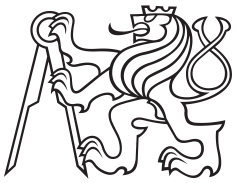
Master Thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Simulink External Mode for Rapid Prototyping Platform

Bc. Jakub Nejedlý

Supervisor: Ing. Michal Sojka, Ph.D.
Field of study: Cybernetics and Robotics
Subfield: Robotics
May 2019

Acknowledgements

Děkuji doktoru Michalu Sojkovi za pomoc a rady při vedení mého projektu. Chtěl bych mu také poděkovat za nabídku pracovat na zajímavé, užitečné práci. Také bych chtěl touto formou poděkovat i svým rodičům, bratrovi Matějovy a přítelkyni Anetě za podporu při tvorbě této práce.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, 24. May 2019

.....

Abstract

The goal of the diploma thesis was implement the external mode into the RPP library for Simulink. The work was performed on the TSM570 development kit from Texas Instruments. The library testing tools were improved during the work. The network thread was modified to receive Ethernet data. The library version has been modified to respect and work with the latest releases of Mathworks. As part of this work, the network communication with the LwIP library was put into operation, the connection speed and stability improved. The commissioning of external mode has not been achieved at work.

Keywords: RPP, external-mode, Simulink, lwip, TSM570 Hercules

Abstrakt

V rámci této diplomové práce bylo cílem implementovat externí mode do knihovny RPP pro Simulink. Práce byli prováděny na vývojovém kitu TSM570 od firmy Texas Instruments. Během práce byli vylepšeny testovací nástroje knihovny a upravena činnost řídících vláken pro zprávu přijatých dat. Verze knihovny byla upravena na poslední vydané verze Mathworks. V rámci práce se podařilo zprovoznit síťovou komunikaci s knihovnou LwIP, zvýšit rychlost a stabilitu připojení. Zprovoznění exteního modu nebylo v práci dosaženo.

Klíčová slova: RPP, external-mode, Simulink, lwip, TSM570 Herkules

Překlad názvu: Externí mód pro Simulink na platformě RPP

Contents

Figures

Tables

Chapter 1

Introduction

The goal of the thesis was implement the external mode into the RPP library. The Industrial Informatics Department has designed the RPP library as a tool to provide rapid development and testing of save machine control process or data acquisition applications. The RPP library is based on the FreeRTOS operating system and contains files that support running the library on multiple platforms. The most widely supported board is the RPP platform, developed directly at the department for Porshe. Other supported boards are the commercially distributed development kit TSM570 and the control unit designed by Eaton fitted with the same TSM570 microprocessor. The library uses the community-developed LwIP stack. It is designed to support full stack functionality with minimal memory requirements. Therefore, LwIP is primarily used on embedded hardware with low memory capacity.
The external mode allows communication between the target, where is the simulation running, and the computer, on which the controller is being developed. During the development of the simulation, the optimization of constants for proper operation takes up a substantial part of the work. When developing on an embedded device, it is necessary to run the program directly on the controller to properly test it. Download and compilation times considerably extend constancy optimization time. The external mode allows data sharing on the operation of the controller directly in the target code and also allows the new settings of the tuned model parameters to be sent and used in the code without the need for a new compilation.
It is necessary to create a function, that allows its correct setting on the transmitter to enable the external mode. There is necessary to properly set up the communication channel and the remaining communication parameters, such as speed, bandwidth, parity, or communication port. For the server-based part on the target device, it is necessary to implement the ETHERNET interface supporting the TSM570 platform. The work was modified for the

latest versions of software released by Mathworks. The main goal of this work for the implementation of external mode is to reduce the time and financial demands for the development of new applications. The TSM570 development kit from Texas Instruments is used to develop critical safety applications. The kit was developed to meet TÜV NORD's safety certification like ASIL D and SIL 3 [TD15]. The kit is recommended for use in the automotive, aviation and production industry. The theoretical part of the thesis can be used as a basic overview of the TSM570 controller. It also includes insight into the libraries used and their capabilities. In the last part, there are presented possibilities of the current implementation of ETHERNET communication and describes testing tools created for debugging possible network errors.// The theoretical part of the thesis contains the description of used hardware and its properties in chapter 2. The libraries and software, used in the RPP library or this project, are described in chapter 3.
The hardware description chapter focuses mainly on the description of the TSM570 platform in chapter 2.1. In the next section, 2.2 is described the use of a particular model. Peripherals used for work, are serial interfaces 2.2.1, which provides communication during testing and provides error statements. The vectored interrupt manager 2.2.2 is necessary to process the interruption correctly to process the data from the ETHERNET peripheral. The hardware section also includes a description of the ETHERNET periphery in chapter 2.3. An integrated MAC controller is described in subsection 2.3.1, and communication with an external physical interface 2.3.2. The last part deals with the core of the TSM570 module with the ARM CORTEX-R4 processor 2.4. The second part of the theoretical part 3 deals with a description of Mathworks software and the libraries used in the project. Section 3.1 describes the working of the simulations, and in chapters 3.1.2 and 3.1.3 is a way in which the external mode works, the benefits and what is necessary for its proper working. The next section describes the development library 3.2 and the relevant subchapters describes the architecture and program tools for testing the proper functioning of the board and all peripherals. Chapter 3.3 is describing the LwIP stack used by the RPP library. The last part 3.4 deals with the theoretical functioning of ETHERNET and discusses some of the particular communication protocols, that have been used in the library.
The part 4 is following partial tasks, which have to be completed for the commissioning of the external mode. The first section 4.2 describes the network structure and network interfaces options. The next section describes how the *ping* test command was implemented. In the next section 4.5, Iperf test tool was implemented as a part of the test software. Subsequently, the RPP main driver of the ETHERNET driver was modified. The function of the main eth thread is described in section 4.6, where the transfer of data from the EMAC controller to the LwIP stack is described. The last part of 4.7 describes an external mode developed on IID as a Linux target for Simulink. If porting a Linux target to the latest version of Simulink will be successful, then it should be possible to use a large part to the TSM570 development kit.

Chapter 2

Descriptions of hardware parts

This chapter describes the use of hardware and its parameters. The chapter contains informations about the development background of TSM570 controllers and its parameters. Furthermore, the section discusses the properties and capabilities of the TSM570 kit and peripherals used during the development of the RPP library.

2.1 Controler TSM570LS

Texas Instruments has created the family TSM570 of microcontrollers to accelerate the development of secure control applications. The series modules are designed to simplify the development of embedded controlling systems and to provide a low redundancy level certificated at Safety Integrity Level 3 (SIL3) [TD15]. Also, the modules are certified by TÜV NORD to ASIL D. TÜV NORD is internationally recognized as independent assessor of quality and safety.
Currently, Texas Instruments is distributing the TSM570LC and TSM570LS groups. The TSM570LS development kits features Cortex-R4 processors, designed for critical safety operations with two 32-bit cores. Cores works in Lockstep mode with ECC-Protected Caches [Ins15]. Also, the kits have a built-in flash memory with a capacity from 1 to 2MB, either with ECC security. A 128 or 160 KByte of RAM with ECC.

2.2 Development kit TSM570LS31

The TMS570LS31 development kit, introduced by Texas Instruments, is a comprehensive development platform, with detailed documentation and manufacturing background. A block diagram of the microcontroller is shown in the image 2.1.

The development kit is equipped with 10/100Mbit network interface with

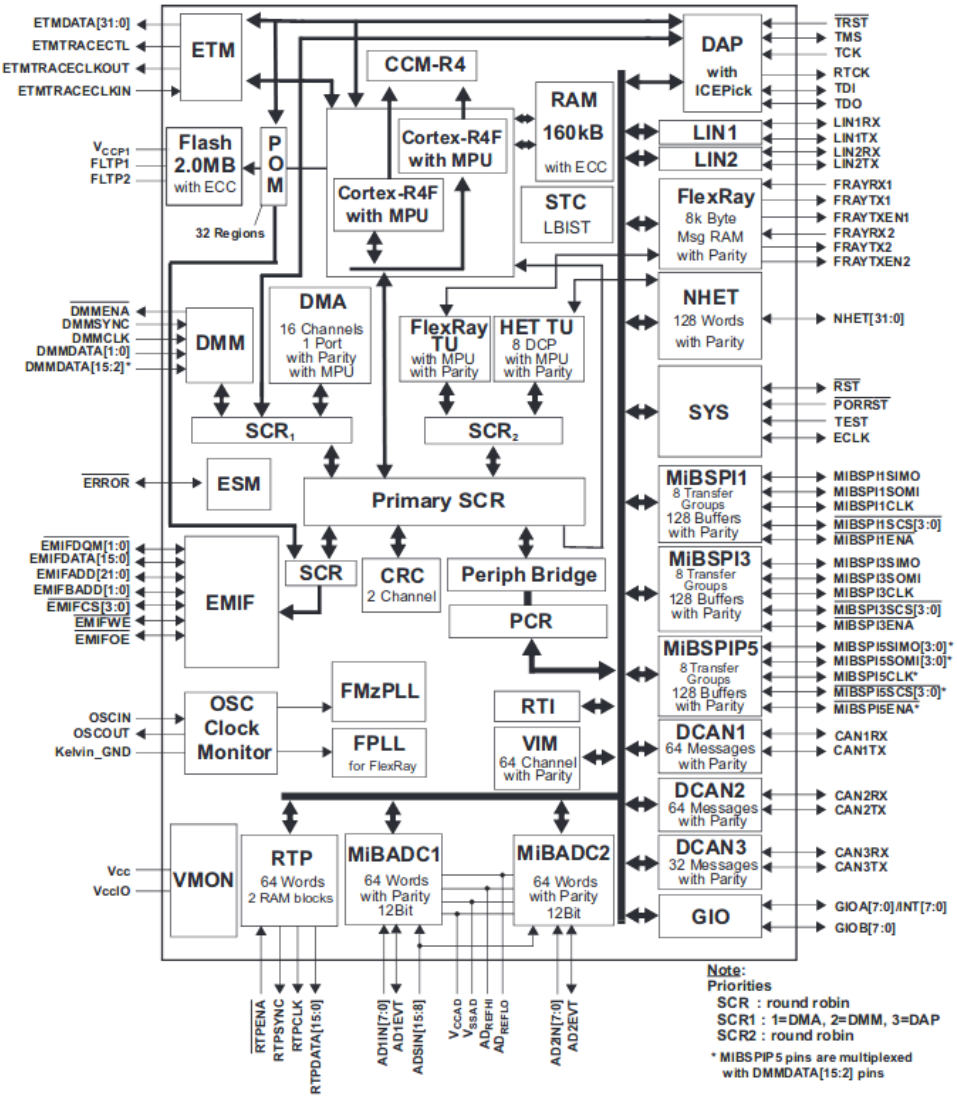


Figure 2.1: Functional block diagram of TSM570. Taken from [Ins15]

PHY DP83640 chip, external JTAG connector, UART, and 5-12V power supply. The kit is also equipped with a programmable High-End Timer (NHET). The timer is controlled by reduced number of instructions specially defined for peripheral timers and is brought to the I/O port. NHET can be

used for width modulation and is generally suited for actuator control with complex and accurate time pulses. The board is equipped with a High-End Timer Transfer Unit (HET-TU) to transfer data to/from the main memory. Unit checks data and provides errorless transport.
The microcontroller TSM570 has two 12-bit AD converters with 24 input channels, of which both converters can share up to eight channels. Also, the converters are equipped with their 64-word RAM protected by a parity bit. The Direct Memory Access Controller (DMA) has 32 requests and 16 channels. The DMA also has parity protection on its memory. The DMA unit allows you to read and write data from memory without CPU interaction. This approach saves CPU time to more critical operations. A Memory Protection Unit (MPU) is part of the DMA to prevent error during data transfer.
The TSM570 chip contains the CORTEX-R4 processor, which has two identical cores that process the same instructions. The instruction results are compared at each step to identify the possible error and warn the user or main application. To avoid common mode impacts, the instructions for the processor are processed with axially symmetrical cores. A 5-cycle delay occurs at the input of the test core. The output from the main core is also delayed, but after processing. So the CCM-R4 Compare unit gets the same instruction process by both cores at the same time. As shown in the diagram 2.2.
This CORTEX-R4 processor architecture enables high diagnostic capability

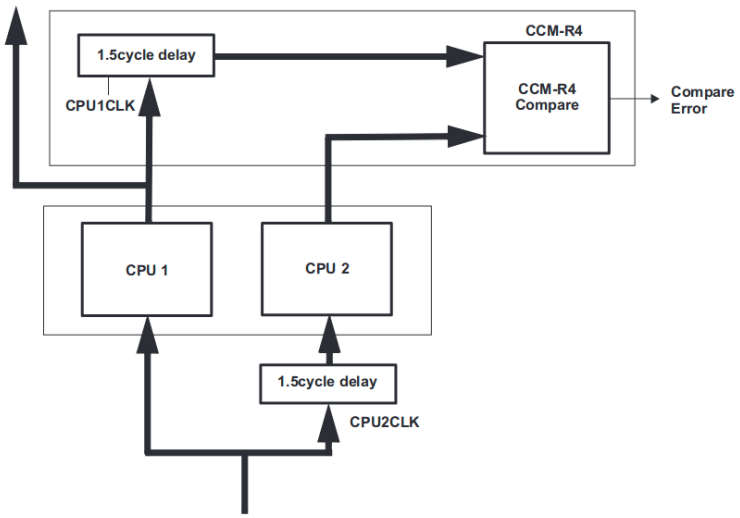


Figure 2.2: Safety redundancy scheme of TSM570. Taken from [Ins15]

by comparing the results of both cores in each processor cycle with only a small delay. In case of error detection, the system can run the procedure with minimal delay and put the board in emergency mode. This approach may reveal a large percentage of hardware errors caused by the processor.
Another possible source of hardware errors, is system memory RAM or Flash. The RAM is equipped with ECC on the development kit. Error-correcting code memory is a security feature, that allows fixing single errors within a single 32-bit word. The method can detect double errors, but can't repair

them. Multiple errors within one security word can't be detected or repaired. The CORTEX processor, installed on the development kit, is equipped with a self-test controller (STC). The STC system is composed of an LBIST module to check CPU during initialization. Another part of STC is PBIST module, that is used to test both RAM and peripheral RAM. All peripheral blocks are equipped with their memory. Peripherals with memory protected by parity bit are MibSPI, FlexRay, MibADC, CAN, and NHET. The Memory Protection Unit (MPU) is used to protect memory and monitor access to program memory or share memory. The LBIST and PBIST tests are designed to save the processor time by attempting a software test. Another available static memory test is the Cyclic Redundancy Check Controller (CRC). A 32-bit MCRC unit enables the checksum test in the memory. The unit provides memory tests on four channels.
All of the above security features are designed to ensure maximum hardware and program reliability. The development kit is designed to create critical safety applications. The introduction of the tested kit allows faster hardware and software development. That allows the customer to reduce costs and accelerate the delivery of the final product.Then, the kit base can be used to control or monitor a device safely. Kit could be used in the automotive, aviation and manufacturing industry.

2.2.1 Serial interfaces

The TSM570LS31 board provides two serial communication peripherals. One of the communication peripherals is intended solely for SCI communication. The second of the peripherals is SCI/LIN, which means that it is shared and the user can choose which mode will be used. The core of the module is SCI. SCI hardware features are enhanced to achieve LIN compatibility. The SCI module on the development board is an universal asynchronous receiver/transmitter. The SCI can be used to communicate over the RS-232 port or through the Keyword Protocol 2000, used in the automotive industry. The LIN communication standard is single-master/multiple-slave with multicast transmission between network nodes [TI18].
The SCI periferal supports full-duplex and half-duplex operations with non-return-to-zero (NRZ). The NRZ is technology to transmit logical 0 or 1. Zero bit is presented by voltage change and 1 bit is represented by staying on previos voltage level. Both interfaces have configurable frame of bits from 3 to 13 bits per character. Main part of the character is data word length from one to eight bits. The interfaces can also set address, parity and stop bits. Communication baud rate suppors maximaly 2^{24} speed.
The serial port is widely used on the board during development due to the ease of commissioning and the ability to report the error. This information can significantly facilitate the development of kit-based programs.

2.2.2 Interrupt handling

The interrupt is used to perform a simple routine outside the main thread of the program. The interrupt serves to inform the main program of an incoming message or other external events. There are several rules for creating interruptions. Because the execution of the main thread is stopped when it is interrupte, it is necessary that the interrupt is non-blocking. Furthermore, the interruption should be as short as possible and ideally not interfere with the critical sections of the program. Mostly, during the interrupt, it is only to give a semaphore to which the main threads code responds.
In the processor, the vectored interrupt manager (VIM) holds the interrupt handler. The manager assists in prioritizing and managing many interrupt sources present on the device. It goes to set the interrupt bit and the CPU switches from performing regular program stream to interrupt service routine (ISR).
The ARM processor provides two interrupt approach. Firstly fast interrupt requests (FIQs), and standard interrupt requests (IRQs). When an interrupt is received, the processor disables any other interruptions and evaluating the incoming. The software supports four interrupt processing options for evaluation.
The first method, useable for older versions of the code, is based on previous designs of the Cortex-R4 processor family. The method reads interrupt vectors from memory positions 0x18 (IRQ) or 1x1C (FIQ) [TI18]. Subsequently, the main routine reads the registry offset to determine the source of the interrupt. This approach is recommended only for ported programs from older microcontrollers such as TMS470.
The second, more advanced, way to handle interrupts is to automatically provide the applications vector address when registering an interrupt. However, before enabling the interrupt itself, it is necessary to initialize the interrupt vector table (VIM RAM). If VIM receives an interrupt, then it loads the ISR address from the table and stores it in the IRQVECREG or FIQVECREG interrupt handler respectively. After that, the standard routine, described in point one, may use the jump to the address filled by the VIM module.
The third method of hardware interruption is direct sending to the ISR. This way of interruption is possible only for IRQ, not for FIQ. This specific interrupt function must be explicitly enabled in the vector enable (VE) register bit. Then it is possible for the operator to interrupt reading the address directly from the VIM interface and not using the standard operation via memory address 0x18.
The last approach uses a program-controlled interruptions. The application uses only its tools and does not use VIM elements. This case is possible, but it is necessary for the application to fill in the request for the interruption in the source and also to switch off the corresponding part of the VIM. This approach can be accomplished by modifying the IRQVECREG registry or writing 1 to INTREQ to the appropriate position [TI18].

2.3 EMAC and PHY

The Ethernet adapter is created from the Ethernet Media Access Control (EMAC) and physical layer (PHY) on the development kit. EMAC is used to manage both received and outgoing data. The PHY chip is used to transfer data over the media and is realized on the kit by the DP83848Q circuit. The synchronous parallel interface serves to transfer information between the EMAC and the chip. This interface has two variants: Media Independent Interface (MII) and Reduced Media Independent Interface (RMII). The MII variant works at 25MHz and uses a 4-bit bus. The second variant works at twice the frequency.
Block scheme of adapter is in figure 2.3.

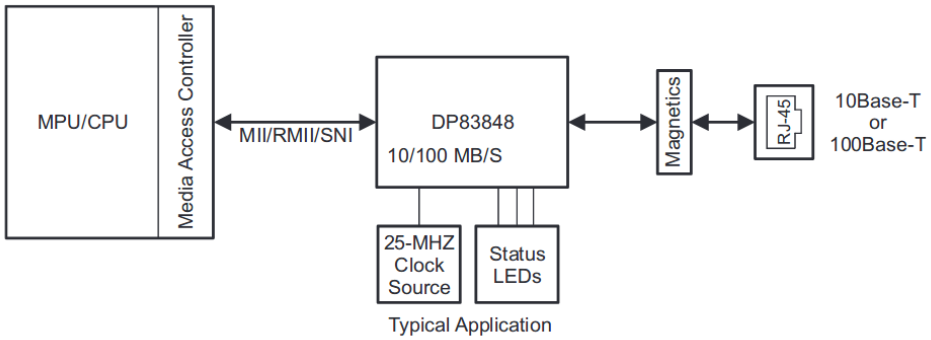


Figure 2.3: EMAC and PHY structure. Taken from [Ins13]

2.3.1 EMAC

The Ethernet communication is on the development kit TSM570 realized by three parts, the EMAC control module, the EMAC module, and the MDIO module. The EMAC control module serves as the primary communication interface between the EMAC processor core and the MDIO module. TSM570's standard interface must implement different transmission rates and methods with respect to standard IEEE 802.3 MII [Ins13]. This standard defines the standardized interface to communicate with different connected PHY chips. The method must implement both full-duplex and half-duplex and including collision CSMA/CD protocol in half-duplex communication.
The structure of the Ethernet module is presented in the diagram 2.4. The EMAC control module is used for communication of the EMAC module via

Host interfaces to the CPU. The DMA bus is used for direct access to internal and external memory. Advantage of DMA is that memory access is done without CPU requirement. The internal CPPI RAM module is mapped to the same extent as the EMAC and MDIO control registers. The MDIO bus is used to communicate and set up a PHY chip. Both modules of EMAC and MDIO combine interrupts into four interrupt signals. The Vectored Interrupt Manager processes all signals from the combiner and sends them to the CPU.

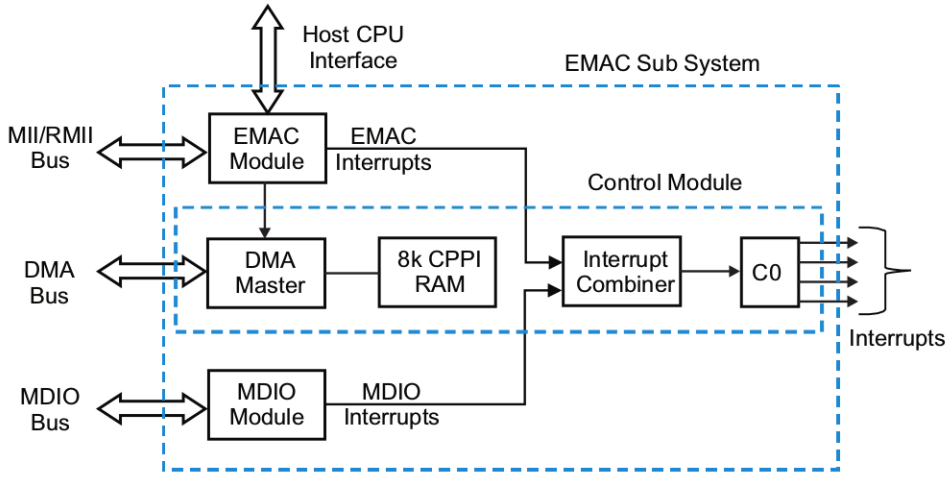


Figure 2.4: EMAC a MDIO interrupt schema. Taken from [TI18]

Media Independent Interface

The MII bus is described in the 2.5 schema. The pins are used for communication between the EMAC module and the MDIO module with PHY chip. MII_TXD [0-3] is a four pins data channel used by EMAC module to communicate with the PHY chip. MII_RXD [0-3] is used to send data from PHY to EMAC. Both bus lines use parallel transmission. The transmission rate is controlled by clock pins MII_TXCLK and MII_RXCLK. Nominal frequencies are 2.5MHz and 25MHz, which is used to control the communication rate of 10 and 100Mbit/s [TI18].
The MII_TXEN is used to control communication over MII_TXD[0-3]. Indicates the intention of the EMAC chip to transmit. The MII_RXER signal is used to transmit an error during data transmission along with another MII_RXDV pin to control the data transfer from the PHY chip [TI18].
The MII_COL and MII_CRS signals are used to control half-duplex traffic. MII_COL is used to detect collision over the data pins and pin MII_CRS is asserted if communication is performed on shared media from at least one source.

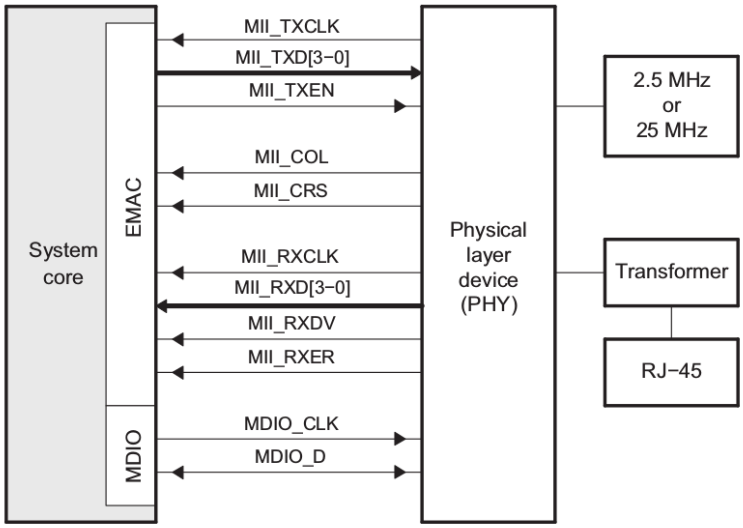


Figure 2.5: MII connections. Taken from [TI18]

Reduced Media Independent Interface

RMII communication is very similar to MII. For both RMII_TXD [0-1] and RMII_RXD [1-0] communication pins, the data pins are reduced at half to MII. At the same time, the transfer rate is doubled for RMII. Furthermore, the CRS and DV pins are multiplexed into RMII_CRS_DV [Ins13]. In the schema 2.6 are all RMII communication pins.

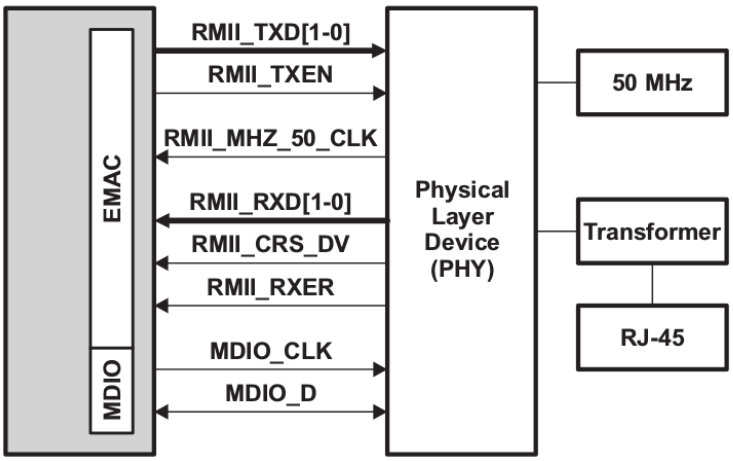


Figure 2.6: RMII connections. Taken from [TI18]

2.3.2 Management data input output

The EMAC module uses the MDIO module to access up to 32 PHY registers. MDIO uses the MDIO_CLK clock controlled by the EMAC module, because communication with the PHY chip is initiated by the EMAC module which is initialized first by CPU. The MDIO interface supports up to 32 PHY devices [Ins13]. With the special command set for controlling MDIO communication, it is possible to access just one particular register in the PHY chip. With it, the communication protocol allows both read and write from the active register using the two-way pin MDIO_D. The communication speed is determined by the maximum clock signal frequency that is standardized at 2.5MHz. Schematics 2.7 describes the structure of the data message sent over the channel.

MII MANAGEMENT SERIAL PROTOCOL	<idle><start><op code><device addr><reg addr><turnaround><data><idle>
Read Operation	<idle><01><10><AAAAA><RRRRR><Z0><xxxx xxxx xxxx xxxx><idle>
Write Operation	<idle><01><01><AAAAA><RRRRR><10><xxxx xxxx xxxx xxxx><idle>

Figure 2.7: MDIO frame structure. Taken from [Ins13]

MDIO message structure

- Start - Serves to initiation message and contents of <01> sequence. That sequence converts channel into active mode from idle.
- Opcode - Indicates operation mode for read is sequence <01> and for write <10>.
- PHY Address - Address space of five bits to select one of the connected PHY chips.
- Register Address - Address space to select specicfic register at PHY.
- TA (Turnaround) - Idle bit of time between register address and data field. PHY device must confirm access by adding 1 or idle bit. In case of reading, PHY drive the register after TA.
- Register Data - requested or sent data.

On figures 2.8 and 2.9 are presented time relationship between time link MCD and MDIO data link.

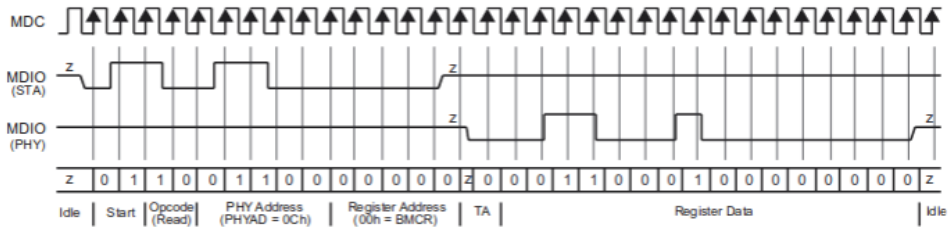


Figure 2.8: MDIO read frame. Taken from [Ins13]

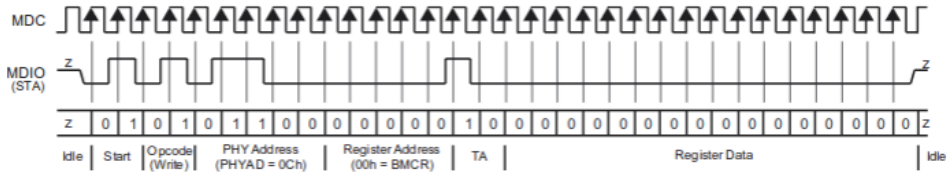


Figure 2.9: MDIO write frame. Taken from [Ins13]

2.4 ARM CORTEX–R4

ARM processors are now spread in most consumer electronics. The processors have established themselves through low power consumption, especially in mobile phones and tablets. Due to the relatively high computational power and small code size, the processors also have a high degree of embedding in devices. The development of ARM architecture began in Britain in the 1980s. The ARM CORTEX-R processor family has a 32-bit structure and is optimized for hard real-time and safety-critical applications. The Cortex-R4 processor was launched in 2011.
The CORTEX-R4 kernel contains the 32-bit ARM and Thumb2 instruction set [Lim11]. While the kernel is running, it is possible to work with both instruction sets according to the program code requirements. The use of both sets makes it possible to reduce the size of the compiled code with a slight reduction in computing speed. The processor has 1.25MB of Flash memory available on the TSM570. The exact structure of CORTEX-R is patented by ARM Holding.

Chapter 3

Descriptions of software parts

In this section will be described used the software. I was implementing a part of the Rapid Prototyping platform to provide external mode. Rapid prototyping platform is described in section 3.2. To provide IP stack is used LwIP. Short summary about LwIP is in section 3.3.
The external mode is parts of Simulink. This product and its features are described in section 3.1. The final part of this section is a description of a basic ethernet standard which had to respect during implementation.

3.1 Simulink

Simulink is a product created by Mathworks as an add-on product to Matlab. It supports system-level design, simulation automatic code generation, graphical environment for modeling and test and verification of dynamic systems. It allows rapid prototyping of virtual models. The extensive pre-defined block library included in Simulink allows drag-and-drop operation. The user is able to create a model that otherwise require hours of implementing. Simulink supports linear and non-linear models with continuous, sampled or hybrid time. These features allow creating models with minimal effort [Soc19].

3.1.1 Embedded Coder

The Embedded Coder is used to generate fast, small, and memory optimized code in C/C++. Coder supports most of the series-built embedded processors on the market. Compared to Simulink Coder and Matlab coder, it includes enhanced optimization for embedded devices [Mat19]. The TLC files generate a program, based on simulation created at Simulink.

TLC Files

Files with extension *.tlc* are used to generate C/C++ code. They are enhanced with additional commands for Coders created by Mathworks. Currently, TLC files are the only way to compile blocks and targets [ac18b].
TLC files can be divided into two groups. One is used to define the code of each block in simulation and is used according to the simulation. The second group is the files created for the target hardware. This target consists of several files and defines main and compilation procedures.

3.1.2 External mode

The external mode is great Simulink feature for rapid prototyping. The external mode is used to exchange data with the generated code from the simulation. Simulink add-on Embedded coder or Simulink coder are used to generating C or C++ programs respecting the model. Coders are using TLC files to compile programs. These programs could be built for many platforms. Embedded coder also allows generating code for embedded platforms like TSM570HDK used in this thesis. One of main advantage and that is both side communication.
External mode allows exchanging data between Simulink instance and tuned model [ac18a]. The tuned model can send data from scopes existing during compiling of simulation. To properly tune the parameters of the model is very useful to saw internal values in the model. Scopes allow tune model rapidly and without multiple long recompiling.
The second function of external mode is to send commands to set up constants and parameters of the tuned model. Order is sent from Simulink to target. Target calls function, that set the new parameter in a defined part of memory. Tuneable parameters have to be set before the compilation of code. This possibility of change allows tuning model without recompiling and reuploading. These updates save many time prototyping engineers.

During generating code in embedded coder is necessary to select checkbox with external mode. The external mode is created only for the prototyping purpose and doesn't make sense to use it in the final build. Than compilator add low priority level thread used to communicate with Simulink instance on a computer.

3.1.3 Design of external mode

External mode communication is based on client/server architecture. Simulink instance running on a computer works as a client and transmit requests to the target. The server responds by executing incoming requests like accept parameter changes or upload signal data.
Simulink external mode use layer system. That means both engines Simulink and model core allow to add independent transport layer. Transport layer respective transport layers on both sides of the medium must support functions to format, transmit and receive messages and data packets. This design allows us to create and use different transport layers. Targets like GRT and ERT supports TCP/IP and RS-232 serial communication [ac18a]. Target RTWin supports shared memory communication. The external mode main type of communication TCP/IP and Serial link are described in sections 3.1.3 and 3.1.3.

TCP/IP

TCP/IP connection uses ethernet network to send information about server and target. Files with built-in transport layer implementation for client and server are *rtiostream_interface.c* and *rtiostream_tcpip.c* [ac18a].
To connect Simulink client to target is required to set some parameters. To provide TCP/IP connection is specified three parameters. These parameters are detailed described lower.

- Target network name: network name of the computer with an external program. By default, this is the computer on which the Code composer product is running. The name has to be defined like string delimited by single quotes, such as 'target' or IP address also delimited by single quotes, such as '148.27.151.12'.
- Verbosity level: number describing the level of detail of the information displayed during transfer. Value 0 means no information displayed and detailed information are represented by value 1.

- Port number: default value is 17725. The user-defined port number must be valued between 256 and 65535 to avoid a port number conflict.

Arguments can be delimited by white space and must be specified in the following order:

```
<TargetNetworkName> <VerbosityLevel> <ServerPortNumber>
```

Serial

The external mode can also use RS-323 serial link. Files used as transport layer are *ext_serial_transport.c* and *rtiostream_serial.c* for client and *ext_svr_serial_transport.c* and *rtiostream_serial.c* to compile server.
Also, the serial connection has parameters to set up a connection [ac18a]. These parameters are described in detail as follows.

- Serial port ID: The port ID of the host. This must be specified as string or integer. Typical structure of port ID is */dev/ttyUSBX*
- Verbosity level: this number has the same meaning as in TCP/IP connection. Value 0 means no information and detailed information are displayed by value 1.
- Baud rate: Specify an integer value, the Default value is 57600.

Arguments parsing is same as in TCP/IP. Delimited by white space and must be specified in the following order:

```
<VerbosityLevel> <SerialPortID> <BaudRate>
```

3.2 Rapid prototyping platform

The RPP is developing on the department of industrial informatics at Czech Technical University in Prague under leading Michal Sojka and Michal Horn.

The platform was initially developed for Texas Instruments TMS570 safety microcontroller. During contract with Eaton Corporation was platform ported to other board such as RM48 HDK and TMS570 HDK development kits. The platform software consists of code generation target - Simulink Embedded Coder, low-level C library and testing and debugging tools for hardware and software parts [MSH17].
The RPP is working under FreeRTOS which is necessary to develop non-trivial applications. The FreeRTOS is a simple real-time operating system. The core has minimal ROM and RAM overhead and typical binary are under 12k bytes. Another big advantage is thas FreeRTOS is distributed under open source MIT license. The license allows using core commercially.

3.2.1 Architecture

The RPP Library is structuralized to 5 layers described in figure 3.1. The structure was following several rules. The Top-down dependency. A lower layer depends on any of a higher layer. Every layer implements a unified interface, so upper layers depend on the lower layer interface.

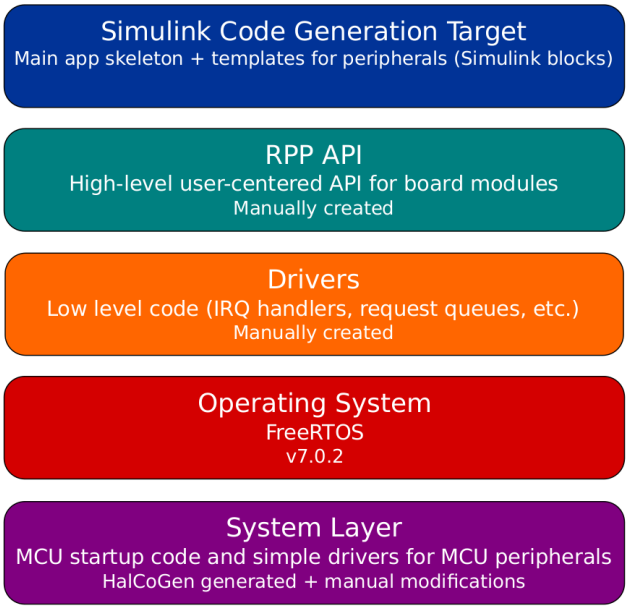


Figure 3.1: Architecture of RPP library. Taken from [MSH17]

System Layer

This layer contains system functions and data definitions. The system contains interrupts mapping, clock definition, MCU start-up, and self-tests. Main parts of this layer were generated by HalCoGen tool.

Operating System layer

The layer containing the FreeRTOS or different operating system. The system in this layer could easily change. For example, is possible to compile the library with the POSIX version.
The currently used version of FreeRTOS is 8.2.2. Actually released version of FreeRTOS was 10.2.1. Version isn't backward compatible with FreeRTOS V8.x.

Drivers layer

This layer contains implementation for control and service all peripherals. Functions in this layer provide IRQ handling, threats management and software queues. Actually supported peripherals are ADC, CAN, Ethernet, LOUT, DAC, and DIN [MSH17].

RPP Layer

The top layer of the library provides encapsulation of Driver layer and simplify manipulate with peripherals. Function in this layer unifies initialization, sending, receiving and canceling the peripherals.

3.2.2 Software test

Application *rpp-test-sw* contains tools for testing and control of the entire board and functions of peripherals. Package contains own binary of the RPP

Library and all headers and other files necessary for building and downloading the application.
The testing suite has own command line. After downloading code to target open through RS-323 serial. Basic serial setup is 115200-8-N-1 and possible commands could be listed by writing *help*.

3.3 Lightweight TCP/IP stack

LwIP is a small implementation of the TCP/IP protocol suite. LwIP is an independent project developed initially by Adam Dunkels at the Computer and Networks Architectures (CNA) lab at the Swedish Institute of Computer Science (SICS) [com19c]. The community is now actively continuing in development and maintenance under a BSD-style license.
LwIP is focusing on providing full-scale TCP with limited memory resources. That makes the library suitable for embedded systems with at least 60 kilobytes of free RAM and room in ROM for around 40 kilobytes of code.

Features

- IP - supports IPv4 and IPv6 over multiple network interfaces. None of the kits supports more than one interface.
- ICMP - the protocol used to maintenance and debugging. The function of the protocol is used in the newly implemented ethernet debugging module as part of RPP.
- TCP - with congestion control, fast recovering and retransmitting and sending SACKs, RTT estimation [AD19]
- UDP - including experimental UDP-lite extensions, actually not used in RPP.
- DHCP - include AutoIP and DHCPv6. DHCP support is planned into RPP but not implemented yet.
- Many other functions like Neighbor discovery (ND), IGMP, PPPoS, PPPoE and DNS. RPP actually not using these functions.

Applications

LwIP also provides some complete applications for easier use. These applications include HTTP server, SNT protocol for sharing time and Iperf.
This module wasn't used to provide stress tests of aggregation LwIP with RPP because aplication are too large to include them with LwIP.

3.4 ISO-OSI model

An International Organization for Standardization introduced in 1984 reference model The Open Systems Interconnection (OSI). The model has served as most basic computer networking elements. The original purpose of development was to provide a basic set of design for manufacturers. This set enables communication between different products of various branches. The OSI model introduced hierarchical layer system and functions necessary to communicate device-to-device.
The layered approach has a few advantages. Separating networking functions simplify the code of each layer. Network problem can easily be handled and solved. The modularity of the system also allows extensibility and easy implementing add-ons into the system.
The OSI model contains seven layers. Each layer has well-defined purposes and functions providing protocols. Every layer has a different level of abstraction and should be created where a different level of abstraction is needed [Mil19].
The seven OSI layers are defined :

- Physical - Provide communication over medium
- Data link - Provide control of transmission error and routes over a local network
- Network - Routes the information between different networks
- Transport - Provides end to end communication threads
- Session - Handles problems which are not communication issues
- Presentation - Provides right interpretation of transmission data
- Application - Provides services for applications

In RPP applications are used physical, Data link, network, and transport layer as they are defined in the OSI model. If it's necessary session and presentation layers are used only as part of applications running over the transport layer. One of these applications is the external mode for Simulink.

3.4.1 Physical layer

The physical layer is designed to transmit raw bits over the communication medium. The design issue has to make sure that one side sends a 1 bit and receiver interprets it as 1 bit. The physical layer has to solve the problems like voltage on data representation, timing in microseconds to send data and half-duplex or full-duplex communication [Mil19]. On physical layer is also defined initial the transmission and ending sequence. The physical layer is designed by the manufacturer. He also establishes a number of used pins and used chips and components.

3.4.2 Data link

Data link layer uses data frames to provide transmission error-free data stream. The main task of the layer is using function fiding errors in raw data. Data are split into frames and then used functions like CRC. After processing frame the acknowledgment frames sent back by the receiver. In case of detecting error are sent request mostly resent the whole frame again. The header of the layer contains a source and destination MAC address [nSTB14].

3.4.3 Network layer

The network layer controls communication over networks. The main role of the network layer is to determine how packets are routed to the destination. Also providing congestion control and accounting. Accounting is also important for selecting the path of the packet with the best metric. Metric mostly describes the time of traveling through this node.
The route table contains destination networks, gateway, metric and source interface. The table has static and dynamic records. Dynamic records are updated after receiving the synchronization packet reflecting the current

network load. This approach also allows connecting heterogeneous networks. This layer contains the IP protocol. All routers operate at this layer.

3.4.4 ICMP

Internet Control Message Protocol is the protocol of the network layer. The protocol serves to report errors and providing information about Ip packet processing. Typical error reported by ICMP is a host isn't reachable or service isn't available. Widely used ICMP tools are ping or traceroute. Structure of the ICMP header is presented in figure 3.2. Content of the header:

Table 1-2. Internet Control Message Protocol - Basic Headers

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Version				IHL				TOS/DSCP/ECN										Total Length													
Identification																		Flags		Fragment Offset									
Time to Live								Protocol										Header Checksum											
Source Address																		Destination Address											
Type								Code										Checksum											

Figure 3.2: Basic ICMP header. Taken from [com19b]

- Version - It should be 4.
- Internet Header Length: The length of the header in words.
- Type of Service - This should be set to 0.
- Total Length - Total length of the header and data.
- Identification, Flags and Fragment offsets - Inherited from the IP header.
- Time To Live - Number of hops this packet will survive.
- Protocol - This should always be 1.
- Header Checksum - Transmit error detecting part.
- Source Address - The source IP address from whom the packet was sent.
- Destination Address - The destination address of the packet.

Ping

Ping is one of the most famous network tools. Ping is used to testing if a network works properly or If is server reachable. Client part of command

consists of sending a request to the target. If a request reaches the target then is sent reply packet. In case that target is unreachable will router of network belong to the same net as target send response ICMP Destination host unreachable. Over the internet is a well-respected rule to send the response on ping request. Ping echo/reply header is presented on figure 3.3.

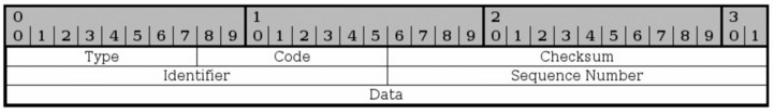


Figure 3.3: Echo/reply header. Taken from [com19b]

3.4.5 ARP

Address Resolution Protocol in the network protocol used to find out or translate IP address to hardware address (MAC). MAC addresses are used to address over a local network. The sending device must know the target MAC address. If ARP cache doesn't contain record about the target IP address. The device sends ARP broadcast request to all devices in the local network. Requests contain information about who is asking and which device should response. All devices on a local network see the message. The only device with target IP will respond the answer and reply ARP message containing its MAC address. After delivering has the device enough information about the target and can send a packet with data.

3.4.6 Transport layer

The transport layer is responsible for delivering data from streams to the right application. The layer has to create sockets and set up source and destination address. Socket also have port or ports number. Ports are on the transport layer to a distinct incoming packet to right application to process them. Before a client can connect to the defined port, the server has to bind port and listen.

TCP

The TCP is the most used protocol of the transport layer used in TCP/IP stack over the internet. This protocol is connection-oriented [Rou19]. That means

establishing and maintaining the connection, throughout communication of applications on both sides. To proper cancel of the connection is necessary to send a sequence of terminating packets.
Another property of TCP is flow control. That meant to provide error-free data transmissions on each stream and retransmission of dropped or wrongly transferred packets [Ass19a]. To provide retransmissions, TCP must send an acknowledgment packet to the source. Acknowledged are packets individually or in a defined window. That window describes how many packets are acknowledged with one packet. In case of an error in the window, all the packet in this window is resent. Windows are useful during downloading a large amount of data.

Initialization of connection

To establish TCP connection is used defined sequence well known as the three-way handshake. This process serves to negotiate the sequence and acknowledgment numbers. After the three-way handshake is possible to start the session. Individual steps are described near [Ass19b].

- Step 1: A client wants to establish a connection with a server. The client sends a segment with Synchronize Sequence Number which informs the server that the client is trying to start communication and with what sequence number it starts his segments.
- Step 2: The server responds to the client response with both numbers Sequence and acknowledgment. Acknowledgment signifies the response of segment it received and a server sequence number which informs where server it is likely to start the segments.
- Step 3: The client acknowledges the response of the server, and they both establish a reliable connection. Now can start actual data transfer.

Terminating of connection

Process of termination is very similar to the initialization sequence [Ass19b]. This four-step process is described below:

- Step 1: The client that wants to close the connection sends a packet with the Finished flag set to 1.

- Step2: The server receives the packet and acknowledges it with the Acknowledgment packet.
- Step 3: The server sends its own segment with the FIN flag set to 1 to the client. To close connection on both sides.
- Step4: The client acknowledges the server's FIN segment and closes the connection.

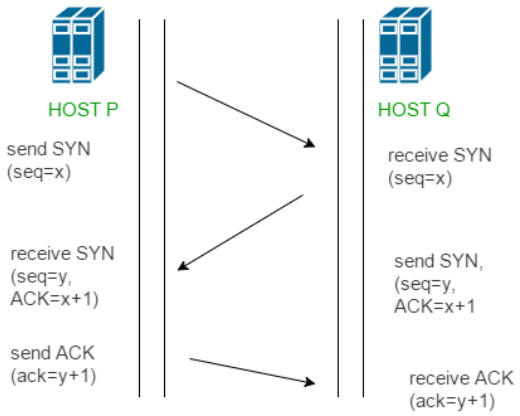


Figure 3.4: Initialization of TCP connection. Taken from [Red19]

UDP

This protocol is used to sending a larger amount of data to one or multiple resources. UDP also doesn't support acknowledgment of delivering packets. That allows using bandwidth for more data. The protocol also hasn't any sequence number and packet could come in a different order then was sent. UDP protocol normally provides higher throughput and shorter response time. These properties make UDP suitable for multimedia like video streaming, IP calls, and some online games.

Chapter 4

Realization

This chapter describes several steps, that have been completed. It was getting familiar with the libraries and running the RPP library blocks in Simulink and compiling the test program. These steps were followed by the network setup and configuration of all elements. The ping command was implemented to test the correct configuration of the network. Subsequently, Iperf command was implemented for stress tests of Ethernet peripheral. During the workload on the periphery, there occur program errors, and it was necessary to look for a problem and modify the memory management. Errors in the system also required a little update on the main Ethernet driver. After all adjustments, connection speed and stability were increased. In the end, I started to work on the ERT target for Linux.

4.1 Test of RPP blocks in Simulink

One of the first steps was to put into operation the existing RPP library in Matlab version 2018b. It is necessary to run the *rpp_setup.m* script located in the *rpp* folder to compile the library into the format required by Simulink. This script compiles all peripheral blocks supported by the RPP library and combines them into the Simulink library [MSH17]. The script also sets the Matlab path to the root library. It is necessary to set the path to the RPP compiler for the script to work properly. The RPP uses a compiler arm_5.1.1 created by Texas Instruments. It is installed by default in the */ti/ccsv5/tools/compiler/arm_5.1.1* folder.
After compiling the library, we can use one of the test schemes, for example,

4.1. The final step for running the program is compiling the code form the blocks using the TLC files. Compilation can be done by using the Embedded codec with the appropriate settings. Then, the code is downloaded to the microcontroller and started.

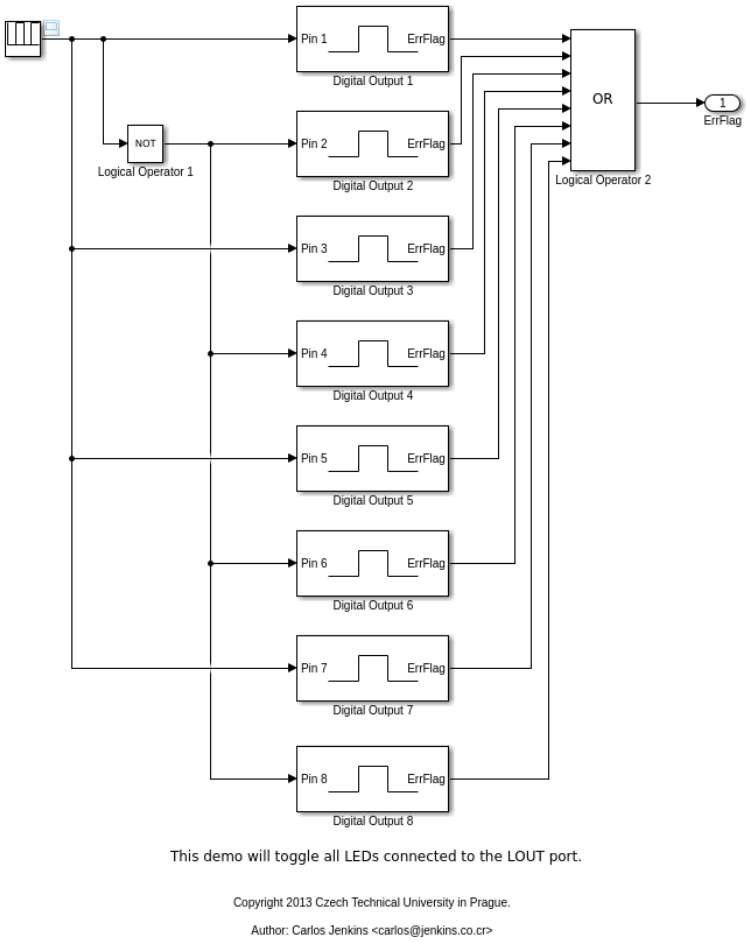


Figure 4.1: Test of LED blink blocks

4.2 Network setup

For the purpose of communicating with the development of the TSM570 platform, a simple network was created. Network is presented in figure 4.2. Configuring a computer for simulink development is not difficult if the router is running a DHCP server. Then, on the computer side, the IP address, netmask and default gateway are obtained. Which is sufficient to communicate with the board. Another option is to assign static data, based on the MAC address

of the host computer. This approach was also used in the project. The computer configuration is listed in table 4.1.
It is possible to turn on the DHCP server during router configuration. However, it is necessary to set a static IP address for the TSM570. Next, the netmask value and the IP address of the router itself must be defined, which then serves as the Default Gateway for the remaining devices. Particularly noteworthy is GW value for setting up the microcontroller.
It is necessary to edit the values in the appropriate header files to set up Ethernet on a microcontroller. The RPP-controlled microcontroller has a defined MAC address in the *eth.h* header file. The header file has defined values for the precompiler as follows.

```
#define RPP_MAC_ADDR                { 0x12, 0x34, 0x56,
0x78, 0x9A, 0xBC }
#define RPP_IP_ADDR                 0x0A235F19
#define RPP_NETMASK                 0xFFFFFF00
#define RPP_GW                      0x0A235F01
```

The values are defined in hexadecimal values and correspond to the values given in tables 4.1, 4.2 and 4.3. However, it is necessary to define the corresponding STATIC_IP_ADDRESS value defined in the LwIP header file in *lwipopts.h* for proper operation. Since the current implementation of RPP does not support DHCP request, the stack cannot get an IP address automatically assigned.

Device	IP	IP (hex)
PC	10.35.95.157/DHCP	0x0A235F9D
TSM570	10.35.95.25	0x0A235F19
Router	10.35.95.1	0x0A235F01

Table 4.1: IP addresses of network interfaces

Device	Maska	Maska (hex)
PC	255.255.255.0	0xFFFFFF00
TSM570	255.255.255.0	0xFFFFFF00
Router	255.255.255.0	0xFFFFFF00

Table 4.2: Mask of network interfaces

Device	GW	GW (hex)
PC	10.35.95.1/DHCP	0x0A235F01
TSM570	10.35.95.1	0x0A235F01
Router	DHCP	DHCP

Table 4.3: GW addresses of network interfaces

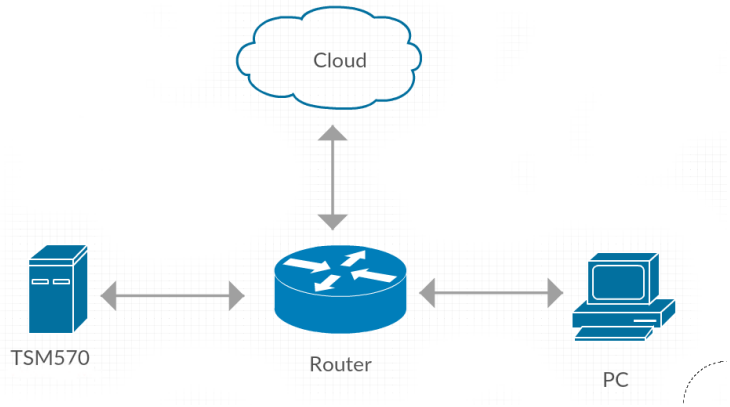


Figure 4.2: Network scheme

4.3 Ping implementation

The ping command has been implemented into the test program *test-sw*. The test program provides a command line for the user. The terminal sends individual commands through the serial link to the microcontroller where they are evaluated and the results are sent back by the link. For proper implementation of the new command, it is necessary to follow a prescribed structure.
The *commands* folder contains all terminal command files. There are also two files that provide data structures for all commands. The file *cmd.h* defines the array of *cmd_des_t* descriptors defined for commands. Each command must add its instance to this field. In addition, the *cmd.c* file contains imports of all the command files that are sorted by the target controller. At the end of the file is edited the *cmd_list_main* array and added the imported features again depending on the target hardware.
Each command is divided into a header file and the implementation itself. Therefore, the ping files are divided into the header file *cmd _ping.h* and the implementation in the *cmd_ping.c*. Header files usually have a simpler structure and text cmd_ping.h is not an exception. The file contains error status values and a data structure definition of the command.
The second file *cmd_ping.c* contains all the necessary includes. Furthermore, all functions are necessary for proper working of the command. At the end of the file is defined the structure of the command, including help and function pointer on the main function. The help of the function describes usage and parameters. The *ethping* has only one parameter the destination IP address. The command is shown in the example.

```
ethping [destininationping]
```

The main function of *ethping* command is *cmd_do_eth_ping*. This function provide initialization test, argument parsing and check, and then sets up sockets. When the socket is initialized, function starts the main loop and call *ping_send* function. If sending funtion returns positive value, then is started *ping_recv* function which measures time and expect ICMP request. Function *ping_prepare_echo* construct ICMP header and return message. Output of *ethping* in the case of error-free transfer is listed in example.

```
ping: send to 8.8.8.8 time=20 ms
ping: send to 8.8.8.8 time=19 ms
ping: send to 8.8.8.8 time=22 ms
```

Time out of *ethping* response is 1000ms. If it is excited, then program produces output iwhich is shown in example.

```
ping: send to 8.8.8.8 time=1000ms - timeout
```

4.4 Memory management

There were system errors during testing of the *ping* command. The FreeRTOS operating system allows you to use a few different ways to manage free memory. Memory management uses Heap that defines methods for allocating, freeing, and possibly accumulating free memory. FreeRTOS includes 5 different complex heap implementations [com19a]. At the same time, Heap with more excellent memory management requires the use of more complex feature code. This code increases the CPU load on memory management and, as a result, the CPU useful performance decreases. Another consequence is an increase in the ROM requirement for the compiled code. All 5 implementations are described below.

- Heap 1 - The simplest implementation does not support freeing memory. An error condition occurs when you try to release.
- Heap 2 - Features support freeing memory but do not allow any memory block connections.
- Heap 3 - Thread safety implementation. Use wrapped functions malloc() an free().

- Heap 4 - More advanced implementation of freed memory management. It allows the connection of neighboring blocks of freed memory and their splitting and redistribution.

- Heap 5 - Extension of Heap 4. Heap 5 connecting nonadjacent memory areas. Therefore, free memory is always processed as one block [com19a].

The Heap 1 implementation was initially used in the RPP library. This implementation, while using the Ethernet periphery, cause system crashes when trying to free up memory. Even the implementation of Heap 2 did not address memory problems sufficiently. Despite the possibility of freeing up memory blocks, the system was still failing. The memory was significantly fragmented by reusing malloc command. The malloc will be unable to allocate enough block of memory and fails. Heap 3 implementation does not provide any memory management enhancements over Heap 2.
The Heap 4 implementation solves the problem. That heap is actualy used in the RPP library, along with other adjustments, to achieve program stability even on the Ethernet periphery. In the original documentation, Heap 4 is recommended for memory management in case of periodic tasks, work with queues, traffic lights, mutexes.

4.4.1 LwIP memory management

Other memory adjustments were made directly in the LwIP library. The header file *lwipopts.h* allows setting the memory options for the threads created by the LwIP stack. The file contains two parts to edit. Parts that deals with memory are TCPIP thread and Memory Options. Only TCPIP_THREAD_STACKSIZE has been reset to 1024 in the TCPIP thread section. Part Memory Options contains these parameters.

```
#define MEM_ALIGNMENT                   4       /* default 1    */
#define MEM_SIZE                        65536   /* default 1600 */
#define MEMP_NUM_PBUF                   48      /* default 16   */
#define MEMP_NUM_TCP_PCB                16      /* default 5    */
#define PBUF_POOL_SIZE                  128     /* default 16   */
```

Resetting these values has led to the stabilization of the Ethernet connection and the speed of communication.

4.5 Iperf implementation

The Iperf is a widely recognized and used internet connection stress test tool. The test is based on server-client communication and consists of sending a large volume of data from the client to the server and back [ac19]. The Iperf server part of the communication is implemented on the microcontroller.
Like the *ethping* command, the *ethiperf* command is implemented as part of a test program. So it is divided into two files: *cmd_iperf.h* and *cmd_iperf.c*. The implementation includes the main function *cmd_do_eth_lwiperf* which, after testing the correct initialization of the Ethernet peripheral, opens port 5001. Iperf communication is addressed by default to port 5001. The function *lwiperf_accept* is listening on port. The function takes care of receiving data and using the last defined *lwiperf_recv* function. This function takes care of sending data back to the source. It is necessary to translate and download the test software code to the TSM570 microcontroller to run the test. Then it is necessary to initialize the Ethernet peripheral using the command *ethinit*. After initialization, the Iperf server can be started using the command listed below.

```
ethiperf
```

If the initialization is correct,then the corresponding message is displayed in the terminal, *Iperf initialized* and a client can then start the test using the control computer. Here is the testing command:

```
iperf -c 10.35.95.25
```

4.5.1 Iperf testing of communication

Testing had begun after all modifications had been made to the RPP Ethernet driver and FreeRTOS memory management. During the repair process, the communication speed increased from 256KB/s to the final 4MB/s. Testing was performed with the command:

```
iperf -c 10.35.95.25 -f K -i 1 -t 1000
```

It consists of a basic command, a switch for formatting the statement in KBytes (-f K), a periodic print switch to determine the possible fall of communication (-i 1) and a communication length switch set to 1000 seconds (-t 1000).
The communication appeared to be very stable during multiple long memory tests.

4.6 Ethernet driver

The RPP library Ethernet driver is based on the *rpp_eth_recv_raw_thr* function, which is initialized and then waits for the interrupted activated semaphore to be enabled. The corresponding interrupt is generated by the EMAC module and informs the main thread about incoming frames.
In the first part the function itself performs iteration through the link list of frames described in the picture 4.3. Also, in our case, each of the blocks, which are being processed are enriched about the LwIP frame. In the second part of the function, it performs the appropriate remapping of Texas Instruments headers into LwIP format. In the final part, the memory blocks of the already processed frames are released.

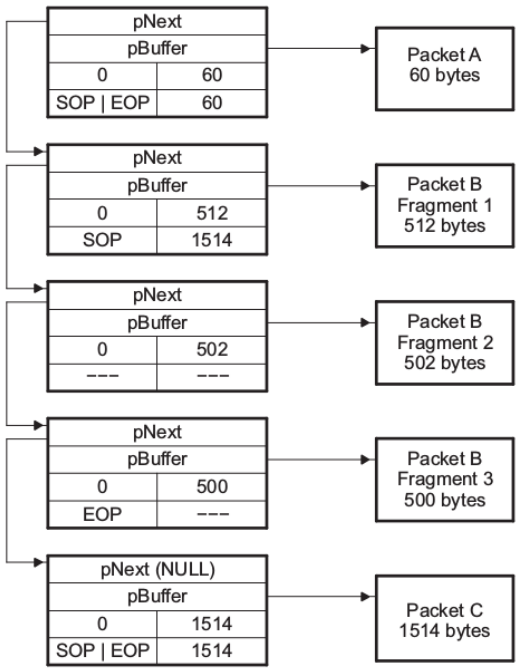


Figure 4.3: Descriptor Linked List. Taken from [TI18]

4.7 ERT Linux

ERT Linux is a target created in the IID department for use on Linux systems. Target allows you to compile programs that run on devices with a Linux operating system using Embedded Coder.
On this target was ported from Matlab version 2013b to version 2018b. Also, Target includes an external mode communicating via TCP / IP. With succesfully ported system, it would be used the main target part to the target RPP library for the TSM570 microcontroller. The list and function of each file are described below.

- ert_linux.tlc - The primary file is containing configuration about the code generation. The file is specifying target language, hard/soft real-time, degree of mutex optimization, or base step rate.
- ert_linux.tmf - Template makefile serves to create a makefile. The generated makefile is used to compile the final program.
- ert_linux_file_process.tlc - File wrapping *ert_linux_main.tlc*. It contains import of libraries like external mode.
- ert_linux_genfiles.tlc - It is currently unused file by most of the targets.
- ert_linux_main.tlc - The file contains the main function and all the necessary functions for running the program. It also includes functions for receiving and sending data using external mode.
- ert_linux_select_callback_handler.m - The file allows to set the compilation parameters in Simulink.
- ert_linux_setup.m - The file serves to add a target to others.
- sl_customization.m - The file is currently used to set the external channel communication channel, only.

Chapter 5

Conclusion

The goal of this thesis was implement and test an external mod for the latest version of Mathworks Simulink. External mode communication was not succesfully implemented using a TCP/IP connection and added to the RPP library. The implementation should be tested on a TSM570 microcontroller created by Texas Instruments.

The Simulink blocks of the RPP library were ported to the latest version of Matlab 2018b. Along with that were made the correct compiler setup and Matlab Path settings. Furthermore, it was necessary to use the appropriate version of the Embedded Code and perform the setting of the compilation parameters. Correct compilation of the library was tested directly on the TSM570 with the latest version of Matlab by running the available peripherals.
The next step to implement the external mode was to make a simple network. The network was addressing with the static addresses used for the microcontroller. Network interface was edited in the appropriate LwIP header files. The test command *ethping* was implemented to test the correct network configuration. The command was added as part of the complex test program *test-sw*. This program is used in the RPP library to test the periphery just like an Ethernet peripheral.
Removing network problems and configuring libraries allow to the *ethping* command has been appropriately tested and enabled. A function *ethiperf* has been implemented in the *test-sw* program to perform the stress test. During the stress tests, there was significant instability while receiving more massive data.
It was necessary to make adjustments in the main thread to resolve the data reception errors. In the function *rpp_eth_recv_raw_thr*, responding to the

master semaphore of the Ethernet data stream. Minor modifications in the inbound data mapping system for the LwIP, caused that communication stability has increased significantly. The subsequent tests, using the newly implemented function *ethiperf*, measure higher transfer speed.
With the stable connection, it was possible to start work on the external mode itself. While working on the TSM570's external module, it turned out that the detection of problems is challenging due to development on Embedded devices. Therefore, it was decided to start work on ERT Linux, which is a target with an external mode. The ERT Linux developed at the IID department to compile simulation directly on Linux devices. In this project, it is possible to evaluate the external mode to work more efficiently and test all supported functions. Consequently, it will be possible to easily port external mode functions directly to the TSM570.

The work offers exciting prospects for the future. EATON Corporation Inc, which has asked for the elaboration of an external mode for the RPP, is still interested in its elaboration. For the company, completing an external mode may mean saving jobs for several teams. Teams are using the RPP library for prototyping controllers on their platforms based on the TSM570 controller from Texas Instruments.
In these circumstances, I hope to be able to complete the work on the external mode to fulfill the potential benefits of this work.

Appendix A

Bibliography

[ac18a] Mathworks author collective, *Simulink coder, user's guide 2018b*, Mathwotks, 2018.

[ac18b] Mathworks authors collective, *Target language compiler 2018b*, Mathwotks, 2018.

[ac19] Iperf authors collective, *iperf - the ultimate speed test tool for tcp, udp and sctp*, `https://iperf.fr/`, Available at 21.5.2019.

[AD19] Leon Woestenberg Adam Dunkels, *Lightweight ip stack, overview*, `http://www.nongnu.org/lwip/2_1_x/index.html`, Available at 20.4.2019.

[Ass19a] Cisco Certified Network Associate, *Tcp explained*, `https://study-ccna.com/tcp-explained/`, Available at 10.5.2019.

[Ass19b] ———, *Tcp three-way handshake*, `https://study-ccna.com/tcp-three-way-handshake/`, Available at 8.5.2019.

[com19a] FreeRTOS comunity, *Memory management*, `https://www.freertos.org/a00111.html`, Available at 20.5.2019.

[com19b] FrozenTux comunity, *Icmp headers*, `https://www.frozentux.net/iptables-tutorial/chunkyhtml/x281.html`, Available at 1.5.2019.

[com19c] LwIP comunity, *lwip - lightweight tcp/ip*, `https://lwip.fandom.com/wiki/LwIP_Wiki`, Available at 19.4.2019.

[Ins13] Texas Instruments, *Dp83848-ep, data manual*, `http://www.ti.com/lit/ds/symlink/dp83848-ep.pdf`, 2013.

[Ins15] _____, *Tms570ls series 16/32-bit risc flash microcontrolle*, http://www.ti.com/lit/ds/spns141g/spns141g.pdf, 2015.

[Lim11] ARM Limited., *Cortex-r4 and cortex-r4f*, docs-api-peg.northeurope.cloudapp.azure.com/assets/ddi0363/g/DDI0363G_cortex_r4_r1p4_trm.pdf, 2011.

[Mat19] Mathworks, *Generate c and c++ code optimized for embedded systems*, https://www.mathworks.com/products/embedded-coder.html, Available at 20.5.2019.

[Mil19] Rachelle Miller, *The osi model: An overview*, SANS Institute, 2019.

[MSH17] Carlos Jenkins Michal Sojka and Michal Horn, *Simulink code generation target for texas instruments tms570 platform*, Czech Technical University in Prague, 2017.

[nSTB14] ndrew S. Tanenbaum and Herbert Bos, *Modern operating systems*, Global edition, 2014.

[Red19] Vivek Reddy, *Computer network | tcp 3-way handshake process*, https://www.geeksforgeeks.org/computer-network-tcp-3-way-handshake-process/, Available at 15.5.2019.

[Rou19] Margaret Rouse, *Transmission control protocol*, https://searchnetworking.techtarget.com/definition/TCP, Available at 8.5.2019.

[Soc19] Signal Procesing Society, *Simulink tutorial*, https://ewh.ieee.org/r1/ct/sps/PDF/MATLAB/chapter8.pdf, Available at 10.5.2019.

[TD15] Jay Thomas and Siddharth Deshpande, *Foundational software for functional safety*, Texas Instruments, 2015.

[TI18] Texas Instruments, *Tms570ls31x/21x 16/32-bit risc flash microcontroller technical reference manual*, http://www.ti.com/lit/ug/spnu499c/spnu499c.pdf, 2018.

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Nejedlý Jakub** Personal ID number: **434826**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

Branch of study: **Robotics**

II. Master's thesis details

Master's thesis title in English:

Simulink External Mode for Rapid Prototyping Platform

Master's thesis title in Czech:

Externí mód pro Simulink na platformě RPP

Guidelines:

1. Make yourself familiar with code generation using Embedded Coder in Matlab/Simulink and with Rapid Prototyping Platform (RPP) project, which was created at supervisor's department.
2. Port the support for Ethernet communication from RPP hardware to similar hardware platform called HydCtr and test its correct functionality.
3. Implement support for external mode to the code generated from Simulink so that the generated code can communicate with Simulink using a TCP connection. This will allow seeing hardware signal waveforms in Simulink as well as changing parameters of blocks, running in the board, from Simulink.
4. Prepare several test applications that demonstrate functionality of the solution for several peripherals such as GPIO, ADC, … and for different sampling periods (1 ms, 10 ms, 100 ms).
5. Document your solution both in the thesis text and in the source code.

Bibliography / sources:

[1] C. Jenkins, M. Horn, M. Sojka, „Simulink code generation target for Texas Instruments TMS570 platform", internal project documentation, 2017.
[2] Mathworks, Simulink® Coder™ User's Guide, R2013a, 2013.

Name and workplace of master's thesis supervisor:

Ing. Michal Sojka, Ph.D., Embedded Systems, CIIRC

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **11.01.2019** Deadline for master's thesis submission: **24.05.2019**

Assignment valid until: **30.09.2020**

Ing. Michal Sojka, Ph.D.	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Ing. Pavel Ripka, CSc.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

© ČVUT v Praze, Design: ČVUT v Praze, VIC