ALEXANDER TECHNOLOGICAL INSTITUTE OF THESSALONIKI

Development of an Embedded GOOSE Router According to IEC 61850-90-5

Author:
Konstantinos AVRAMIDIS

Advisor:
Albert RUIZ

Supervisor:
Periklis CHATZIMISIOS

Thessaloniki, June 2014

Contents

List of Figures

List of Tables

1 Introduction

Few words about the SmartGrid
The consumption of electric power has increased as industry has developed. Environmental problems originating in the use of fuel to provide energy have exacerbated the need to develop new, renewable energy sources, such as wind and solar power. These factors increase the complexity of power networks and make it more difficult for power operators to manage power networks reliably.

Hence, new concepts of power-network management technology, such as the Smart grid, and intelligent devices, such as Intelligent Electrical Devices (IEDs), have been developed for various facilities and to create new applications [1].

The Smart Grid is somewhat intangible in definition and relevant scope. It is a term which embraces an enhancement of the power grid - not just a traditional upgrade of the grid, but a long-term vision for a future power system.

The Smart Grid is a modernization of the hole electricity system so that it monitors, protects and automatically optimizes the operation of its interconnected elements - from the central and distributed generator through the HV transmission network, to industrial users, buildings, energy storage installations, electric vehicles, thermostats and other household devices. It will have to face challenges such as integration of renewable generation and storage devices, increase of consumer participation, communications and computational ability [2].

Few words about IEC 61850
IEC 61850 is an important new international standard for substation automation and communication. IEC 61850 was developed by the International Electrotechnical Commission's (IEC) Technical Committee 57 (TC57) architecture for electric power systems. It was designed to achieve interoperability between different vendors. As exposed in [3], IEC 61850 is an innovative approach that requires a new way of thinking about substation automation that will result in very significant improvements in Smart Grid communications.

Although IEC 61850 was developed for system interfaces between power facilities for the automation of substations, its application field is expanding rapidly. Currently, almost all power systems can be interfaced with other devices by using the IEC 61850 standard. Thus, interface devices with IEC 61850 are desirable. The IEC

61850 standard for the PMU interface has also developed as IEC 61850-90-5 [1].

Few words about PMUs
Phasor Measurement Units (PMUs) are devices which measure the phase of voltage and current in power systems, using a common time source for synchronization. Time synchronization allows synchronized real-time measurements of multiple remote measurement points on the grid. PMUs are also commonly referred to as synchrophasors. They are considered the most important measuring devices in the future Smart Grid [4].
The IEC 61850-90-5 is an extension for PMU communication and defines how to use IEC 61850 services for PMU data transfer.

Brief Purpose
According to IEC 61850-90-5, PMU data shall be exchanged with Generic Object Oriented Substation Event (GOOSE) messages and Sampled Values (SV) messages. These two are communication protocols defined in IEC 61850-8.1 and IEC 61850-9.1 respectively. They are not connection-oriented and are used only for point-to-point streaming. This is the reason, GOOSE and SV messages run directly over Ethernet.

The purpose of this project is to develop a device with proper firmware which roots GOOSE frames to the internet. This project is a part of a bigger one and focuses only in routing GOOSE messages. By routing these frames, the user can obtain real time information about an electrical power system or maybe just a single IED.

Chapters 3 and 4 expose extended information about PMUs and the IEC 61850 Standard respectively. They are more detailed approaches to these topics.

Chapter 5 and 6 define the tools used, including hardware and software.

Chapter 7 describes the software development and partial tests made to reach the final application code, which is specified in chapter 8.

Finally, chapter 9 include the conclusions of the project.

2 Purpose of the Project

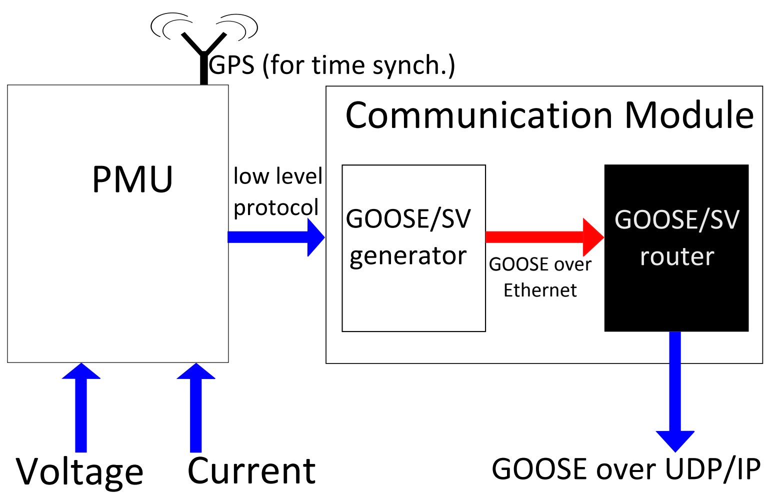


Figure 2.1: Scheme of the GOOSE router within the communication module

IEC 61850-90-5 covers PMU communications and proposes to use GOOSE and SV protocols to exchange data.

A PMU measures the phasor of voltage and current and it is synchronized by the Global Positioning System (GPS). It exchanges these measurements with a communication module. This communication module, at first, generates a GOOSE/SV message with PMU measurements. This message runs directly over Ethernet for point-to-point communication (between devices).

The purpose of the project is to implement a device that routes GOOSE messages to the internet (the black box in figure 2.1). The implementation of a GOOSE messages routing device could solve the routing issue, according to IEC 61850-90-5.

For this project some initial constraints were settled:

- **Fast data processing**. PMUs are expected to generate high data traffic. For this reason, the GOOSE router shall be able to forward incoming GOOSE messages at a high rate.

- The GOOSE router prototype should be designed as an **industrial solution**. This means that it should be based on manufactured components.
- For the implementation only **open source tools** should be used.
- **Low price**. The price of the whole prototype should as low as possible. The components used were the most adequate, for the needs of the project.

Finally, the development of this project should help the author to acquire skills for embedded software development.

3 PMUs Overview

3.1 Introduction to PMUs

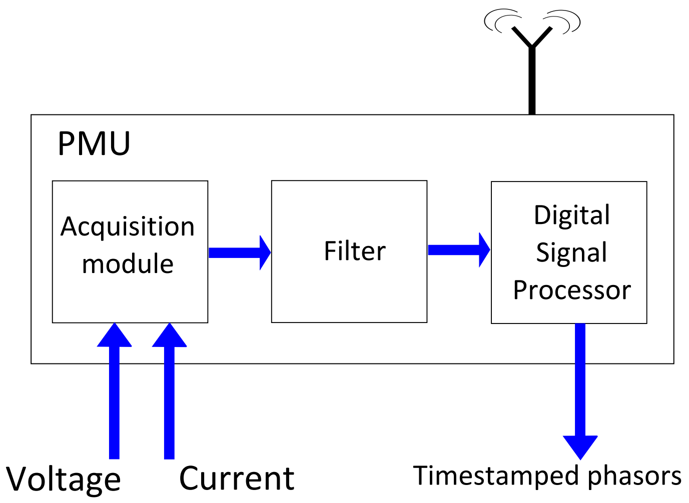


Figure 3.1: General layout of a PMU

One of the main challenges of the Smart Grid is to increase the reliability of power grids. This is in part motivated by the high costs derived from large blackouts. Roughly speaking, a large blackout occurs as a result of a cascading series of failures in the grid. An example of a large blackout is the one experienced in North America in August 2003, which caused losses on the order of billions of dollars. These situations occur due to the high interconnection and interdependence of the elements of the grid and the lack of wide-area situational awareness in today's power grid [5].

Wide-area situational awareness refers to the monitoring of the grid across large geographical areas aimed at obtaining a detailed and accurate picture of the overall grid performance. To that end, a PMU provides voltage and current phasor measurements of the power line. These measurements are time-stamped with an accurate global reference clock-by means of Global Positioning System (GPS). By exploiting

Figure 3.2: Schweizer Engineering Laboratories PMU SEL-487E

https://www.selinc.com/SEL-487E/

time synchronization, measurements taken by different PMUs can provide a snapshot of the state of the grid for each time instant.

Figure 3.1 presents a general layout of a PMU. As depicted in the figure, PMUs rely on a GPS time signal for extremely accurate time-stamping of phasor measurement.

A GPS satellite receiver provides a precise timing pulse, which is correlated with sampled voltage and current inputs - typically the three phase voltages of a substation and the currents in lines, transformers, and loads terminating at the substation. From these data samples, positive-sequence voltages and currents are calculated and timestamped so that the exact microsecond when the phasor measurement is taken is permanently attached to it. The device assembles a message from the time stamp and the phasor data in a format defined in IEEE C37.118, which can then be transmitted to a remote site over any available communication link. Positive-sequence phasor data from all substations equipped with such devices are collected at an appropriate central site using a data concentrator or exchanged between local units for protection/control applications [4].

4 IEC 61850 and GOOSE Messages

4.1 Introduction to IEC 61850

IEC 61850 has been elaborated by the ad-hoc group ”Substation Control and Protection Interfaces” of IEC Technical Committee 57 ”Power systems management and associated information exchange”.

This standard was first designed for the standardisation of communication in Substation Automation Systems (SAS). However, just a few years after the start of IEC 61850 project (in 1995), utility and vendor experts of non-substation related application domains began to realise both the benefits of a single international standard for the electrical energy supply system and the powerful approach and content of IEC 61850. [6]

The objective of IEC 61850 is to specify requirements and to provide a framework to achieve interoperability among Intelligent Electronic Devices (IEDs). This framework specifies the manner that devices should organize information, so it is consistent across all types of devices.

4.1.1 Benefits of IEC 61850

IEC 61850 and its features deliver substantial benefits to users that understand and take advantage of them. Rather than simply approaching an IEC 61850 based system in the same way as any other system, a user that understands and takes advantage of the unique capabilities will realize significant benefits that are not available using legacy approaches.

The most significant benefit of IEC 61850 is the interoperability with different vendors. This means that different devices from different manufacturers can actually communicate and exchange data seamlessly. Figure 4.1 presents the present situation in automation in the electric power system - multiple communication protocols: DNP, MODBUS, LON etc. On the other hand, figure 4.2 shows an automation system with true seamless interoperability.

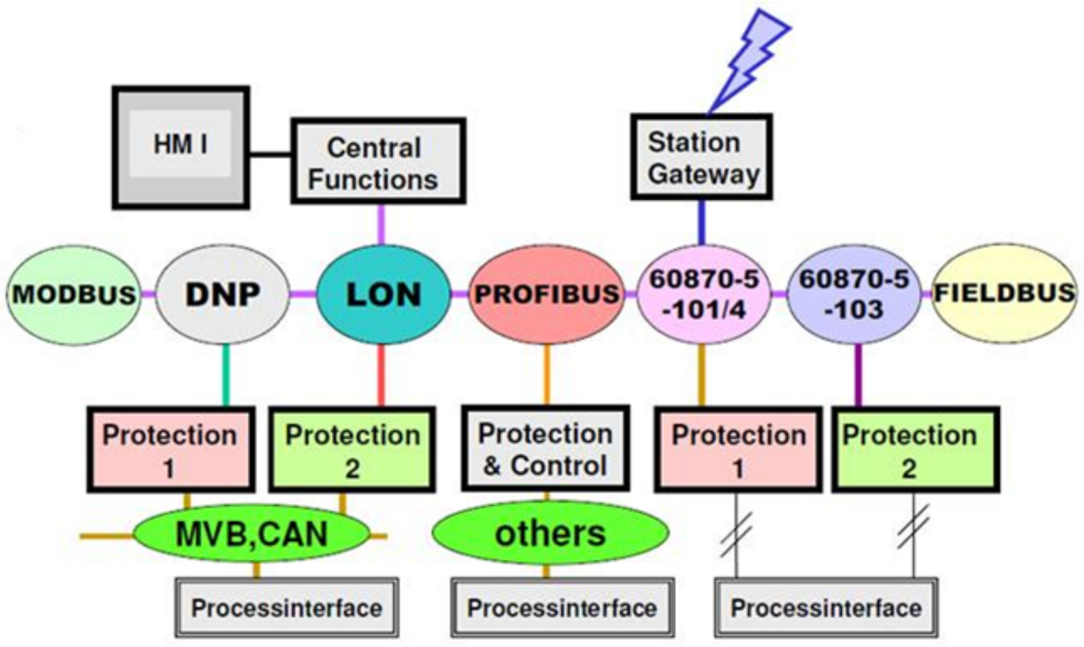


Figure 4.1: Present automation system in the electricity sector

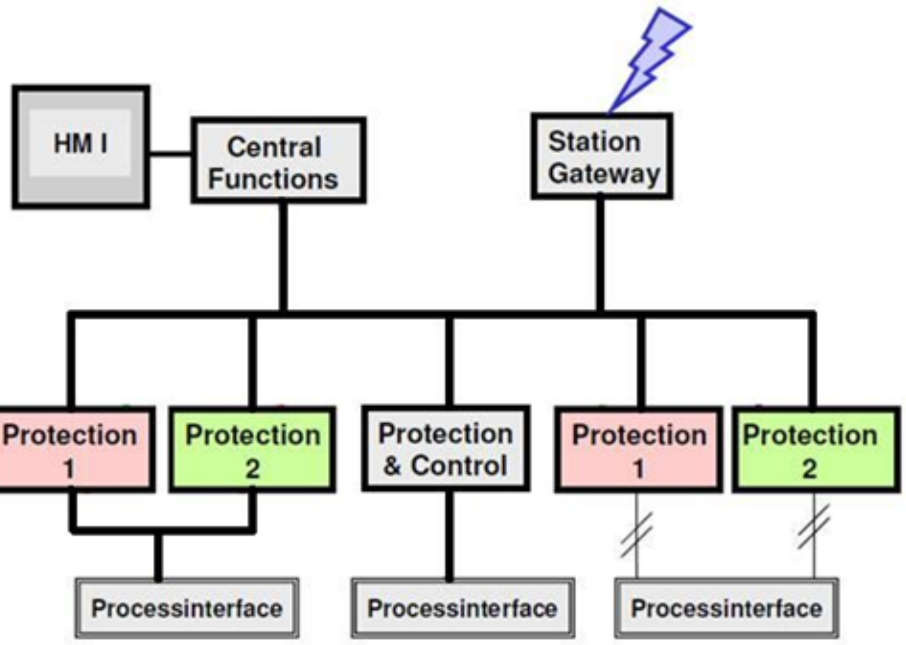


Figure 4.2: Desired automation system using IEC 61850

Some extra benefits derived from interoperability of IEC 61850:

- **Lower Installation Cost**: IEC 61850 enables devices to quickly exchange data without having to wire separate links for each device.
- **Lower Transducer Costs**: With IEC 61850 protocol ”translators” are not required.
- **Lower Commissioning Costs**: The cost to configure and commission devices is drastically reduced because IEC 61850 devices don’t require as much manual configuration.
- **Lower Equipment Migration Costs**: Because IEC 61850 defines more of the externally visible aspects of the devices besides just the encoding of data on the wire, the cost for equipment migrations is minimized.
- **Lower Extension Costs**: Because IEC 61850 devices do not have to be configured to expose data, new extensions are easily added without having to reconfigure devices.
- **Lower Integration Costs**: IEC 61850 networks are capable of delivering data without separate communications front-ends or reconfiguring devices.
- **Implement New Capabilities**: The advanced services and unique features of IEC 61850 enables new capabilities that are simply not possible with most legacy protocols.

4.2 Communication Protocols used in IEC 61850

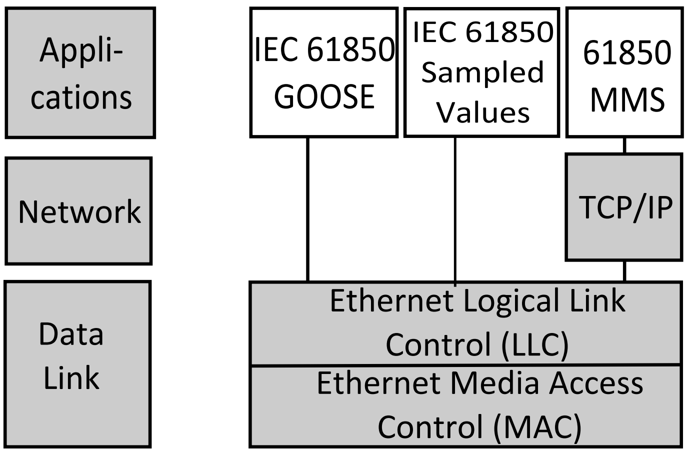


Figure 4.3: Illustration of protocol stack of general data

IEC 61850-7-2 proposes some abstract communication services to [7]:

- Connect and disconnect
- Read data
- Write data
- Manage reports
- Manage alarms
- Send sampled values
- Time synchronization
- File transfer

Protocols IEC 61850-8-1 [8], IEC 61850-9-1 [9] and IEC 61850-9-2 [10] define how this services are mapped into real protocols.

Current mappings in the standard are to Manufacturing Message Specification(MMS), GOOSE and SV. As shown in figure 4.3, SV and GOOSE applications run directly over Ethernet data frame, thereby eliminating processing of any middle layers; the MMS operates over TCP/IP. [3]

4.3 GOOSE Protocol and IEC 61850-90-5

IEC 61850 proposes a fast and reliable communication system, between IEDs in a substation, which is not connection-oriented. Furthermore, it is only for point-to-point streaming and is based on light weight messages. One of the protocols associated with this system is the GOOSE protocol. As mentioned also before, GOOSE runs directly over Ethernet.

Table 4.1 presents fields that can be found at the GOOSE PDU and their value types.

IEC 61850-90-5 is a new (2012) extension of IEC 61850 for PMU data transfer. This chapter specifies GOOSE and SV broadcast over UDP protocol (figure 4.4).

Table 4.1: GOOSE PDU fields

Field name	**No**	**Value type**
gocdRef	[0]	IMPLICIT VISIBLE-STRING
timeAllowedtoLive	[1]	IMPLICIT INTEGER
datSet	[2]	IMPLICIT VISIBLE-STRING
goID	[3]	IMPLICIT VISIBLE-STRING OPTIONAL
t	[4]	IMPLICIT UtcTime
stNum	[5]	IMPLICIT INTEGER
sqNum	[6]	IMPLICIT INTEGER
test	[7]	IMPLICIT BOOLEAN DEFAULT FALSE
confRev	[8]	IMPLICIT INTEGER
ndsCom	[9]	IMPLICIT BOOLEAN DEFAULT FALSE
numDatSetEntries	[10]	IMPLICIT INTEGER
allData	[11]	IMPLICIT SEQUENCE OF Data
security	[12]	ANY OPTIONAL

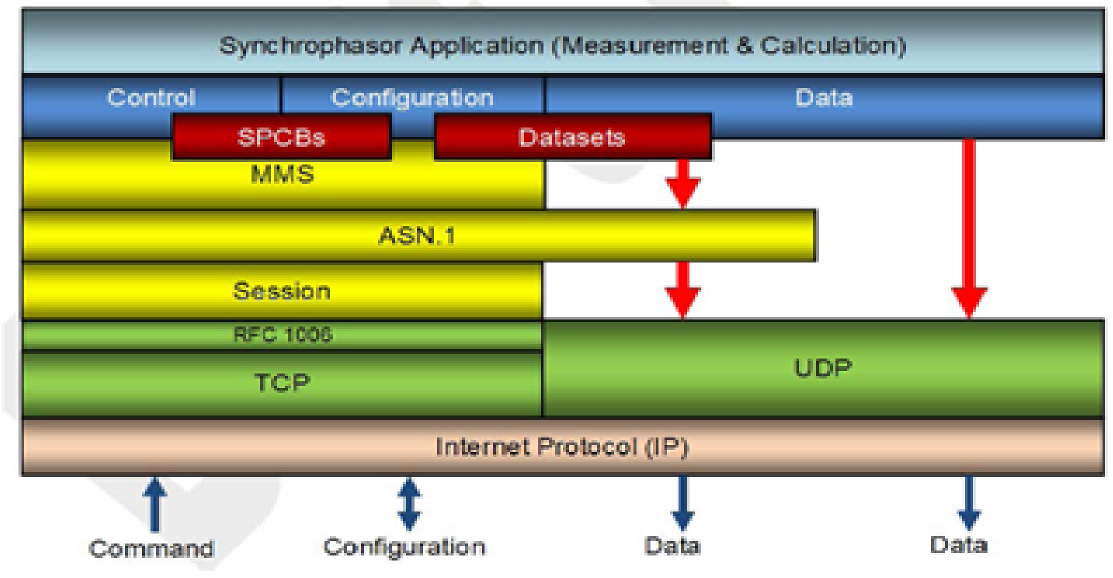


Figure 4.4: General overview of the mapping of Synchrophasor Services

5 GOOSE Router Prototype

In this project it has been implemented a GOOSE message router prototype. This chapter describes the hardware that has been used.

5.1 Hardware components

Figure 5.1 shows the objective of the project. The GOOSE message received from a local network is forwarded to a public network (internet). This requires two Ethernet ports, one for input and another for output.

The prototype has been implemented with an ARM Cortex-M3 platform and an external Ethernet controller. Both components are connected via SPI connection (Figure 5.2).

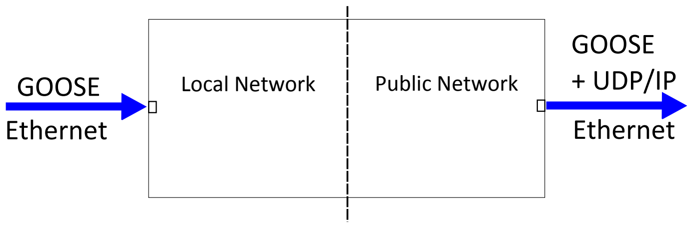


Figure 5.1: Objective of the application prototype

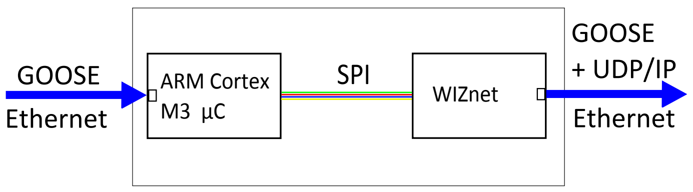


Figure 5.2: Components of the application prototype

The choice of the hardware was based on requirements exposed in chapter 2. The components of the prototype are (see figure 5.3):

- **Mbed LPC 1768** platform (on base board)
- **WIZnet W5100** Ethernet controller.

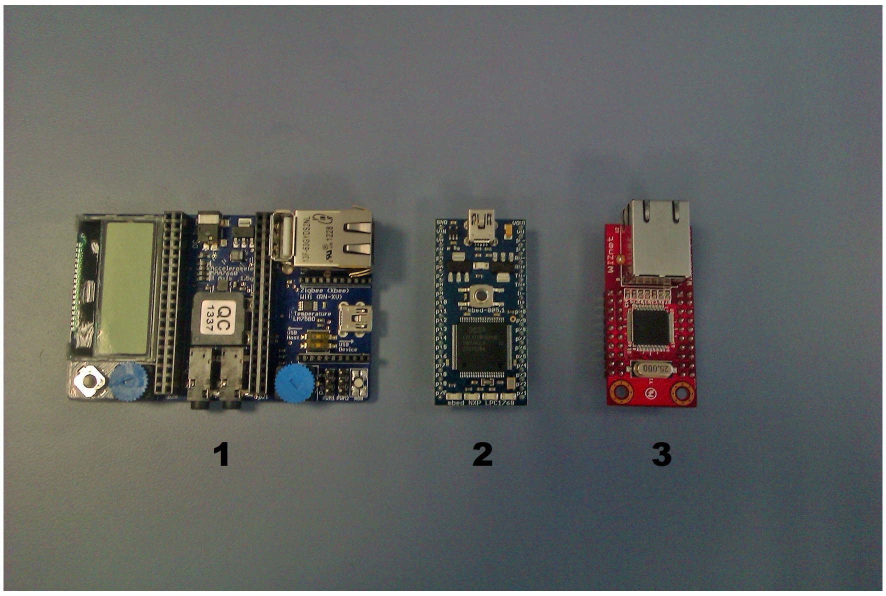


Figure 5.3: 1. Base Board, 2. Mbed LPC 1768, 3. WIZnet W5100

5.2 Mbed LPC 1768

Mbed LPC 1768 is an ARM micro-controller development board designed for rapid prototyping. It contains a 32-bit ARM Cortex-M3 core running at 96MHz. It includes lots of interfaces including built-in Ethernet, USB Host and Device, CAN, SPI, I2C, ADC, DAC, PWM and other I/O interfaces. Figure 5.4 shows the commonly used interfaces and their locations (source http://www.mbed.org/platforms/mbed-LPC1768/). The peripheral complement of the LPC 1768 includes (among other blocks):

- 512 kB of flash memory
- 64 kB of data memory
- Ethernet MAC
- 4 UARTs
- SPI interface
- Four general purpose timers

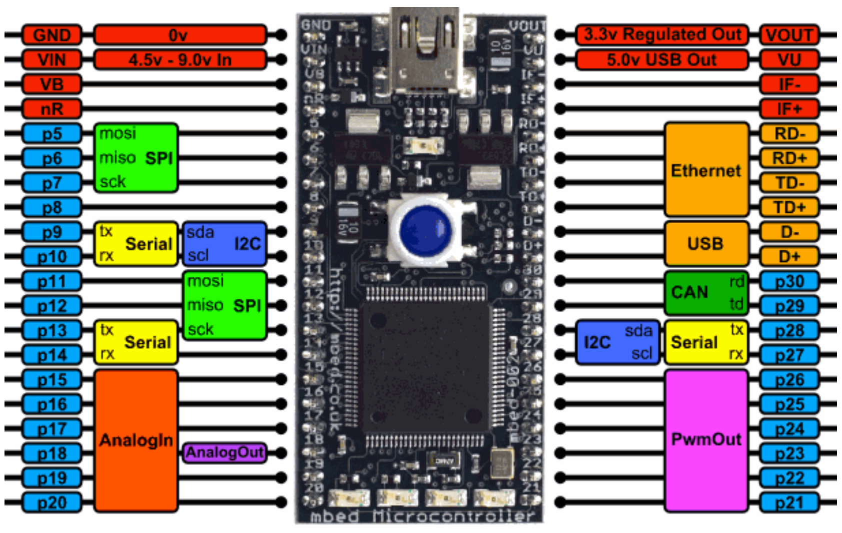


Figure 5.4: Pinout of the mbed board

- Real-Time Clock (RTC)
- 70 general purpose I/O pins

5.3 ARM Cortex-M3 architecture

The ARM Cortex-M3 is a series of general purpose 32-bit microprocessors, which offer high performance and very low power consumption. The Cortex-M3 offer many new features, including a Thumb-2 instruction set, low interrupt latency, hardware divide, interruptible/continuable multiple load and store instructions, automatic state save and restore for interrupts, tightly integrated interrupt controller with Wakeup Interrupt Controller, and multiple core buses capable of simultaneous accesses (see http://www.arm.com/products/processors/cortex-m/index.php).

Pipeline techniques are employed so that all parts of the processing and memory systems can operate continuously. Typically, while one instruction is being executed, its successor is being decoded, and a third instruction is being fetched from memory [11].

The Cortex-M addresses the requirements for the 32-bit processor in the following ways:

- Greater performance efficiency: allowing more work to be done without increasing the frequency or power requirements.
- Low power consumption: enabling longer battery life, especially critical in portable products including wireless networking applications.

- Enhanced determinism: guaranteeing that critical tasks and interrupts are serviced as quickly as possible and in a known number of cycles.
- Improved code density: ensuring that code fits in even the smallest memory footprints
- Ease of use: providing easier programmability and debugging for the growing number of 8-bit and 16-bit users migrating to 32 bits.
- Lower cost solutions: reducing 32-bit-based system costs close to those of legacy 8-bit and 16-bit devices and enabling low-end, 32-bit microcontrollers to be priced at less than 1 US dollar for the first time.
- Wide choice of development tools: from low-cost or free compilers to full-featured development suites from many development tool vendors.

5.4 WIZnet W5100 board

The W5100 is a full-featured, single-chip Internet-enabled 10/100 Ethernet controller designed for embedded applications where ease of integration, stability, performance, area and system cost control are required. The W5100 has been designed to facilitate easy implementation of Internet connectivity without OS. The W5100 is IEEE 802.3 10BASE-T and 802.3u 100BASE-TX compliant.

The W5100 includes fully hardwired integrated Ethernet MAC and PHY, TCP, UDP, IPv4, ICMP, ARP, IGMP and PPPoE. It includes 16Kbytes internal buffer for data transmission. The w5100 offers 4 sockets, programmable through SPI. For easy integration, three different interfaces like memory access way, called direct, indirect bus and SPI, are supported on the MCU side [12].

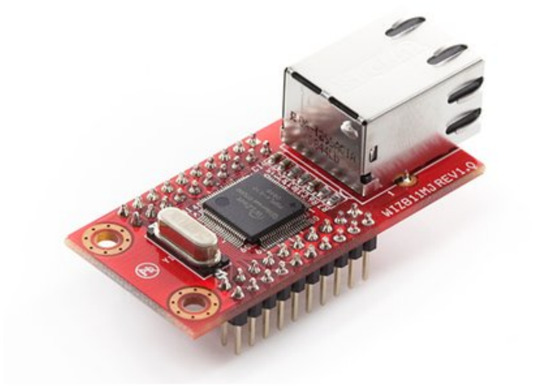


Figure 5.5: WIZnet W5100

6 Software Development Tools

This chapter exposes what software tools were used for the development of the application.

6.1 First steps: Assembly development

The philosophy of the low-level language Assembly helps developers to figure out how and which registers are needed to be configured and also be as close to hardware as one can be. For this reason the author was proposed to develop a small program using a simple editor and GNU toolchain for ARM architecture, as an exercise. In annex A there is an example to turn on a led.

6.1.1 GNU toolchain for ARM architecture

The GNU toolchain is a blanket term for a collection of programming tools produced by the GNU Project. It can be used for programming applications for different architectures, including ARM Cortex-M3.

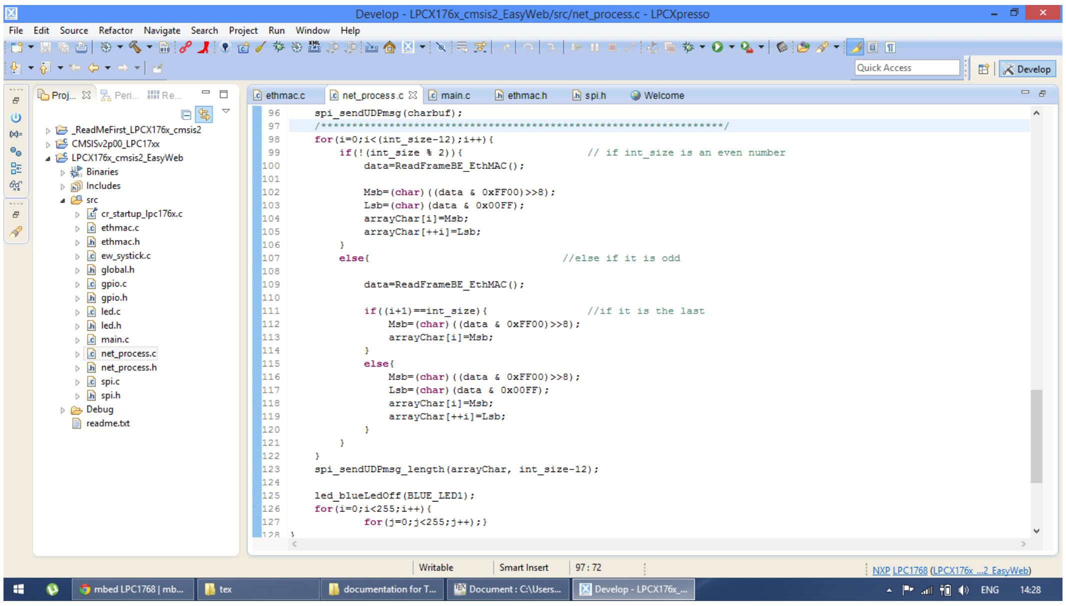
Projects included in the GNU toolchain are:

- GNU make: Automation tool for compilation and build. GNU Compiler Collection (GCC): Suite of compilers for several programming languages.
- GNU Binutils: Suite of tools including linker, assembler and other tools.
- GNU Bison: Parser generator.
- GNU m4: m4 macro processor.
- GNU Debugger (GDB): Code debugging tool.
- GNU build system (autotools).

The GNU Compiler Collection (GCC) is a compiler system produced by the GNU Project supporting various programming languages. GCC is a key component of the GNU toolchain. The Free Software Foundation (FSF) distributes GCC under the GNU General Public License (GNU GPL). GCC has played an important role in the growth of free software, as both a tool and an example (see also http://en.wikipedia.org/wiki/GNU_Compiler_Collection).

GNU toolchain is available for free in http://www.mentor.com/embedded-software/sourcery-tools/sourcery-codebench/editions/lite-edition/.

6.2 NXP LPCXpresso IDE



```
    spi_sendUDPmsg(charbuf);
    /*****************************************************************/
    for(i=0;i<(int_size-12);i++){
        if(!(int_size % 2)){                    // if int_size is an even number
            data=ReadFrameBE_EthMAC();

            Msb=(char)((data & 0xFF00)>>8);
            Lsb=(char)(data & 0x00FF);
            arrayChar[i]=Msb;
            arrayChar[++i]=Lsb;
        }
        else{                                 //else if it is odd

            data=ReadFrameBE_EthMAC();

            if((i+1)==int_size){                //if it is the last
                Msb=(char)((data & 0xFF00)>>8);
                arrayChar[i]=Msb;
            }
            else{
                Msb=(char)((data & 0xFF00)>>8);
                Lsb=(char)(data & 0x00FF);
                arrayChar[i]=Msb;
                arrayChar[++i]=Lsb;
            }
        }
    }
    spi_sendUDPmsg_length(arrayChar, int_size-12);

    led_blueLedOff(BLUE_LED1);
    for(i=0;i<255;i++){
        for(j=0;j<255;j++);}
}
```

Figure 6.1: LPCXpresso interface

Final software was developed with NXP's LPCXpresso. The LPCXpresso IDE is a highly-integrated software development environment for NXP's LPC micro-controllers. It is a low-cost development tool platform, available directly from NXP (see http://www.lpcware.com/lpcxpresso/code-red). It includes the GNU toolchain for ARM and some extra tools to develop high-quality software solutions. LPCXpresso builds on its Eclipse foundation by including many enhancements that simplify development with NXP LPC micro-controllers.

The LPCXpresso platform supports all of NXP's LPC family of microcontrollers, including those based on ARM7, ARM9 and Cortex-M.

Some of NXP LPCXpresso's key features are as follows:

- Eclipse-based IDE.
- Free Edition supports code sizes up to 256 kb after activation and can be upgraded to unlimited code size by purchasing a Pro Edition license.
- Supports C++ application and library projects.
- Instruction Trace support.
- Contains helpful coding examples.

6.3 CMSIS code layer

6.3.1 Background of CMSIS

The Cortex-M micro-controllers are gaining momentum in the embedded application market, as more and more products based on the Cortex-M processor and software that support the Cortex-M processor are emerging. At the end of 2008, there were more than five C compiler vendors, and more than 15 embedded Operating Systems (OS) supporting the Cortex-M processor. There are also a number of companies providing embedded software solutions, including codecs, data processing libraries, and various software and debug solutions. The Cortex Micro-controller Software Interface Standard (CMSIS) was developed by ARM chip manufacturers to help users of the Cortex-M micro-controllers to develop their embedded application quickly and reliably.

The CMSIS was started in 2008 to improve software usability and inter-operability of ARM micro-controller software. It is integrated into the driver libraries provided by silicon vendors, providing a standardized software interface for the Cortex-M processor features, as well as a number of common system and I/O functions. The library is also supported by software companies including embedded OS vendors and compiler vendors.

The aims of CMSIS are to:

- Improve software portability and reusability
- Enable software solution suppliers to develop products that can work seamlessly with device libraries from various silicon vendors
- Allow embedded developers to develop software quicker with an easy-to-use and standardized software interface
- Allow embedded software to be used on multiple compiler products
- Avoid device driver compatibility issues when using software solutions from multiple sources

The first release of CMSIS was available from fourth quarter of 2008 and has already become part of the device driver library from microcontroller vendors. The CMSIS is also available for Cortex-M0.

6.3.2 Areas of standardization

The scope of CMSIS involves standardization in the following areas:

- Hardware Abstraction Layer (HAL) for Cortex-M processor registers: This includes standardized register definitions for NVIC, System Control Block registers, SYSTICK register, MPU registers, and a number of NVIC and core feature access functions.
- Standardized system exception names: This allows OS and middleware to use system exceptions easily without compatibility issues.
- Standardized method of header file organization: This makes it easier for users to learn new Cortex microcontroller products and improve software portability.
- Common method for system initialization: Each Micro-controller Unit (MCU) vendor provides a SystemInit() function in their device driver library for essential setup and configuration, such as initialization of clocks. Again, this helps new users to start to use Cortex-M micro-controllers and aids software portability.
- Standardized intrinsic functions: Intrinsic functions are normally used to produce instructions that cannot be generated by IEC/ISO C.* By having standardized intrinsic functions, software reusability and portability are considerably improved.
- Common access functions for communication: This provides a set of software interface functions for common communication interfaces including universal asynchronous receiver/transmitter (UART), Ethernet, and Serial Peripheral Interface (SPI). By having these common access functions in the device driver library, reusability and portability of embedded software are improved. At the time of writing this book, it is still under development.
- Standardized way for embedded software to determine system clock frequency: A software variable called SystemFrequency is defined in device driver code. This allows embedded OS to set up the SYSTICK unit based on the system clock frequency.

The CMSIS defines the basic requirements to achieve software reusability and portability. MCU vendors can include additional functions for each peripheral to enrich the features of their software solution. So using CMSIS does not limit the capability of the embedded products.

The role of these layers is summarized in Figure 6.2.

6.3.3 Organization of CMSIS

The CMSIS is divided into multiple layers as follows:

Core Peripheral Access Layer

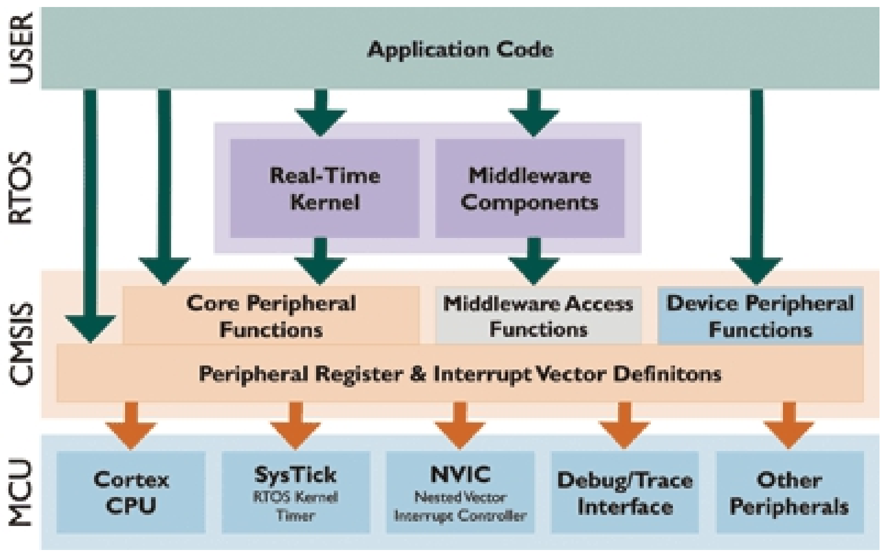

USER
RTOS
CMSIS
MCU
Application Code
Real-Time Kernel
Middleware Components
Core Peripheral Functions
Middleware Access Functions
Device Peripheral Functions
Peripheral Register & Interrupt Vector Definitons
Cortex CPU
SysTick
RTOS Kernel Timer
NVIC
Nested Vector Interrupt Controller
Debug/Trace Interface
Other Peripherals

Figure 6.2: CMSIS Struture

http://www.electronicproducts.com/Software/Development_Tools_and_Software/Software_interface_standard_gives_new_framework.aspx

- Name definitions, address definitions and helper functions to access core registers and core peripherals

Middleware Access Layer

- Common method to access peripherals for the software industry (work in progress)
- Targeted communication interfaces include Ethernet, UART, and SPI.
- Allows portable software to perform communication tasks on any Cortex microcontrollers that support the required communication interface

Device Peripheral Access Layer (MCU specific)

- Name definitions, address definitions, and driver code to access peripherals

Access Functions for Peripherals (MCU specific)

- Optional additional helper functions for peripherals

6.3.4 Using CMSIS

Since the CMSIS is incorporated inside the device driver library, there is no special setup requirement for using CMSIS in projects. For each MCU device, the MCU vendor provides a header file, which pulls in additional header files required by the device driver library, including the Core Peripheral Access Layer defined by ARM (see 6.3).

These header files contain the peripheral register definitions and access functions for the Cortex-M3 processor peripherals like NVIC, System Control Block registers, and SYSTICK registers. They also contain declaration of CMSIS intrinsic functions to allow C applications to access instructions that cannot be generated using IEC/ISO C language.

A notification is that in some cases, the intrinsic functions in CMSIS could have similar names compared with the intrinsic functions provided in the C compilers, whereas the CMSIS intrinsic functions are compiler independent.

The C files contain implementation of CMSIS intrinsic functions that cannot be implemented in the header files using simple definitions.

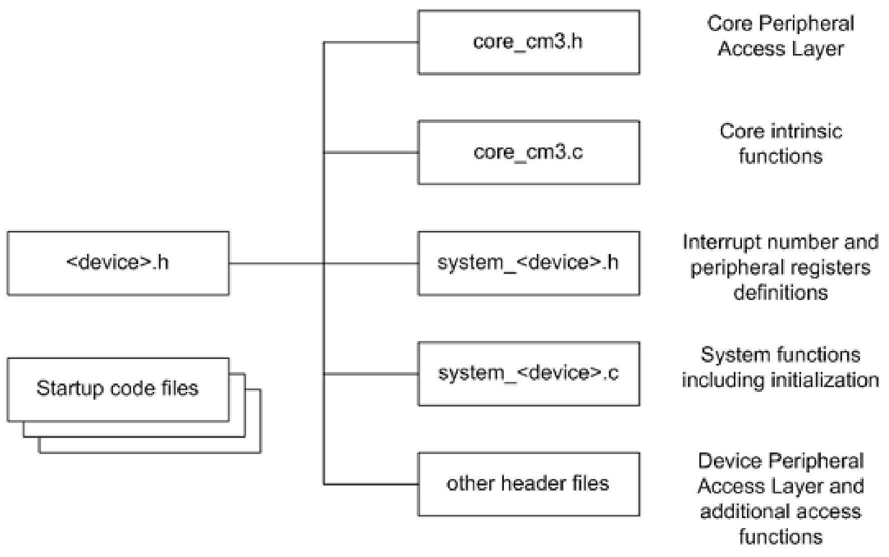

core_cm3.h
Core Peripheral Access Layer
core_cm3.c
Core intrinsic functions
<device>.h
system_<device>.h
Interrupt number and peripheral registers definitions
Startup code files
system_<device>.c
System functions including initialization
other header files
Device Peripheral Access Layer and additional access functions

Figure 6.3: CMSIS Files

There is also a system header file, which contains microcontroller specific interrupt number definitions and peripheral register definitions. The system C file contains a microcontroller specific function called SystemInit for system initialization.

In addition, CMSIS compliant device drivers also contain start-up code (which contains the vector table) for various supported compilers, and CMSIS version of intrinsic functions to allow embedded software access to all processor core features on different C compiler products.

6.3.5 Benefits of CMSIS

The main advantage is much better software portability and reusability. Besides easy migration between different Cortex-M3 microcontrollers, it also allows software to be quickly ported between Cortex-M3 and other Cortex-M processors, reducing time to market.

For embedded OS vendors and middleware providers, the advantages of the CMSIS are significant. By using the CMSIS, their software products can become compatible with device drivers from multiple microcontroller vendors, including future microcontroller products that are yet to be released. Without the CMSIS, the software vendors either have to include a small library for Cortex-M3 core functions or develop multiple configurations of their product so that it can work with device libraries from different microcontroller vendors.

The CMSIS has a small memory footprint (less than 1 KB for all core access functions and a few bytes of RAM). It also avoids overlapping of core peripheral driver code when reusing software code from other projects.

Since CMSIS is supported by multiple compiler vendors, embedded software can compile and run with different compilers. As a result, embedded OS and middleware can be MCU vendor independent and compiler tool vendor independent. Before availability of CMSIS, intrinsic functions were generally compiler specific and could cause problems in retargetting the software in a different compiler.

Since all CMSIS compliant device driver libraries have a similar structure, learning to use different Cortex-M3 microcontrollers is even easier as the software interface has similar look and feel (no need to relearn a new application programming interface).

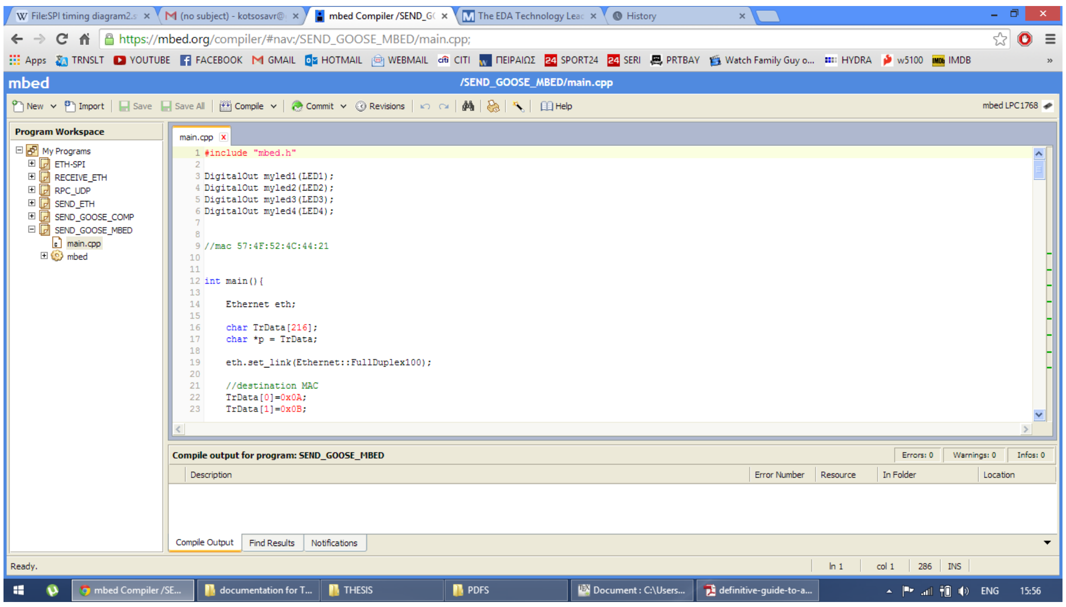
CMSIS is tested by multiple parties and is Motor Industry Software Reliability Association (MISRA) compliant, thus reducing the validation effort required for developing your own NVIC or core feature access functions. [11]

6.4 Mbed online environment

Mbed provides an online development environment for some ARM Cortex-M platforms. It includes a variety of libraries, an editor and a compiler for fast prototyping. The mbed environment is available through web browsers (as shown in Figure 6.4).

There is also a community where developers upload their work or contribute to back fixes and libraries as help to other developers.

Online environment contributes very much in developing from the aspect of time and effectiveness. However, for the final application code, it was preferred for the author not to use the ready made code, but to develop from scratch. Annex A shows an example of an application developed with this environment.



https://mbed.org/compiler/#nav:/SEND_GOOSE_MBED/main.cpp;

/SEND_GOOSE_MBED/main.cpp

Program Workspace

- My Programs
  - ETH-SPI
  - RECEIVE_ETH
  - RPC_UDP
  - SEND_ETH
  - SEND_GOOSE_COMP
  - SEND_GOOSE_MBED
    - main.cpp
    - mbed

main.cpp

```
#include "mbed.h"

DigitalOut myled1(LED1);
DigitalOut myled2(LED2);
DigitalOut myled3(LED3);
DigitalOut myled4(LED4);


//mac 57:4F:52:4C:44:21


int main(){

    Ethernet eth;

    char TrData[216];
    char *p = TrData;

    eth.set_link(Ethernet::FullDuplex100);

    //destination MAC
    TrData[0]=0x0A;
    TrData[1]=0x0B;
```

Compile output for program: SEND_GOOSE_MBED

Errors: 0 Warnings: 0 Infos: 0

Description	Error Number	Resource	In Folder	Location

Ready.

Figure 6.4: Online mbed environment running in Google Chrome

7 Software Development and Partial Tests

7.1 GPIO programming

The mbed board has 4 LEDs. These LEDs are connected to LPC 1768 with certain GPIO pins. For example LED 1 is connected to pin P1.18 (pin 18 which belongs to port 1). The LED is on after the set up of 3 configuration registers, PINSEL3, FIO1DIR2 and FIO1MASK2 and the status register FIO1PIN2 (to control LED status). Table 7.1 shows which values have to be written to these registers to turn on LED 1 and Figure 7.1 is showing the result [13].

Table 7.1: Settings for turning on LED 1

Register	Value	Explanation
PINSEL3	0x00	Initialization of register
FIO1DIR2	0xFF	Controls the direction of each port pin
FIO1MASK2	0x00	Writes, sets, clears and reads to the port
FIO1PIN2	0xFF	Shows current state of digital port pins

7.2 SPI programming

As seen in chapter 5, LPC 1768 is connected with the W5100 via SPI. In this chapter it is explained how SPI module of the LPC 1768 was set and how the W5100 was configured via SPI commands.

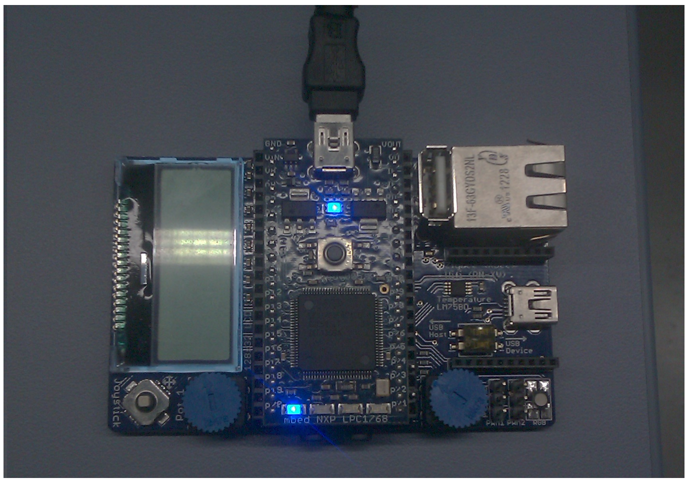


Figure 7.1: LED 1 is turned on

7.2.1 Introduction to SPI

The Serial Peripheral Interface (SPI) bus is a synchronous serial data link that operates in full duplex mode. Multiple slave devices are allowed with individual slave select lines, but only one master device is allowed. SPI is a four-wire serial bus, contrasting with three-, two-, and one-wire serial buses.

The SPI bus specifies four logic signals:

- SCLK: serial clock (output from master);
- MOSI: master output, slave input (output from master);
- MISO: master input, slave output (output from slave);
- SS: slave select (active low, output from master).

Alternative naming conventions are also widely used:

- SCLK: SCK, CLK: serial clock (output from master)
- MOSI: SIMO, SDO, DO, DOUT, SO, MTSR: serial data out; data out, serial out, master transmit slave receive
- MISOMISO: SOMI, SDI, DI, DIN, SI, MRST: serial data in; data in, serial in, master receive slave transmit
- SS: nCS, CS, CSB, CSN, nSS, STE, SYNC: chip select, slave transmit enable (active low, output from master)

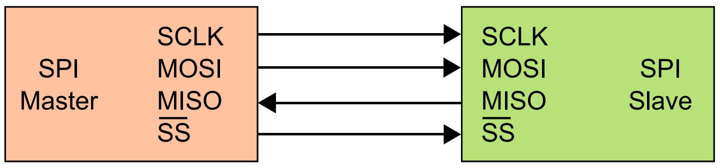


Figure 7.2: SPI bus: single master and single slave

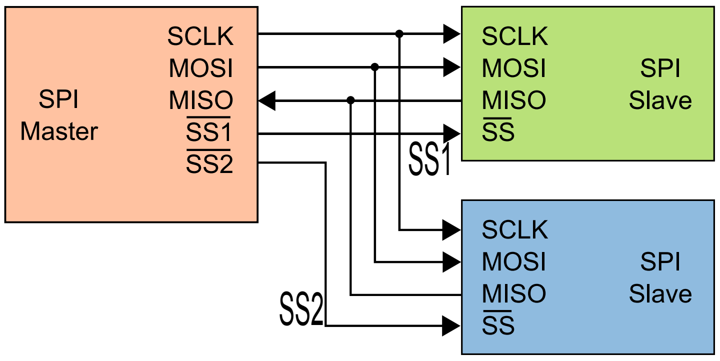


Figure 7.3: SPI bus: single master and multiple slaves

Figure 7.2 shows single master/single slave communication, while Figure 7.3 exposes single master/multiple slave communication. If a single slave device is used, the SS pin may be fixed to logic low if the slave permits it. Some slaves require a falling edge of the chip select signal to initiate an action, which starts conversion on a high-low transition. With multiple slave devices, an independent SS signal is required from the master for each slave device.

To begin a communication, the bus master first switches the logic 0 for the desired slave over the SS. SS is switched to logic 0 because it is active low, meaning its off state is a logic 1. Afterwards, the master configures the clock, using a frequency less than or equal to the maximum frequency the slave device supports. Such frequencies are commonly in the range of 10 kHz – 100 MHz. If a waiting period is required (such as for analog-to-digital conversion), then the master must wait for at least that period of time before starting to issue clock cycles. After clock configuration, the data is transferred and after that the master switches SS back to logic 0. During each SPI clock cycle, a full duplex data transmission occurs:

- The master sends a bit on the MOSI line; the slave reads it from that same line.
- The slave sends a bit on the MISO line; the master reads it from that same line.

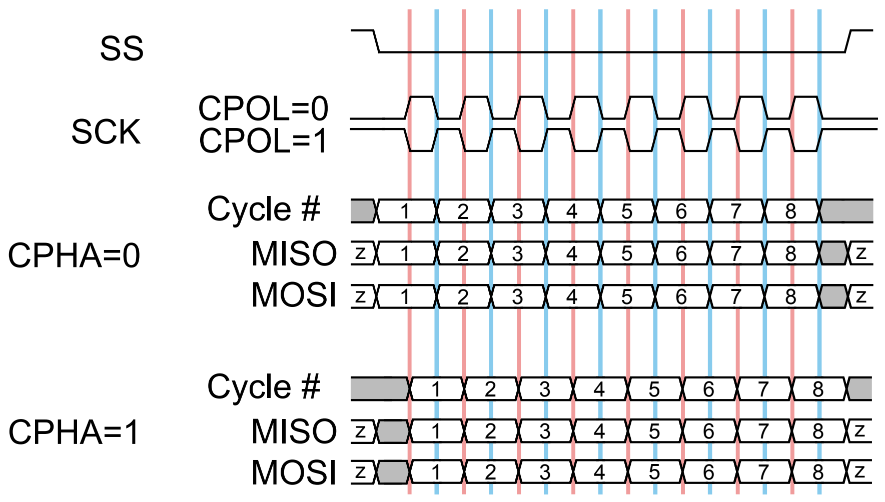


Figure 7.4: Timing diagram showing clock polarity and phase

In addition to setting the clock frequency, the master must also configure the clock polarity and phase with respect to the data. To do so, most micro-controllers use CPOL and CPHA respectively, as internal registers. Figure 7.4 shows the different states of polarity and phase. At CPOL=0 the base value of the clock is zero:

- For CPHA=0, data are captured on the clock's rising edge (low-high transition) and data is propagated on a falling edge (high-low clock transition).
- For CPHA=1, data are captured on the clock's falling edge and data is propagated on a rising edge.

At CPOL=1 the base value of the clock is one (inversion of CPOL=0):

- For CPHA=0, data are captured on clock's falling edge and data is propagated on a rising edge.
- For CPHA=1, data are captured on clock's rising edge and data is propagated on a falling edge.

That is, CPHA=0 means sample on the leading (first) clock edge, while CPHA=1 means sample on the trailing (second) clock edge, regardless of whether that clock edge is rising or falling. Note that with CPHA=0, the data must be stable for a half cycle before the first clock cycle.

The MOSI and MISO signals are usually stable (at their reception points) for the half cycle until the next clock transition. SPI master and slave devices may well sample data at different points in that half cycle. This adds more flexibility to the communication channel between the master and slave.

Below can be seen the part of the code that enables the SPI interface.

```
int spi_enable(void){
   LPC_SC_TypeDef *systemControl = LPC_SC;
   // First we turn ON power for the SPI module
   // This is done in the PCONP register
   // SPI is activated with the bit 8 in PCONP register
   // bit8 = 0 -> no power (deactivated)
   // bit8 = 1 -> power (activated)
   systemControl->PCONP |= BIT8; // SPI is activated

   // Then we set clock for the SPI module
   // This is done in the PCLKSEL0 register
   // Clock for SPI is modified in bits 17:16 in PCLKSEL0
      register
   // 17:16 = 00 -> CCLK/4
   // 17:16 = 01 -> CCLK
   // 17:16 = 10 -> CCLK/2
   // 17:16 = 11 -> CCLK/8
   systemControl->PCLKSEL0 &= ~(BIT17|BIT16);  // CCLK/4
}
```

7.2.2 SPI for the mbed board

In this section it is exposed how the SPI module is configured in the mbed board. The registers which are modified are SPCR, SPCCR and SPDR. In table 7.2 it is shown bit-by-bit how the value of SPCR is set.

SPCCR controls the frequency of a master's SCK. The register indicates the number of SPI peripheral clock cycles that make up an SPI clock. In Master mode, this register must be an even number greater than or equal to 8. SPCCR has a value equal to 0xFA, in order the SCK to be 96kHz.

Finally, data register SPDR provides the transmit and receive data for the SPI. Transmit data is provided to the SPI0 by writing to this register.

Table 7.2: SPCR Register bit values and explanation, bits 0,1 are not used

Bit	Value	Explanation
Bit2	0	SPI controller sends 8 bits
Bit3	0	CPHA - Data is sampled on the first rising edge of CLK
Bit4	0	CPOL - CLK is active high
Bit5	1	The device is an SPI master
Bit6	0	SPI data is transferred MSB (bit 7 first)
Bit7	0	SPI interrupts are inhibited
Bit8	0	Bits 11:8 are 1000 respectively. This means that 8 bits are transferred
Bit9	0	
Bit10	0	
Bit11	1	

7.2.3 Learning and testing SPI for W5100 chip

As mentioned before, the W5100 chip supports SPI interface and behaves as an SPI slave. W5100 accepts two types of commands - Read and Write. These commands are 4 bytes long and permit to modify registers in the W5100. The first byte stands for the command, second and third for the address of the register to be modified and the forth for the data. Table 7.4 summarizes the two commands. The 4 bytes are transferred with the most significant bit(MSB) first and least significant bit(LSB) last [12].

Table 7.3: SPI Commands

Command	OP-Code Field		Address Field	Data Field
Write	0xF0	1111 0000	2 bytes	1 byte
Read	0x0F	0000 1111	2 bytes	1 byte

In order to test W5100 SPI interface, it has been used Microchip's MCP2210 USB-to-SPI Protocol Converter (see Figures 7.5 and 7.6). It is a general purpose SPI master. This chip comes with testing software called SPI Terminal (Figure 7.7).

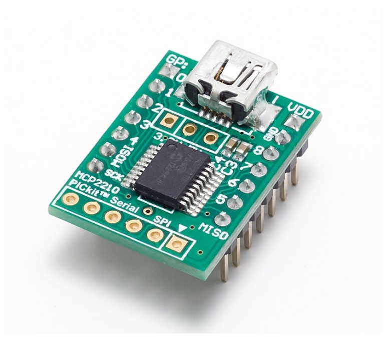


Figure 7.5: MCP2210 USB-to-SPI Protocol Converter

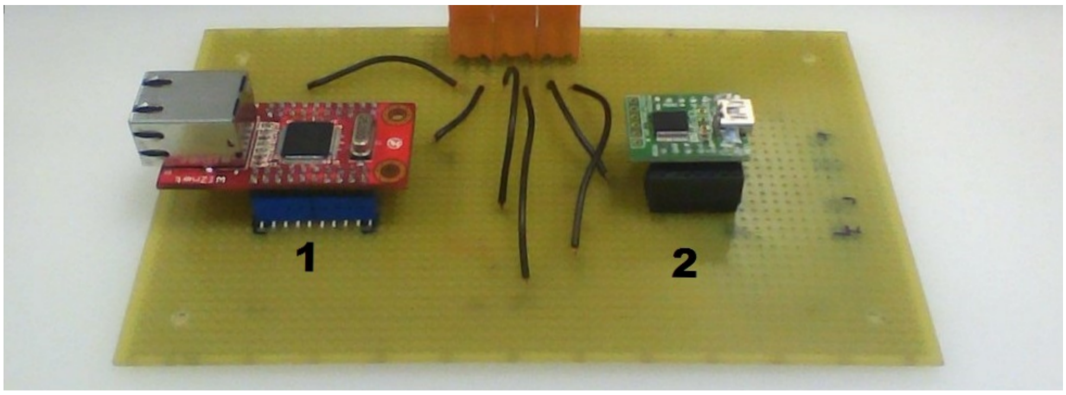
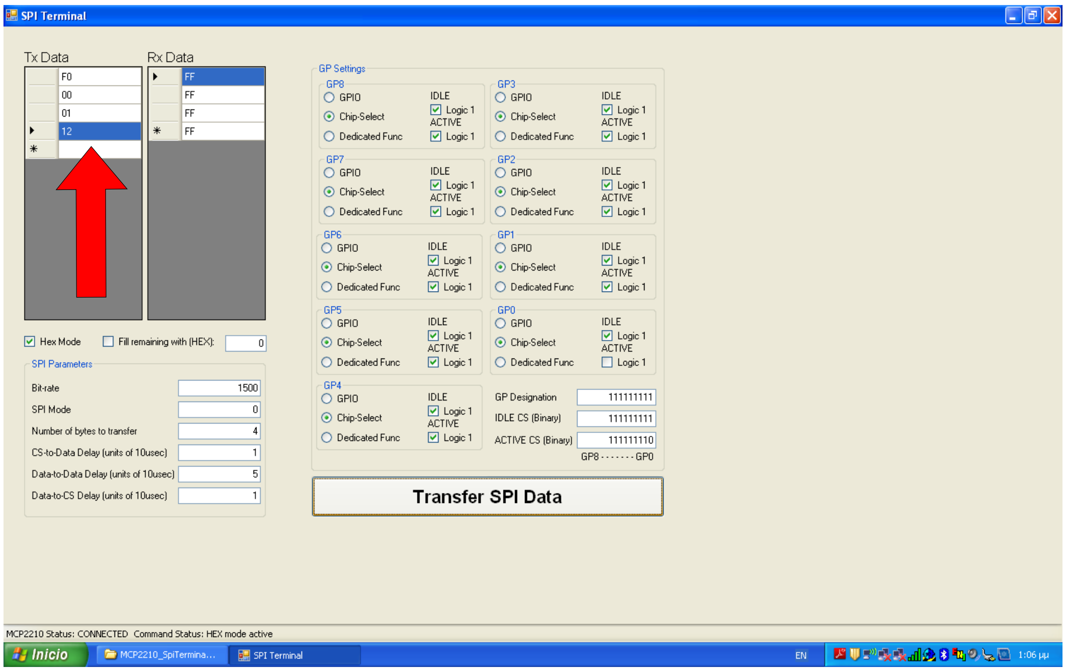


Figure 7.6: Testboard **1.**W5100 chip, **2.**MCP2210 USB-to-SPI Protocol Converter

Figure 7.8 shows the result of a write command captured with an oscilloscope. As it can be seen, the first byte is 0xF0 and stands for the operation command, then two bytes 0x00 and 0x01 stand for the address 0x01 of the W5100 memory and the last byte is 0x12 that stands for data. In annex A.2, it is also described an example of the Read operation. Figure 7.9 shows how the oscilloscope is connected to the testboard.



SPI Terminal

Tx Data	Rx Data
F0	FF
00	FF
01	FF
12	FF

GP Settings

	Mode	IDLE	ACTIVE
GP8	( ) GPIO [✓ Chip-Select] ( ) Dedicated Func	✓ Logic 1	✓ Logic 1
GP7	( ) GPIO [✓ Chip-Select] ( ) Dedicated Func	✓ Logic 1	✓ Logic 1
GP6	( ) GPIO [✓ Chip-Select] ( ) Dedicated Func	✓ Logic 1	✓ Logic 1
GP5	( ) GPIO [✓ Chip-Select] ( ) Dedicated Func	✓ Logic 1	✓ Logic 1
GP4	( ) GPIO [✓ Chip-Select] ( ) Dedicated Func	✓ Logic 1	✓ Logic 1
GP3	( ) GPIO [✓ Chip-Select] ( ) Dedicated Func	✓ Logic 1	✓ Logic 1
GP2	( ) GPIO [✓ Chip-Select] ( ) Dedicated Func	✓ Logic 1	✓ Logic 1
GP1	( ) GPIO [✓ Chip-Select] ( ) Dedicated Func	✓ Logic 1	✓ Logic 1
GP0	( ) GPIO [✓ Chip-Select] ( ) Dedicated Func	✓ Logic 1	☐ Logic 1

✓ Hex Mode ☐ Fill remaining with (HEX): 0

SPI Parameters

Bit-rate	1500
SPI Mode	0
Number of bytes to transfer	4
CS-to-Data Delay (units of 10usec)	1
Data-to-Data Delay (units of 10usec)	5
Data-to-CS Delay (units of 10usec)	1

GP Designation	111111111
IDLE CS (Binary)	111111111
ACTIVE CS (Binary)	111111110

GP8 - - - - - - - GP0

Transfer SPI Data

MCP2210 Status: CONNECTED Command Status: HEX mode active

Figure 7.7: Write operation example

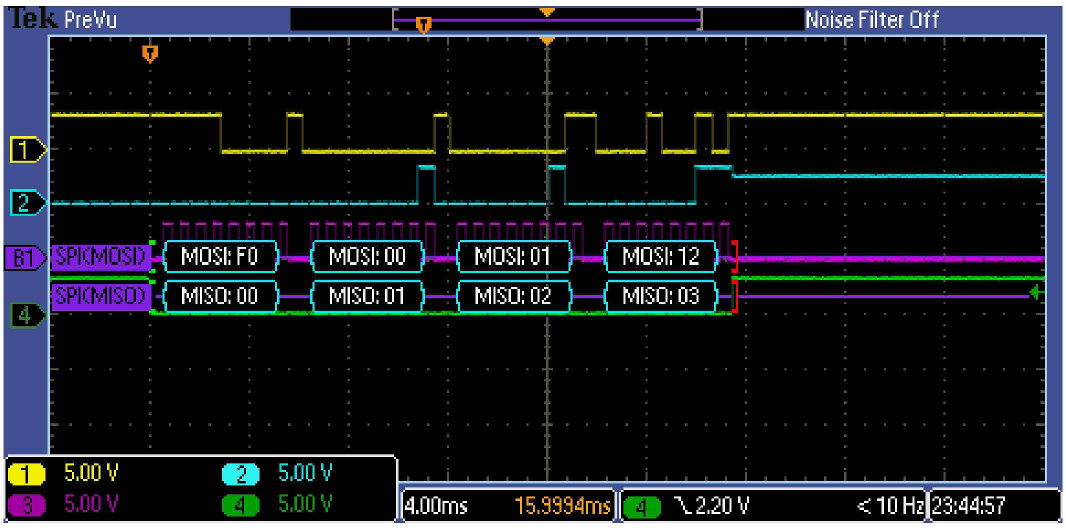


Figure 7.8: Example of write operation in oscilloscope

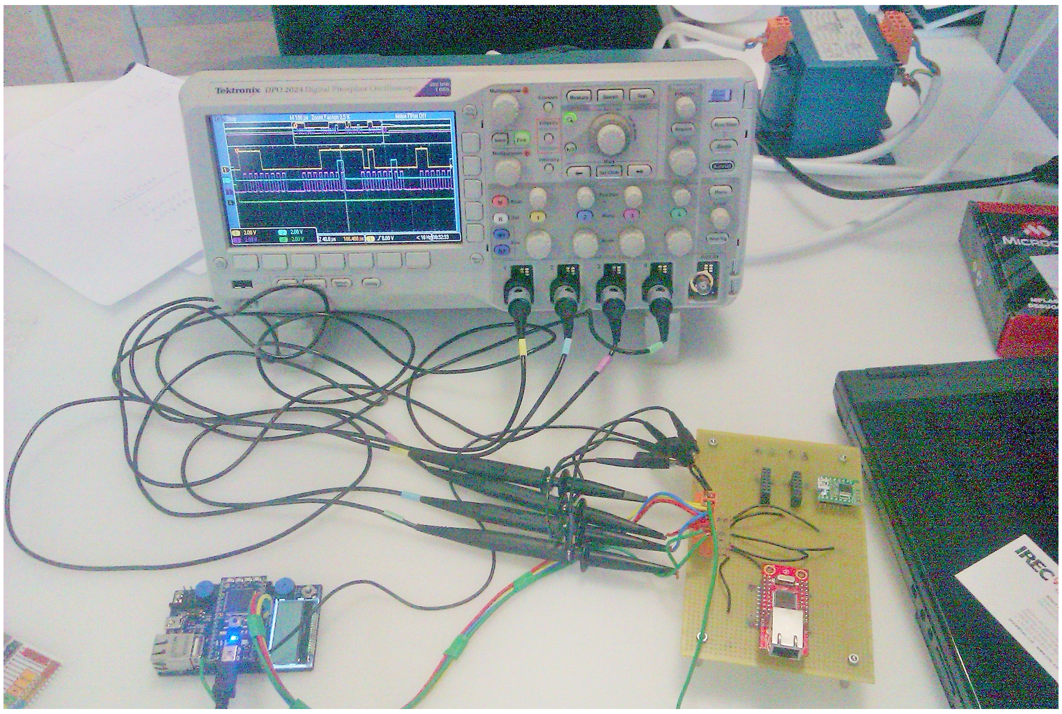


Figure 7.9: Connection between the mbed board, the W5100 chip and the oscilloscope

7.2.4 W5100 configuration

In order to configure the W5100, some common registers needed to be set up. The table 7.4 shows which values were written to the registers (in certain addresses) for the configuration of W5100.

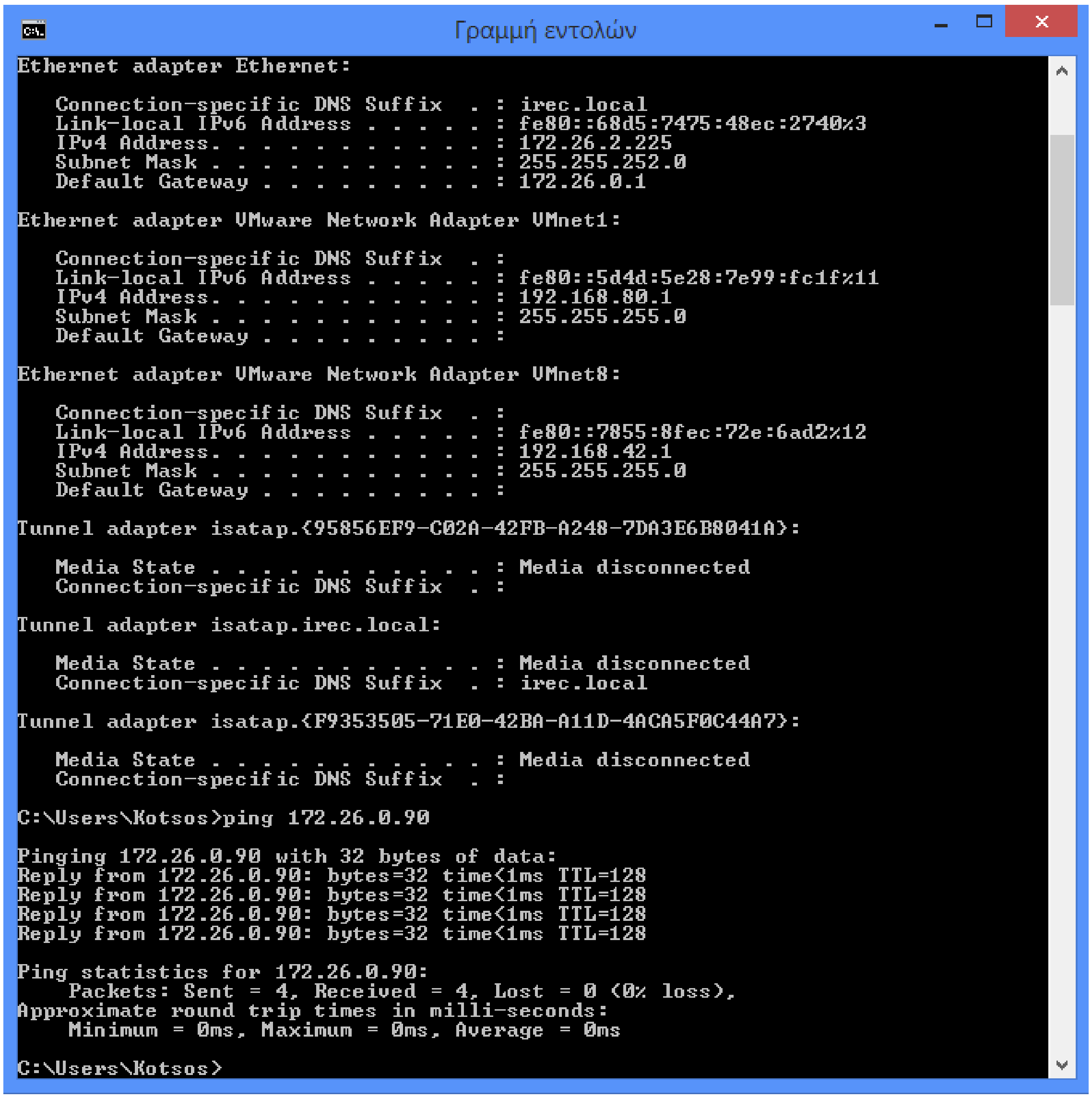
Figure 7.10 shows the result of a "ping" after the configuration of the W5100.

All the proper configurations for the SPI communication are shown in the part of code below.

Table 7.4: Basic Settings for the W5100

Register name	Address	Written Value	Explanation
Mode	0x00	0x80	Pin enabled
Interrupt Mask	0x16	0xC1	IP Conflict and Destination unreachable enabled
Retry Time-value	0x17 0x18	0x0F 0xA0	400ms timeout
Retry Count	0x19	0x08	The number of re-transmission
Gateway Address	0x01 0x02 0x03 0x04	0xAC 0x1A 0x00 0x01	The gateway address is 172.26.0.1
Source Hardware Address	0x09 0x0A 0x0B 0x0C 0x0D 0x0E	0x0A 0x0B 0x0C 0x0D 0x0E 0x0F	The MAC address of the Board is set as 0A:0B:0C:0D:0E:0F
Subnet Mask Address	0x05 0x06 0x07 0x08	0xFF 0xFF 0xFC 0x00	The Subnet Mask is 255.255.252.0
Source IP Address	0x0F 0x10 0x11 0x12	0xAC 0x1A 0x00 0x59	Source IP Address (mbed IP) is set as 172.26.0.89

```
spi_WriteToWiznet(0x00, 0x00, 0x80);//  *MR reset and
   ping enable

                           //   *IMR:IP Conflict BIT7
spi_WriteToWiznet(0x00, 0x16, 0xC1);//  *Destination
   unreachable BIT6
                           //   *Occurrence of Socket 0
                              Socket Interrupt BIT0

spi_WriteToWiznet(0x00, 0x17, 0x0F);//  *
spi_WriteToWiznet(0x00, 0x18, 0xA0);//  *RTR 400ms
   timeout

spi_WriteToWiznet(0x00, 0x19, 0x08);//  *RCR

spi_WriteToWiznet(0x00, 0x01, 0xAC);//172 *
spi_WriteToWiznet(0x00, 0x02, 0x1A);//26 *
spi_WriteToWiznet(0x00, 0x03, 0x00);//0  *
spi_WriteToWiznet(0x00, 0x04, 0x01);//1  *GAR

spi_WriteToWiznet(0x00, 0x09, 0x0A);//0A *
spi_WriteToWiznet(0x00, 0x0A, 0x0B);//0B *
spi_WriteToWiznet(0x00, 0x0B, 0x0C);//0C *
spi_WriteToWiznet(0x00, 0x0C, 0x0D);//0D *
spi_WriteToWiznet(0x00, 0x0D, 0x0E);//0E *SHAR random
   MAC address
spi_WriteToWiznet(0x00, 0x0E, 0x0F);//0F
   *0A.0B.0C.0D.0E.0F

spi_WriteToWiznet(0x00, 0x05, 0xFF);//255 *
spi_WriteToWiznet(0x00, 0x06, 0xFF);//255 *
spi_WriteToWiznet(0x00, 0x07, 0xFC);//252 *
spi_WriteToWiznet(0x00, 0x08, 0x00);//0  *SUBR
spi_WriteToWiznet(0x00, 0x0F, 0xAC);//172 *
spi_WriteToWiznet(0x00, 0x10, 0x1A);//26 *
spi_WriteToWiznet(0x00, 0x11, 0x00);//0  *
spi_WriteToWiznet(0x00, 0x12, 0x59);//89 *SIPR

spi_WriteToWiznet(0x00, 0x1A, 0x03);//  *RX MEM SIZE
   2kB RMSR 8kB TO SOCKET 0
spi_WriteToWiznet(0x00, 0x1B, 0x03);//  *RX MEM SIZE
   2kB RMSR 8kB TO SOCKET 0

spi_WriteToWiznet(0x04, 0x00, 0x02);//  *S0_MR = 0x02
   UDP MODE
a=spi_readFromAddress(0x04, 0x00);

spi_WriteToWiznet(0x04, 0x01, 0x01);//1  *S0_CR OPEN
   SOCKET 0

for(dummy = 0 ; dummy < 100 ; dummy++){
      // Waiting
}

spi_WriteToWiznet(0x04, 0x04, 0xD4);//  *
spi_WriteToWiznet(0x04, 0x05, 0x31);//  *S0_PORT 54321
```



```
Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . : irec.local
   Link-local IPv6 Address . . . . . : fe80::68d5:7475:48ec:2740%3
   IPv4 Address. . . . . . . . . . . : 172.26.2.225
   Subnet Mask . . . . . . . . . . . : 255.255.252.0
   Default Gateway . . . . . . . . . : 172.26.0.1

Ethernet adapter VMware Network Adapter VMnet1:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::5d4d:5e28:7e99:fc1f%11
   IPv4 Address. . . . . . . . . . . : 192.168.80.1
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :

Ethernet adapter VMware Network Adapter VMnet8:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::7855:8fec:72e:6ad2%12
   IPv4 Address. . . . . . . . . . . : 192.168.42.1
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :

Tunnel adapter isatap.{95856EF9-C02A-42FB-A248-7DA3E6B8041A}:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Tunnel adapter isatap.irec.local:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . : irec.local

Tunnel adapter isatap.{F9353505-71E0-42BA-A11D-4ACA5F0C44A7}:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

C:\Users\Kotsos>ping 172.26.0.90

Pinging 172.26.0.90 with 32 bytes of data:
Reply from 172.26.0.90: bytes=32 time<1ms TTL=128
Reply from 172.26.0.90: bytes=32 time<1ms TTL=128
Reply from 172.26.0.90: bytes=32 time<1ms TTL=128
Reply from 172.26.0.90: bytes=32 time<1ms TTL=128

Ping statistics for 172.26.0.90:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Users\Kotsos>
```

Figure 7.10: Pinging with the W5100 chip

7.3 Ethernet programming

NXP provides users of LPCXpresso with examples. The author found an example of the implementation of the Ethernet module (an example of an HTTP Server) and adapted it to the project's needs.

A test was implemented using an extra mbed board as a shown in Figure 7.11. The extra mbed board works as an Ethernet message sender. It was programmed to send messages periodically, using the online environment. The code for this operation is shown below.

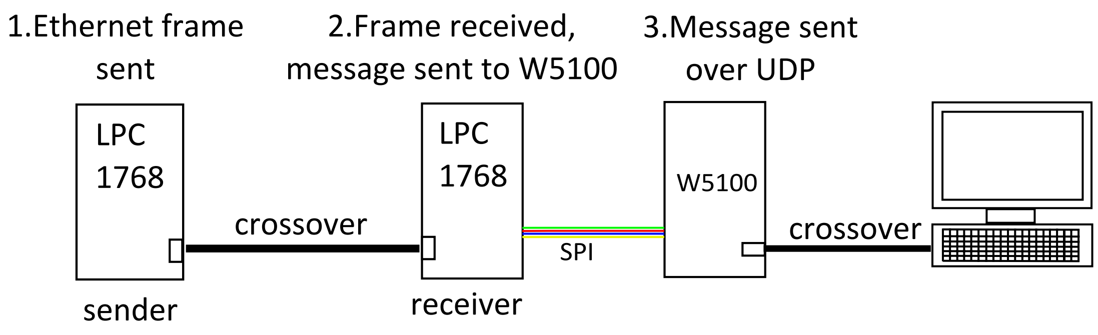


Figure 7.11: Ethernet testing

```
#include "mbed.h"

DigitalOut myled1(LED1);
DigitalOut myled2(LED2);
DigitalOut myled3(LED3);
DigitalOut myled4(LED4);


int main(){

    Ethernet eth;

    char TrData[100];
    char *p = TrData;

    eth.set_link(Ethernet::FullDuplex100);

    //set destination MAC, your server's MAC address or
       broadcast address, example shows broadcast
    TrData[0]=0x0A;
    TrData[1]=0x0B;
    TrData[2]=0x0C;
    TrData[3]=0x0D;
    TrData[4]=0x0E;
    TrData[5]=0x0F;
```

```
//set source MAC, the mbed's MAC address (use your own)
TrData[6]=0xA1;
TrData[7]=0x02;
TrData[8]=0xF7;
TrData[9]=0xF1;
TrData[10]=0xC8;
TrData[11]=0x74;

//set ethertype, just use IPv4, here is 0x0800
TrData[12]=0x00;
TrData[13]=0xAB;

//set the transmission data
TrData[14]=0x48;//H
TrData[15]=0x45;//E
TrData[16]=0x4C;//L
TrData[17]=0x4C;//L
TrData[18]=0x4F;//O
TrData[19]=0x5F;//_
TrData[20]=0x4D;//M
TrData[21]=0x41;//A
TrData[22]=0x4E;//N
TrData[23]=0x5F;//_
TrData[24]=0x48;//H
TrData[25]=0x4F;//O
TrData[26]=0x57;//W
TrData[27]=0x5F;//_
TrData[28]=0x41;//A
TrData[29]=0x52;//R
TrData[30]=0x45;//E
TrData[31]=0x5F;//_
TrData[32]=0x59;//Y
TrData[33]=0x4F;//O
TrData[34]=0x55;//U
```

```
    TrData[35]=0x3F;//?
    TrData[36]=0x3F;//?

    for(int i=37;i<99;i++){
        TrData[i]=0x58;//X
    }
    TrData[99]='Z';


    while(1) {
        wait(1);

           // Needed after startup.
        if(eth.link()){
                                                                       //
           Checking if there is Ethernet connection turn on
           LED4

        myled1=1;

        //send packet
        eth.write(p, sizeof(TrData));
        eth.send();

        wait(1);
        myled1=0;
        }
    }
}
```

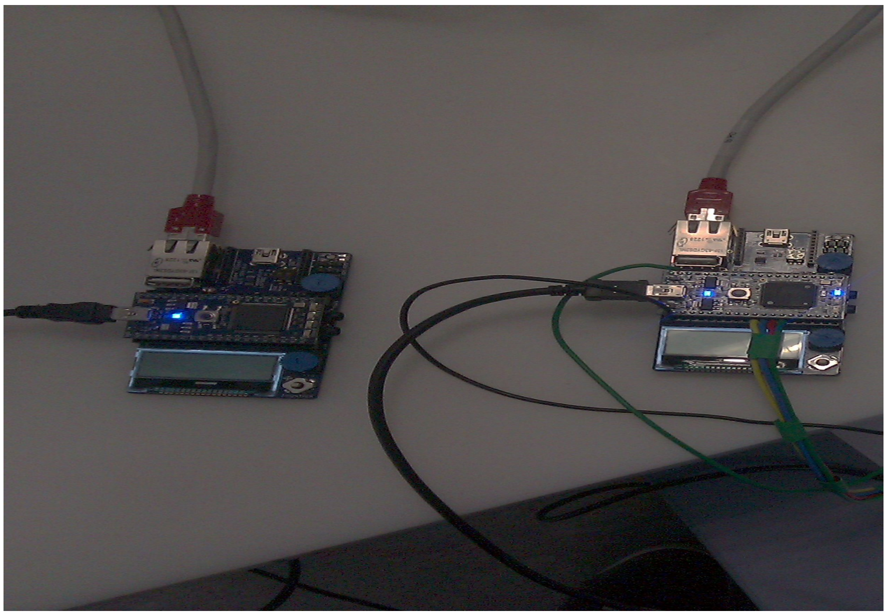
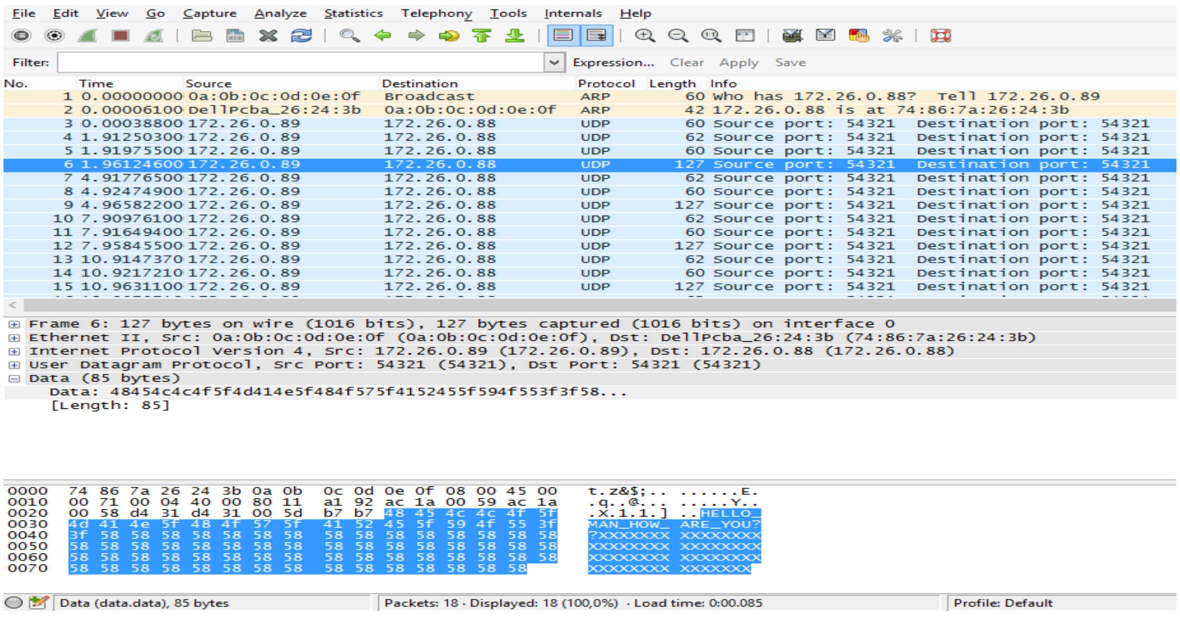


Figure 7.12: Board-to-Board Ethernet Communication


File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply Save

```
No.       Time        Source             Destination        Protocol Length Info
        1 0.00000000 0a:0b:0c:0d:0e:0f  Broadcast          ARP         60 Who has 172.26.0.88?  Tell 172.26.0.89
        2 0.00006100 DellPcba_26:24:3b  0a:0b:0c:0d:0e:0f  ARP         42 172.26.0.88 is at 74:86:7a:26:24:3b
        3 0.00038800 172.26.0.89        172.26.0.88        UDP         60 Source port: 54321  Destination port: 54321
        4 1.91250300 172.26.0.89        172.26.0.88        UDP         62 Source port: 54321  Destination port: 54321
        5 1.91975500 172.26.0.89        172.26.0.88        UDP         60 Source port: 54321  Destination port: 54321
        6 1.96124600 172.26.0.89        172.26.0.88        UDP        127 Source port: 54321  Destination port: 54321
        7 4.91776500 172.26.0.89        172.26.0.88        UDP         62 Source port: 54321  Destination port: 54321
        8 4.92474900 172.26.0.89        172.26.0.88        UDP         60 Source port: 54321  Destination port: 54321
        9 4.96582200 172.26.0.89        172.26.0.88        UDP        127 Source port: 54321  Destination port: 54321
       10 7.90976100 172.26.0.89        172.26.0.88        UDP         62 Source port: 54321  Destination port: 54321
       11 7.91649400 172.26.0.89        172.26.0.88        UDP         60 Source port: 54321  Destination port: 54321
       12 7.95845500 172.26.0.89        172.26.0.88        UDP        127 Source port: 54321  Destination port: 54321
       13 10.9147370 172.26.0.89        172.26.0.88        UDP         62 Source port: 54321  Destination port: 54321
       14 10.9217210 172.26.0.89        172.26.0.88        UDP         60 Source port: 54321  Destination port: 54321
       15 10.9631100 172.26.0.89        172.26.0.88        UDP        127 Source port: 54321  Destination port: 54321

⊞ Frame 6: 127 bytes on wire (1016 bits), 127 bytes captured (1016 bits) on interface 0
⊞ Ethernet II, Src: 0a:0b:0c:0d:0e:0f (0a:0b:0c:0d:0e:0f), Dst: DellPcba_26:24:3b (74:86:7a:26:24:3b)
⊞ Internet Protocol Version 4, Src: 172.26.0.89 (172.26.0.89), Dst: 172.26.0.88 (172.26.0.88)
⊞ User Datagram Protocol, Src Port: 54321 (54321), Dst Port: 54321 (54321)
⊟ Data (85 bytes)
    Data: 48454c4c4f5f4d414e5f484f575f4152455f594f553f3f58...
    [Length: 85]

0000  74 86 7a 26 24 3b 0a 0b  0c 0d 0e 0f 08 00 45 00   t.z&$;.. ......E.
0010  00 71 00 04 40 00 80 11  a1 92 ac 1a 00 59 ac 1a   .q..@... .....Y..
0020  00 58 d4 31 d4 31 00 5d  b7 b7 48 45 4c 4c 4f 5f   .X.1.1.] ..HELLO_
0030  4d 41 4e 5f 48 4f 57 5f  41 52 45 5f 59 4f 55 3f   MAN_HOW_ ARE_YOU?
0040  3f 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   ?XXXXXXX XXXXXXXX
0050  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
0060  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58 58   XXXXXXXX XXXXXXXX
0070  58 58 58 58 58 58 58 58  58 58 58 58 58 58 58      XXXXXXXX XXXXXXX

Data (data.data), 85 bytes | Packets: 18 · Displayed: 18 (100,0%) · Load time: 0:00.085 | Profile: Default
```

Figure 7.13: Wireshark result with message.

8 Final Application and Results

8.1 Description

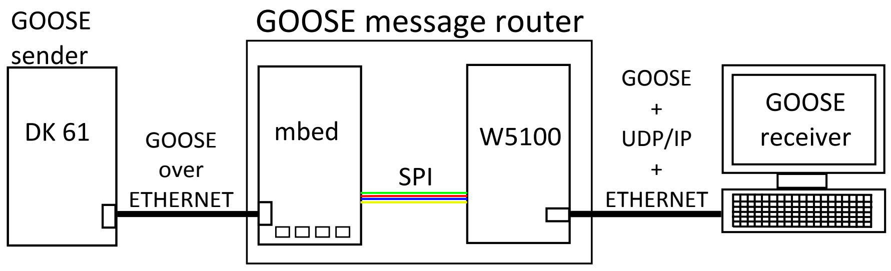


Figure 8.1: Functionality of the GOOSE router

Figure 8.1 shows the functionality of the GOOSE router developed. The GOOSE router receives GOOSE messages(which are mounted directly over Ethernet) and adds the proper UDP/IP header. Figure 8.2 shows the actual equipment used.

For the GOOSE sender it has been used the Development kit DK61 manufactured by Beck GmbH. The board includes the commercial IEC 61850 library PIS-10 (library version: 1.0) developed by SystemCorp. This server sends real GOOSE messages periodically to the GOOSE router. Details about the GOOSE messages generated are explained in annex A.3.

The GOOSE receiver is a computer, which uses WireShark to show the incoming GOOSE + UDP/IP message.

8.2 Final software

In figure 8.3 is depicted the UML diagram of the final software. At first, the initialization of the LEDs and the SPI Interface for the mbed board is executed. After that, the initialization of the W5100 and the Ethernet module. The W5100 socket needs some time to get into UDP mode and there is a possibility not to be ready on time. If it is not, then there is a short delay.

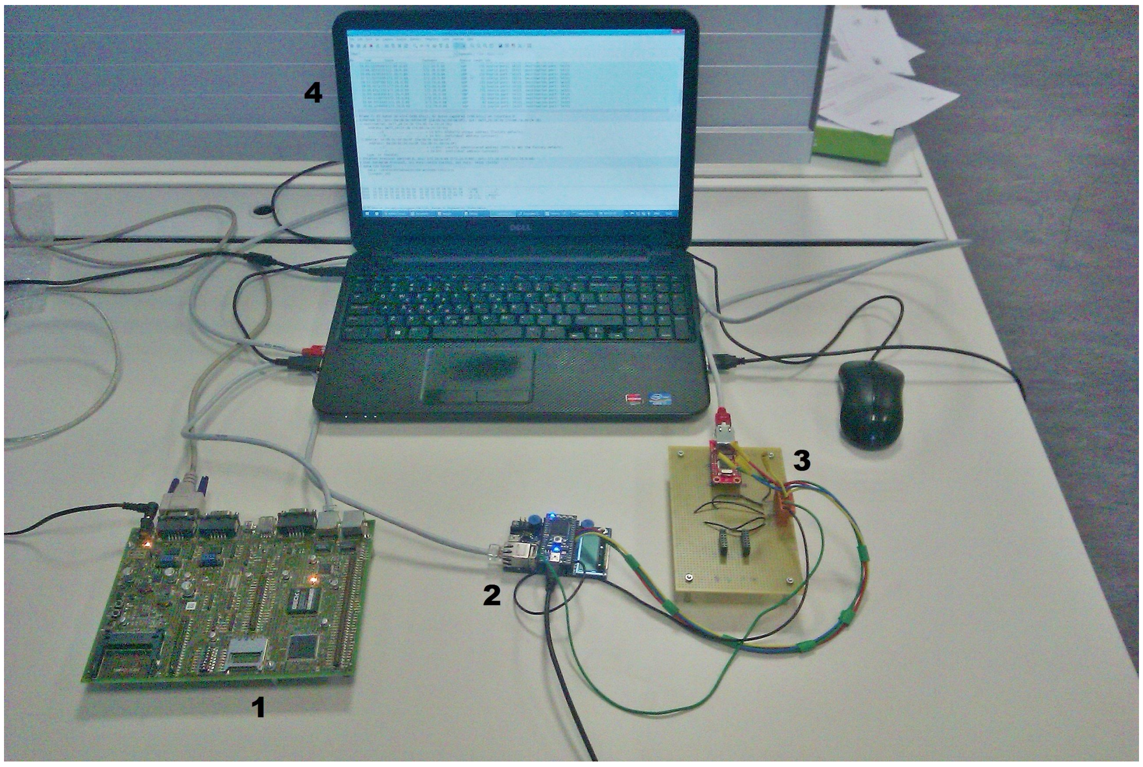


Figure 8.2: 1.DK61 GOOSE sender, 2.Mbed board, 3.W5100, 4.GOOSE+UDP/IP receiver

Next, the mbed Ethernet socket changes status to OPEN and ”listens” for incoming GOOSE messages. When a GOOSE message arrives, it is processed. The message is read byte by byte. The first 12 bytes containing source and destination hardware address are cut out.

Then a UDP message containing a message about the size of the incoming one is sent. Finally, UDP header is added to the GOOSE data part and the GOOSE + UDP/IP message is sent.

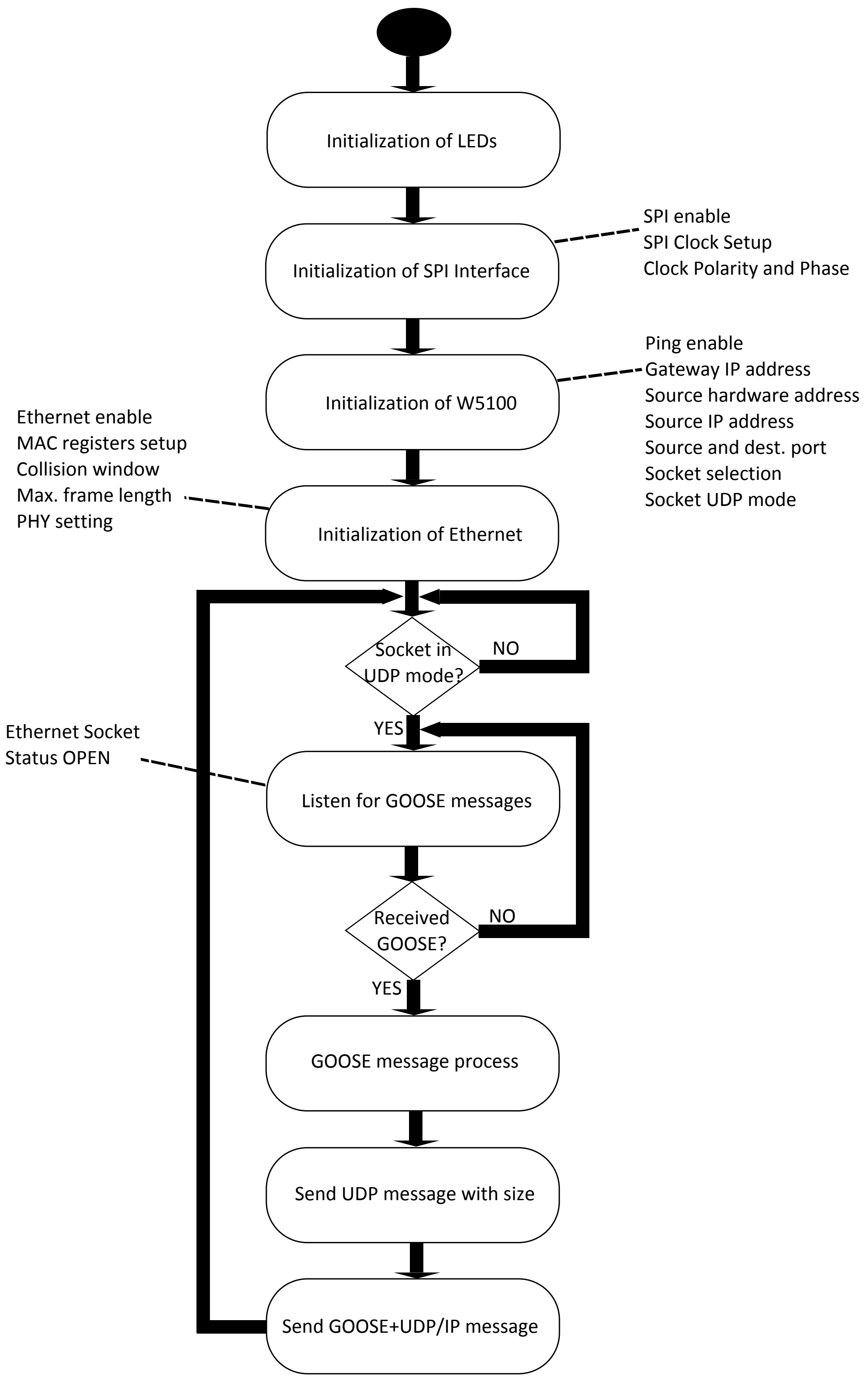
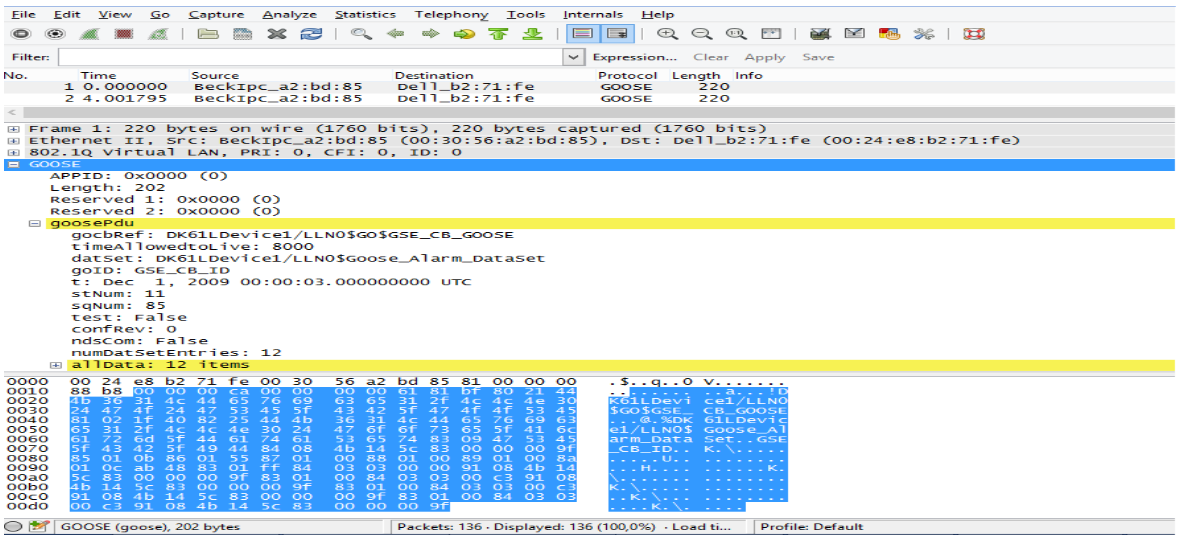


Figure 8.3: UML activity diagram

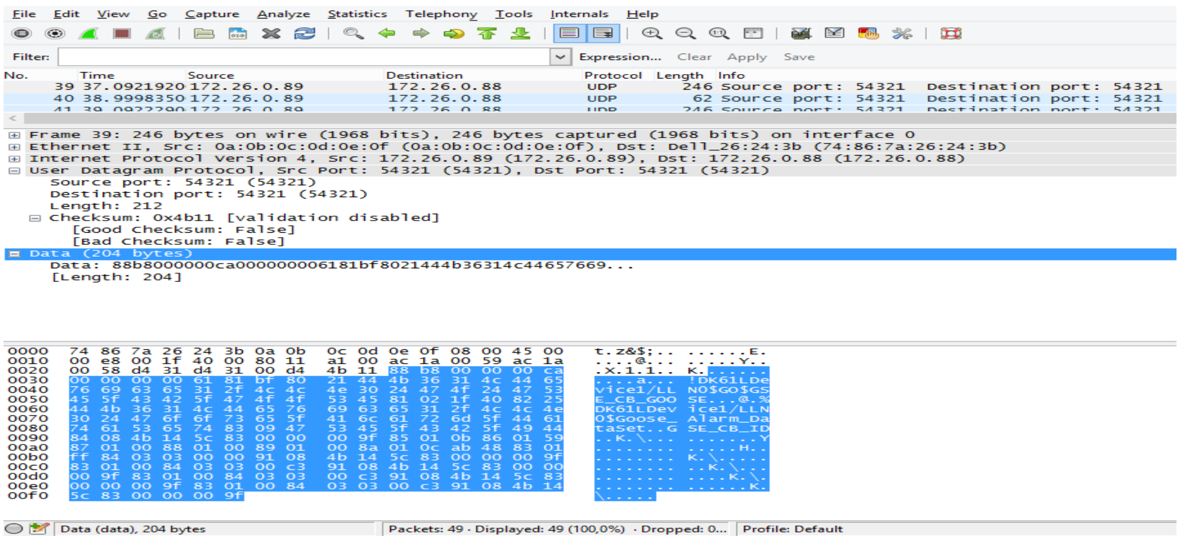
8.3 Final result

Figure 8.4, shows the WireShark capture of a GOOSE message as described by IEC 61850. This message is generated by the DK61 GOOSE sender. In figure 8.5, it is presented the WireShark capture of a routed GOOSE message.



```
No.      Time          Source                    Destination          Protocol  Length  Info
       1 0.000000      BeckIpc_a2:bd:85          Dell_b2:71:fe        GOOSE      220
       2 4.001795      BeckIpc_a2:bd:85          Dell_b2:71:fe        GOOSE      220

⊞ Frame 1: 220 bytes on wire (1760 bits), 220 bytes captured (1760 bits)
⊞ Ethernet II, Src: BeckIpc_a2:bd:85 (00:30:56:a2:bd:85), Dst: Dell_b2:71:fe (00:24:e8:b2:71:fe)
⊞ 802.1Q Virtual LAN, PRI: 0, CFI: 0, ID: 0
⊟ GOOSE
     APPID: 0x0000 (0)
     Length: 202
     Reserved 1: 0x0000 (0)
     Reserved 2: 0x0000 (0)
   ⊟ goosePdu
       gocbRef: DK61LDevice1/LLN0$GO$GSE_CB_GOOSE
       timeAllowedtoLive: 8000
       datSet: DK61LDevice1/LLN0$Goose_Alarm_DataSet
       goID: GSE_CB_ID
       t: Dec  1, 2009 00:00:03.000000000 UTC
       stNum: 11
       sqNum: 85
       test: False
       confRev: 0
       ndsCom: False
       numDatSetEntries: 12
     ⊞ allData: 12 items

0000  00 24 e8 b2 71 fe 00 30  56 a2 bd 85 81 00 00 00   .$..q..0 V.......
0010  88 b8 00 00 00 ca 00 00  00 00 61 81 bf 80 21 44   ........ ..a...!D
0020  4b 36 31 4c 44 65 76 69  63 65 31 2f 4c 4c 4e 30   K61LDevi ce1/LLN0
0030  24 47 4f 24 47 53 45 5f  43 42 5f 47 4f 4f 53 45   $GO$GSE_ CB_GOOSE
0040  81 02 1f 40 82 25 44 4b  36 31 4c 44 65 76 69 63   ...@.%DK 61LDevic
0050  65 31 2f 4c 4c 4e 30 24  47 6f 6f 73 65 5f 41 6c   e1/LLN0$ Goose_Al
0060  61 72 6d 5f 44 61 74 61  53 65 74 83 09 47 53 45   arm_Data Set..GSE
0070  5f 43 42 5f 49 44 84 08  4b 14 5c 83 00 00 00 9f   _CB_ID.. K.\.....
0080  85 01 0b 86 01 55 87 01  00 88 01 00 89 01 00 8a   .....U.. ........
0090  01 0c ab 48 83 01 ff 84  03 03 00 00 91 08 4b 14   ...H.... ......K.
00a0  5c 83 00 00 00 9f 83 01  00 84 03 03 00 c3 91 08   \....... ........
00b0  4b 14 5c 83 00 00 00 9f  83 01 00 84 03 03 00 c3   K.\..... ........
00c0  91 08 4b 14 5c 83 00 00  00 9f 83 01 00 84 03 03   ..K.\... ........
00d0  00 c3 91 08 4b 14 5c 83  00 00 00 9f               ....K.\. ....

GOOSE (goose), 202 bytes | Packets: 136 · Displayed: 136 (100,0%) · Load ti... | Profile: Default
```

Figure 8.4: WireShark capture with GOOSE message generated by the GOOSE sender



```
No.      Time         Source         Destination     Protocol  Length  Info
      39 37.0921920  172.26.0.89    172.26.0.88     UDP          246  Source port: 54321   Destination port: 54321
      40 38.9998350  172.26.0.89    172.26.0.88     UDP           62  Source port: 54321   Destination port: 54321
      41 39.0922290  172.26.0.89    172.26.0.88     UDP          246  Source port: 54321   Destination port: 54321

⊞ Frame 39: 246 bytes on wire (1968 bits), 246 bytes captured (1968 bits) on interface 0
⊞ Ethernet II, Src: 0a:0b:0c:0d:0e:0f (0a:0b:0c:0d:0e:0f), Dst: Dell_26:24:3b (74:86:7a:26:24:3b)
⊞ Internet Protocol Version 4, Src: 172.26.0.89 (172.26.0.89), Dst: 172.26.0.88 (172.26.0.88)
⊟ User Datagram Protocol, Src Port: 54321 (54321), Dst Port: 54321 (54321)
     Source port: 54321 (54321)
     Destination port: 54321 (54321)
     Length: 212
   ⊟ Checksum: 0x4b11 [validation disabled]
       [Good Checksum: False]
       [Bad Checksum: False]
⊟ Data (204 bytes)
     Data: 88b8000000ca00000000006181bf8021444b36314c44657669...
     [Length: 204]

0000  74 86 7a 26 24 3b 0a 0b  0c 0d 0e 0f 08 00 45 00   t.z&$;.. ......E.
0010  00 e8 00 1f 40 00 80 11  a1 00 ac 1a 00 59 ac 1a   ....@... .....Y..
0020  00 58 d4 31 d4 31 00 d4  4b 11 88 b8 00 00 00 ca   .X.1.1.. K.......
0030  00 00 00 00 61 81 bf 80  21 44 4b 36 31 4c 44 65   ....a... !DK61LDe
0040  76 69 63 65 31 2f 4c 4c  4e 30 24 47 4f 24 47 53   vice1/LL N0$GO$GS
0050  45 5f 43 42 5f 47 4f 4f  53 45 81 02 1f 40 82 25   E_CB_GOO SE...@.%
0060  44 4b 36 31 4c 44 65 76  69 63 65 31 2f 4c 4c 4e   DK61LDev ice1/LLN
0070  30 24 47 6f 6f 73 65 5f  41 6c 61 72 6d 5f 44 61   0$Goose_ Alarm_Da
0080  74 61 53 65 74 83 09 47  53 45 5f 43 42 5f 49 44   taSet..G SE_CB_ID
0090  84 08 4b 14 5c 83 00 00  00 9f 85 01 0b 86 01 59   ..K.\... .......Y
00a0  87 01 00 88 01 00 89 01  00 8a 01 0c ab 48 83 01   ........ .....H..
00b0  ff 84 03 03 00 00 91 08  4b 14 5c 83 00 00 00 9f   ........ K.\.....
00c0  83 01 00 84 03 03 00 c3  91 08 4b 14 5c 83 00 00   ........ ..K.\...
00d0  00 9f 83 01 00 84 03 03  00 c3 91 08 4b 14 5c 83   ........ ....K.\.
00e0  00 00 00 9f 83 01 00 84  03 03 00 c3 91 08 4b 14   ........ ......K.
00f0  5c 83 00 00 00 9f                                  \.....

Data (data), 204 bytes | Packets: 49 · Displayed: 49 (100,0%) · Dropped: 0... | Profile: Default
```

Figure 8.5: WireShark capture with UDP message containing the GOOSE message

9 Conclusions

In this work, we have implemented a GOOSE message router for PMUs, according to IEC 61850-90-5.

PMUs can play a key role in wide area monitoring system of the Smart Grid. Actually, it was a major topic in the last five IEEE PES Innovative Smart Grid Technologies.

In 2012, IEC 61850-90-5 was published. This is one chapter of IEC 61850 standard that defines how to transmit synchrophasor data according to IEEE C37.118. This standard proposes GOOSE messages and SV messages to exchange PMU information. Especially, it proposes routing GOOSE messages over UDP/IP protocol.

An ARM Cortex-M3 micro-controller has been programmed to receive GOOSE messages and send them through SPI to an Ethernet controller, the W5100. This Ethernet controller adds UDP/IP header to the GOOSE message and sends it through the internet.

The GOOSE router prototype covers all the primary constraints given in chapter 2. The fast LPC 1768 micro-controller contributes for fast data processing and SPI interface for faster data transmission. Also UDP/IP protocol is used for this reason. The prototype was based only on economic, manufactured components and all the software development tools used were open source. Finally, the author had to obtain skills for embedded software development, in order to complete the prototype.

The GOOSE router could be a positive impact to the Smart Grid, because it can help for a better monitoring of it, by receiving real time measurements, from distance. Having such capability reduces the chance of big black outs to occur. Moreover, can be used with protection relays.

Furthermore it could be applied to more futuristic applications for IEDs inside a Smart House. Another application could be the remote controlling of house appliances and other smart devices.

Annex A Learning Applications and Testing

A.1 GPIO programming

Examples of turning on LED1.

A.1.1 Developed with Assembly

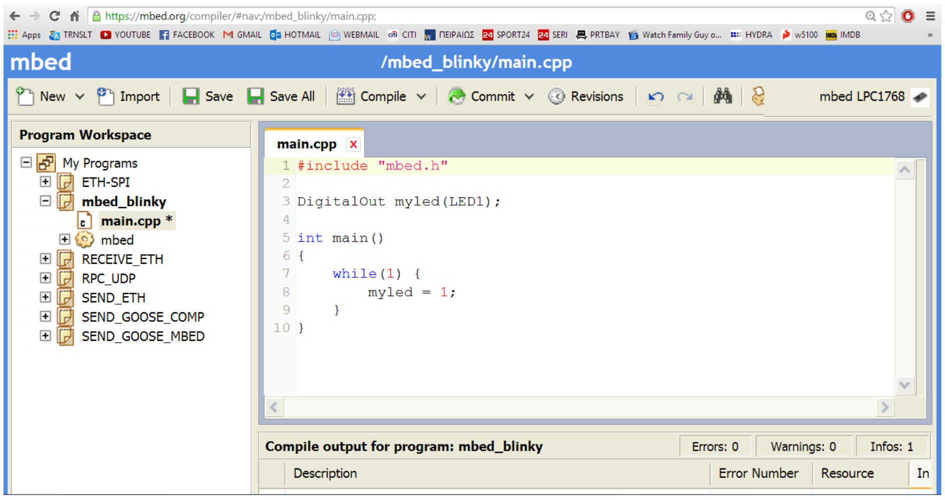
```
.equ PINSEL3, 0x4002C00C
.equ STACK_TOP, 0x20000800
.equ FIO1DIR2, 0x2009C022
.equ FIO1PIN2, 0x2009C036
.equ FIO1MASK2, 0x2009C032
.text
.syntax unified
.thumb
.global _start
.type start, %function
_start:
.word STACK_TOP, start

start:
   ldr r0, =PINSEL3//initialization of reg. PINSEL3 for
      GPIO port1.18
   mov r1, #0x00
   str r1, [r0]
   ldr r0, =FIO1DIR2 //reg. FIO1DIRx controls the
      direction of each port pin
   mov r1, #0xFF
   str r1, [r0]

   ldr r0, =FIO1MASK2 //reg. FIO1MASKx writes, sets,
      clears and reads
   mov r1, #0x00
   str r1, [r0]

   ldr r0, =FIO1PIN2//reg. FIO1PINx current state of
      digital port pins
   mov r1, #0xFF
   str r1, [r0]

loop: //endless loop
b loop
.end
```

A.1.2 Developed with C

```
int led1_on(unsigned char const port, unsigned char const
   pin){

   LPC_GPIO_TypeDef *gpio;

   gpio = LPC_GPIO1; //(LPC_GPIO_TypeDef *) (0x2009C000UL
      + 0x00020)

   gpio->FIODIR2 |= (1<<2);

   gpio->FIOMASK2 &= ~(1<<2);

   gpio->FIOPIN2 |= (1<<2);
```

```
    return 0;
}
```

A.1.3 Developed with the online Environment



```
#include "mbed.h"

DigitalOut myled(LED1);

int main()
{
    while(1) {
        myled = 1;
    }
}
```

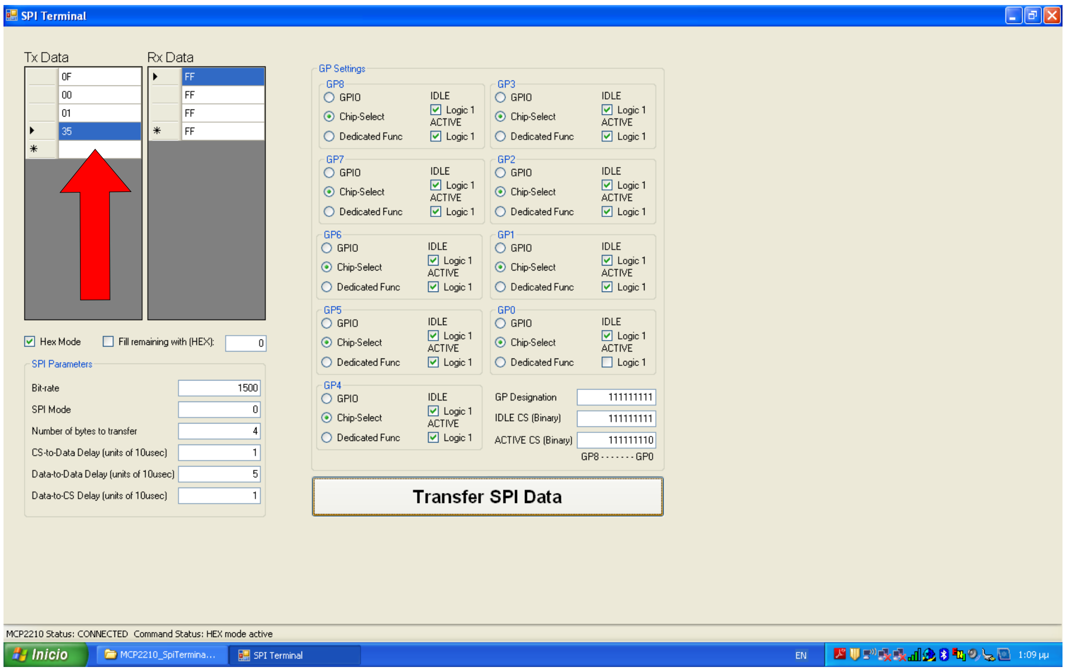
Figure A.1: LED1 on with online environment

A.2 SPI MCP2210 SPI Terminal

Read operation. The TX Data column contain the bytes wanted to be transmitted. The first byte is the read command, the next 2 bytes are for the memory address. The last byte is not affecting the result, so we can put any one-byte-value.

A.3 Final Test with GOOSE Server

The GOOSE Server generates messages about the status of some onboard switches. As shown in figure A.4, the statuses of the switches no. 5, 6, 7, and 8 are 1, 0, 0 and 1 respectively. Figure A.5 shows the GOOSE message with the status of the switches as True, False, False and True.



SPI Terminal

Tx Data

0F
00
01
35

Rx Data

FF
FF
FF
FF

[✓] Hex Mode [ ] Fill remaining with (HEX): 0

SPI Parameters

Bit-rate	1500
SPI Mode	0
Number of bytes to transfer	4
CS-to-Data Delay (units of 10usec)	1
Data-to-Data Delay (units of 10usec)	5
Data-to-CS Delay (units of 10usec)	1

GP Settings

	Mode	IDLE	ACTIVE
GP8	GPIO [✓ Chip-Select] Dedicated Func	[✓] Logic 1	[✓] Logic 1
GP7	GPIO [✓ Chip-Select] Dedicated Func	[✓] Logic 1	[✓] Logic 1
GP6	GPIO [✓ Chip-Select] Dedicated Func	[✓] Logic 1	[✓] Logic 1
GP5	GPIO [✓ Chip-Select] Dedicated Func	[✓] Logic 1	[✓] Logic 1
GP4	GPIO [✓ Chip-Select] Dedicated Func	[✓] Logic 1	[✓] Logic 1
GP3	GPIO [✓ Chip-Select] Dedicated Func	[✓] Logic 1	[✓] Logic 1
GP2	GPIO [✓ Chip-Select] Dedicated Func	[✓] Logic 1	[✓] Logic 1
GP1	GPIO [✓ Chip-Select] Dedicated Func	[✓] Logic 1	[✓] Logic 1
GP0	GPIO [✓ Chip-Select] Dedicated Func	[✓] Logic 1	[ ] Logic 1

GP Designation	111111111
IDLE CS (Binary)	111111111
ACTIVE CS (Binary)	111111110
	GP8 - - - - - - - GP0

Transfer SPI Data

MCP2210 Status: CONNECTED Command Status: HEX mode active

Figure A.2: Read operation, SPI Terminal

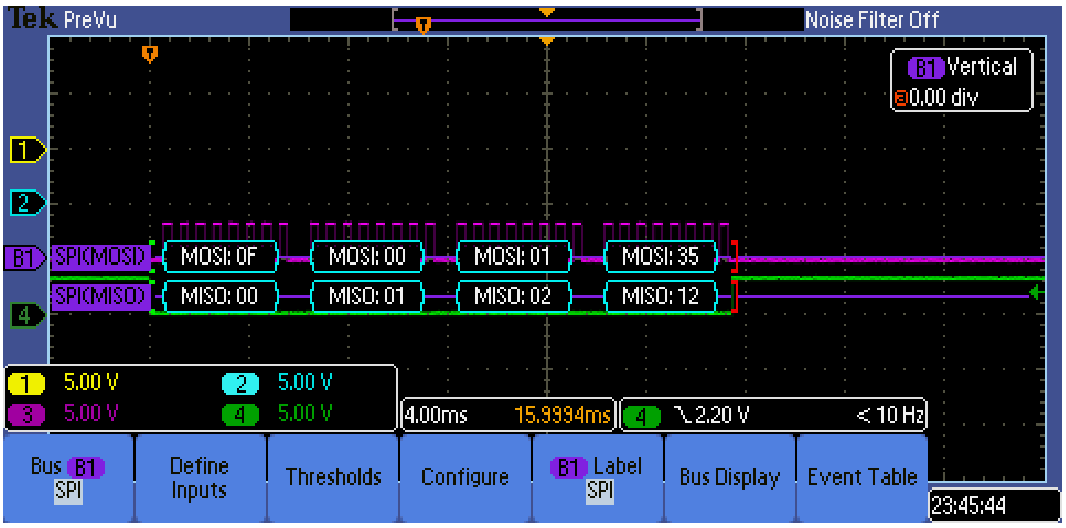


Figure A.3: Example of read operation in oscilloscope

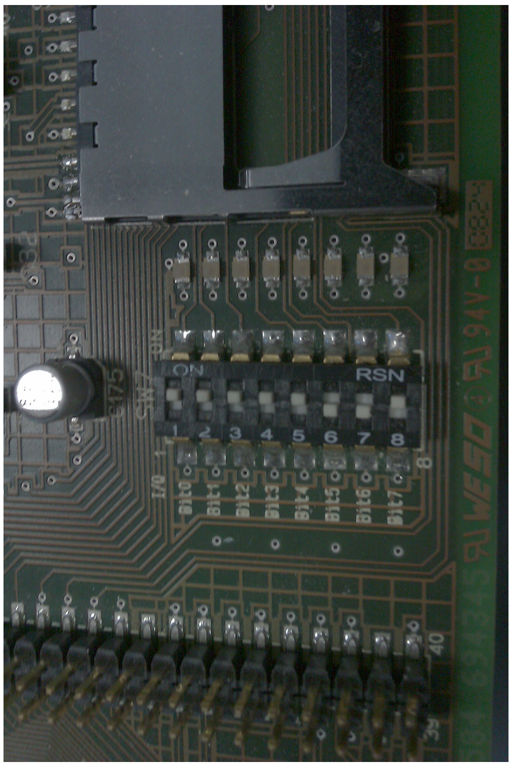
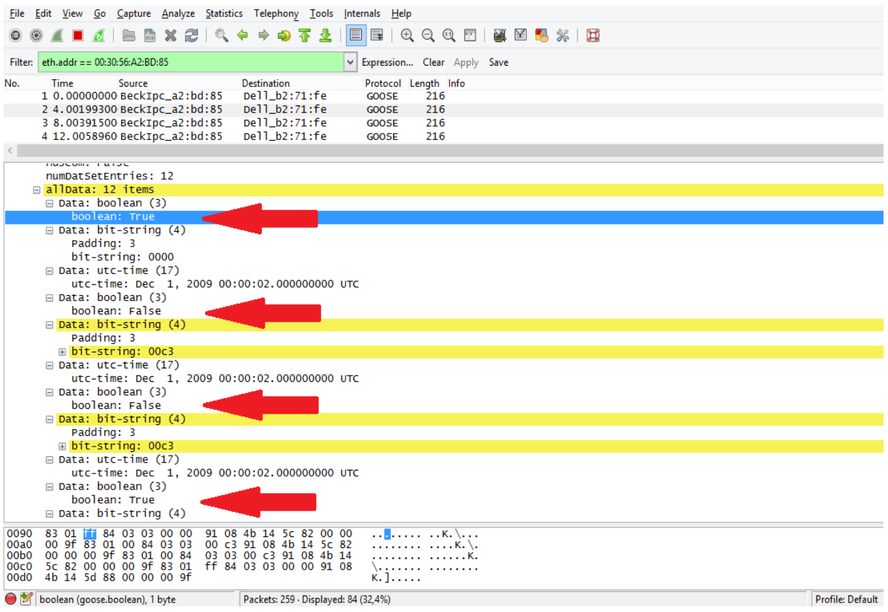


Figure A.4: Status of the switches


File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Expression... Clear Apply Save

```
Filter: eth.addr == 00:30:56:A2:BD:85

No.   Time       Source              Destination      Protocol  Length  Info
    1 0.00000000 BeckIpc_a2:bd:85    Dell_b2:71:fe    GOOSE      216
    2 4.00199300 BeckIpc_a2:bd:85    Dell_b2:71:fe    GOOSE      216
    3 8.00391500 BeckIpc_a2:bd:85    Dell_b2:71:fe    GOOSE      216
    4 12.0058960 BeckIpc_a2:bd:85    Dell_b2:71:fe    GOOSE      216

      numDatSetEntries: 12
    ⊟ allData: 12 items
      ⊟ Data: boolean (3)
          boolean: True
      ⊟ Data: bit-string (4)
          Padding: 3
          bit-string: 0000
      ⊟ Data: utc-time (17)
          utc-time: Dec  1, 2009 00:00:02.000000000 UTC
      ⊟ Data: boolean (3)
          boolean: False
      ⊟ Data: bit-string (4)
          Padding: 3
        ⊞ bit-string: 00c3
      ⊟ Data: utc-time (17)
          utc-time: Dec  1, 2009 00:00:02.000000000 UTC
      ⊟ Data: boolean (3)
          boolean: False
      ⊟ Data: bit-string (4)
          Padding: 3
        ⊞ bit-string: 00c3
      ⊟ Data: utc-time (17)
          utc-time: Dec  1, 2009 00:00:02.000000000 UTC
      ⊟ Data: boolean (3)
          boolean: True
      ⊟ Data: bit-string (4)

0090  83 01 ff 84 03 03 00 00  91 08 4b 14 5c 82 00 00   ........ ..K.\...
00a0  00 9f 83 01 00 84 03 03  00 c3 91 08 4b 14 5c 82   ........ ....K.\.
00b0  00 00 00 9f 83 01 00 84  03 03 00 c3 91 08 4b 14   ........ ......K.
00c0  5c 82 00 00 00 9f 83 01  ff 84 03 03 00 00 91 08   \....... ........
00d0  4b 14 5d 88 00 00 00 9f                            K.].....

boolean (goose.boolean), 1 byte    Packets: 259 · Displayed: 84 (32,4%)    Profile: Default
```

Figure A.5: WireShark capture of GOOSE message with the status of each switch

Bibliography

[1] J.-D. Lee, S.-J. Lee, J.-H. Bae, and D.-Y. Kwon, “The PMU Interface using IEC 61850,” in *ICT Convergence (ICTC), 2013 International Conference on*, 2013, pp. 1125–1128.

[2] A. Ruiz and I. Cairo, “KIC Instinct - Traffic Requirements of Microgrid Control Algorithms and IEC 61850,” Institut de Recerca en Energia de Catalunya, Tech. Rep., 2013.

[3] R. Mackiewicz, “Overview of IEC 61850 and Benefits,” in *Transmission and Distribution Conference and Exhibition, 2005/2006 IEEE PES*, 2006, pp. 376–383.

[4] D. Hart, D. Uy, V. Gharpure, D. Novosel, D. Karlsson, and M. Kaba, “Pmus - a new approach to power network monitoring,” 2001.

[5] J. Alonso-Zarate, J. Matamoros, D. Gregoratti, and M. Dohler, *Smart Grid Communications and Networking*. Cambridge University Press, 2012, ch. 6, p. 28.

[6] A. Ruiz and I. Cairo, “Substation Communications, Introduction to IEC 61850,” Institut de Recerca en Energia de Catalunya, Tech. Rep., 2013.

[7] IEC, “IEC 61850-7-2: Abstract Communication Service Interface,” IEC, Tech. Rep., 2003.

[8] ——, “IEC 61850-8-1: Mapping to MMS,” IEC, Tech. Rep., 2003.

[9] ——, “IEC 61850-9-1: Sampled Values Over Serial Unidirectional Multidrop Point to Point Link,” IEC, Tech. Rep., 2003.

[10] ——, “IEC 61850-9-2: Sampled Values over ISO/IEC 8802-3,” IEC, Tech. Rep., 2004.

[11] ARM, *The Definitive Guide to the ARM Cortex-M3 Second Edition*.

[12] WIZnet, *W5100 Datasheet Version 1.1.8*.

[13] NXP, *UM10360 LPC17xx User manual*.