystem – care about memory for processes
- Network Interface functions – the end low layer driver function
- Internet Checksum functions – packet checksum calculation
- Abstraction API – for global stack interfacing

In term of process model, LWIP is intended to operate as the single process, whereas user is interfacing stack API (Application Programming Interface) accessing this single process, or stack is divided into the two process in mutual collaboration comprised from the API and the TCP/IP process. In case of the divided API and TCP/IP, both process communication using an inter-process communication (IPC) using software semaphores, message passing and memory sharing.

The memory management uses chained structures with necessary variables and pointers to the payloads. For example, the packets are represented as a structure as well as the network driver structure, which represents the certain mapped network devices.

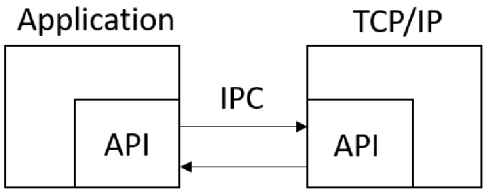


Figure 1 Division between TCP/IP and Application

The memory manager splits memory into the small chunks and places small structure with flag used/unused there. This allows to maintain memory and prevent fragmentation. Also, the manager maintains only dedicated part of the memory so cannot occupy room of other blocks

The packet buffers are example of the chained structure instances. The packet may reside in a *PBUF_RAM*, *PBUF_ROM* or might be stored in the fastest way to the pre-located static *PBUF_POOL* memory. The *PBUF_ROM* memory is suitable for constant data to be sent, *PBUF_RAM* data are used to send packet and *PBUF_POOL* for its fast access is used for incoming packets. Note that in the MCU is available only the RAM and the FLASH memory for protentional data, so packet of the ROM character is both read/write accessible. The *PBUF* structure contains following parts such pointer of the next *PBUF* instance pbuf, pointer of packet payload, length of packet, length chained packets, flags and reference bits. The data of payload pointer are stored in the though frame of pbuf as the last member of structure. In term of size is the biggest *PBUF_POOL*, because of incoming packet size (huge amount of received payload data). Figure 2 express chained packets of types PBUF_RAM and PBUF_ROM, whereas PBUF_ROM has external payload storage.

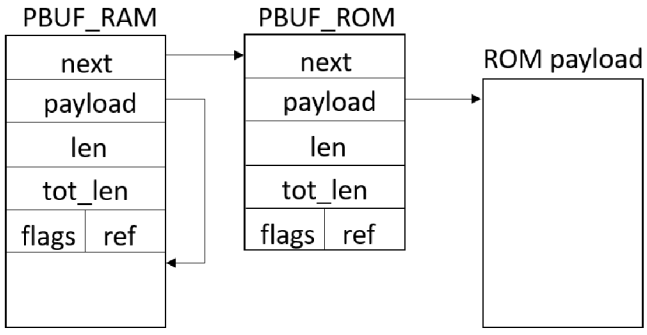


Figure 2 General PBUF_RAM packet chained with PBUF_ROM packet

Network interfaces are also handled using chained structure defined as *netif*. Structure contains pointer for chaining, name of interface, interface number, IP address, netmask, gateway field and state flag, but especially contain two function pointers for low layer packet handling during receiving and transmitting. IP and UDP protocols are presented as structures as well.

2.1.1 LWIP Stack footprint

The Adam Dunkels's proposal of the LWIP was tested only for the program RO (Read Only) memory usage (data, instruction). Also, reached results are old – fashioned in term of the implementation (compiled for x86 and 6502), note that reference document was issued in the year 2001. Dunkels's original stack takes 23kB of the flash memory and there is not specified application of usage.

In point of view from LWIP original proposal, the STM recently issued document of the LWIP stack for its ARM M-4 Cortex microcontrollers STM32F4xx (year 2013) [4]. STM compiled LWIP in result of 98kB used FLASH (RO data and RO program), and 33kB of the RAM usage in case of the HTTP 1.0. Note that the STM uses the hardware abstract functions and the used microcontroller is much more advanced in compare to 6502 used by Dunkels (STMF4xx family represents 32 – bit ARM M-4 Cortex MCU).

Table 1 Dunkels's LWIP footprint

Module	Size [B]
TCP	11461
Support functions	4149
API	3847
UDP	1264
IP	1211
ICMP	714
Total	**22646**

Table 2 STM32F4xx HTTP server demo footprint

Module	RO code FLASH [B]	RO data FLASH [B]	RW data SRAM [B]
Ethernet driver and interface	2828	0	9360
LWIP memory management and IP modules	18634	20	19978
Application Modules: general initialization	6988	52385	1581
STM Peripheral drivers and board support	3720	5	16
Others (stack, heap, ect.)	8456	4573	32
Total	**40626**	**57091**	**32770**

Regards to the footprint of the LWIP stack is mandatory to mention the µIP stack developed for the 8-bit microcontrollers with the footprint < 1kB of FLASH and <100B of the RAM, also by Adam Dunkels [2]. Constraints caused by footprints will be discussed in the MCU related chapter 4.

2.2 ARM microcontroller introduction

It may be said with exaggeration that the ARM based 32-bit MCUs rule the world. The Advanced RISC Machines are used not only in the miniaturized embedded solution, but also in the devices with high computing performance [5].

Although the RISC (Reduced Instruction Set Computing) is part of the name abbreviation, most of the used instruction is CISC (Complex Instruction Set Computing). ARM company does provide only the IP core for the silicon vendors which produce MCU with their customized peripherals.

Once of the main advantage is a core native debug interface standard JTAG/SWD, what is also only direct entry point to the core. Consequently, the core provides ethe entry high performance AXI and simpler APB bus for interfacing with other important block of MCU such as clock section, memories (SRAM, FLASH), etc. ARM is designed for ARM - 32b instructions, but also provides option of the THUMB2 16/32b instruction set.The ARM microcontrollers are marked as the Harvard architecture devices, because data and program can be accessed simultaneously. This device processes with virtualize memory, so even if FLASH and RAM is content from different HW memories, user accesses them in linear order.

ARM cores are divided into:

- ARM – A: High performance devices such as mobile phones, tablets and small computers. This type of core supports 7 modes of operation with its own processing necessary registers. Only this core uses full scale of ARM instructions. ARM instruction might be used as very powerful using Assembler fashion programming.
- ARM – R: Devices with the highest timing requirements. This core is used in application, whereas is needed timing accuracy (e.g. during ISR routines) such as very precise motor control.
- ARM – M: Brief description in following subchapter.

2.2.1 ARM M – family

The ARMv7M architecture of ARM-M devices was developed to offer better industrial leading-edge system performance, support natively C/C++ coding, deterministic instruction and interrupts timings [6]. M – family requirement was to allow producing enhanced low-cost miniaturized applications, whereas full performance of core may be needed. The ARM company market offers a few M-cores sorted in increasing order of performance and complexity.

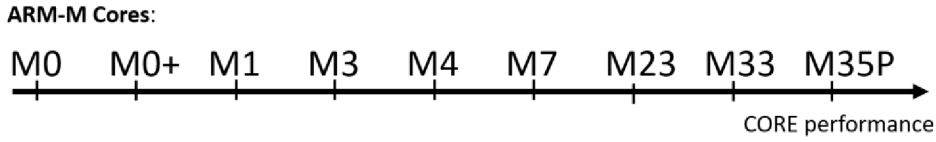


Figure 3 ARM-M Cores Overview

The devices support two (three) operational modes in frame of the Thumb State [5]. The number of modes is very limited in compare to "A" and "R" ARM devices. There is the privileged handler mode for exception (interrupt) processing and privileged/unprivileged thread mode for normal instruction execution. The unprivileged thread mode prevents from accessing sensitive parts of the MCU such memory or peripherals and uses its own stack pointer – ensures advanced reliability of the system. However, the unprivileged mode does not have to be used at all, especially in the simple applications.

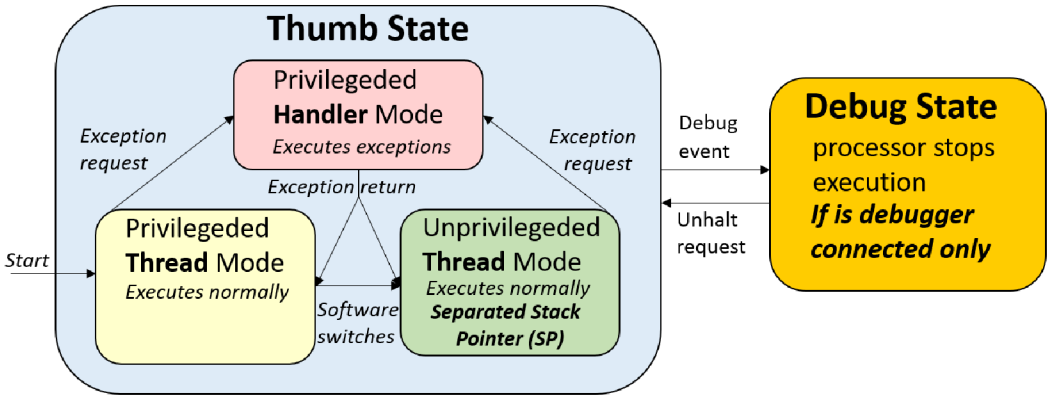


Figure 4 ARM-M operation states and modes

The register bank is composed from 16 registers of 32-bit width, whereas first 12 of them are for general purposes. The 13th register is Stack Pointer and can be MSP (Main SP) or PSP (Process SP) according to the current state of operation. Then, there is a register to hold return address, program counter. The ARMs have a few of status registers a xPSR (Application, Execution, Interrupt), what also ensures higher reliability of the system.

One of the most mentioned feature of the ARM high performance is the three-stage pipeline. The three-stage pipeline is implementation begins in M3 family and then in the better cores. The pipelining allows fetch-execute-store fashion of execution, what allows to process most of the instruction in one clock cycle such multiplication. Note that clock the frequency may exceed 100 MHz up to 200 MHz [5].

3 SYSTEM PROPOSAL

This chapter discusses important hardware blocks used in the LEN/LES 2 board. At the first is discussed power supply unit, secondly there is brief introduction to the proprietary MAC-PHY block. The third part expresses properties of the used MCU ADuCM4050 application. Consequently, there are subchapters about universal interfacing units – such as USB/UART interface and connectors. The detailed view on the embedded system is Figure 5. Schematic of the proposed system is enclosed as the appendix.

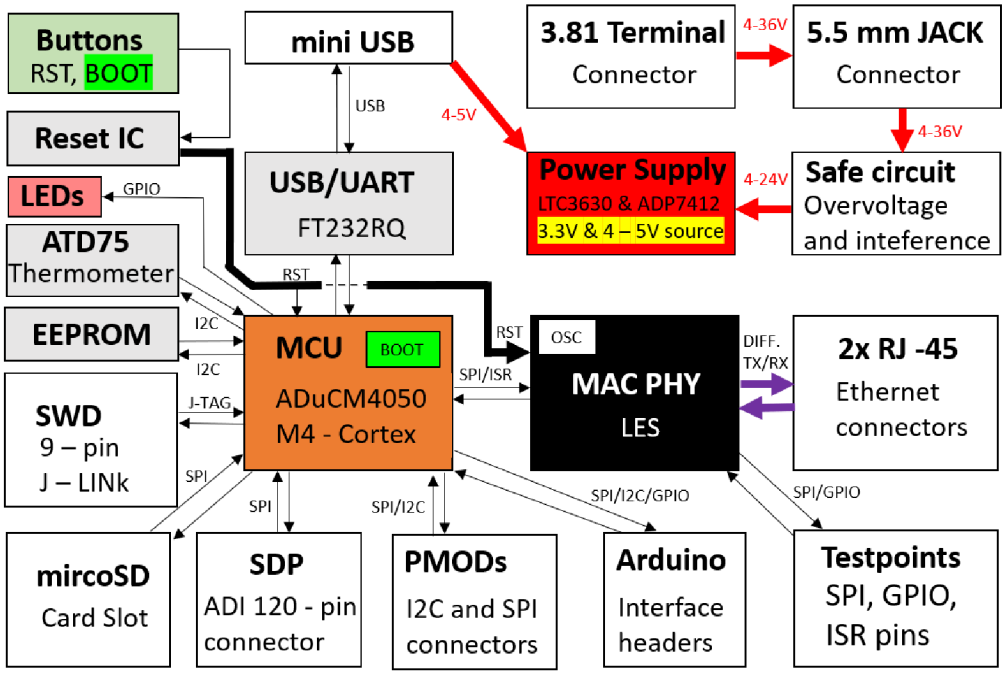


Figure 5 Hardware architecture overview

The Hardware architecture depicts blocks and its signal and external power connections. The white blocks represent passive mechanical connectors for board interfacing – note that there are only around the core. The core of the architecture is MAC-PHY block and MCU (brown and black). The light blue color marks ICs used for auxiliary functions such as reset circuit, USB to UART etc. The red block contains power supply circuitry and provides power supply 3.3V to the system. Note that whole system is powered by 3.3V except the USB/UART – internally powered by the LDO. There is 4-5V power source stabilized by the 5V LDO ADP7412, the usage is only for the SDP interface connector. The green block is composed from two buttons, first for reset circuit, the second for the BOOT pin of the MCU, note that signal connection between BOOT and MCU is not drawn – expressed only by distinct green color. For signalization there are a few LEDs dedicated (pink).

The simple thin arrows are used for SPI/I2C/GPIO/ISR connections. The thick black arrow leads between MCU and MAC-PHY block of external reset requirement. The uncomplete thick red arrow should outline power supply options.

3.1 Power Supply unit

3.1.1 Preliminary

One of the typical features of the miniaturized devices is range of power supply voltage and low power requirements. Because of the used MAC-PHY PFGA-based block (discussed in the next chapter), the low power requirements may be abandoned. During the pre-iteration of the Low Complexity Ethernet system was measured consumption of the whole system including MCU around 300 mA (Figure 5).

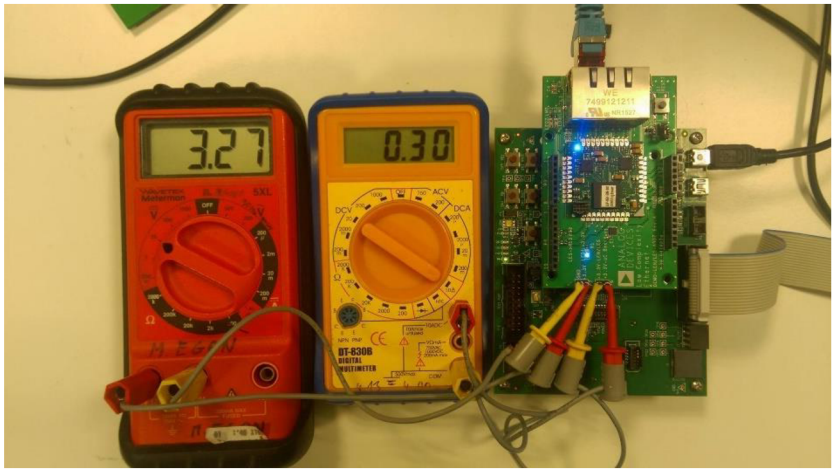


Figure 6 LES power consumption measurement

Auxiliary consumption measurement is depicted in figure 5. There is voltmeter on the left, ammeter in the middle and the very first LES device with the hosting ADuCM4050 board ADZS - ADuCM4050 EZKIT – the universal MCU evaluation board [7]. The EZKIT ensured up to 500mA current at 3.3V for hosted application. During the measurement, the LES was linked with the PC (processed packet on the ARP layer – traffic load). The current consumption reached approximately 300mA, what did not vary across adjusting traffic significantly. Note that the EZKIT was supplied using its USB interface by desktop PC - DELL OPTIPLEX.

However, ADI (Analog Devices International) stopped production of EZKIT and started to produce more practical and economic replacement EV-COG-AD4050 [14]. Replacement board though does not have enough power supply unit because of its purpose – evaluating microcontroller ADuCM4050 in lower power efficient applications. The result was that MAC-PHY block was kept in brown-out state so reliable system with ADuCM4050 and MAC-PHY module in collaboration is needed – the diploma thesis assignment.

3.1.2 LTC3630 down converter

The Analog Devices acquired Linear Technology (year 2017) and became company with wide offer of power supply converters or stabilizers. Logically, usage of corporal IC for power supply is required. Described circuit is part of the red block in figure 5.

Global power supply requirements of the system:

- 5V for SDP requirement and 3.3V for LES, MCU and related circuits
- USB – V_{BUS} 5V power supply source (down to 4.2V in worst case)
- Industrial 24VDC ±10V source, overvoltage and interference safe circuits
- Direct 3.3V voltage power supply source – exceptional, debug purposes
- Provide 500mA at 3.3V, voltage cannot decrease down to 3.20V
- Provide 5V for SDP purposes and 3.3V for MCU and LES (next subchapter)

LTC3630 was chosen as an appropriate down converter due to following properties [8]:

- Operating range 4 – 65VDC
- Adjustable 50 – 500mA output current
- All-in including switch, only external inductor working point passives needed
- Low – dropout in case of $V_{IN} \approx V_{OUT}$, switch R_{DSON} 1.9Ω at 100% duty cycle
- Low profile footprint 3.5mm x 5mm with thermal pad

There are a few evaluated typical applications in the official LTC3630 datasheet [8]. The best suits "*4V to 65V Input to 3.3V Output, 500mA Step-Down Converter*" from the first page. Because LTC3630 contains optional pins (unused in mentioned application), the original schematic was enhanced (Figure 6). Especially, there is a pin for output current settings, which is originally left floated ≈ maximal current.

The output current I_{OUT} (500mA) is set by switched I_{PEAK} which are related as:

$$I_{OUT} = I_{PEAK} \div 2 \rightarrow I_{PEAK} = 2 \times I_{OUT} \rightarrow I_{PEAK} = 1.0A \quad (3.1)$$

$$R_{I_{SET}} = I_{PEAK} \times 0.2 \times 10^6 = 1 \times 0.2 \times 10^6 = 200k\Omega \quad (3.2)$$

The equation 3.2 calculates value of 200kΩ resistor which is connected to the I_{SET} pin.

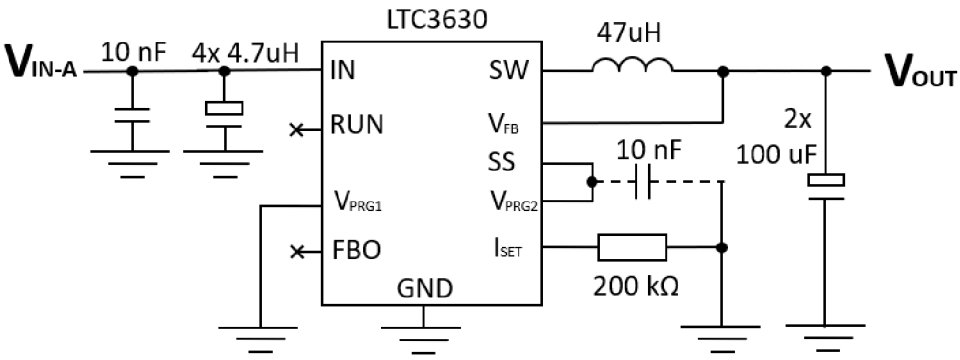


Figure 7 Schematic of the LTC3630 power supply

The value 47µH of the switching inductor at the terminal SW was obtained from the graph of figure 7. Value 47µH is on in the optimal center of the recommended inductance graph. As it is recommended in the datasheet, quality MSS – 1048 family inductor by Coilcraft vendor was chosen. Estimated switching frequency should be approx. 80kHz.

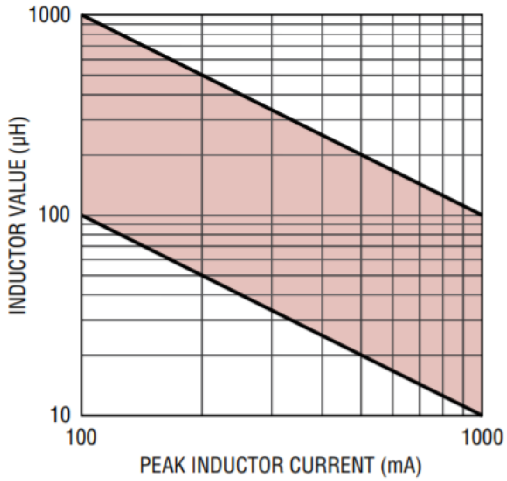


Figure 8 Recommended inductor value graph, assumed from [8]

Although there are formulas for the output and input capacitor calculation, vendor recommend increased values of mentioned discrete parts. However, by experienced colleagues in application department was recommended to use greater value of capacity in V_{OUT} pin represented by cascade of the recommended capacitors. The 4x 4.7µF and 10nF were placed in the input in order to bypass it appropriately. The output stage of the converter is bypassed by vendor recommended 2x 100µF capacitors. Note that type of chosen capacitor is X5R and X7R (material of the dielectric - stable capacity across voltage). There is not sketched connection of thermal PAD in the schematic – connected to GND.

Terminals RUN, and FB were left floating, because FB (Feedback) takes its place only in the application of LTC3630 cascade. The RUN pin is dedicated for voltage lockout, floating RUN sets input voltage lockout to internal 3.5V ÷ 3.7V of input voltage.

3.1.3 ADP7142 LDO stabilizer

The main source voltage 3.3V is provided by LTC3630 described upper. However, for ADI SDP purposes, also 5V voltage is required. Due to economy drive, just the LDO was used. There may be occur problem with providing 5V, if input voltage is lower than 5V – output voltage of the LDO is never higher than input. The ADP7412 is originally ADI LDO circuit and there is fixed 5V instance - ADP7142ACPZN5.0 – figure 9.

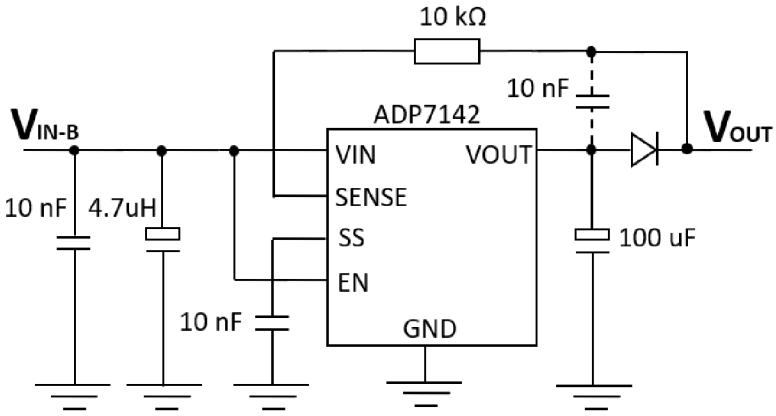


Figure 9 Schematic of the ADP7142 power supply

The input and output of the IC are bypassed by capacitors (recommend values are 2.2 µF in the input and output [9]). However, greater capacitors were placed at the terminals. The Pin SS controls soft start of the IC, whereas tied capacitor of the value 10 nF corresponds with calculated time:

$$SS_{TIME} = t_{STARTUP(0pF)} + (0.6 \times C_{SS}) \div I_{SS} \qquad (3.3)$$

Whereas the $t_{STARTUP(0pF)}$ is constant of value 380µs as well as the bias I_{SS} = 1.15µA

The calculate time is:

$$SS_{TIME} = 380 \times 10^{-6} + (0.6 \times 10^{-8}) \div 1.15 \times 10^{-6} = 5.5ms \qquad (3.4)$$

The feedback composed from the 10kΩ resistor and optional the 10nF capacitor causes stabilized 5V output voltage (SS adjusts loop gain if VIN is not equal to VOUT), whereas capacitor may prevent circuit oscillations. EN pin connected to VIN permanently enables LDO function. The diode at the output stage prevents reverse current flow – discussed in the subchapter of protection circuitry. Note that the input capacitors are quality X7R/X7S (stable capacity across work voltage range).

3.1.4 Power supply protection circuitry

Refers to the previous subchapter, a few protective discrete parts were used. The main requirement was to prevent overvoltage passed to the power supply terminal damage the circuitry. Generally, digital circuits need interference suppression, thus filters were implemented as well.

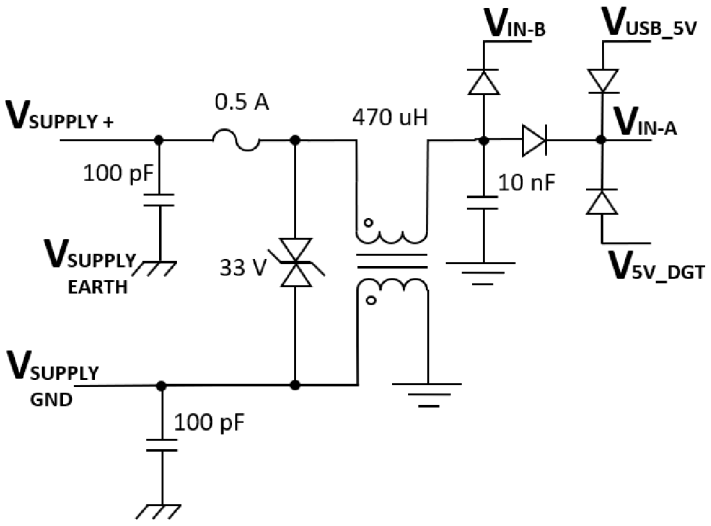


Figure 10 Schematic of power protection circuit

The figure 10 depicts the noise suppression and the overvoltage safe circuit. Note that the enclosed external power supply may be used as positive $V_{SUPPLY\ +}$ ground (negative) $V_{SUPPLY\ GND}$ and optional $V_{SUPPLY\ EARTH}$ for enhanced noise suppression. From the input point of view, the input current goes through the PTC – based resettable 0.5A poly-fuse [11], which increase its resistance in case of higher than nominal current, because of semiconductor current heating. Also, there are symmetrically tied 100 pF capacitors to the earth potential. The current through the fuse may be increased by 33V transil, which vice versa decrease its resistance in case of higher than nominal voltage [10].

Consequently, 470uH common mode choke suppresses symmetrical and unsymmetrical interference. The inductor works as the serial inductance in the first mode and suppresses usual high frequency signals. The second mode of operation is, when there is some interference noise in both input lines. The common mode choke produces magnetic flux in its the core, which causes signal subtraction in the magnetic domain – therefore also in signal domain. Choke output is bypassed by additional 10 nF capacitor.

Then, there is a few of power Schottky diodes to block reverse current flow in case of multiply connected power sources [12]. Basically, the diodes prevent to force voltages V_{USB_5V} and V_{5V_DGT} in case of used $V_{SUPPLY+}$ and vice versa in another 2 combinations (the three diodes connected to the V_{IN-A} node). The diode of the V_{IN-B} and diode of figure 9 may prevent reverse current in case of used SDP daughterboard with external power supply. The 100pF capacitors contains C0G dielectric (very stable) and the rest of capacitors are of X7R dielectric type.

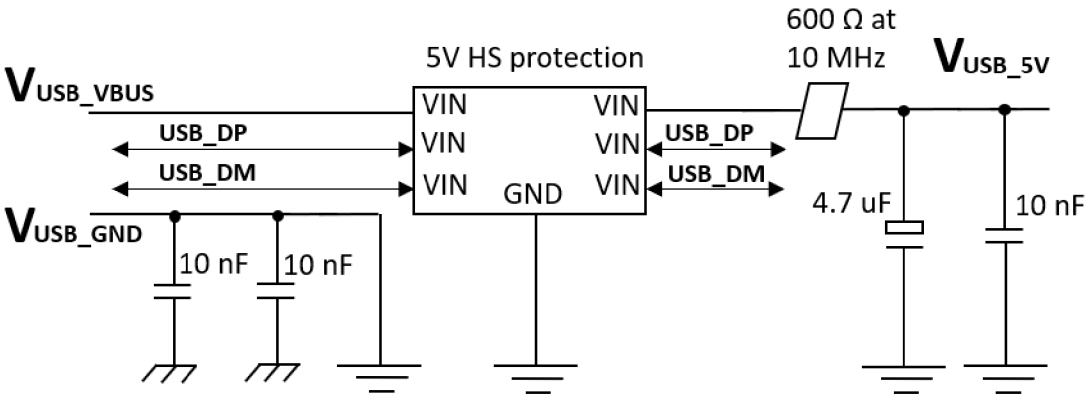


Figure 11 USB power and signal protection circuit

Figure 11 illustrates the USB overvoltage (might never take place) and signal interference protection. All the lines are protected against increased voltage by protective circuit IC [13]. Additionally, there is placed a ferrite bead with convenient 600Ω at 10MHz value. Power supply line is also bypassed by two capacitors connected to the V_{USB_5V} node. Also, as in the previous cases the chosen capacitor were used with the X7R dielectric.

The chapter 3.1 and proposed circuits are based on the results of the currently produced EV-COG-AD4050 [14] and its ancestor ADZS - ADuCM4050 EZKIT [7].

3.2 Proprietary Low Complexity Ethernet LES

Following chapter is intended to sketch necessary properties of the unreleased – secret prototype with MAC – PHY service implemented. As was mentioned previously in the power supply articles, the FPGA based device has significant consumption depicted in figure 6 (300 mA at normal network traffic).

To describe MAC-PHY as a black box is necessary to say, that in this application is interfaced by host processor using the SPI and a few optional GPIO pins (ISR). Another user part of interface – network interface is supporting convenient double socket "RJ – 45" with internal transformers. In the block are implemented the 10/100-BASE Ethernet domains. Currently is prototype continually developed and would result in the complex SoC chip.

Figure 12 Proprietary MAC-PHY prototype illustration

3.3 ADuCM4050 Cortex M4-F microcontroller

The ARM cores are sold and implemented by a lot of silicon vendors. Typically, certain vendor uses convenient core such as M3 or M4, adds its specific peripherals what results in microcontroller for specific usage – see chapter 2.2 and related sources.

The ADuCM4050 might be highlighted for its integrated power management – field of the application in low power sensors for medical, industrial, agricultural and other low power sensor-based applications [15]. The power management offers 3 sleep modes with various depth of system hibernation and support fast wake – up. The deepest sleep mode guarantees consumption around 40 nA and the MCU may be woken using 4 various interrupts. Typical consumption in active mode is 400 µA/MHz [16].

For the general IoT purposes is the ADuCM4050 equipped with up to 128 kB SRAM and 512 kB flash memory. The SRAM could be divided into 32 kB partitions and part of SRAM is optional 4 kB cache for efficient processing. The "F" letter in the M4-F designation expresses the floating-point unit with various support to floating computing. The rest of the features such peripherals might be marked as convenient.

However, in this application the power management will not be used due to significant LES consumption (see 3.1.1). The exact model of the used MCU is the ADUCM4050BCPZ-U2 in the 64 – LCSFP package (9x9x0.75mm).

Following hardware peripherals are used (corresponds to figure 5):

SPI (Serial peripheral Interface):

- SPI 0 – ADI SDP
- SPI 1 – SPI PMOD socket, microSD slot, SPI/I2C Arduino header
- SPI 2 (highest performance [16]) – MAC-PHY module, EEPROM

I2C (Inter Integrated Circuit – Two Wire Interface):

- I2C 0 – I2C PMOD socket, ADI SDP, SPI/I2C Arduino header, Thermometer ADT75

UART (Universal Asynchronous Receiver and Transmitter):

- UART 0 – via USB/UART connected to USB
- UART 1 – UART/GPIO Arduino header, ADI SDP

SPORT (Serial PORT – parallel SPI):

- SPT 0 – ADI SDP

WAKE External Interrupts:

- WAKE 0 – Arduino UART/GPIO header, PMOD SPI
- WAKE 1 – MAC-PHY module
- WAKE 2 – Arduino UART/GPIO header, SDP
- WAKE 3 – Arduino UART/GPIO header

ADC (Analog to Digital Converter) channels:

- ADC 0 – Arduino ADC header, SDP
- ADC 1 – Arduino ADC header, SDP
- ADC 2 – Arduino ADC header, SDP
- ADC 3 – Arduino ADC header, SDP
- ADC 4 – Arduino ADC header
- ADC 5 – Arduino ADC header, SDP
- ADC 6 – SDP
- ADC 7 – not used

The ADuCM4050 allows (LF) low frequency and (HF) high frequency external clock sources, both instances were implemented – LF using 32.7680 kHz and HF using 26 MHz XTAL.

Then, there are features such as GPIO, Timer, BOOT or various VCC and GND pins, which detailed view is not in range of the thesis – see attached schematic and datasheets [15] [16].

3.4 System connector interfaces

One of the main aspects for useful universal embedded system is synoptically sorted interfacing. For development purposes is important to raise enough number of available peripherals and feature - in the best case all of them. However, problems caused by extended wiring may occur (especially in ADC or high-speed digital communication).

Following connectors are required:

- SDP platform – 120-pin interface with GPIO, TIMER pins, UART, SPI, I2C, ADC, SPORT and WAKE peripheral
- Arduino interface – 4 female 2.54mm headers. In this document divided to Power header, ADC header, SPI/I2C header and UART/GPIO header.
- PMOD interface – two row 90° 2.54mm female header according to the Digilent specification, implemented SPI and I2C.
- microSD slot – SPI based slot for removable flash microSD card storage
- RJ – 45 (double) – network connectivity with the system (MAC-PHY)
- J – link (9 – pin) for ARM – debugging/flashing interface
- Power supply 5.5mm jack barrel – for powering using conventional connectors
- Socket 3.81mm terminal – for powering in industrial conditions
- microUSB – for UART interfacing, used USB/UART FT232RQ converter

3.4.1 SDP platform

The SDP platform was found by the ADI in order to reuse central elements in system demonstration [17]. Basically, the SDP demonstration system comprises from (figure 13):

- Motherboard – The FPGA/MCU based universal board, with 120 – pin connectors to interface up to 2 SDP daughterboard devices. Motherboard has USB connector for PC interfacing. Motherboard usually powers daughterboard. There are supported interfaces such an ADC, SPORT, UART, SPI, GPIO/WAKE various parallel ports and I2C (for daughterboard identifying).

- PC – convenient desktop or laptop, running application software

- Daughterboard – demonstrated module or system board. Daughterboard may sink power supply current from motherboard, but in high consumption cases may be used its external power source. Each daughterboard should have EEPROM with stored ID (accessed via I2C). Kind of protocol to access demonstrated circuit is optional.

This proposed system may be understood as motherboard instance. However, function is very limited in compare to dedicated universal motherboard devices. For more information see attached schematic – page 5 or source [17] and its related.

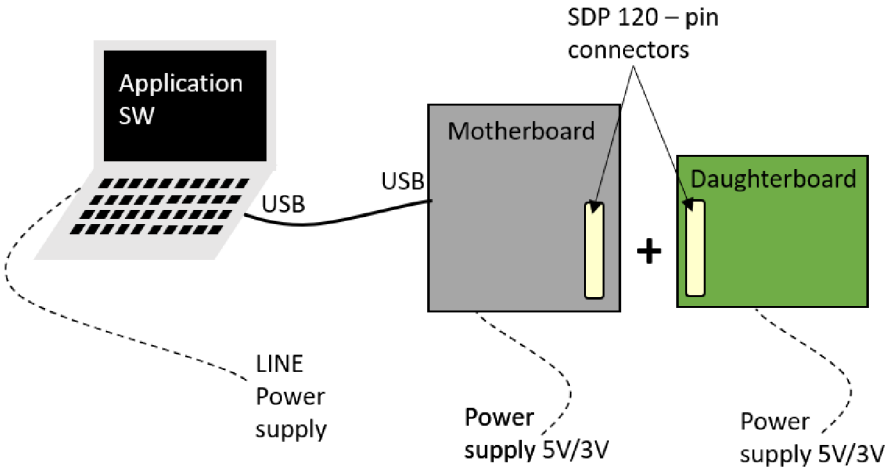


Figure 13 Block diagram of the SDP system

3.4.2 Arduino interface

The Arduino standard is one of the most popular standards for amateur technics and programmers. The Arduino is open-source platform based on easy to use software and hardware. From the embedded point of view, The Arduino board may be understood as universal development board with certain implemented MCU. The most famous is the UNO board with 8 – bit Atmel ATmega328 [18]. Although 8 – bit processor board might be mark as insufficient for high performance application, the UNO Arduino contains one important interface - special female 2.54m headers sorting known as the Arduino interface. Figure 14 illustrates sorting of 4 headers. There is power header, ADC header, SPI/I2C header and UART/GPIO header. Nowadays exist a lot of Arduino shields – daughterboard of Arduino, which are hosted by motherboard MCU board and became helpful for easy development in professional embedded field.

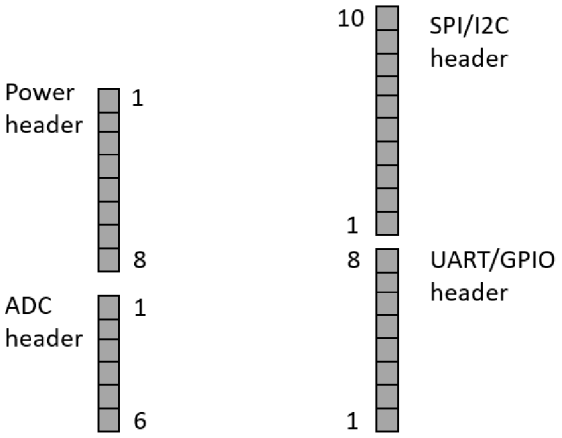


Figure 14 The Arduino interface header sorting

In the LEN/LES 2 board was attempted to follow Arduino interface pinout as much as possible. Following chart expresses matches and differences. Note that some requirements could not be follow (missing ATmega timer-related output, the ADuCM4050 provides complex timer outputs).

Table 3 Arduino interface pinout

	Original	LEN/LES2 demo
1	NC	NC
2	IOREF	IOREF
3	RESET	RESET
4	3.3V	3.3V
5	5V	5V
6	GND	GND
7	GND	GND
8	VIN	VIN

1	ADC0	ADC0
2	ADC1	ADC1
3	ADC2	ADC2
4	ADC3	ADC3
5	ADC4/I2C	ADC4
6	ADC5/I2C	ADC5

	Original	LEN/LES 2 demo
10	ADC5/IC2_SCL	GPIO/I2C_SCL
9	ADC4/IC2_SDA	GPIO/I2C_SDA
8	AREF	M_REF
7	GND	GND
6	GPIO/SPI_SCK	GPIO/SPI_SCK
5	GPIO/SPI_MISO	GPIO/SPI_MISO
4	GPIO/OC2A/SXPI_MOSI	GPIO/SXPI_MOSI
3	GPIO/OC1B/SPI_CS	GPIO/SPI_CS
2	GPIO/OC1A	GPIO/RTC1_SS2
1	GPIO/CLK0/ICP1	GPIO/TMR2_0

8	GPIO/AIN1	GPIO/WAKE0
7	GPIO/AIN0/OC0A	GPIO/TMR2_0
6	GPIO/XCK/T1/OC0B	GPIO/TMR1_0
5	GPIO/XCK/T0	GPIO
4	GPIO/WAKE1/OC2B	GPIO/WAKE3/TMR2_0
3	GPIO/WAKE0	GPIO/WAKE2
2	GPIO/UART_TX	GPIO/UART_TX
1	GPIO/UART_RX	GPIO/UART_TX

Table 3 represents pinout and deviation table of the Arduino Headers. Red color highlights deviation in function in compare to original Arduino. Originally, ATmega MCU were used and contains timers with various output functions such Output Compare etc. (OCXY).

ADuCM4050 does not support this fashion of timer output – there is a few universal timer outputs instead. The ADC header deviated in pins 5 and 6 where ADuCM4050 does not support multiplexing with I2C stage. In the UART/GPIO header are deviated pins 8 – 5. The 8 – 7 deviates, because ADuCM4050 does not support analog comparator on used pins and 6 – 7 because of external timer clocks (also not supported). The same problem occurred in the SPI/I2C header pins 4 – 1 and because ADuCM4050 does not multiplex ADC in the I2C stage (pins 10 – 9).

Anyway, important features such external interrupt or wake interrupt are followed. Also SPI, I2C, UART and ADC stages are connected to Arduino convenient headers.

3.4.3 PMOD interface

The PMOD is originally Digilent output standard for low frequency interfacing [19]. PMOD is intended for devices of 3V or 5V logic and currents appropriate to digital circuits (at least 2 mA). Logically, the current should not exceed low values of usual digital processing. Nowadays exist a lot of pluggable modules using PMOD interface

such as UART/USB converter, RTC module and various wireless modules. PMOD currently specifies 6 kind of interface and some interface has A and B extinguishing. In the LEN/LES 2 board are used I2C PMOD and SPI type 2 PMOD. Note that PMOD is usually 2.54mm two-row socket with 12 – 24 pins with 90° angle.

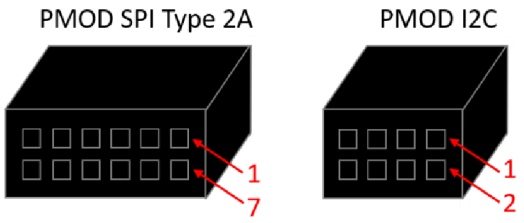


Figure 15 SPI Type 2A, I2C PMOD connectors

Table 4 PMOD SPI Type 2A, I2C pinout

PMOD SPI Type 2A

SS	MOSI	MISO	SCK	GND	VCC	INT	RESET	N/S	N/S	GND	VCC
1	2	3	4	5	6	7	8	9	10	11	12

PMOD I2C

SCL	SCL	SDA	SDA	GND	GND	VCC	VCC
1	2	3	4	5	6	7	8

As it is shown at Figure 15 and Table 4, there are more than data lines in case of SPI Type 2A. The INT pin is dedicated for interrupt purposes and is connected to WAKE0 – the highest priority interrupt of the ADuCM4050. The RESET pin may be output pin for slave reset. N/S (Not Specified) are connected to GPIO (see schematic page 5).

3.4.4 USB/UART converter interface

The ADuCM4050 does not support USB stage, so converter for universal PC interfacing is needed. The FTDI USB/UART converter FT232RQ is quite mostly used IC in embedded application [20]. The USB differential signal passed to the input of FT232 is processed and converted to UART protocol. Note that PC, which is sending USB signal requires installed and supported virtual COM ports.

Anyway, the FT232RQ supports conversion of another protocols such RS232 or RS485, thus more than UART RX and UART TX is required in term of the IC pinout. The mentioned chip is used only as USB/UART converter and the rest of the input pins is terminated using 10 kΩ. Using internal LDO is whole active chip powered by 3.3V, so UART is in the 3.3V LOGIC mode (ADuCM4050 supports only this mode). The outputs are left floating except N_RTS (Request to Send) – may be used for flow control with N_CTS cooperation. Note that UART RX and UART TX signals must swapped when connecting to the MCU UART stage.

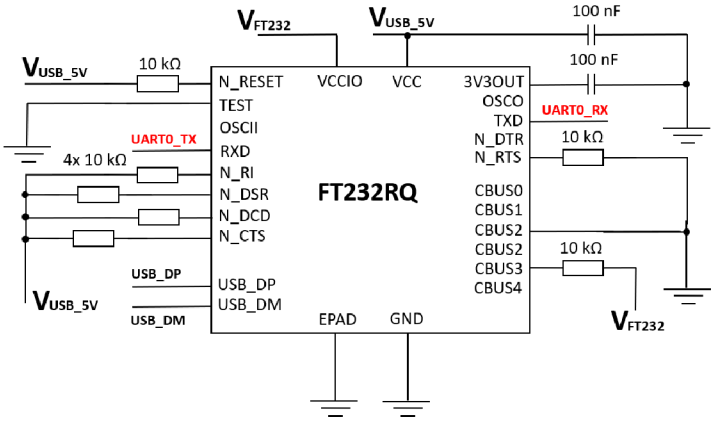


Figure 16 Schematic of USB/UART converter

3.4.5 Auxiliary circuitry

The scope of the diploma thesis is limited, so following chapter is only brief comment to the rest of used IC is the design.

Reset Circuit ADM6315-29D2ARTZR7 [21] supervises power voltage and generates reset signal, when voltage drops below its trip-point 2.93V to prevent brown out of the MAC-PHY module at the first (ADuCM4050 operates at much lower voltage level). The reset signal is then connected to the NAND gate, so MAC-PHY can be reset by MCU GPIO signal or by reset circuit (page 3 of schematic).

The ADT75 is 1° C precision thermometer with the I2C output and its address is set by external resistor [22]. Thermometer should be placed further from the MAC-PHY and MCU otherwise will measure its power dissipation.

To distinguish more LEN/LES 2 boards according to the MAC addresses, there is used EEPROM 25LC01A of the storage size 128kB with lock (read-only) capability [28].

It is planned to use a few LEDs for signalizing POWER, RESET, and 2 GPIO LEDs. For boot and reset options, the RESET and BOOT button is intended (pull upped to VCC).

4 HARDWARE PROPOSAL AND DEBUG

This chapter focuses on the hardware PCB parts placement and PCB layout. The placement of the parts had been done by writer of the thesis, however layout services such as routing and layering had been done by ADI layout specialist – Pat Sheahan (internal ADI regulation). The precise hardware layout is included in appendix (the end of the document). Note that board is 4 – layer.

4.1 Parts placement

Refer to chapter 4, there is needed a few restrictions during the parts placement. For example, the Arduino interface requires specific placement of all 4 connectors regards to coordinates. Also, the MAC-PHY block should be placed as close as possible to its Ethernet connector. Generally, useful components such interface connectors should be placed to correct places for comfortable usage and debug by user/developer. The result of placement discussion is following placement proposal of figure 17. The scale of the picture approx. corresponds with real intended dimensions (if page size is A4).

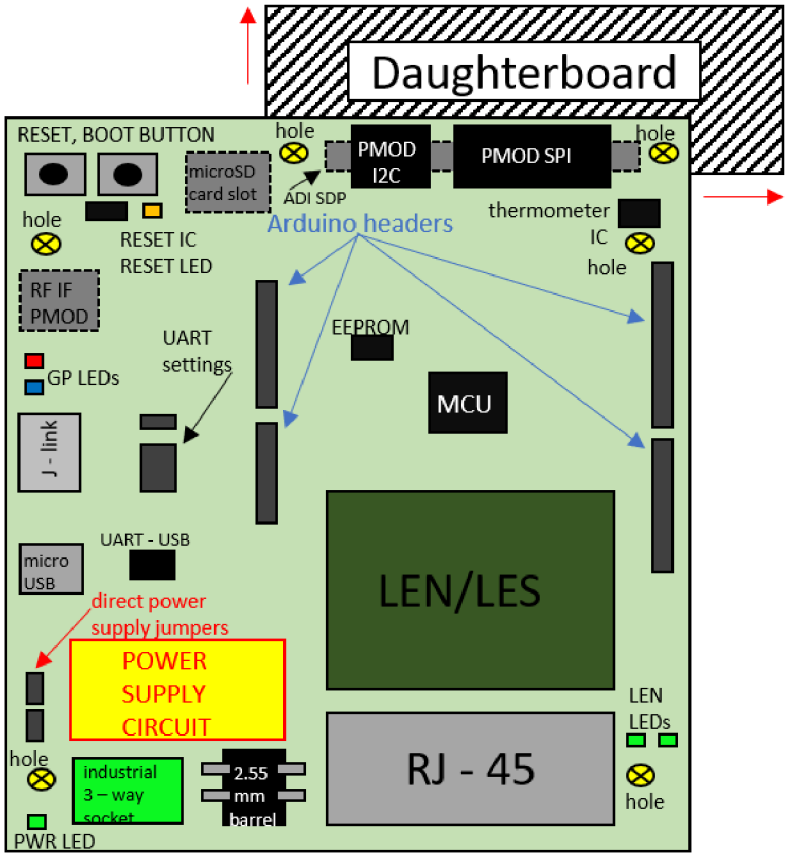


Figure 17 LEN/LES 2 board

4.1.1 MAC-PHY – RJ 45

The most important trace connection is between MAC-PHY and RJ-45 connector, so both parts are placed close to each other, and its interfacing differential signals +Tn, – Tn and +Rn, -Rn are routed with the same length using meander trace (n is port number: 1, 2). In case of not matched traces could occur errors during delayed signal of any ethernet line. Note that the MAC-PHY block requires 25MHz oscillator, which must be placed close as well.

4.1.2 Arduino interface

As it is explained in the chapter 3 (Arduino section), the correct sorting and placement for Arduino shield compatibility is needed [18]. The Arduino top orientation of figure 17 is portrait, so upper left connectors contains power signals. Also, between the Arduino interface and RJ-45 connector should be leave at least 2-3cm place, because the Arduino shields are often bigger than dimensions of the connector interface.

4.1.3 MCU, EEPROM

Regards to intended high speed SPI communication (up to 2.5Mbps) between MCU and the MAC-PHY port, the distance between them should be small and no sources of interference should be located around. On the other hand, the EEPROM shall be used only during startup to read MAC address bytes, so its placement has no special restrictions.

4.1.4 PMOD SPI, PMOD I2C, ADI SDP, RF connector

PMOD connectors usually lay on the PCB (axial connector orientation), so the best location in any PCB edge, that's why are both PMODs placed to the top edge of the board. Also, at least centimeter clearance between them may improve connector's accessibility. The ADI SDP motherboard connector shall be placed at the bottom side of the board (because daughterboard connector is on the top) and except holes for bolting has special requirement. To explain this requirement, see Figure 17 and the hatched daughterboard. The red arrow shows that daughterboard could be arbitrarily high and wide of the arrow directions, but the left bottom corner of the daughterboard (from the point of view Figure 17) must be aligned as depicted. This fact limits placing dimensional components and stand-offs around the SDP connector such. The RF connector is 8-pins two row connector of nRF24L01+ transmitter/receiver and shall be placed to the edge for compatibility.

4.1.5 Buttons

As the conventional buttons are by finger accessible almost everywhere, they should be placed to the corner of the LEN/LES 2 board. RESET button capability only resets whole system (drives reset circuit), but BOOT button except boot functionality of the MCU may be used as GPIO button of the microcontroller.

4.1.6 Slot microSD

The slot for the microSD card is usually placed at the bottom of the boards, so this board shall not be exception. Also, placing to the edge of the board improves accessibility

4.1.7 USB micro, J-LINK – pin

For device connectivity, it is necessary to place USB connector to the any edge of the board. J-LINK debug/flash port shall be placed to the edge as well for better connectivity. For flawless communication between USB and MCU the USB-UART FDTI converter shall be placed close to the USB connector. Note that J-LINK connector P10 is only way to debug/flash MCU (except UART0 in the boot mode [7][16]).

4.1.8 LEDs

There are four LEDs indicating LEN/LES 2 system. The orange LED placed in the up-left corner will be used when is the system in reset state (RESET button pressed or reset IC triggered undervoltage). The green LED is placed in the left-down corner shall light when the board is powered (3.3V power source supplies voltage). Two general purpose LEDs of red and blue color located on the left-upside are connected via pull-ups to 3.3V and MCU.

4.1.9 Power Source

Many times, placement of the switched power source is tricky task. However, in this design there is no sensitive device, as the ethernet PHY communicates on the much higher frequency than LTC3630 switches (estimated frequency according to the current and load is 80kHz [8]). For better accessibility, the source circuitry will be placed to the separate part of the (left-down corner) as well as its connector (DC barrel and three-way terminal).

4.1.10 Thermometer ADT75

The thermometer is placed as far as possible from the MAC-PHY (expected increased temperature around this block) to measure approximate average temperature of the board (left up).

4.1.11 The remaining components of the schematic

In the schematic (included in attachments in the end of this document) there are a few DNI connectors for debugging, especially of MAC-PHY block (e.g. SPI), as well as direct connector for 3.3V supply (in case of main power source failure) and most of the crucial traces is connected by 0Ω resistors (some of the also DNI) for disconnections or probe accessibility. Thus, resistor, which may be changed were not placed to the plastic parts such as header female connector (to do not damage plastic by heat). The crystals for MCU and protection diodes were placed to the related components.

4.2 Hardware getting started and hardware bugs

The LEN/LES 2 board had been manufactured by analog device PCB vendor, so no manual soldering or placement was needed. The real board photograph is following picture (figure 18). For a first time, board was powered using power source with current limit and no fatal shortcut was detected.

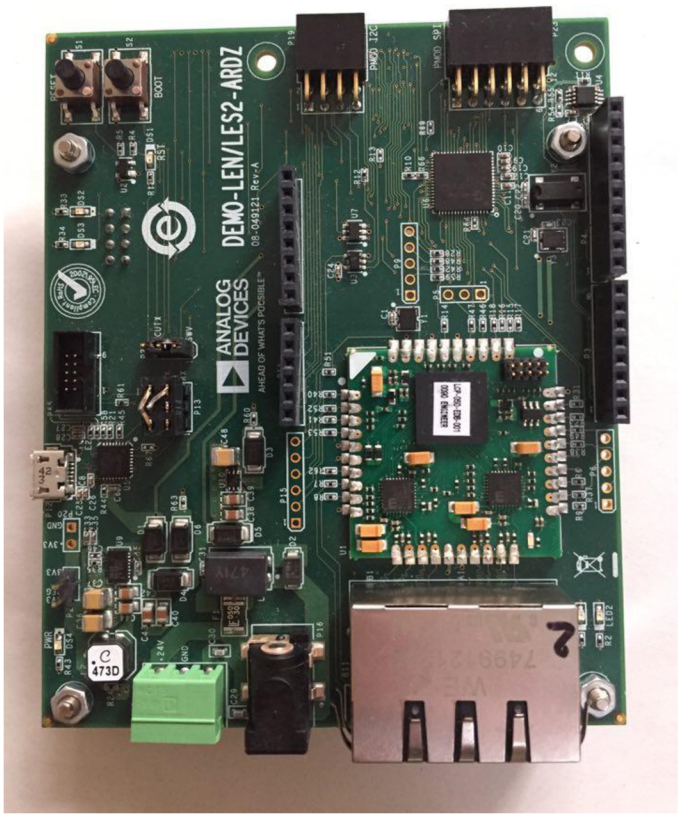


Figure 18 LEN/LES 2 board

However, the first bug was found as the green LED DS4 is placed in reverse mode. To debug this is needed only remove and rotate appropriately DS4.

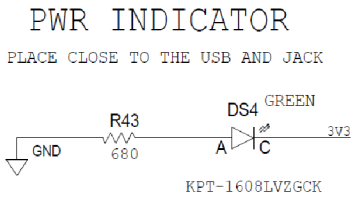


Figure 19 Wrong LED orientation

Originally intended capability to disconnect completely power supply is not possible during misconnection of 3V3 and 3V3_REG. Figure 20 depicts problem, whereas R24 must be removed and wire placed between 3V3 and right side of resistor (MAC-PHY power supply). This problem is not fatal, and board can work normally, but if P26 jumper is removed, MAC-PHY is still powered.

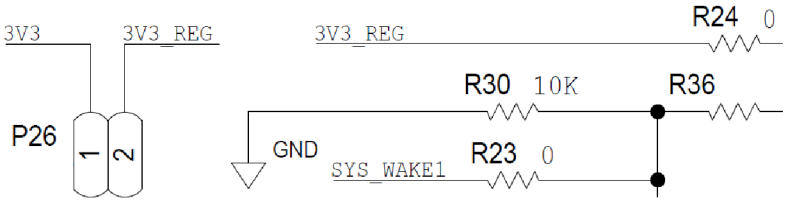


Figure 20 3V3 and 3V3 MAC-PHY misconnection

The third found bug is misunderstanding when choosing two-row male pin header P13. The jumper purpose was to set up to 4 different UART settings using two jumpers. All of parts used in the system are in ADI corporal library and this header was there as well. However, header in the library has pinout not usual to header, but pinout as integrated circuit. That is why is not possible to use 2 jumpers as originally intended. To fix this bug is needed to use wires terminated by female header or create female matching reduction. To understand this problem more, see schematic included in the appendix.

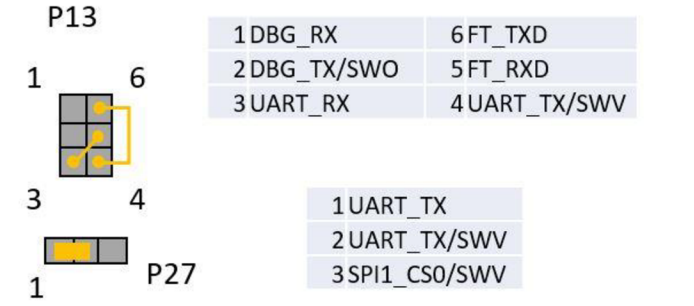


Figure 21 Wrong P13 pinout

Of course, the firmware development and creating external equipment was done gradually, however to approve system consumption in usual mode following figure 22 depicts measurement of current consumption during TCP bidirectional traffic (LWIP TCP-layer firmware is running and exchanges string messages with the PC).

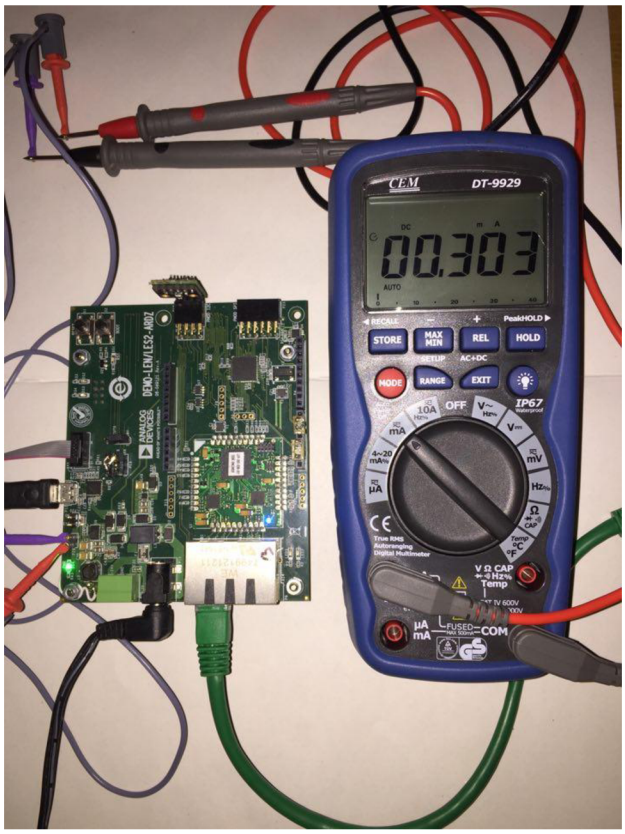


Figure 22 LEN/LES 2 board normal consumption

Figure 22 shows, that during normal operation (with LWIP stack and TCP bidirectional communication with 0.3s period, payload > 50 bytes) and whole system takes 303 mA at voltage 3.034V provided by on board power source. Note that the same system with powered off MAC-PHY takes 8.3 mA (estimated MCU consumption running on the frequency 26 MHz is ≈ 1.5 mA [15]).

Arduino interface, RF (nRF24L01+) interface, PMODs and SDP was tested successfully as well as buttons and LEDs. The most important part MAC-PHY also communicates with the MCU and basic firmware of static frame send/receive approved that system is ready to implement main assigned goals (for more see following chapter).

All of three currently known bugs do not prevent normal operational of the MCU and MAC-PHY (if developer does not fix them).

5 LWIP IMPLEMENTATION

There is a lot of customized LWIP implementation available under free license. As was mentioned in chapter 2, originally LWIP is based on Adam Dunkel's stack. In fact, the basic LWIP implementation to custom system may include only porting layer implementation and system settings definition file. This chapter may be helpful for someone who attempts implement LWIP for a first time (always is needed device such as MAC-PHY).

5.1 Getting started with MAC-PHY

To use the MAC-PHY with LWIP, necessary functionalities had to be tested and defined. To keep the proprietary driver of MAC-PHY secret, wrapper layer MAC_PHY has been implemented and lower than this layer is not published in this document. The first firmware functionality is UDP Syslog frame send and receive with string payload of the common first call "Hello World" with MAC address of device and frame number. Syslog is UDP based protocol intended for debug messages, so it's useful for test purposes [25].

For the test, two LEN/LES 2 were used. First board worked as the Syslog transmitter, meanwhile the second as the receiver. Transmitter worked on its own with periodically triggered frame sending with no logging and receiver's firmware sent all received valid frames to the UART console (logged on the PC via putty terminal). However, for illustration, following Figure 23 depicts caught frames by WireShark (transmitter connected to the PC), Figure 24 mentioned communication of two LEN/LES 2 boards.

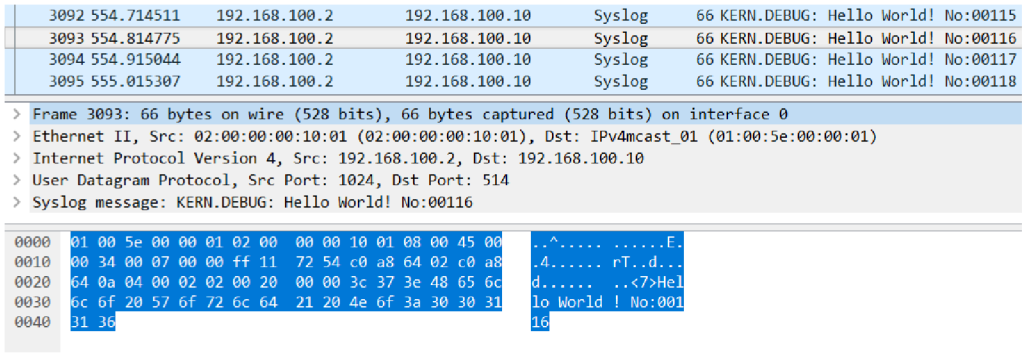


Figure 23 Wireshark Syslog frame receiving

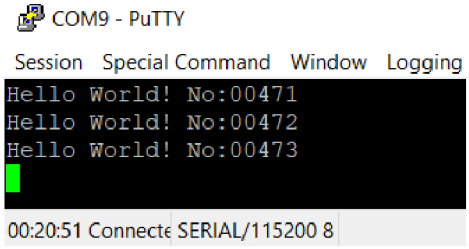


Figure 24 Frame Syslog payload received by LEN/LES 2 receiver

```
#define MAC_LNG    (uint32_t)(6)
#define FRAME_LNG  (uint32_t)(66)
#define PLD_OFFSET (uint32_t)(45)

//#define RX 1 /* When receiver should be flashed */
//#define TX 1 /* When transmitter should be flashed */

#ifdef TX

 uint8_t device_mac[MAC_LNG] = {0x2, 0x0, 0x0,a
                                0x0, 0x10, 0x1};

 uint8_t multicast_mac[MAC_LNG] = {0x1, 0x0, 0x5e,
                                   0x0, 0x0, 0x1};

 uint8_t udpframe[FRAME_LNG] = {

   0x01, 0x00, 0x5e, 0x00, 0x00, 0x01, /* Multicast address */
   0x02, 0x00, 0x00, 0x00, 0x10, 0x01, /* Device MAC */
   0x08, 0x00,           /* IP Ethertype (0x0800) */
   /* IP Header */
   0x45,                 /* IP V4 -- header length 20 bytes */
   0x00,                 /* differentiated services field   */
   0x00, 0x34,           /* total length (length of ASCII + 31)*/
   0x00, 0x07,           /* identification */
   0x00, 0x00,           /* flags/fragment offset */
   0xFF,                 /* time-to-live 255*/
   0x11,                 /* protocol -- UDP */
   0x00, 0x00,           /* header checksum  */
   192u, 168u, 100, 2u, /* Receiver's IP */
   192u, 168u, 100, 10u,/* Transmitter's IP */

   /* UDP Header */
   0x04, 0x00,             /* UDP source port 1024 */
   0x02, 0x02,             /* UDP destination port 514 Syslog */
   0x00, 0x20,             /* length (total 20, ASCII + 11) */
   0x00, 0x00,             /* UDP checksum (0 = don't check) */
   /* Syslog Message */
   0x3C, 0x37, 0x3e,       /* KERNAL DEBUG message */
   /* message contents (ASCII) */

   /* Prepared 21 space characters for further payload*/
   ' ' , ' ' , ' ' , ' ' , ' ' , ' ' , ' ' , ' ' ,
   ' ' , ' ' , ' ' , ' ' , ' ' , ' ' , ' ' , ' ' ,
   ' ' , ' ' , ' ' , ' ' , ' '
 };

```

```
#elif RX
uint8_t *p_frame = NULL;
uint32_t rx_lng = 0;
uint8_t device_mac[MAC_LNG] = {0x2, 0x0, 0x0,
                               0x0, 0x10, 0x2};
uint8_t multicast_mac[MAC_LNG] = {0x1, 0x0, 0x5e,
                                  0x0, 0x0, 0x1};

#endif
int main(int argc, char *argv[]) {

    uint16_t frame_num = 0;
    /* IDE generated pin multiplex */
    adi_initComponents();
    /*Initialize ADuCM4050 LEN/LES 2 board peripherals */
    platform_len_les2_init();
    /* Initialize MAC-PHY */
    MAC_PHY_start_up();
    /* Set device's MAC address */
    MAC_PHY_static_mac_add(device_mac);
#ifdef TX
    /* Calculate necessary IPv4 checksum */
    Ipv4_CalcOutGoingChecksum(udpframe);


    while(1){
        /* Send frame and wait 100 ms and increment frame
           number*/
        MAC_PHY_send_frame(udpframe, FRAME_LNG);
        sprintf((char*)udpframe + PLD_OFFSET,
                "Hello World! No:%05d", frame_num);
        frame_num++;
        tim_delay_ms(100);
    }
#elif RX
    MAC_PHY_static_mac_add(multicast_mac);

    while(1) {

        /* Poll for receive frame (non-blocking request)*/
        MAC_PHY_receive_frame();
        /* Get the frame */
        p_frame = MAC_PHY_get_buff(&rx_lng);
        /* If valid frame received, print its number, content
         * and clear receiving buffer */
        if(rx_lng != 0) {
            print_frame(p_frame + PLD_OFFSET,
                        FRAME_LNG, frame_num);
            frame_num++;
            MAC_PHY_clear_buff();
        }
    }
#endif

}
```

The code snipped mentioned here may be helpful for beginners implementing ethernet system with different MAC-PHY, especially pre-generated ***udpframe*** array with filled

multicast MAC address. Note that many MAC-PHYs does not have it included. Be careful, the IPv4 header checksum must be calculated.

Important is after initializing MAC-PHY fill the MAC address table with expected addresses (line 67, 83). For example, current MAC-PHY was not capable to receive multicast packet without adding multicast MAC (line 83).

5.2 LWIP porting

According to the [26] and [27] (getting started sections), only four files of whole LWIP stack may be edited to customize stack for specific MAC-PHY (Ethernet periphery). So, for a first time (of the LWIP implementation) is the best way to download free LWIP available at [26] and only override settings of foreign customization. The first is *lwipopts.h*, which contains definitions customizing default LWIP stack settings such service enabling (e.g. whether DHCP, UDP, TCP … are enabled). Then very important definitions are whether LWIP runs with or without RTOS (this LWIP runs without) and the settings of checksum calculation. MAC-PHY used in the LEN/LES 2 board does not calculate frame checksums on its own, so software calculations are needed (so defined to be set in *lwipopts.h*). Following code snippet of the *lwipopts.h* includes mentioned definitions of NO_RTOS – NO_SYS LWIP with software checksum calculations and disables RTOS LWIP related modules etc. Note that memory alignment for 32-bit MCU shall be 4.

```
1.  #ifndef __LWIPOPTS__H__
2.  #define __LWIPOPTS__H__
3.
4.  #ifdef __cplusplus
5.  extern "C" {
6.  #endif
7.
8.
9.
10.  #define WITH_RTOS                       0
11.  #define NO_SYS                          1
12.  #define CHECKSUM_BY_HARDWARE            0
13.  #define SYS_LIGHTWEIGHT_PROT            0
14.  #define MEM_ALIGNMENT                   4
15.  #define LWIP_ETHERNET                   1
16.  #define LWIP_DNS_SECURE                 7
17.  #define TCP_SND_QUEUELEN                9
18.  #define TCP_SNDLOWAT                    1071
19.  #define TCP_SNDQUEUELOWAT               5
20.  #define LWIP_NETCONN                    0
21.  #define LWIP_SOCKET                     0
22.  #define RECV_BUFSIZE_DEFAULT            2000000000
23.  #define LWIP_STATS                      0
24.  #define LWIP_CHECKSUM_CTRL_PER_NETIF    1
25.
26.  #endif
```

The second file and the most important is *ethernetif* module adapting representing layer between LWIP and custom MAC-PHY (ethernet periphery). The *ethernetif* will be discussed later. To be precise, required files to be change are also *cc.h* and *sys_arch.h*. The *cc.h* includes definitions of the data types (LWIP is not written using stdint.h library, so e.g. instead of uint8_t type is used u8_t etc.) The sys_arch.c includes mainly necessities for RTOS based LWIP, so both may be simply reused in this case.

5.2.1 Porting layer ethernetif

As was discussed upper, this module is the most necessary to be customized. This module interfaces input/output of used ethernet hardware and the module quality and reliability will be shared and inherited across whole LWIP. Also, it is recommended to attempt override any *ethernetif* then bare metal development.

In total, 7 functions are implemented in this layer including initialization, input, output and system time management (LWIP counts with 1ms timer). This means, that except input/output driver for customized MAC-PHY is needed to implement also reliable source of the 1 millisecond ticks. Initialization function is easiest and concerns only simple ethernet interface initialization, eventually storing static MAC address table. Then very important settings of the *netif* structure such an ARP setting, hardware address settings and set the flag signalizing link up [26][27]. The *netif* structure is passed through whole LWIP and includes also device IP address etc.

```
1. static void low_level_init(struct netif *netif) {
2.
3.      uint8_t mac_addr[6] = { 0 };
4.      /* Start MAC-PHY, read MAC address (from EEPROM)
5.         and store to the device*/
6.      MAC_PHY_start_up();
7.      MAC_PHY_read_MAC(mac_addr);
8.      MAC_PHY_static_mac_add(mac_addr);
9.
10.       netif->flags |= NETIF_FLAG_LINK_UP;
11. #if LWIP_ARP || LWIP_ETHERNET
12.      /* set MAC hardware address length */
13.       netif->hwaddr_len = ETH_HWADDR_LEN;
14.      /* set MAC hardware address */
15.      netif->hwaddr[0] = mac_addr[0];
16.      netif->hwaddr[1] = mac_addr[1];
17.      netif->hwaddr[2] = mac_addr[2];
18.      netif->hwaddr[3] = mac_addr[3];
19.      netif->hwaddr[4] = mac_addr[4];
20.      netif->hwaddr[5] = mac_addr[5];
21.      /* maximum transfer unit */
22.      netif->mtu = 1500;
23. #if LWIP_ARP
24.      netif->flags |= NETIF_FLAG_BROADCAST | NETIF_FLAG_ETHARP;
25. #else
26.      netif->flags |= NETIF_FLAG_BROADCAST;
27. #endif
28. #endif
29. }
```

The output function simple assembles packet content in pbuf chained structures (see chapter 3) and sends.

```
1. static err_t
2. low_level_output(struct netif *netif, struct pbuf *p) {
3.
4.      err_t errval = 0;
5.      struct pbuf *q;
6.      uint8_t buffer[1024];
7.      uint32_t bufferoffset = 0;
8.
9.      for (q = p; q != NULL; q = q->next) {
10.          memcpy(buffer + bufferoffset, q->payload, q->len);
11.          bufferoffset += q->len;
12.       }
13.       errval = MAC_PHY_send_frame(buffer, bufferoffset);
14.
15.       return errval;
16.   }
```

The input function could be tricky to debug because of frame receiving, if the valid frame is received must be divided into *pbuf* for further LWIP processing.

```
1. static struct pbuf * low_level_input(struct netif *netif) {
2.
3.      /* Buffering for LWIP*/
4.      struct pbuf *p = NULL;
5.      struct pbuf *q = NULL;
6.
7.      /* Length of packet */
8.      uint32_t len = 0;
9.      /* Remaining bytes for copy distinguish */
10.       uint32_t bufferoffset = 0;
11.       /* Buffer on the received characters */
12.       uint8_t *buffer;
13.
14.       /* get received frame */
15.       if (MAC_PHY_receive_frame() != MAC_PHY_OK)
16.           return NULL;
17.
18.       /* Get internally stored buffer and obtain the size of the
19.        * packet and put it into the "len" variable. */
20.       buffer = MAC_PHY_get_buff(&len);
21.       if (len > 0) {
22.           /* We allocate a pbuf chain of pbufs from
23.            the LWIP buffer pool */
24.           p = pbuf_alloc(PBUF_RAW, len, PBUF_POOL);
25.       }
26.       if (p != NULL) {
27.           bufferoffset = 0;
28.           for (q = p; q != NULL; q = q->next) {
29.               /* Copy data in pbuf */
30.               memcpy((uint8_t*) ((uint8_t*) q->payload),
31.                       (uint8_t*) ((uint8_t*) buffer +
32.                       bufferoffset), q->len);
33.               bufferoffset = bufferoffset + q->len;
```

```
34.           }
35.       }
36.       /* Clear internal buffer for further receiving*/
37.       MAC_PHY_clear_buff();
38.       return p;
39.
40.   }
41.
```

System tick source functions are implemented here, function included inside of them should return unsigned number of length 32-bit which is each millisecond incremented.

```
1. u32_t sys_jiffies(void) {
2.     return tim_get_1ms_tick();
3. }
4.
5. u32_t sys_now(void) {
6.     return tim_get_1ms_tick();
7. }
```

Following two functions had been copied and unchanged, in fact only wraps already written functions to pass them to the LwIP

```
1. void ethernetif_input(struct netif *netif) {
2.     err_t err;
3.     struct pbuf *p;
4.
5.     /* move received packet into a new pbuf */
6.     p = low_level_input(netif);
7.
8.     /* no packet could be read, silently ignore this */
9.     if (p == NULL)
10.          return;
11.
12.      /* entry point to the LWIP stack */
13.      err = netif->input(p, netif);
14.
15.      if (err != ERR_OK) {
16.          LWIP_DEBUGF(NETIF_DEBUG, ("ethernetif_
17.             input: IP input error\n"));
18.          pbuf_free(p);
19.          p = NULL;
20.      }
21.  }
22.
23.  err_t ethernetif_init(struct netif *netif) {
24.      LWIP_ASSERT("netif != NULL", (netif != NULL));
25.
26.  #if LWIP_NETIF_HOSTNAME
27.      /* Initialize interface hostname */
28.      netif->hostname = "LWIP";
29.  #endif /* LWIP_NETIF_HOSTNAME */
30.
31.      netif->name[0] = IFNAME0;
32.      netif->name[1] = IFNAME1;
33.
```

```
34.  #if LWIP_IPV4
35.  #if LWIP_ARP || LWIP_ETHERNET
36.  #if LWIP_ARP
37.      netif->output = etharp_output;
38.  #else
39.      /* The user should write ist own code in
40.       low_level_output_arp_off function */
41.      netif->output = low_level_output_arp_off;
42.  #endif /* LWIP_ARP */
43.  #endif /* LWIP_ARP || LWIP_ETHERNET */
44.  #endif /* LWIP_IPV4 */
45.
46.  #if LWIP_IPV6
47.      netif->output_ip6 = ethip6_output;
48.  #endif /* LWIP_IPV6 */
49.
50.      netif->linkoutput = low_level_output;
51.
52.      /* initialize the hardware */
53.      low_level_init(netif);
54.
55.      return ERR_OK;
56.  }
```

Mentioned code allows almost immediately run LwIP. However for the comfortable LwIP initialization was created function, whereas is called ***lwip_init()***, ***netif_add()***, checked if the link is up calling ***netif_is_link_up()*** to prompt ***netif_set_up().*** Function ***netif_add()*** requires static *netif* structure as the first argument and following three arguments are IP address of this device, netmask and gateway, last two arguments shall be addresses of the structure-functions ***ethernetif_init*** and ***ethernetif_input*** (previously discussed) This is fair enough to set up LwIP. To update LwIP is necessary to call periodically functions ***ethernetif_input()***, ***sys_check_timeouts()***. Both of last-mentioned function ensures processing of the received frames may be called as update functions.

Calling sequence of the initialization routines and calling periodically two update functions (for example in the while loo) is LwIP ready to process ping request from the PC. The address to be processed by ping command is IP address passed to ***netif_add().***

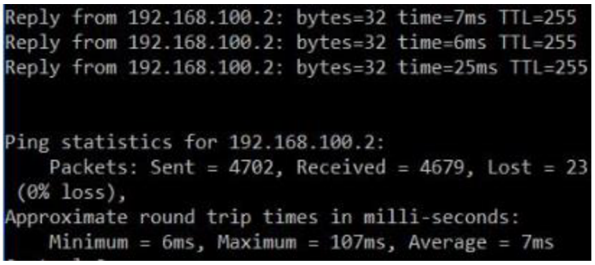


Figure 25 Calling ping command (no other task processed)

Note that described system to this point works as system with statically assigned IP address (host device e.g. PC must be located on the same network 192.168.100.x).

5.2.2 DHCP and LWIP

To use the LWIP device as the convenient ethernet device, must be capable to negotiate for IP address using DHCP protocol. Basic requirement is already fulfilled - UDP enable, because DHCP uses UDP layer. Expected behavior of the device is that will ask for IP address, so from the point of view may be marked as DHCP client.

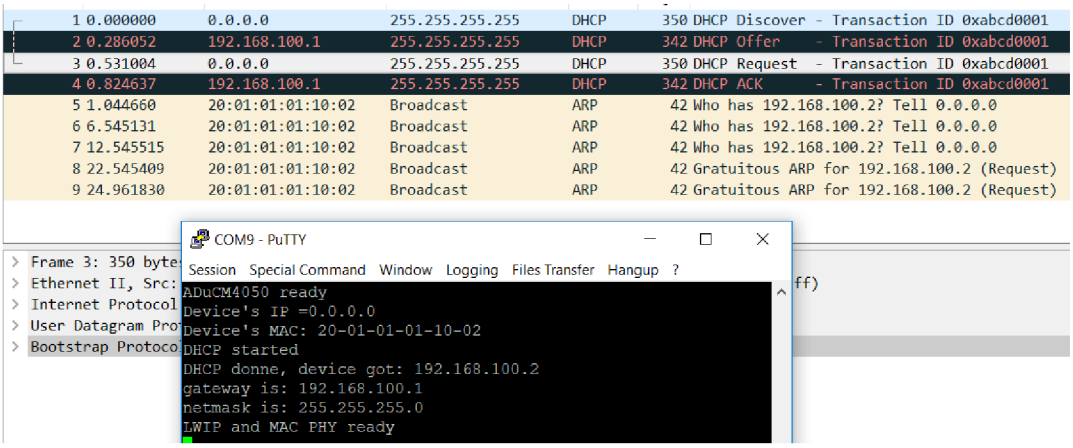


Figure 26 DHCP LWIP client

To enable DHCP is necessary to use after functions during LWIP startup (6.2.1) function ***dhcp_start()*** and wait until communication is between DHCP client and host is done (Figure 26). Note that all addresses of the *netif* during ***netif_add()*** were set to zero (except MAC hwadrr)**.**

```
1.    printf("DHCP started\r\n");
2.    dhcp_start(&gnetif);
3.    uint32_t mscnt = 0;
4.    LWIP_initialized = true;
5.
6.    while (gnetif.ip_addr.addr==0) {
7.        LWIP_CheckRecvIncTime();
8.        tim_delay_ms(DHCP_FINE_TIMER_MSECS);
9.        dhcp_fine_tmr();
10.         mscnt += DHCP_FINE_TIMER_MSECS;
11.         if (mscnt >= DHCP_COARSE_TIMER_SECS*1000) {
12.                 dhcp_coarse_tmr();
13.                 mscnt = 0;
14.         }
15.     }
```

This code snippet except starting DHCP and waiting for address assign also includes function ***LWIP_CheckRecvIncTime()***, which encapsulates ***ethernetif_input()***, ***sys_check_timeouts()*** – necessary to receive anything from the DHCP server. Also, definition: LWIP_DHCP, LWIP_AUTOIP, LWIP_DHCP_AUTOIP_COOP, LWIP_IPV4 to 1 must be added into to *lwipopts.h*. The DHCP server must negotiate with the device (e.g. default windows DHCP server).

6 DEMONSTRATION SYSTEM HARDWARE

To highlight system potential and usage, appropriate and simple demonstration must be assembled. Unfortunately, limited resources does not allow complex and practical-useful system and is not purpose of this thesis.

To figure competence of the system, very simple air conditioning was proposed. Air conditioning is represented by the created Arduino shield and contains brushless fan and circuitry necessary to drive fan using PWM. Then, the fan blows ambient air on the board, especially on the on-board ADT75 thermometer and airstream may be adjusted. Also, resistor load may be placed close to the ADT75 in purpose to heat the air around and simulate temperature increasing. Let us say, that this could be basic regulated system.

6.1 Fan-based air conditioning

At the first was intended to use low power 5VDC PC fan driven directly by LEN/LES 2 MCU PWM stage. However, unavailability of the PWM driven fan resulted in cheap and convenient DC brushless fan with tentative controller – adjustable voltage source driven by PWM. The vendor Sunon guarantees [29], that fan works from 3.5V to 6V with proportional revolutions of the fan. The problem is, that the fan contains inert circuitry to drive brushless topology and operates within specific range, this means that below 3.5V fan does not operate and so on fluent revolution control cannot be reach (at 3.5V fan operates with specific revolutions). Power device like this is not good by PWM, so voltage control is needed.

To control voltage is suitable to use well-known LDO LM317 [31]. Because of its dropout voltage (always higher than 1.25V), 12V supply is needed. This device adjust voltage with basic simple formula:

$$V_{LDO} = V_{ADJ} + 1.25V \tag{6.1}$$

So, only the PWM – Voltage converter is needed. Suitable way is to filter PWM waveforms using the LOW-PASS filter (such simple RC) and then correct this voltage using some operational amplifier (also needed to do not load RC). To do correction means, that maximal PWM filtered output voltage cannot exceed 3.3 V (MCU powered by 3.3 V) and this would not reach 6 V according to the formula 6.1. So, approximately 5V must be produced from the initial 3.3V (needed amplify approximately 1.5x). The operational amplifier may be single-ended and for these purposes is appropriate LM358 [30] in non-inverting mode. Repeated note, that VCC voltage is terminated as 12 VDC.

Using basic OA knowledge, gain in non-inverting mode is calculated as:

$$G = 1 + \frac{R_2}{R_1} = \frac{2350}{4700} = 1.5 \tag{6.2}$$

The RC filter frequency is calculated as 169Hz using 9.4 kΩ resistor and100 nF capacitor. Filter frequency may be dependent on the carrier frequency of PWM modulation. In this case, timer running PWM has base frequency 26kHz.

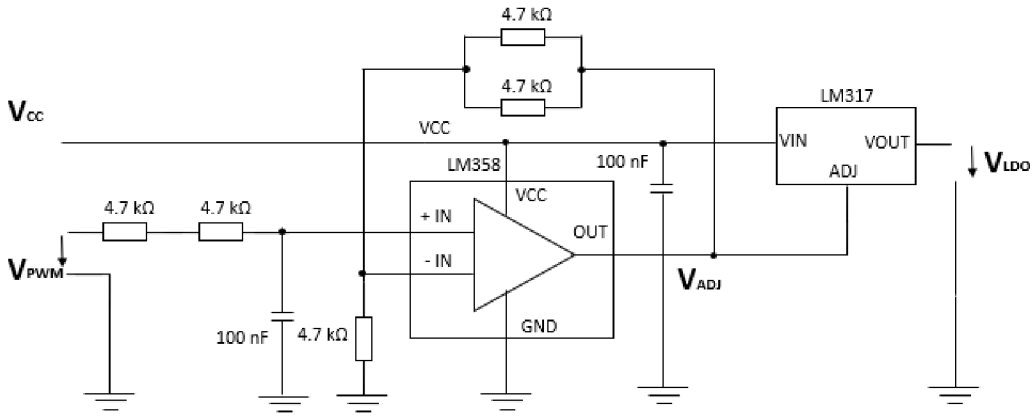


Figure 27 Fan control circuit

The circuit of Figure 27 had been manufactured using universal PCB to fit into the Arduino header. Following relation between PWM duty cycle and V_{ADJ} and V_{LDO} has been measured (Figure 28). Also, additional protection diode may be placed at the output and V_{CC} could by bypassed using high capacitance (LEN/LES 2 board already has).

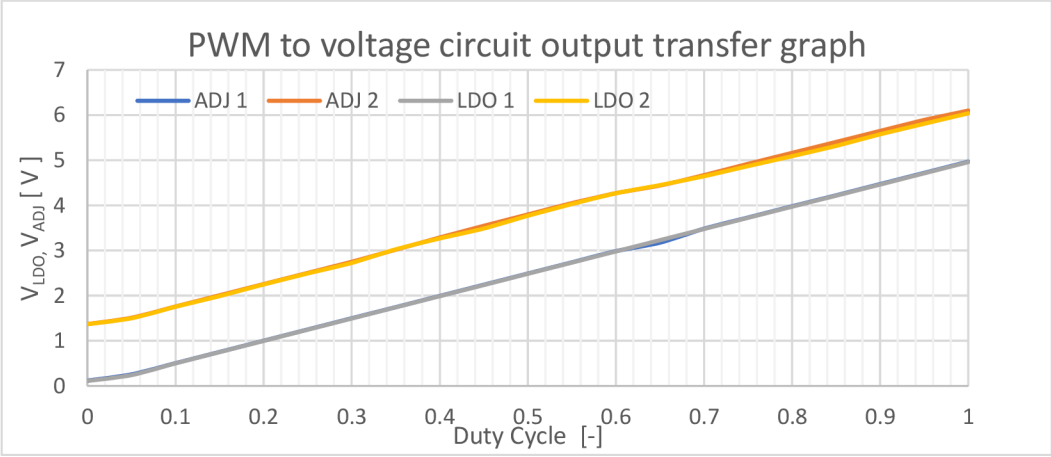


Figure 28 Fan circuit voltage

Note that Figure 28 depicts voltage with loaded circuits by the mentioned fan (205 mA at 5V). Two shields with described circuitry were created for further usage.

7 TCP/IP MODBUS APPLICATION

Regards to complexity, as the first approach to Low Complexity Ethernet demonstration system were chosen TCP/Modbus. Intended system uses PC laptop as MODBUS master (client) and two LEN/LES 2 boards as MODBUS slaves (masters).

7.1 MODBUS introduction

The MODBUS protocol is widely accepted and known industrial protocol based in the late 70's [23]. The MODBUS communication hardware interface may be serial RS-232, RS-485, various optical communication interfaces even wireless. For this diploma thesis is most suitable Ethernet TCP/IP based - the TCP/Modbus.

Basically, the MODBUS is only the application layer sorting transmitted data to the acceptable form. It is necessary to mention, that protocol is of the master-slave hierarchy. This means, that only master may start request and slave must reply. Slave is intended to listen and reply, slave never starts any transaction [23].

The MODBUS protocol master request consists of the slave device addressing value, function number, length of the message, data to be read/written, and CRC and descriptors of the data amount (length of bytes). However, for TCP/IP purposes may the CRC abandoned due to TCP/IP packet checksum. The TCP/IP is acknowledge-based protocol, so request may be re-transmitted until reply is received. Also, there is possibility to use no-acknowledge-based UDP protocol. TCP/Modbus always occupies port 502.

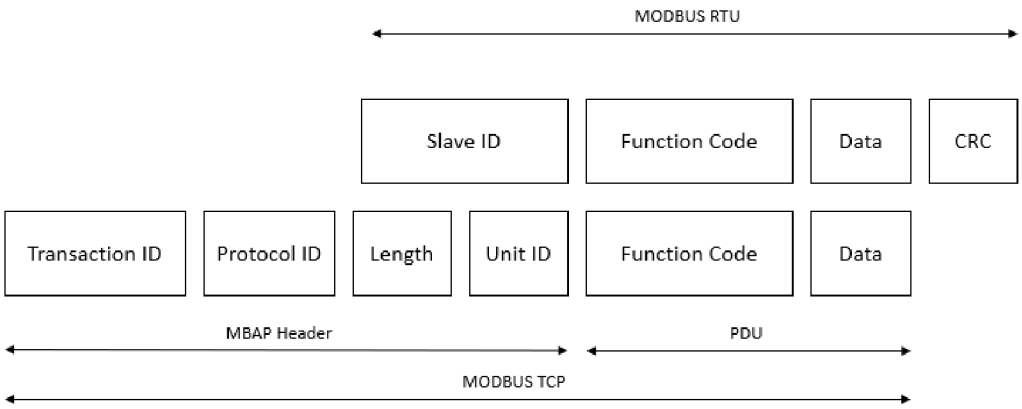


Figure 29 Modbus RTU vs TCP/MODBUS

Figure 29 depicts comparison between RTU frame and MODBUS TCP frame. Basically, there is depicted, what TPC/IP packet must contain to be as the TCP/MODBUS frame. The transaction ID is 16-bit number generated be slave to distinguish packet order (in case that some replies from the slave were delayed and received after later requests), slave just replies the same transaction ID. Protocol ID for Modbus TCP is 0x0000.

Length describes in 16-bit number the length of further data payload. The Unit ID contains only 8-bit address number of the slave device. The Function code and Data are responsible for writing/overriding data to processed.

Function code	Description		Type	Access
0x01	Read DO	Read coil state	Bool	Read
0x02	Read DI	Read input state	Bool	Read
0x03	Read AO	Read holding register	16-bit	Read
0x04	Read AI	Read input register	16-bit	Read
0x05	Write single DO	Write coil	Bool	Write
0x06	Write single AO	Write holding register	16-bit	Write
0x0F	Write multiple DO	Write multiple coils	Bool	Write
0x10	Write multiple AO	Write multiple register	16-bit	Write

Figure 30 TCP/Modbus commands

The figure 30 depicts possible commands included in the function code. It is obvious that coil or holding register may be read/overridden (e.g. coil can be used as switch, holding register as PWM duty cycle value). On the other hand, input or input register is read-only, thus is suitable for measurement values (e.g. temperature).

Byte	Request	Byte	Reply
0xAE	Transaction identifier MSB	0xAE	Transaction identifier MSB
0x3C	Transaction identifier LSB	0x3C	Transaction identifier LSB
0x00	Protocol identifier MSB	0x00	Protocol identifier LSB
0x00	Protocol identifier MSB	0x00	Protocol identifier LSB
0x00	Length of message MSB	0x00	Length of message MSB
0x06	Length of message LSB	0x04	Length of message LSB
0x01	Slave device address	0x01	Slave device address
0x02	Function code	0x02	Function code
0x00	Address of register to be overridden MSB	0x01	Number of following bytes
0x00	Address of register to be overridden LSB	0x01	Value of the coil
0x00	Addressing the specific coil MSB		
0x01	Addressing the specific coil LSB		

Figure 31 TCP/Modbus example transaction read coil (ON-state)

Figure 31 depicts typical communication cycle, there is master request on the left and slave's reply on the right. In addition, the MODBUS has capability to reply error message (invalid register or coil to read/write etc.), what is always only 9-bytes. The first 6-bytes are obtained in the same manure as normally (Figure 31), but the length of payload is only 3-bytes. In case of error code message, the payload contents address of the slave, the value of the function request sent by master with added most significant bit (read coil 0x01 => 0x81) and error code 8-bit value to distinguish reason of the error message. For more information about MODBUS and TCP/MODBUS see [23] and its recommended documents.

7.2 TCP/Modbus demonstration system

To demonstrate capability of the diploma thesis hardware (LEN/LES 2 board), firmware (LwIP) and software (TCP/Modbus master - described later) was invented simple system with following peripherals (also mentioned in chapter 7):

- Button (input DI)
- LED Diode DS3 (coil DO)
- PWM controlled on-board fan (holding register DO)
- On-board temperature measurement (input register AI)
- External temperature measurement (input register AI)

Following subchapters briefly describes master (client) and slave (server) implementation.

7.3 TCP/Modbus master (client) implementation

In the MODBUS terminology, the master is the client who can ask whenever wants its server, who is slave in the point of view. Refers to the intro description of the chapter 8, the PC laptop is intended to be master (client) [23].

For demonstration purposes is enough to use public Python-language based library pyModbusTCP [33].

This library is socket-based [34], and maintains TCP/IP MODBUS session:

- open/close
- acknowledge
- read
- write

Following code snipped expresses all used TCP/Modbus functions in the master implementation. Note that is also available multiple coils override and multiple holding registers override, but for the purposes of the 8.2 are not needed.

```
1. from pyModbusTCP.client import ModbusClient
2.
3. """ Create connection """
4. TCPModbus_Session.host("192.168.100.2")
5. TCPModbus_Session.port(502)
6. TCPModbus_Session.open()
7.
8.
9. """ Read operations"""
10.  button = TCPModbus_Session.read_discrete_inputs(0,1)
11.  LED_DS3 = TCPModbus_Session.read_coils(0,1)
12.  PWM_DutyCycle = TCPModbus_Session.read_holding_registers(0,1)
13.  Ambient_Tempt= TCPModbus_Session.read_input_registers(0,1)
14.
15.  """ Write operations"""
16.  TCPModbus_Session.write_single_coil(0,1)
17.  TCPModbus_Session.write_single_register(0,1000)
18.
19.  """ Close connection """
20.  TCPModbus_Session.close()
```

The master was created as the GUI python application using pyModbusTCP and basic python GUI tkinter [35]. The final appearance of the application is on the following figure. The button calls pyModbusTCP functions using button callbacks.

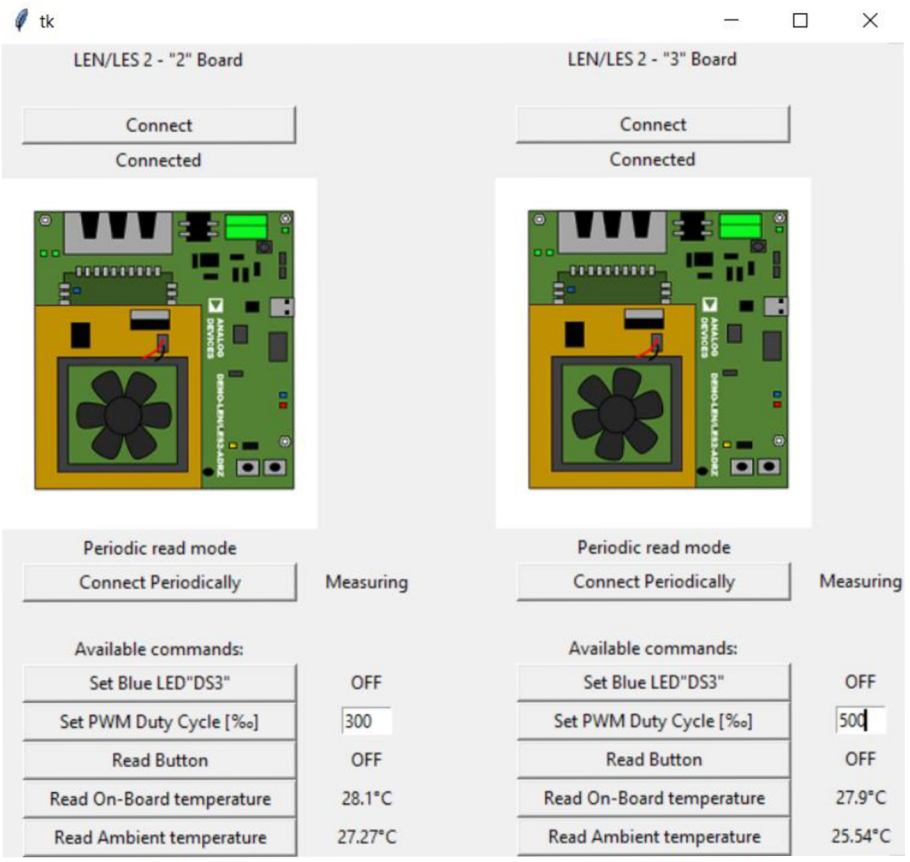


Figure 32 The master TCP/Modbus GUI

7.4 TCP/Modbus slave (server) implementation

Regards to various licensed and limited TCP/Modbus slave C-language libraries was written custom TCP/Modbus layer (library) by the author of the thesis. The TCP/Modbus stack is not completed yet and provides only services necessary to interface master in the settings as was described in the chapter 7.3.

As was mentioned in the chapter 5, current LWIP stack works in polling mode (NO_SYS) and, thus callbacks usage is needed. From the technical point of view, the TCP/Modbus may be understood as TCP/IP protocol, so TCP functions LWIP may be used. Figure 33 depicts theory of MODBUS slave operation. Discussed type *pbuf* is equal to PBUF from the chapter 2.

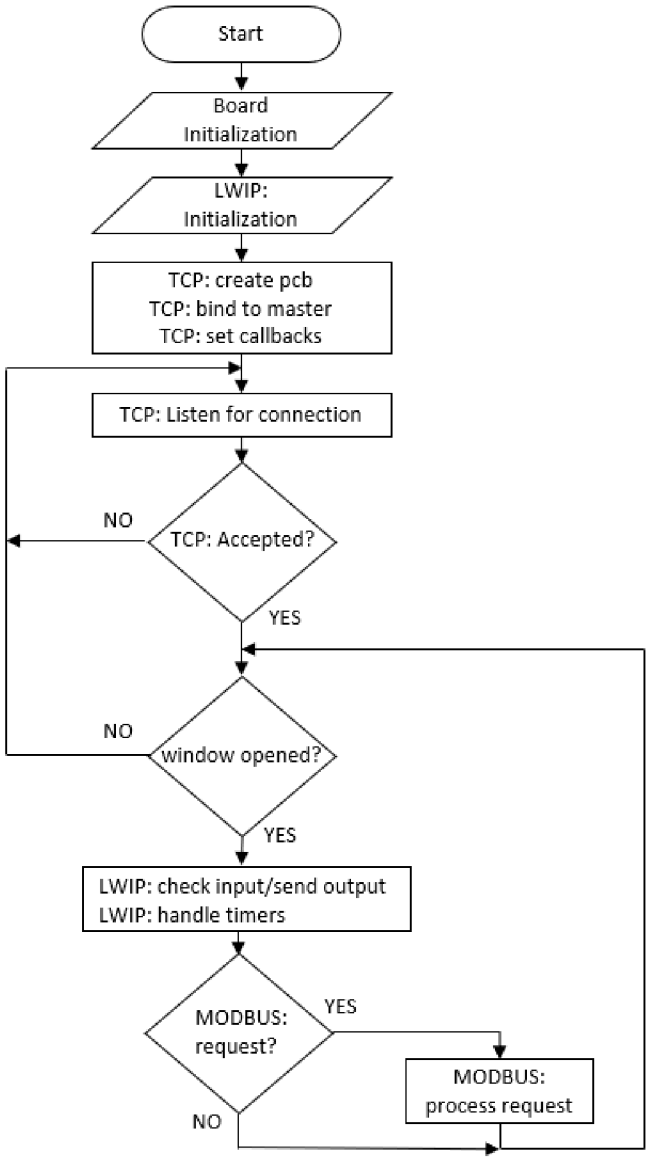


Figure 33 The slave TCP/Modbus diagram

At the first LEN/LES 2 board and LWIP is initialized. Then is created new TCP/IP session calling function ***tcp_new()*** and consequent binding to master with ***tcp_bind()***. Note that ***pcb_new()*** creates *tcp_pbc* which has to be passed to any further TCP functions. Also, the callback of the pointer types: *tcp_accept_fn* and *tcp_recv_fn* must be created and their pointers prepared for consequent passing as an argument of the functions.

The ARP function ***etharp_gratituous()*** can be used to notify master about slave's MAC address. However, it is important to call ***tcp_listen_with_backlog()*** and then function ***tcp_accept()*** with passed function of the pointer type *tcp_accept_fn.* Note that ***tcp_listen_with_backlog()*** will change passed *tcp_pcb* to smaller new *tcp_pbc*. To start receive is necessary to call ***tcp_recv()*** and due to changed *tcp_pbc* is the best to call ***tcp_recv()*** in the callback function of the *tcp_accept_fn* pointer type and pass to the ***tcp_recv()*** its *tpc_pbc*. The argument of the of the ***tcp_recv()*** function must be as well function of the pointer type *tcp_recv_fn.* Receiving callback then controls whole loop. In case of closed connection, receive callback is called with passed NULL *pbuf*, otherwise *pbuf* contains received data (TCP/IP payload). Received data are parsed according to the TCP/Modbus rules and appropriate reply to the request is created. Created response is sent calling ***tcp_write()*** and ***tcp_output()***. Do not forget inform LWIP about received data calling ***tcp_recved()*** and free *pbuf* calling ***tcp_free().*** Note that NULL as the *pbuf* contained in the receiving callback is suitable to signalize "closed windows".

```
1. void modbus_tcp_req_read_single_coil(uint8_t* msg_in,
2.     uint32_t msg_in_lng, uint8_t *msg_out, uint32_t*
   msg_out_lng);
3.
4. void modbus_tcp_req_write_single_coil(uint8_t* msg_in,
5.     uint32_t msg_in_lng, uint8_t *msg_out, uint32_t*
   msg_out_lng);
6.
7. void modbus_tcp_req_read_discrete_input(uint8_t* msg_in,
8.     uint32_t msg_in_lng, uint8_t *msg_out, uint32_t*
   msg_out_lng);
9.
10.  void modbus_tcp_req_write_analog_reg(uint8_t* msg_in,
11.      uint32_t msg_in_lng, uint8_t *msg_out, uint32_t*
   msg_out_lng);
12.
13.  void modbus_tcp_req_read_analog_reg(uint8_t* msg_in,
14.      uint32_t msg_in_lng, uint8_t *msg_out, uint32_t*
   msg_out_lng);
15.
16.  void modbus_tcp_req_read_input_reg(uint8_t* msg_in,
17.      uint32_t msg_in_lng, uint8_t *msg_out, uint32_t*
   msg_out_lng);
```

This snippet is the list of the slave MODBUS functions to process request (e.g. read DO, write AO) and create reply (e.g. read AI, read AO). There is obvious msg_in containing modbus incoming request and msg_out where the response is passed.

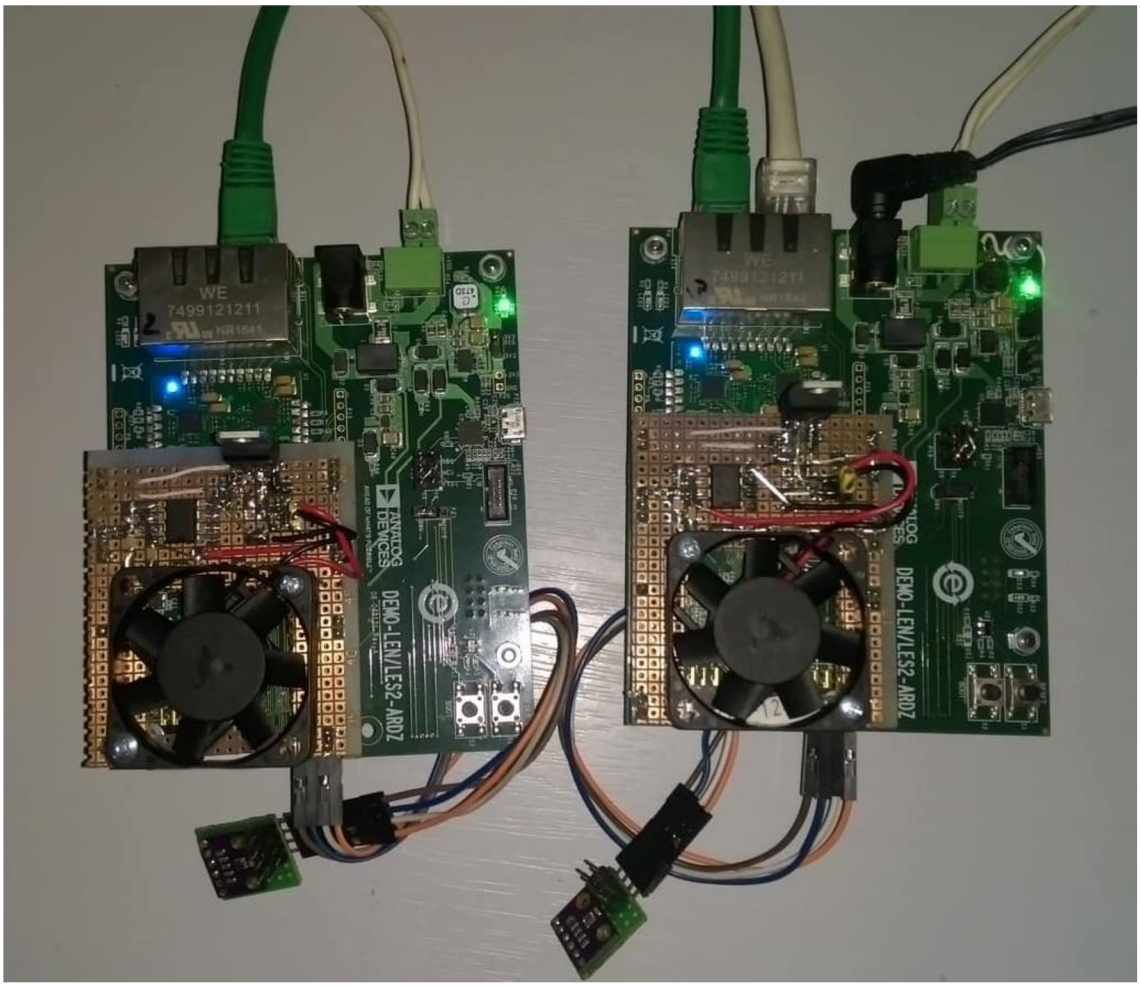


Figure 34 Low Complexity Ethernet Demonstration System

Figure 34 is the photograph of the running TCP/Modbus system using two LEN/LES 2 boards. There are also visible two thermometers BMP280 [32] providing the ambient temperature. Using these thermometers and on-board ADT75 [22] (hidden under the fan) was found, that during usage of the board temperature increases by approximately 3°C in compare to air ambient temperature (FPGA-based MAC- PHY is the source of the heat). Whole system is supplied by external 12 VDC adapter. The right LEN/LES 2 boards works also as the switch – packet which are not addressed to the right board are re-transmitted immediately to the left board (connection via green cable). The grey cable leads to the PC ethernet port with pre-set static IP address. The white two-wire cable ensure power supply loop extended from the left LEN/LES board (static IP application). The whole described system of the Figure 34 is used for conclusion, no other auxiliary connections had been done. The LWIP runs in the polling mode (user accesses raw API).

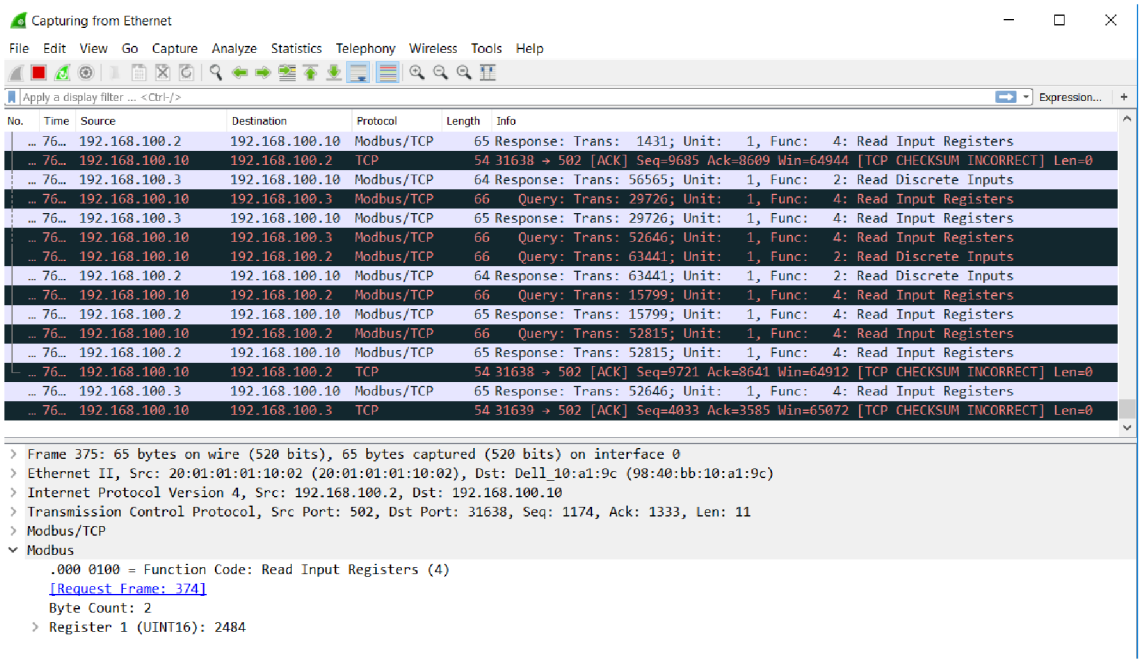


Figure 35 TCP/Modbus master-slave communication

Except the visual functionality of the boards was used Wireshark to sniff communication between LEN/LES 2 boards (slaves) and PC (master) – Figure 35. The Wireshark tool recognizes protocol on the side of the slaves as well as on the side on the master. The packets of the black highlight are outgoing from the PC (master) and signalizes not correct checksum (will be processed in the PC NIC) – source IP address 192.168.100.10. The light-blue packets are responses of the slaves. There are two slaves of the IP addresses 192.168.100.3 and 192.168.100.2. Note that 192.168.100.2 is the left one of Figure 34. At the bottom of Figure 35 is visible typical payload of the slave response – read input register. In this case the ambient temperature was read and number 2484 is 24.84°C (measured by BMP280).

8 CONCLUSION

The main goal of the assignment to propose hardware based on proprietary MAC-PHY and ADuCM4050 was done as the semester thesis. This system proposal covers chapters up to chapter 3. Also, there is brief introduction to LWIP and ARM microcontrollers.

The chapter 4 aims into the first task of the diploma thesis part of assignment hardware proposal – layout section. The whole system called as LEN/LES 2 board was proposed by author of this thesis (schematics, physical placement of parts), however layout finalizing, which concern routing and layering was done by ADI specialist Pat Sheahan. The subchapter 4.2 publishes known mistakes of proposal, but no one of them is critical and may prevent full functionality of the LEN/LES 2 board. Although the firmware and complexity of the system was growing gradually, chapter 5.2 also expresses full-functionality test and current consumption. The consumption was measured approximately 0.3 A at 3.3 V power supply.

Consequent chapter 5 describes NO-SYS LWIP implementation in detail (including function and code snippets) and so may be useful as reference for other engineers, which would like to start with LWIP. Result reached in the chapter 5 is measured average ping 7ms during sent 4702 cycles and received 4679 replies (0.0% loss). As an addition the subchapter 6.2.2 sketches successful LWIP DHCP client implementation (PC as the DHCP server). Note that LWIP is used as the polling raw API accessing mode.

To use LEN/LES 2 boards in application close to industrial was invented system with controlled fan, on-board temperature and ambient temperature measurement. The smaller chapter 6 is about the hardware implementation to ensure fan control.

Finally, chapter 7 aims into the TCP/Modbus master and slave implementation. Meanwhile the master was implemented as GUI Python application with used pyModbusTCP, the slave side was implemented as the bare metal. In total two LEN/LES 2 boards were manufactured so TCP/Modbus system with two slaves was assembled. The distinguish between them was realized using IP and MAC address recognition. The system was fully tested and can demonstrate fan control, temperature measurement and discrete LED switching, button reading. The full application does not occupy more than half of ADuCM4050 FLASH and RAM resources (512kB, 128kB).

However, the weakest part of the system is the python master GUI and may be improved from the visual point of view as well as in term of safety and reliability (uses tentative libraries). To reach better reliability of the system is best to use RTOS based LWIP (running with OS) and use socket layer. Although reached ping discussed upper was reached 7ms, this value can be various across the traffic and so OS LWIP is needed. The LWIP seems to be optimized and regards to limited resources robust enough. Especially, application should be tested and optimized with certified TCP/Modbus device to improve (in this diploma thesis both parts were represented using non-certified equipment).

9 SOURCES

[1] DUNKELS, Adam. Design and Implementation of the LWIP TCP/IP Stack [online]. Swedish Institute of Computer Science, Kista, Sweden, 2001 [cit. 2019-12-06]. Dostupné z: http://dunkels.com/adam

[2] DUNKELS, Adam. UIP - A Free Small TCP/IP Stack [online]. Swedish Institute of Computer Science, Kista, Sweden, 2001 [cit. 2019-12-06]. Dostupné z: http://dunkels.com/adam/

[3] KOLKA, Zdeněk. Počítačové a komunikační sítě [online]. Brno, Czechia, 2007 [cit. 2019-12-06]. Dostupné z: https://moodle.vutbr.cz/course/view.php?id=171162

[4] LWIP TCP/IP stack demonstration for STM32F4x7 microcontrollers: Datasheet [online]. 2013 [cit. 2019-12-06]. Dostupné z: https://www.st.com/en/embedded-software/stsw-stm32070.html

[5] YIU, Joseph. Definitive Guide to ARM(r) Cortex(r)-M3 and Cortex(r)-M4 Processors [online]. ARM Ltd., Cambridge, UK: Newnes, 2014 [cit. 2019-12-06]. Dostupné z: https://www.bookshop.cz/elsevier-science-technology/definitive-guide-to-arm-r-cortex-r-m3-and-cortex-r-m4-processors

[6] ARMv7-M architecture [online]. ARM Ltd., Cambridge, UK, 2010 [cit. 2019-12-06]. Dostupné z: https://developer.arm.com/docs/ddi0403/e/armv7-m-architecture-reference-manual

[7] ADuCM4050EZKIT: Schematic/Manual [online]. 2017 [cit. 2019-12-06]. Dostupné z: https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/adzs-u4050lf-ezkit.html

[8] LTC3630: Datasheet [online]. Milpitas, CA, USA, 2012 [cit. 2019-12-06]. Dostupné z: https://www.analog.com/en/products/ltc3630.html

[9] ADP7142: Datasheet [online]. Norwood, MA, USA, 2019 [cit. 2019-12-06]. Dostupné z: https://www.analog.com/en/products/adp7142.html

[10] SMAJ33CA: Datasheet [online]. 2010 [cit. 2019-12-06]. Dostupné z: https://www.st.com/content/st_com/en/products/protection-devices/eos-10-1000-microsecond-surge-protection/400w-tvs/smaj.html

[11] PTC1812L: Datasheet [online]. 2017 [cit. 2019-12-06]. Dostupné z: https://www.littelfuse.com/products/resettable-ptcs/surface-mount/1812l/1812l110_33.aspx

[12] MBRA160T3: Datasheet [online]. 2017 [cit. 2019-12-06]. Dostupné z: https://www.onsemi.com/pub/Collateral/MBRA160T3-D.PDF

[13] HSP051: Datasheet [online]. 2017 [cit. 2019-12-06]. Dostupné z: https://www.st.com/content/st_com/en/products/protection-devices/esd-protection/high-speed-port-protection/hsp051-4m10.htm

[14] EV-COG-ADUCM4050: Schematic/Manual [online]. 2019 [cit. 2019-12-06]. Dostupné z: https://wiki.analog.com/resources/eval/user-guides/ev-cog-ad4050lz

[15] ADUCM4050: Datasheet [online]. 2019 [cit. 2019-12-06]. Dostupné z: https://www.analog.com/en/products/aducm4050.html

[16] ADUCM4050: Datasheet HRM [online]. 2019 [cit. 2019-12-06]. Dostupné z: https://www.analog.com/en/products/aducm4050.html

[17] RYAN, Rosemary. System Demonstration Platform Facilitates Quick Prototyping and Evaluation: Standard [online]. Limerick, Ireland, 2011 [cit. 2019-12-06]. Dostupné z: https://www.analog.com/en/analog-dialogue/articles/demo-platform-quick-prototyping-evaluation.html#author

[18] Arduino Offical Page [online]. 2019 [cit. 2019-12-06]. Dostupné z: https://www.arduino.cc/

[19] Digilent Pmod™ Interface Specification: Standard [online]. Pullwan, WA, USA, 2011 [cit. 2019-12-06]. Dostupné z: https://www.digilentinc.com/Pmods/Digilent-Pmod_%20Interface_Specification.pdf

[20] FT232RQ: Datasheet [online]. Glasgow, UK, 2019 [cit. 2019-12-06]. Dostupné z: https://www.ftdichip.com/

[21] ADM6315: Datasheet [online]. Norwood, MA, USA, 2019 [cit. 2019-12-06]. Dostupné z: https://www.analog.com/en/products/adm6315.html

[22] ADT75: Datasheet [online]. Norwood, MA, USA, 2019 [cit. 2019-12-06]. Dostupné z: https://www.analog.com/en/products/adt75.html

[23] INTRODUCTION TO MODBUS TCP/IP: Standard [online]. Wixom, MI, USA, 2005 [cit. 2019-12-06]. Dostupné z: https://www.acromag.com/

[24] United Automation - OPC UA: Web Page [online]. 2019 [cit. 2019-12-06]. Dostupné z: http://documentation.unified-automation.com/uasdkhp/1.0.0/html/index.html

[25] Learn more about Syslog Protocol: Thomas Porter [online]. 2007 [cit. 2019-05-15]. Dostupné z: https://www.sciencedirect.com/topics/computer-science/syslog-protocol

[26] Savannah LWIP site [online]. 2019 [cit. 2019-05-02]. Dostupné z: https://savannah.nongnu.org/bugs/?49631

[27] Fandom LWIP wiki [online]. 2019 [cit. 2019-05-02]. Dostupné z: https://LWIP.fandom.com/wiki/LWIP_Wiki

[28] 25LC01A: Datasheet [online]. USA, 2012 [cit. 2019-05-02]. Dostupné z: https://www.microchip.com/wwwproducts/en/25LC010A

[29] EB40100S2-1000U-999: Datasheet [online]. Kaohsiung, Taiwan, 2010 [cit. 2019-05-04]. Dostupné z: http://www.sunon.com/index2/index.php

[30] LM385: Datasheet [online]. Dallas, Texas, USA, 2019 [cit. 2019-05-04]. Dostupné z: http://www.ti.com/lit/ds/symlink/lm158-n.pdf

[31] LM317: Datasheet [online]. Dallas, Texas, USA, 2019 [cit. 2019-05-04]. Dostupné z: http://www.ti.com/lit/ds/symlink/lm317.pdf

[32] BMP280: Datasheet [online]. Reutlingen, Germany, 2019 [cit. 2019-05-04]. Dostupné z: https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BMP280-DS001.pdf

[33] PyModbusTCP [online]. Python Software Foundation [cit. 2019-05-14]. Dostupné z: https://pypi.org/project/pyModbusTCP/

[34] Socket Programming in Python: Nathan Jennings [online]. 2018 [cit. 2019-05-14]. Dostupné z: https://realpython.com/python-sockets/

[35] An Introduction to Tkinter: Fredrik Lundh [online]. 2005 [cit. 2019-05-14]. Dostupné z: http://effbot.org/tkinterbook/

10 QUANTITIES AND ABRREVIATIONS

V_{IN}	*Input voltage of the certain system*
V_{OUT}	*Output voltage of the certain system*
V_{USB_5V}	*Convenient 5V voltage of the USB bus*
$V_{FT232RQ}$	*Output voltage of the internal LDO in case of the FT232RQ*
I_{OUT}	*Output Current*
R_{ISET}	*Resistor for current setting*
SS_{TIME}	*Soft start time*
$V_{SUPPLY+}$	*Positive power supply voltage*
$V_{SUPPLY\ GND}$	*Ground potential of the voltage source (minus)*
$V_{SUPPLY\ EARTH}$	*Earth potential of the source*
netif	*LWIP structure of the network interface*
pbuf	*LWIP structure of the packet*

LWIP	Light-weight TCP/IP stack
µIP	Micro TCP/IP stack (lighter than LWIP)
MAC-PHY	Proprietary prototype implementing MAC layer.
SWD	Serial Wire Debug
JTAG	Joint Test Action Group (standard)
J-LINK	Debug Interface
MAC	Media Access Control
TCP/IP	Transmission Control Protocol/Internet Protocol
IP	Internet Protocol
ICMP	Internet Control Message Protocol
DHCP	Dynamic Host Configuration Protocol
UDP	User Datagram Protocol
IPC	Inter Process Communication
HTTP	Hyper Text Transfer Protocol
API	Application Programming Interface
RAM	Random Access Memory
ROM	Read Only Memory
FLASH	Solid state non-volatile memory, electronically erasable
RO	Read Only

RW	Read Write
SRAM	Static RAM
CISC	Complex Instruction Set Computing
RISC	Reduced Instruction Set Computing
ISR	Interrupt Service Routine
MCU	Microcontroller Unit
ARM	Advanced RISC Machine
OPC – UA	Open Platform Communications – Unified Automation
LEN	Proprietary ADI Low Complexity Ethernet Node module
LES	Proprietary ADI Low Complexity Ethernet Switch module
ADI	Analog Devices International
SoC	System on Chip
TCP/IP	Transmission Control Protocol/Internet Protocol
IP	Internet Protocol
ETH	Ethernet
SPI	Serial Peripheral Interface
I2C	Inter Integrated Circuit (Two Wire Interface)
UART	Universal Asynchronous Receiver and Transmitter
USB	Universal Serial Bus
GPIO	General Purpose Input Output
IO	Input Output
PHY	Physical layer of the OSI ethernet model
OSI	Open System Interconnection
AXI	Advanced Microcontroller Bus Architecture (AMBA) – AXI type
APB	AMBA, version APB
IC	Integrated Circuit
FPGA	Field Programmable Gate Array
VDC	Voltage caused by Direct Current
HW	Hardware
IP	Intellectual Property (in term of IP core)
MSP	Main Stack Pointer
PSP	Process Stack Pointer
xPSR	Process Status Register
LDO	Low Dropout regulator

SS	Soft Start (in term of the LDO)
SDP	System Demonstration Platform (in term of ADI the platforms)
SPORT	Serial PORT, customized ADI dual SPI interface
ADC	Analog to Digital Converter
SW	Software
SW	Switch (e.g. Transistor)
PTC	Positive Temperature Coefficient (e.g. polyfuse)
WAKE	High priority interrupts of the ADuCM4050
LF	Low Frequency
HF	High Frequency
PMOD	Peripheral Module
EEPROM	Electronically Erasable Programmable Read Only Memory
ID	Identifier (number)
LED	Light Emitting Diode
NAND	Negated Output Logical AND Gate
STM	ST microcontroller
DNI	Do not place
OA	Operational Amplifier
RTU	Remote Terminal Unit
MBAP	MODBUS Application Header
PDU	Protocol Data Unit
GUI	Graphical User Interface
DO	Digital Out (Modbus coil)
AO	Analog Out (Modbus holding register)
AI	Analog Int (Modbus input register)

APPENDIX

- The LEN/LES 2 board schematic (5 pages)
- The LEN/LES 2 board layout (10 pages)

RELAY CONTROL CHART

CONTROL	CODE	DEVICE	FUNCTION	CONNECTOR

JUMPER TABLE

JP#	ON	OFF
1		
2		
3		
4		
5		

* SEE ASSEMBLY INSTRUCTIONS

REVISIONS

REV	DESCRIPTION	DATE	APPROVED

P.O SPEC.	BK/BD SPEC.	SOCKET OEM	OEM PART#	HANDLER

TEMPLATE ENGINEER –	DATE	
HARDWARE SERVICES PAT SHEAHAN		
HARDWARE SYSTEMS –		
TEST ENGINEER –		
COMPONENT ENGINEER –		
TEST PROCESS –		
HARDWARE RELEASE –		
DESIGNER –		
PTD ENGINEER VLADIMIR SUSTEK		
CHECKER –		

SCHEMATIC

ANALOG DEVICES

HW TYPE : Customer Evaluation
DEMO-LEN/LES2-ARDZ
–
–
–
–
–
–

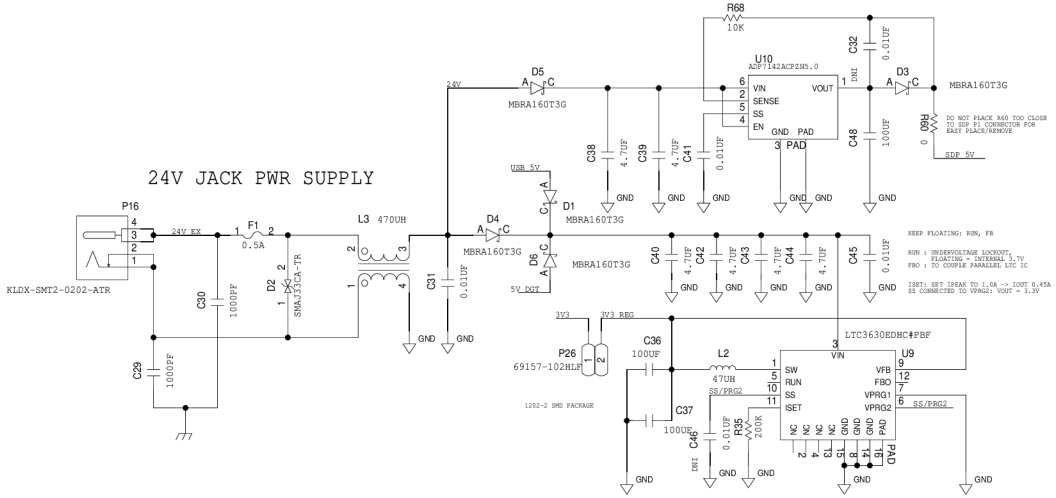
MASTER PROJECT TEMPLATE –
TESTER TEMPLATE no_template

UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES
TOLERANCES
DECIMALS X.XX +-0.010 X.XXX +-0.005
FRACTIONS +-1/32
ANGLES +-2

DRAWING NO. 02-049121
REV. A

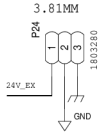
SIZE D | SCALE N/A | CODE ID NO. CodeID | SHEET 1 OF 5

REVISIONS			
REV	DESCRIPTION	DATE	APPROVED



PWR CONNECTOR

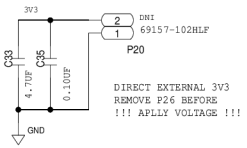
3.81MM



PWR INDICATOR

PLACE CLOSE TO THE USB AND JACK

3.3V POWER SUPPLY



ANALOG DEVICES

SCHEMATIC

HW TYPE : Customer Evaluation
DEMO-LEN/LES2-ARDZ
-

DESIGN VIEW
<DESIGN_VIEW>

DRAWING NO.
02-049121

REV
A

PTD ENGINEER
VLADIMIR SUSTEK

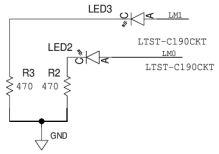
SIZE
D

SCALE
N/A

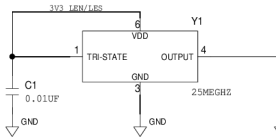
SHEET 2 OF 5

REVISIONS			
REV	DESCRIPTION	DATE	APPROVED

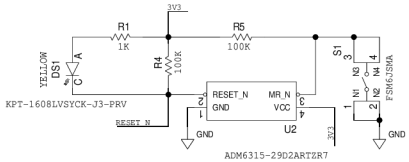
LEN MODE INDICATORS



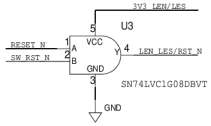
CLOCK SOURCE



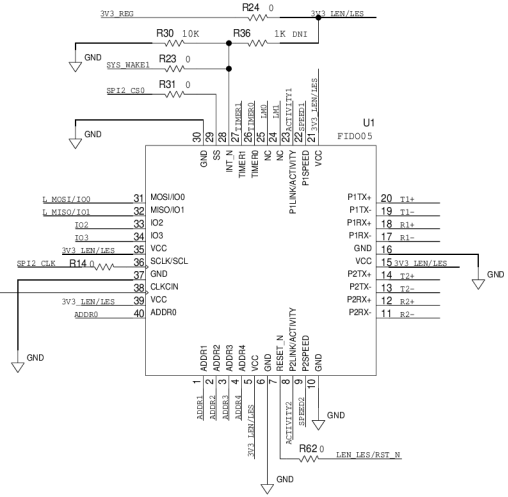
RESET



LES RESET (CONDITIONED)

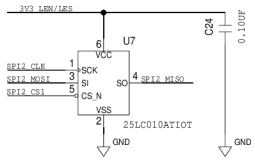


LES CHIP PAC

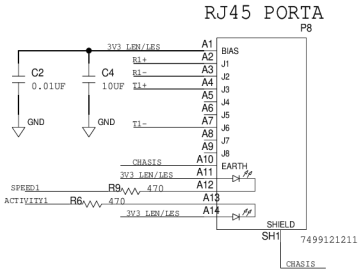


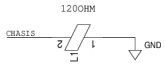
LES/LES MODE SET

EEPROM (MAC OF LES)

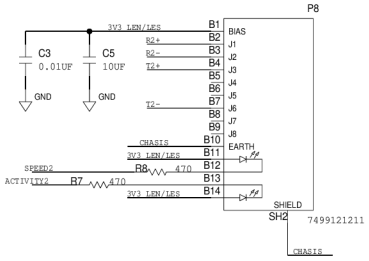


RJ45 PORTA





RJ45 PORTB

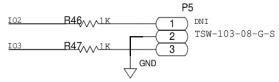


TEST/OUTPUT HEADERS (OPTIONAL)

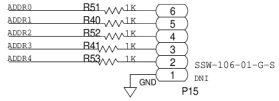
PLACE IN ORDER TO SAFE SPACE

LES SPI TEST

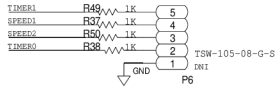
LEN/LES IO



LEN/LES ADDRESSES



LEN/LES TMR AND SPEED

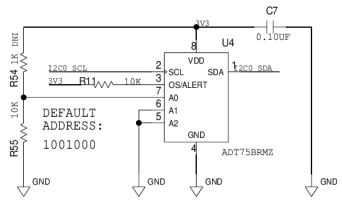


ANALOG DEVICES	SCHEMATIC		
	HW TYPE : Customer Evaluation DEMO-LEN/LES2-ARDZ -		
THIS DRAWING IS THE PROPERTY OF ANALOG DEVICES INC. IT IS NOT TO BE REPRODUCED OR COPIED IN WHOLE OR IN PART, OR USED IN ANY MANNER WITHOUT THE WRITTEN PERMISSION OF ANALOG DEVICES. THE EQUIPMENT SHOWN HEREIN MAY BE PROTECTED BY PATENTS OWNED OR CONTROLLED BY ANALOG DEVICES.	DESIGN VIEW <DESIGN_VIEW>	DRAWING NO. 02-049121	REV A
	PTD ENGINEER VLADIMIR SUSTEK	SIZE D	SCALE N/A
		SHEET 3 OF 5	

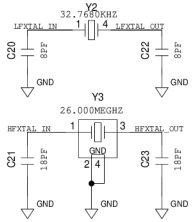
MAIN HOST PROCESSOR OF LES

REVISIONS			
REV	DESCRIPTION	DATE	APPROVED

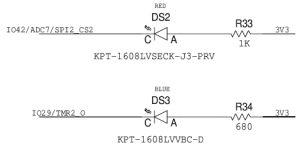
DEMO DATA TEMP SENSOR



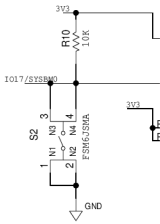
MCU CLOCK LF/HF XTAL



VISUAL OUTPUT (LEDS)



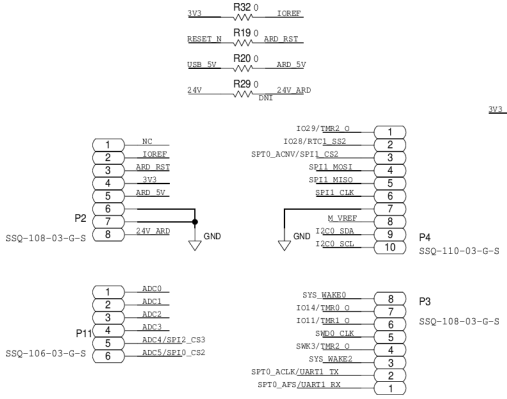
BOOTMODE



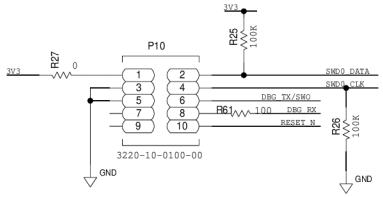
ANALOG DEVICES	SCHEMATIC
	HW TYPE : Customer Evaluation DEMO-LEN/LES2-ARDZ -
DESIGN VIEW <DESIGN_VIEW>	DRAWING NO. 02-049121 REV A
PTD ENGINEER VLADIMIR SUSTEK	SIZE D SCALE N/A SHEET 4 OF 5

REVISIONS			
REV	DESCRIPTION	DATE	APPROVED

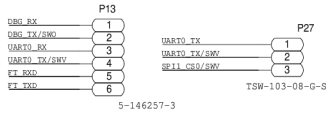
ARDUINO INTERFACE



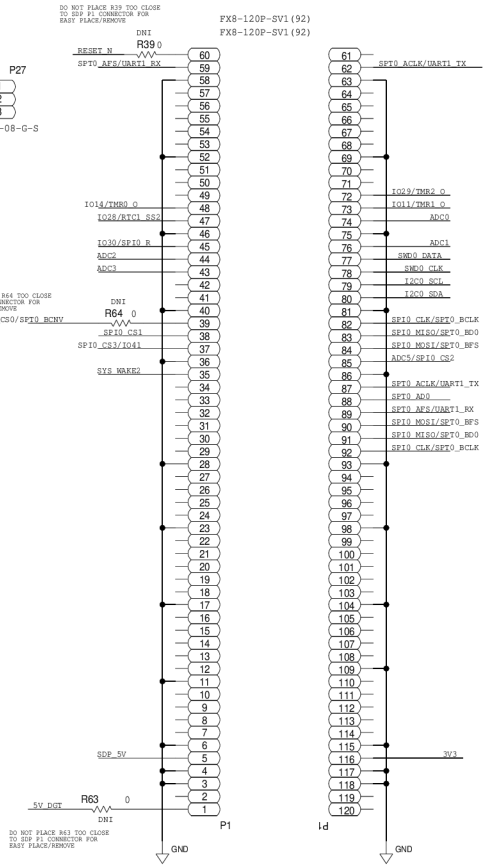
J-LINK DEBUG



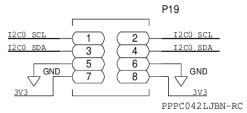
UART AND DEBUG SIGNAL SETTINGS



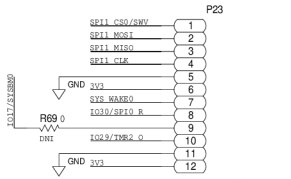
MOTHERBOARD SDP CONNECTOR



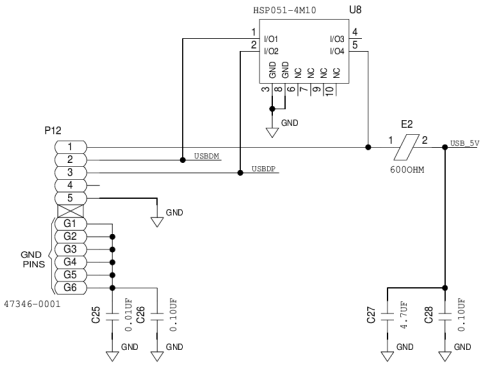
PMOD I2C



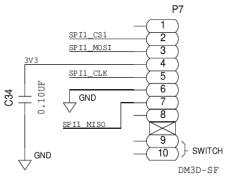
PMOD SPI



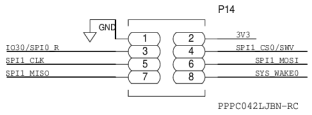
MICRO USB CONNECTOR



MICRO SD CARD SLOT

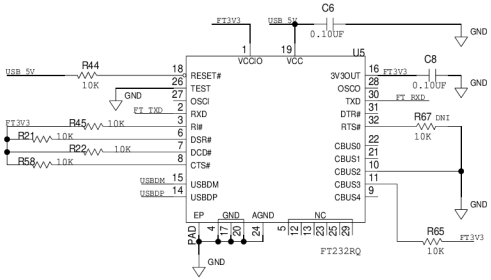


RF MODULE SPI INTERFACE



USB TO UART FTDI CHIP

FTDI TXD TO MCU RXD
FTDI RXD TO MCU RXD

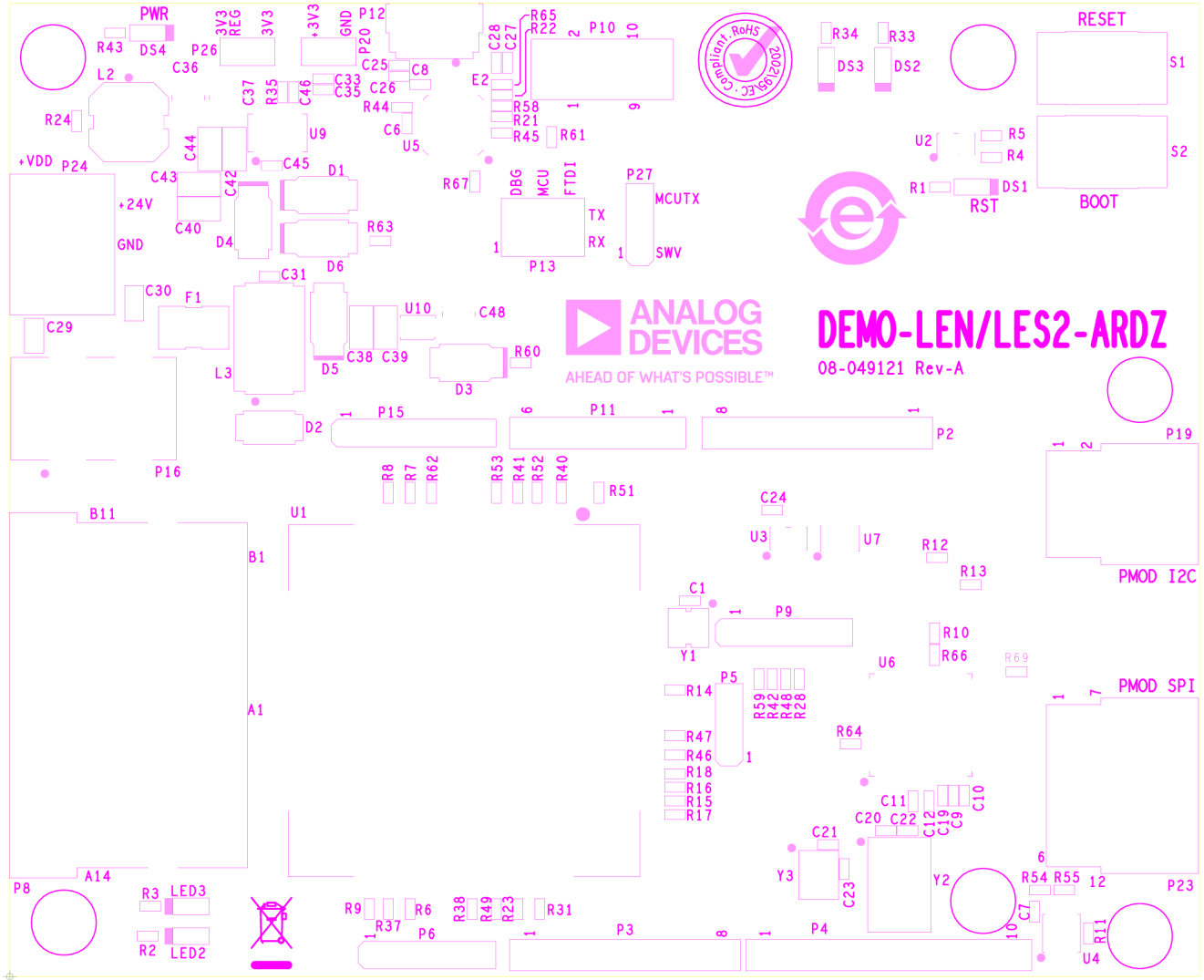


ANALOG DEVICES	SCHEMATIC		
	HW TYPE : Customer Evaluation DEMO-LEN/LES2-ARDZ –		
	DESIGN VIEW <DESIGN_VIEW>	DRAWING NO. 02-049121	REV A
	PTD ENGINEER VLADIMIR SUSTEK	SIZE D	SCALE N/A
		SHEET 5 OF 5	

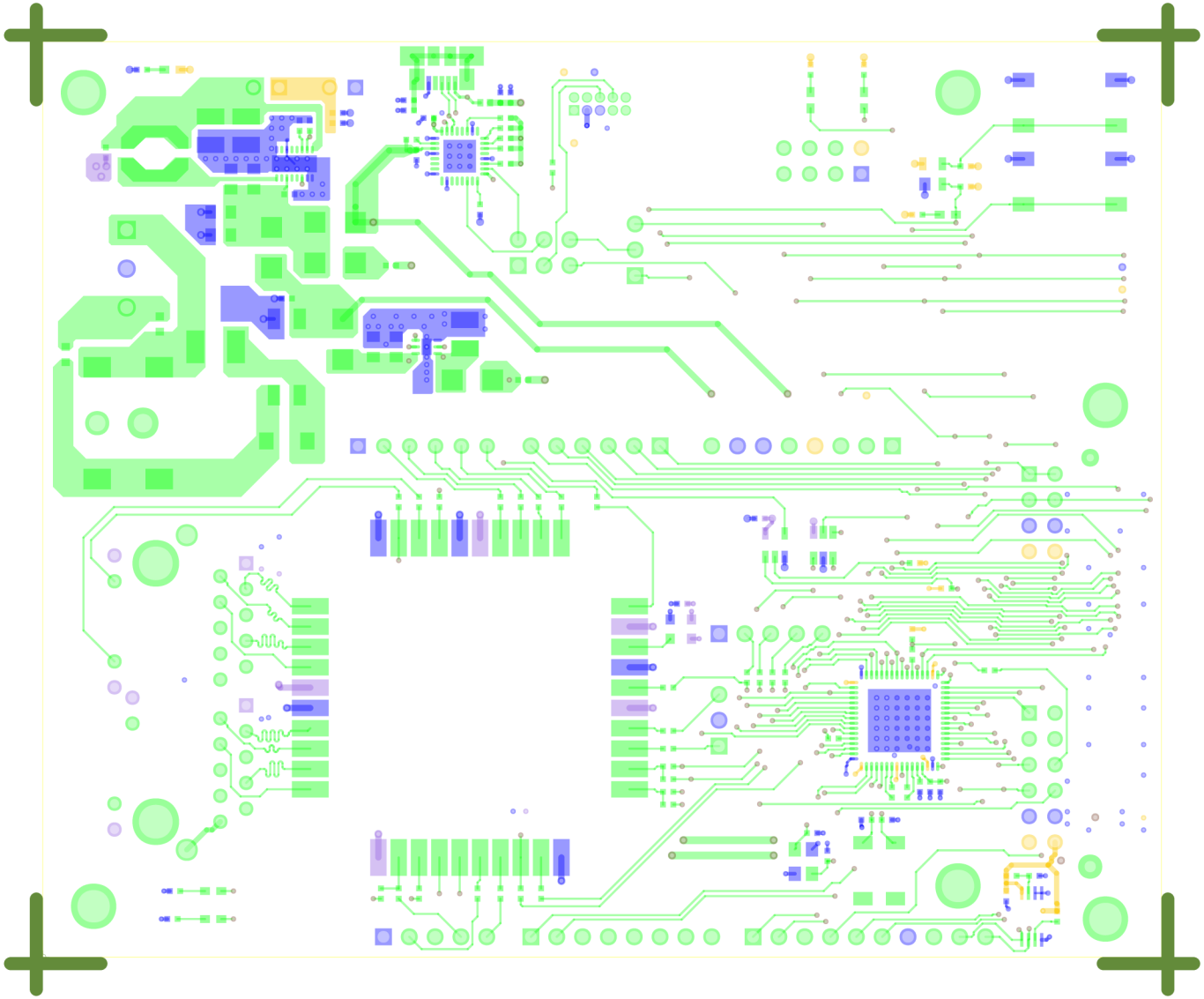
SILKSCREEN PRIMARY

08-049121-03

DEMO-LEN/LES2-ARDZ Rev-A



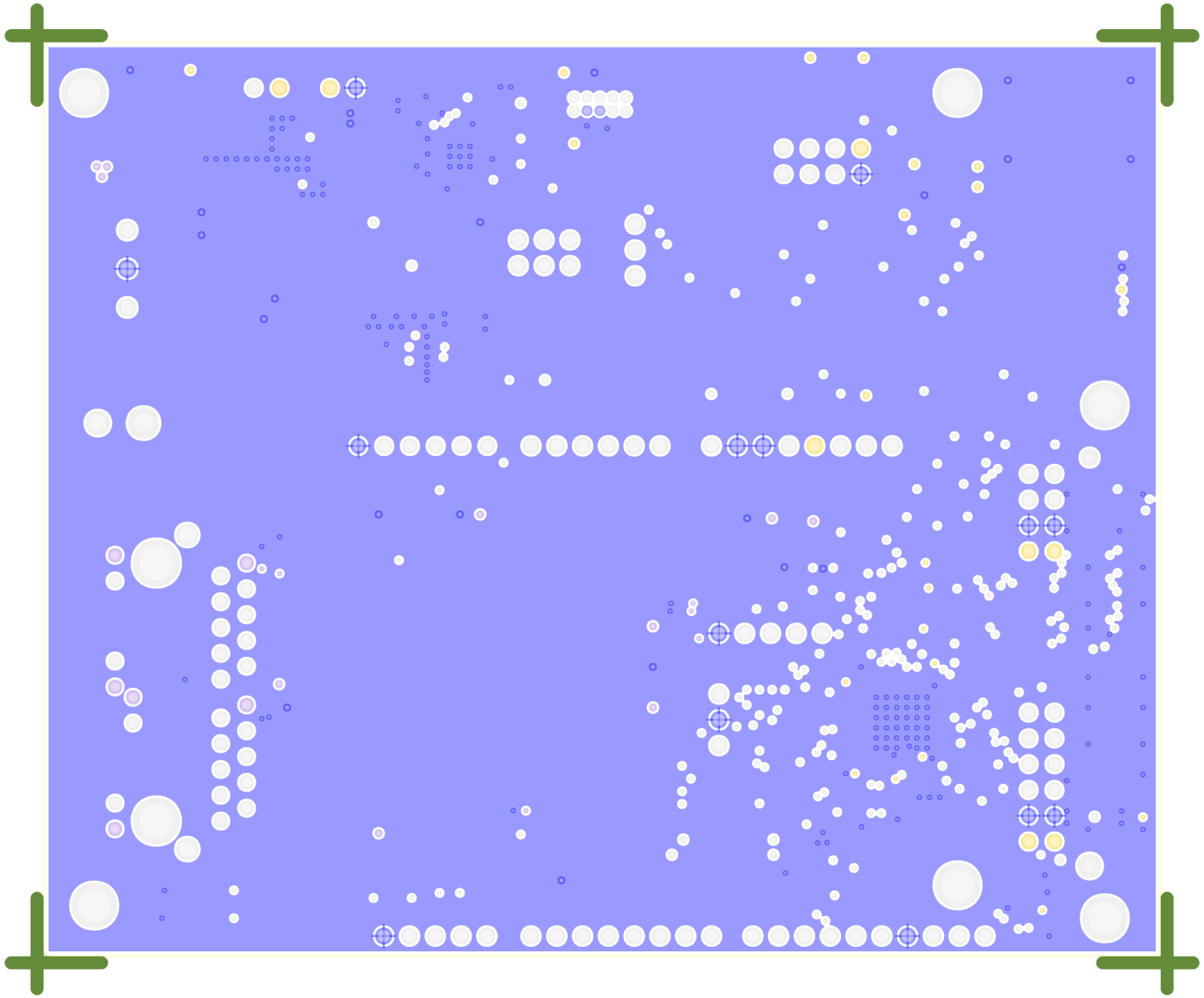
L1 PRIMARY
08-049121-01
DEMO-LEN/LES2-ARDZ Rev-A



L2 Internal GND

08-049121-07

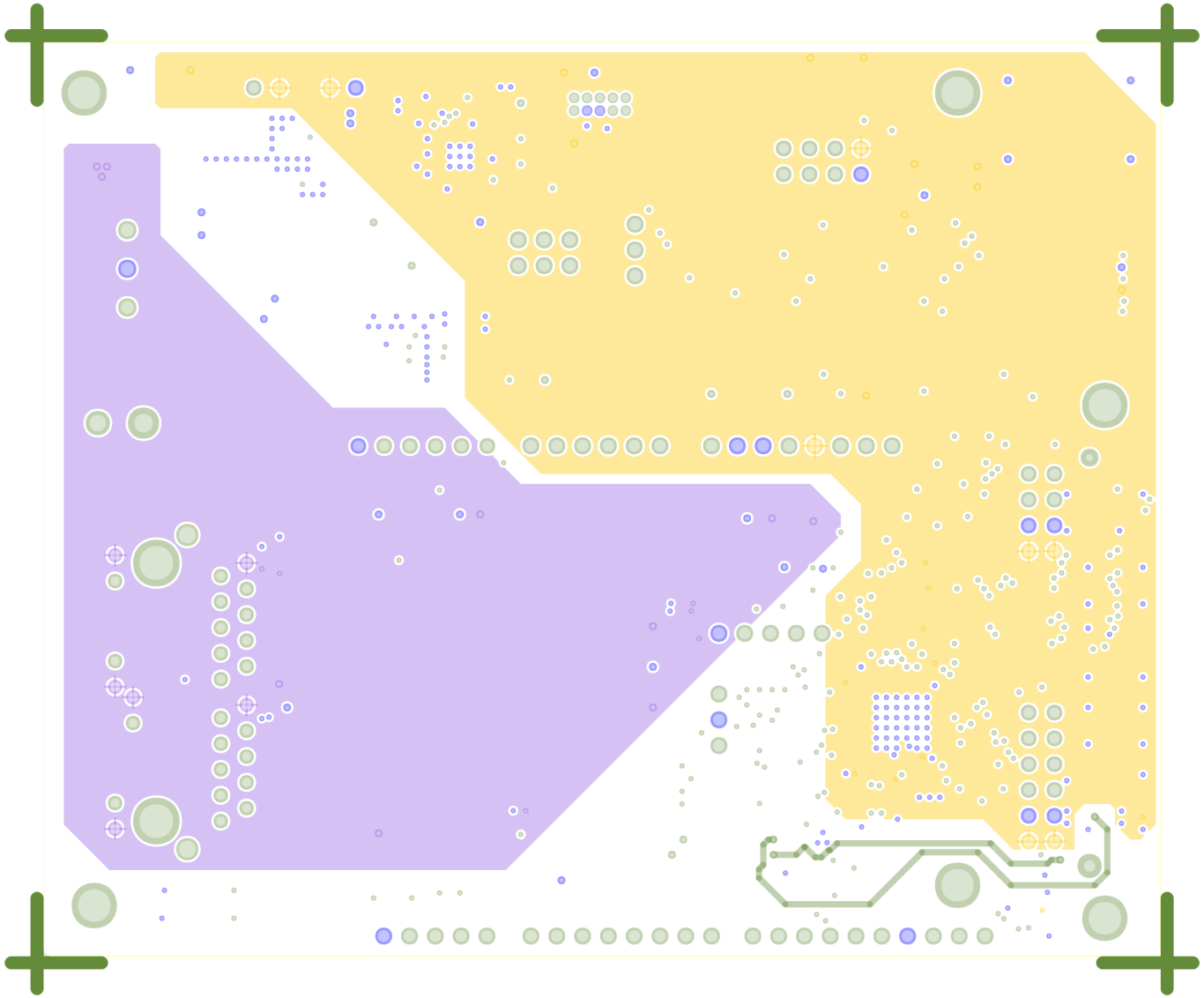
DEMO-LEN/LES2-ARDZ Rev-A



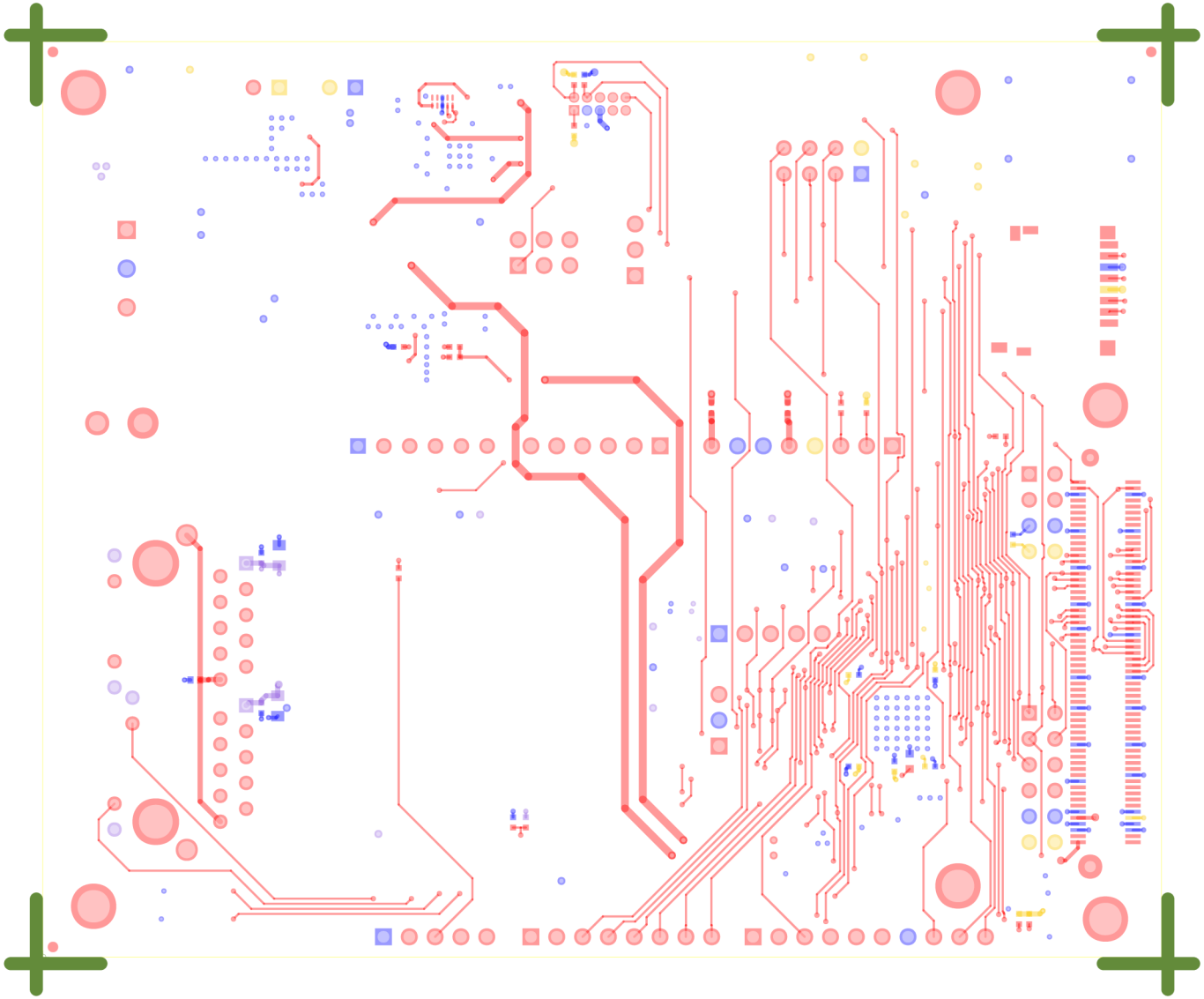
L3 Internal PWR

08-049121-08

DEMO-LEN/LES2-ARDZ Rev-A



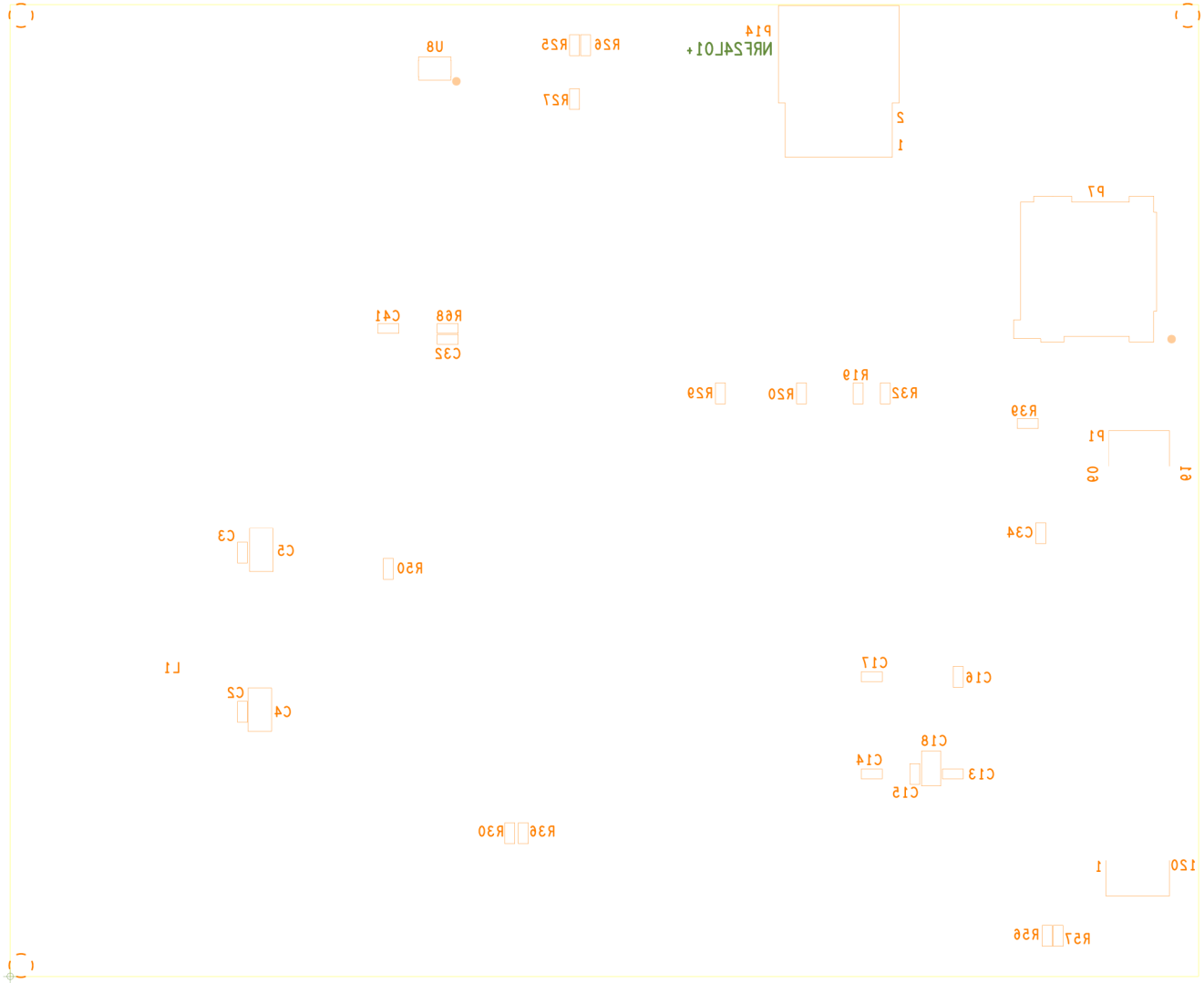
L4 SECONDARY
08-049121-02
DEMO-LEN/LES2-ARDZ Rev-A



SILKSCREEN SECONDARY

08-049121-05

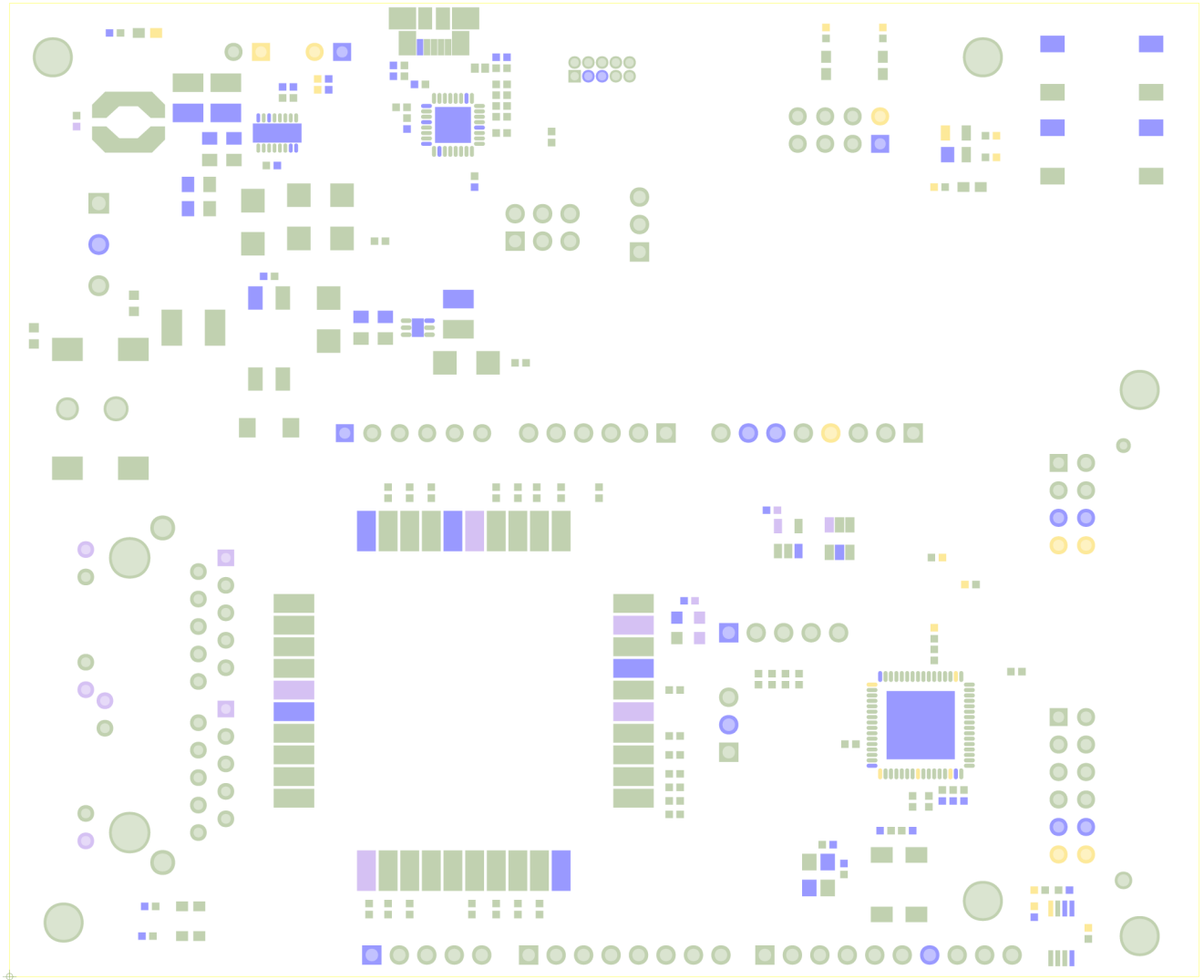
DEMO-LEN/LES2-ARDZ Rev-A



SOLDERMASK PRIMARY

08-049121-04

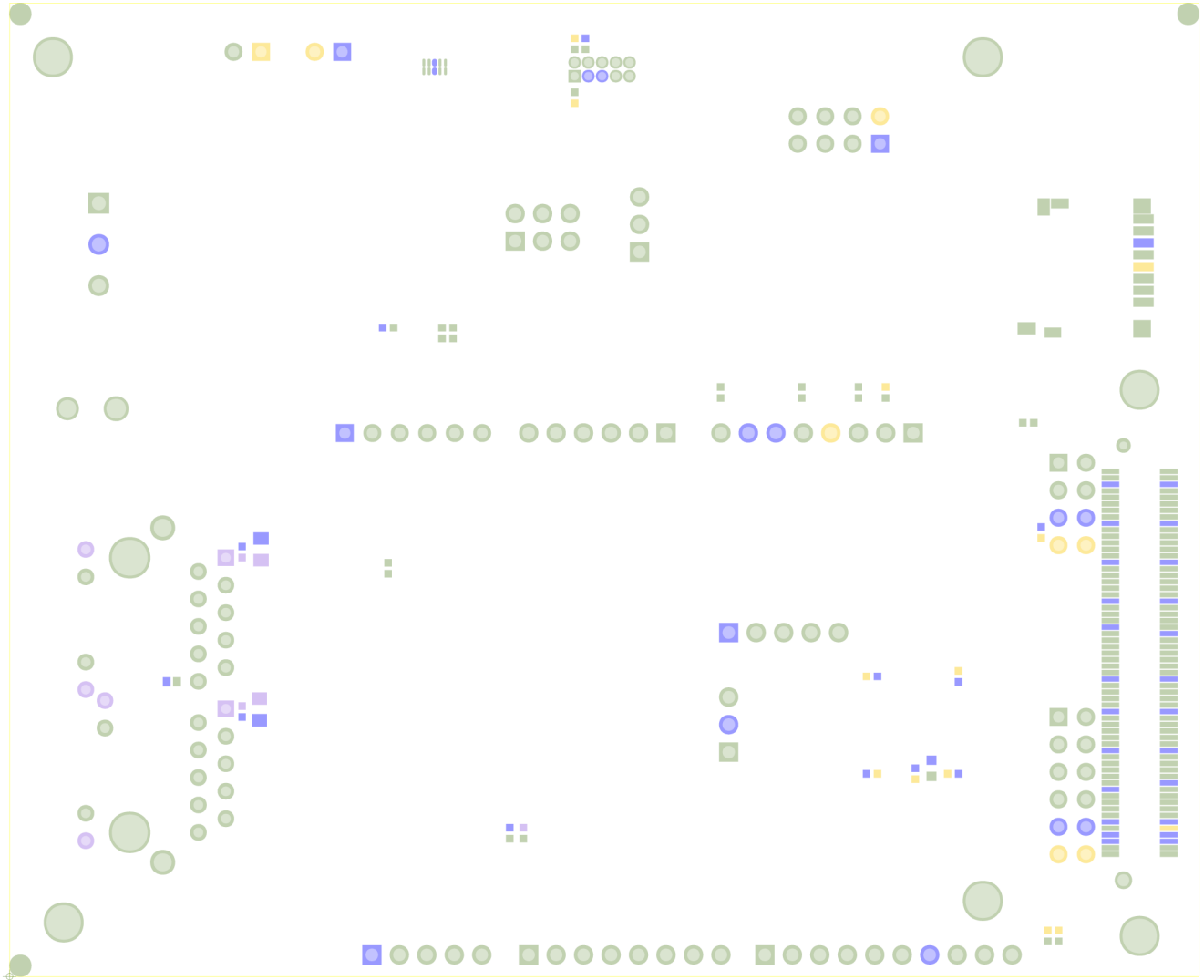
DEMO-LEN/LES2-ARDZ Rev-A



SOLDERMASK SECONDARY

08-049121-06

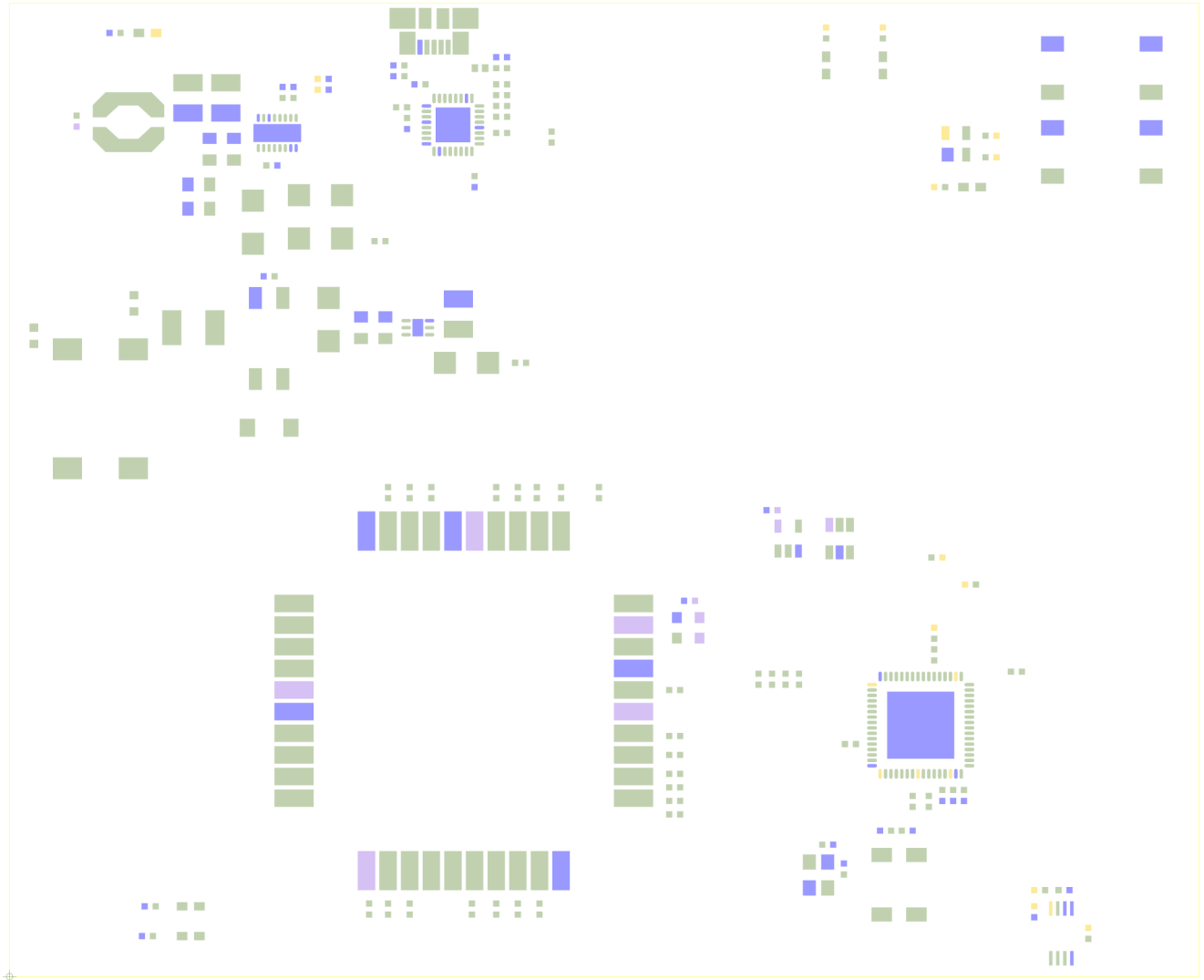
DEMO-LEN/LES2-ARDZ Rev-A



PASTEMASK PRIMARY

08-049121-09

DEMO-LEN/LES2-ARDZ Rev-A



PASTEMASK SECONDARY
08-049121-10
DEMO-LEN/LES2-ARDZ Rev-A