TFTP loading of programs into a Microcontroller's flash memory and evaluation of Microchip's TCP/IP stack with ENC28J60

Kenan Alci

2014-05-28

Project for IK2553 performed at Department of Communication Systems

Examiner and academic adviser
Professor Gerald Q. Maguire Jr.

School of Information and Communication Technology (ICT)
KTH Royal Institute of Technology
Stockholm, Sweden

Abstract

This project began with a microprocessor platform developed by two master's students: Albert López and Francisco Javier Sánchez. Their platform was designed as a gateway for sensing devices operating in the 868 MHz band. The platform consists of a Texas Instruments MSP430F5437A microcontroller and a Microchip ENC28J60 Ethernet controller connected to the MSP430 processor by a Serial Peripheral Interface.

Javier Lara Peinado implemented prototype white space sensors using the platform developed by the earlier two students. As part of his effort, he partially implemented a Trivial File Transfer Protocol (TFTP) system for loading programs into the flash memory of the microcontroller using Microchip's TCP/IP stack. However, he was not successful in loading programs into the flash as the TFTP transfer got stuck at the first block.

The first purpose of this project was to find and fix the error(s) in the TFTP loading of programs into the MSP430's flash memory. The second purpose of this project was to evaluate Microchip's TCP/IP stack in depth. This report describes measurements of UDP transmission rates. Additionally, the TFTP processing rate is measured and the TFTP program loading code is documented. The report concludes with suggestions for possible improvements of this system.

Keywords: TFTP loading, MSP430 flash, IP stack evaluation

Sammanfattning

Projektet startade med en mikroprocessor-plattform som utvecklades av två masterstudenter: Albert López och Francisco Javier Sánchez. Deras plattform var utformad som en inkörsport för avkänning av apparater som arbetar i 868 MHz-bandet. Plattformen består av en Texas Instruments MSP430F5437A mikrokontroller och en Microchip ENC28J60 Ethernet controller ansluten till MSP430-processor med en SPI-gränssnitt (Serial Peripheral Interface).

Javier Lara Peinado genomförde prototypvitt utrymme sensoreranvända plattformen som utvecklades av de två tidigare nämnda studenter. Som en del av sitt arbete genomförde han delvis ett Trivial File Transfer Protocol (TFTP) system för lastning program i flashminne mikrokontroller med hjälp av Microchips TCP / IP-stack. Men han var inte framgångsrik i lastning program i flash som TFTP-överföringen fastnade vid det första blocket.

Det första syftet för detta projekt var att hitta och åtgärda felet(er) i TFTP laddning av program i MSP430 flashminne. Det andra syftet för detta projekt var att utvärdera Microchips TCP/IP- stack på djupet. I denna rapport beskrivs mätningar av UDP överföringshastighet. Dessutom mäts TFTP bearbetningshastighet och TFTP programladdningskoden dokumenteras. Rapporten avslutas med förslag på möjliga förbättringar av systemet.

Nyckelord: TFTP programladdning, MSP430 flashminne, IP-protokollstackenutvärdering

Table of contents

List of Figures

List of Tables

List of acronyms and abbreviations

CCS	Code Composer Studio
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
FET	Flash Emulation Tool
GUI	Graphical User Interface
IC	Integrated Circuit
IDE	Integrated Development Environment
IP	Internet Protocol
JTAG	Joint Test Action Group
LAN	Local Area Network
MCU	Microcontroller Unit
NIC	Network Interface Controller
OS	Operating System
PC	Personal Computer
PoE	Power over Ethernet
PSE	Power Sourcing Equipment
RAM	Random Access Memory
RISC	Reduces Instruction Set Computer
RRQ	Read Request
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
TCP	Transmission Control Protocol
TFTP	Trivial File Transfer Protocol
TI	Texas Instruments
TID	Transfer Identifier
UDP	User Datagram Protocol
USD	United States Dollar
WRQ	Write Request

1 Introduction

This chapter specifies the problems that were addressed in this project, the problems encountered during the project, the goals of the project, and a brief overview of the objectives of the project.

1.1 Problem description

This project began with a microprocessor platform developed by Albert López and Francisco Javier Sánchez as part of their master's thesis project[1]. Their platform was designed as a gateway for sensing devices operating in the 868 MHz band. The platform consists of a Texas Instruments MSP430F5437A microcontroller unit (MCU) [2] and a Microchip ENC28J60 Ethernet controller[3] connected to the MSP 430 processor by a Serial Peripheral Interface (SPI).

Javier Lara Peinado implemented prototype white space sensors using the platform developed by the earlier two students[4]. As part of his effort he partially implemented a Trivial File Transfer Protocol (TFTP) based bootloader to load programs into the MCU's flash memory using Microchip's TCP/IP stack. However, he was not successful in loading programs into the flash memory as the TFTP transfer got stuck at the first block.

The first purpose of this project was to find and fix the error(s) in the TFTP loading of programs into the MSP430's flash memory. Due to this, a user is unable to easily load new software into the processor. Instead, the user must manual program each board using a Joint Test Action Group (JTAG) programmer. This makes it much harder to develop and deploy applications for this platform.

The second purpose of this project was to evaluate Microchip's TCP/IP stack in depth. The reason for this examination is that the MCU is connected to the Ethernet controller by an SPI interface. This means analyzing and documenting the system's performance and if possible identifying bottlenecks. For example, does this SPI's data rate limit the performance of the processor's maximum sending and receiving data rates. As part of this evaluation measurements of UDP transmission rates were made.

Additionally, the TFTP processing rate was measured and the TFTP program loading code was documented.

1.2 Goals

The main goal was to solve the TFTP loading issue in order to improve the usability of the system, specifically to make it easier to write and deploy new applications, such as the test programs to be used to assess the performance of the platform's TCP/IP stack. This lead to the following subtasks:

- Solve the TFTP loading problem – so that programs could be loaded from a TFTP server into the MCU's flash memory,
- Measuring transfer rates with different configurations of the platform,
- Identify bottlenecks in the system, and
- Suggest improvementsto the system.

1.3 Structure of this report

This report exists of five chapters. The first chapter introduced the purpose of the project, stated the project's goals, and defined a series of subtasks. The second chapter provides the readers background information concerning what has already been done and what the reader needs to know in order to understand this report. The third chapter explains the methods and approaches to be used to solve the problems. The fourth chapter evaluates what was done and gives a comprehensive analysis of the measurement results. Finally, the last chapter summarizes our conclusions, describes what was not achieved, suggests future work that could lead to improvements, and reflects upon several issues related to the project.

2 Background

This chapter provides the reader with a survey of related work. This is followed by a description of two protocols (DHCP and TFTP) to allow the reader to better understanding the content of this report.

2.1 What others already have done

As stated in the introduction, this project builds upon previous projects. This section discusses what these previous students did in more detail.

2.1.1 Exploiting wireless sensors

Albert López and Francisco Javier Sánchez developed a gateway to sniff wireless sensor traffic in the 868 MHz band in order to use this data for multiple purposes[1]. The main component of the motherboard is a Texas Instruments' (TI) MSP430F5437A MCU[2]. This MCU was developed for ultra-low power applications. For network connectivity, they used an Ethernet controller. A Microchip ENC28J60[3] Ethernet controller was chosen due its Serial Peripheral Interface (SPI)[5] enabling it to communicate with the MSP430 MCU. An additional advantage of using this Ethernet controller is that there is no need for an external memory as the Ethernet controller integrates a dual port Random Access Memory (RAM) buffer for receiving and sending data packets. Due to the low power consumption of this platform (motherboard and radio daughterboard), the motherboard was designed so that it could be powered by Power over Ethernet (PoE)[6][7]. In addition to the motherboard, they developed a daughterboard with a radio transceiver for the 760 – 928 MHz band that also connects to the MCU via an SPI interface.

2.1.2 Minding the spectrum gaps

Javier Lara Peinado[4]use the two boards developed by López and Sánchez and added network booting functionality. The goal was to have a boot program stored in the flash memory of the processor that upon power up would use the Dynamic Host Configuration Protocol (DHCP) to:(1) get an IP address, (2) learn the name of a file to be loaded and executed, and (3) learn the IP address of the file server from which this file could be retrieved using the Trivial File Transfer Protocol (TFTP).Furthermore, the complete configuration of the gateway was done by means of DHCP options[8], while the installation of software to be run was to be done by TFTP. Unfortunately, he did not complete the implementation of using TFTP to load the code into the MCU's flash memory. However, he did implement software that scans the radio spectrum over a programmed range for "gaps", i.e., white spaces where no devices are transmitting. These measurements of the spectrum occupancy are sent to a server via UDP datagrams.

2.1.3 Fixing the PoE functionality

Julia Alba Tormo Peiró in her master thesis using a number of white space sensors[9]needed to address a problem with the PoE functionality as the PoE power subsystem of the motherboard was not providing enough power to runs the radio scanning process continuously. She successfully fixed this issue and was able to carry out white space sensing with a number of the motherboards together with their daughter board.

2.1.4 Smart Door Lock

Rafid Karim and Haidara Al-Fakhri utilized the motherboard and an existing near field communication board (designed as an Arduino shield) to build a prototype of a network powered NFC capable door lock[10].The main idea of their bachelor's thesis project was to simplify the user's life. For example, a homeowner could send one-time key to a repairperson or give two weeks access to his/her neighbor while he/she is on vacation so the neighbor can water the plants.

2.2 Dynamic Host Configuration Protocol

The Dynamic Host Configuration Protocol (DHCP) is a network protocol that describes how a computer can dynamically obtain network settings from a DHCP server[8], [11]. The DHCP protocol is based on the Internet Protocol (IP) [12] and works with User Datagram Protocol (UDP) [13] packets. The main feature of DHCP protocol is that it reduces the need for human interaction each time a client joins the network. This protocol is used by the embedded platform to connect to the network.

2.3 Trivial File Transfer Protocol

Trivial File Transfer Protocol[14] is a protocol that uses UDP to transfer files. It was first defined in January 1980 by Karen R. Sollins in IEN 133[15]and revised in July 1992 by Karen R. Sollins in RFC 1350 [16].

The simplicity of this protocol is the main reason for its usage in our project. This protocol was designed to be small and easy to implement. The only functionality of TFTP is to read and write files from/to a remote server. The protocol is similarities to other Internet protocols in passing 8-bit bytes of data.

Every transfer begins with a request to read or write a file. A response from the server indicates an open connection between the client and server. Each data packet that is send has a fixed length blocks of 512 bytes that has to be acknowledged by the receiver. When a packet is sent with less than 512 bytes this means that it is the last data packet. A timeout will occur at the recipient when a packet is lost in the network. It is up to the receiver to ask for a retransmission of the packet by the sender. Because of this stop-and-wait protocol, TFTP provides flow control and eliminates the need of reordering the incoming packets.

Almost all errors cause a termination of the connection. An error is signaled by an error packet, which does not have to been acknowledged or retransmitted. There are three types of events that cause errors: (1) not being able to satisfy the request (e.g., file not found, access violation, or no such user), (2) receiving a packet which cannot be explained by a delay or duplication in the network (e.g., an incorrectly formed packet), and (3) losing access to a necessary resource (e.g., disk full or access denied during a transfer). The only case where an error does *not* cause a termination of the connection is when the source port of a received packet is incorrect. In this case, an error packet is sent to the originating host.

2.3.1 Structure of a packet

Since TFTP was designed to be implemented on the top of the UDP, the datagram is carried inside an Internet Protocol packet. The resulting packet has an IP header, a UDP header, a TFTP header, and the TFTP data being sent. In addition, a link layer header is added by the interface to allow the packet to be delivered to its destination. TFTP does not specify any values in the IP header; however, TFTP does set some specific values in the UDP header. The UDP header has four fields. The UDP source and destination ports indicate the UDP ports

used by the sender and receiver. The datagram's length reflects the length of the packet. The optional checksum can be used to detect errors, which may have been occurred. Transfer Identifiers (TIDs) are used for the port numbers in the UDP datagram.

2.3.2 Initial connection

A TFTP client initially sends a write request (WRQ) or read request (RRQ) and expects to receive an ACK for a WRQ or the first data packet in response to a RRQ. This initial communication establishes a transfer. ACK packets contain the block number of the data packet that is being acknowledged. The block numbers begin with one and are incremented for each successive data block. The block number of a positive response to the first write request will be zero.

TIDs are randomly chosen at each end of the connection so probability that the same number is chosen by two clients is very low. These TID's are used for the UDP source and destination ports. A requesting host sends its initial request to the well-known UDP port number 69 of the serving host. The response of the server to the request is a TID chosen by the server itself as its source TID, while source TID from the request message by the requestor is used as the destination TID. This pair of TIDs are used until the transfer ends.

2.3.3 TFTP packets

TFTP has five types of packets with an opcode for each type (see Table 2-1).

Table 2-1: TFTP opcodes

Opcode	Operation
1	Read request (RRQ)
2	Write request (WRQ)
3	Data (DATA)
4	Acknowledgement (ACK)
5	Error (ERROR)

Figure 2-1 shows the read request and write request packet format. The first field is the opcode field (Op #). The opcode indicates if this is a RRQ or WRQ packet. The Filename field contains "octet", "netascii", or "mail". In our case the sender and recipient use netascii mode. When netascii mode is used, the host translates the data in the Filename field into its own string format. The filename field is followed by a byte containing a zero (0) and a mode field. Themode field makes it possible to define other modes of cooperating between pairs of hosts. Because, there is no central authority this must be done with care. A byte containing the value 0 indicates the end of a RRQ or WRQ packet.

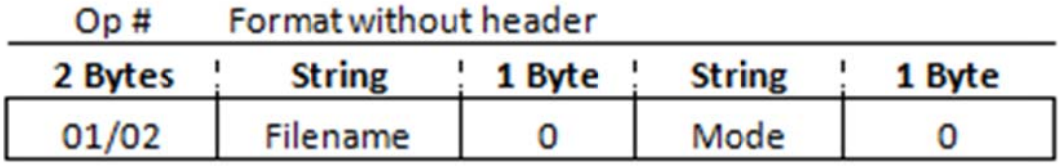


Figure 2-1: RRQ/WRQ

DATA packets transfer the actual data. Figure 2-2 shows the structure of a DATA packet (opcode = 3). The block number begins with one and increments each time a new block of data is send. The data field contains the actual data and is 512 bytes long; if not, this means that data block is the last block.

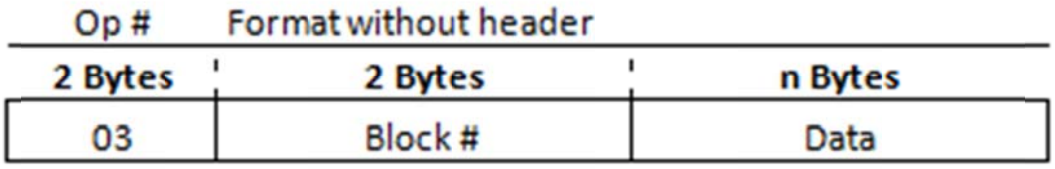


Figure 2-2: DATA packet

Every data packet should be acknowledged to ensure the consistency of the transfer and to enable the other party to send/request the next block. This is done by sending an ACK packet with the opcode 4. Figure 2-3 shows the structure of an ACK packet. The block number in this ACK indicates the block number of the DATA packet being acknowledged. At the start of a transfer a WRQ is acknowledged with block number of zero.

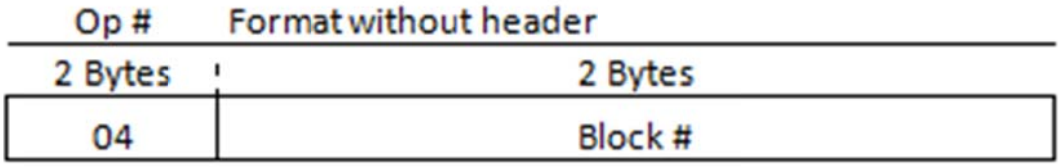


Figure 2-3: ACK packet

Figure 2-4shows the structure of an ERROR packet. The opcode of an ERROR packet is 5. The error code field indicates the type of error. The error message is in netascii and the string and the packet ends with a zero byte.

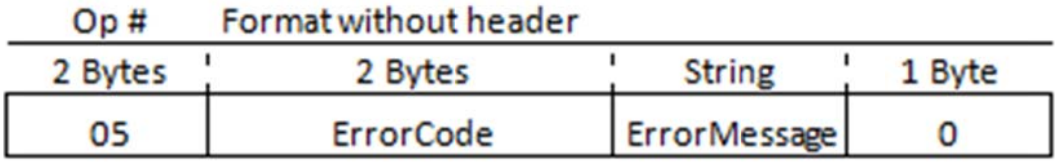


Figure 2-4: ERROR packet

3 Method

This chapter explains how we will achieve the goals of this project. Additionally, the tools that used to realize these goals are discussed.

3.1 Objectives

Several sub goals were defined for this project based upon the goals of the project (as described in Section 1.2). These sub goals are divided into two sets:

1. TFTP boot loading:
    - Connecting the board to the network in such a way that each board has its own IP address,
    - Detect the cause of the TFTP loading problem, and
    - Finally, applying the best solution to solve this problem.
2. IP stack evaluation:
    - Measuring the transfer rate from the MCU to a remote PC (located on the same isolated local area network) and
    - Measuring Ethernet controller buffer to PC transfer rate.

3.2 Hardware

This section discusses the hardware used in this project.

3.2.1 Motherboard

As stated earlier the motherboard has been used in a number of projects (previously described in Section 2.1). The motherboard uses one SPI interface to connecting a daughterboard. This enables the user to attach a new daughterboard without needing to change any other part of the motherboard. The first daughterboard was a radio module for the 868 MHz band (see Section 2.1.1). The second daughterboard was an Arduino NFS shield(see Section 2.1.4).

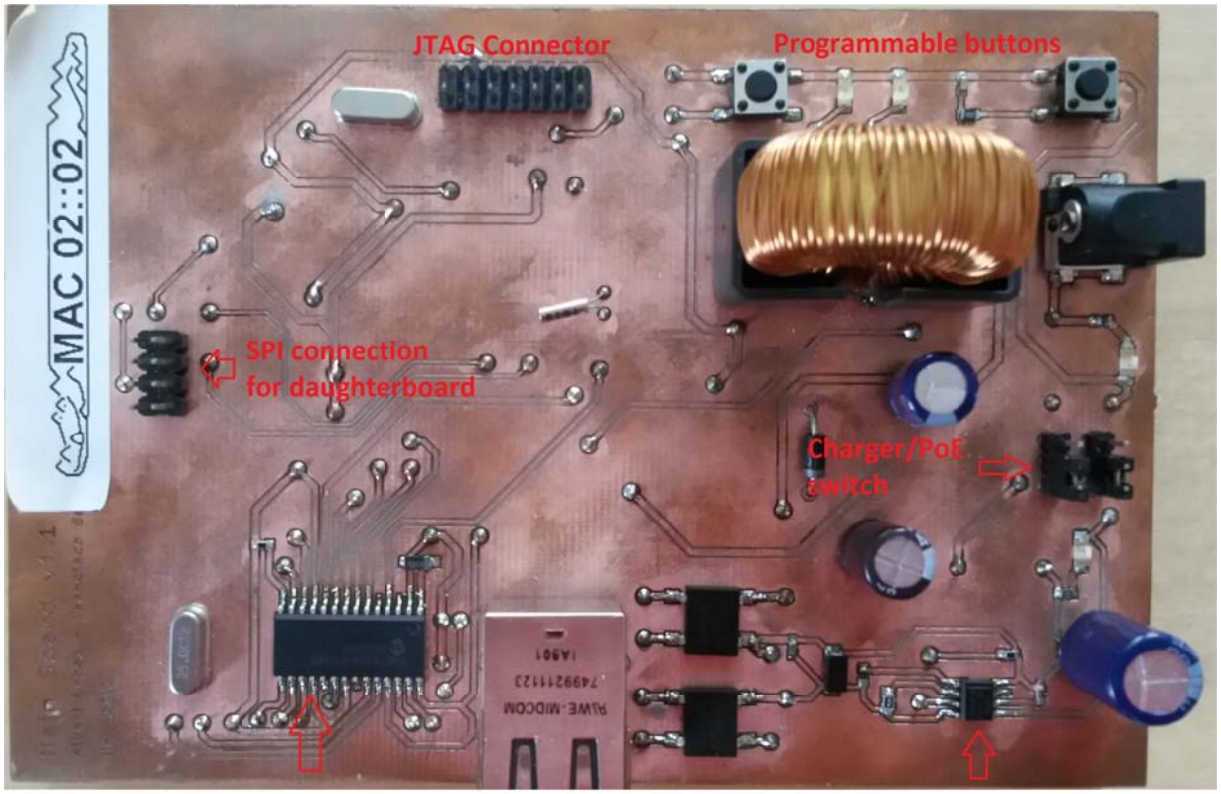
Figure 3-1 shows the front and back of the motherboard. The board consists of two means of powering the supplies powering, processing, networking, and in interface to a daughterboard. This motherboard together with an optional daughter card is an embedded networked computing platform. The motherboard can be powered by an external DC power supply or via PoE. The selection of the power source is up to the user by changing the position of the jumper to choose the desired option. The board can work with any DC supply that provides power between 3.3V and 60V because of the TL2575HV step-down converter[17].PoE is the preferred energy source because the platform will typically be connected to a PoE capable Ethernet switch* to provide both power and network connectivity.

* This switch will act as Power Sourcing Equipment (PSE). Alternative a PoE power injector could be used.

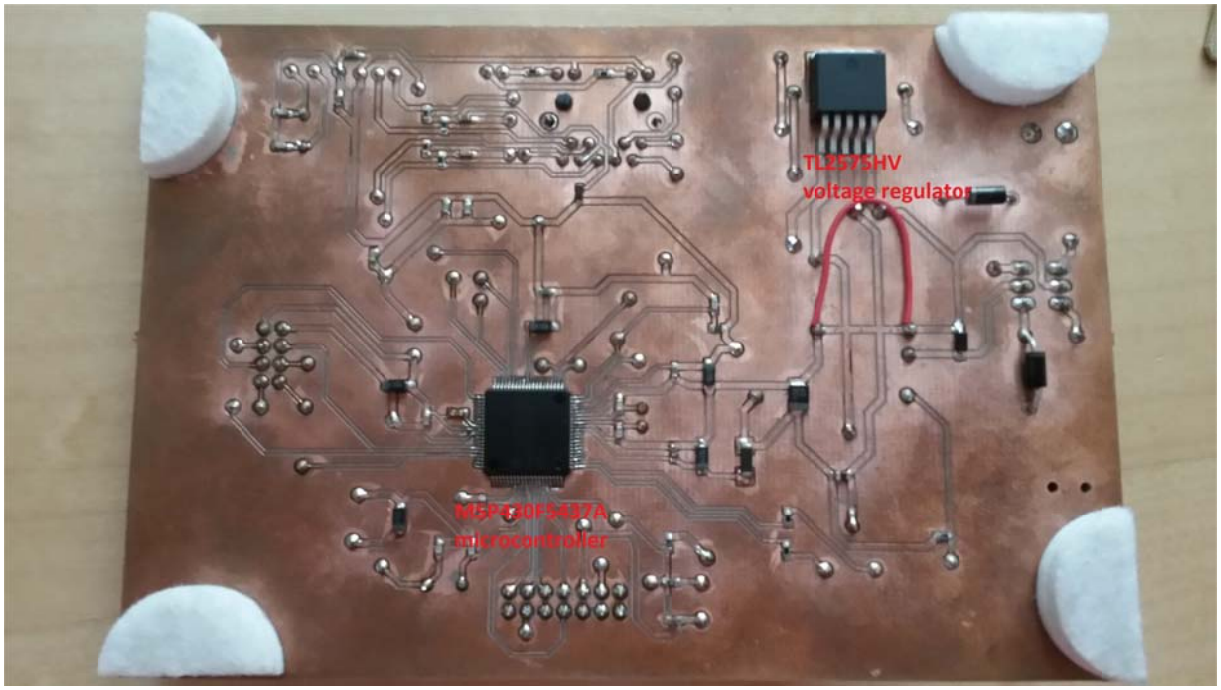
The main component of the board is the Texas Instruments MSP430F5437A microcontroller. This is a 16-bit MCU from TI's MSP430 ultra-low power MCU Reduced Instruction Set Computer (RISC) mixed signals family. This version of the chip has only 4 KB of Static Random Access Memory (SRAM) and 256 KB of Flash memory. This means that we can store relatively large amounts of data and instructions in flash, but we have to be very careful with using the RAM of the microcontroller. This microcontroller has two SPIs: one is, as we already discussed, used to connect the daughterboard and the other is used to connect to the Ethernet controller. The Ethernet controller is a Microchip ENC28J60. This Ethernet controller is connected to an RJ45 socket. To provide PoE functionality, a Texas Instruments TPS2375[18]8-pin integrated circuit (IC) is used. This IC contains all of the features needed to realize an IEEE 802.3af [19] compliant powered device.

There are two possible ways to program the microcontroller. Either using the Bootstrap Loader (BSL) interface or the Joint Test Action Group (JTAG) interface.[20][21] Because, only the JTAG interface is included in the board we used the JTAG interface to program our microcontroller.

Furthermore, the motherboard is equipped with two buttons. One button is a reset button, while the use of the other is programmable, so the developer can do whatever he/she wants. In our case we have programmed this button to jump to the program loaded using TFTP.



(Front)



(Back)

Figure 3-1: Views of the front and back of the motherboard

3.2.2 HP ProCurve Switch 2626

We need both a power supply and network connectivity. For this we have used a HP ProCurve Switch 2626 [22].

3.2.3 Dell Optiplex GX620

A Dell Optiplex GX620 desktop PC runs the DHCP and TFTP servers[23]. This PC is running a openSUSE[24]Linux distribution*. This PC is connected via a Realtek model RTL-8139/8139C/8139C+ Ethernet interface[25] to the switch described above.

3.2.4 MSP430 Programmer

To program the MSP430F5437A we used a Texas Instruments MSP-FET430UIF programmer [26].Figure 3-2 shows this programmer. The programmer can act as a Flash Emulation Tool (FET) for debugging the code line by line and to transfer a program to the flash memory of the microcontroller. The 14 pin JTAG connector is used by the FET to program the flash of the MCU.

*A Linux distribution is an Operating System (OS) build on top of the Linux kernel. These are usually targeted at PCs, but are also available for wide variety of systems up to the supercomputers, or to the smallest systems.

Figure 3-2: TI MSP-FET430UIF

3.3 Software

In this section, we will discuss the software tools we used to achieve our goals.

3.3.1 Wireshark

Wireshark[27]is a very popular free and open-source packet analyzer use to capture packets by setting the Network Interface Controller (NIC) in promiscuous mode*. This program is widely used for troubleshooting, packet analysis, and much more. This functionality is similar to tcpdump [28],but the program has a Graphical User Interface (GUI) that allows the user to easily sort and filter captured packets. Since this project involved sending and receiving packets, Wireshark was used to analyze whether the packets contain the data we expected. For example, we used Wireshark to check if there were DHCP/TFTP transfers between the embedded platform and the PC that serves as a server.

3.3.2 Code Composer Studio

The Integrated Development Environment (IDE) that we used during this project is Texas Instruments' Eclipse IDE [29]based Code Composer Studio (CCS) version 5.4.[30] CCS provides everything necessary to develop a program for the MSP430 MCU family. The negative part of this IDE is the price. The full version is quite expensive. Fortunately, CCS has also a free version that is limited in code size (up to 16 KB) or in time (180 days). Since the TFTP Boot program has a code size of about 14 KB this issue not an obstacle, so we were able to use the free version of this IDE.

*Promiscuous mode is a configuration of the NIC that causes the NIC to receive all incoming frames rather than only the frames that are specifically for this NIC, broadcast frames, or multicast frames that the NIC has indicated it is interested in.

3.4 Connecting the embedded platform to the network

This section explains how the embedded platform is attached to the network and how it obtains an IP address and then downloads programs.

3.4.1 DHCP server

The platform needs an IP address to join the network, to learn what file it is to download, and to learn the IP address of the TFTP server, hence we need a DHCP server. This DHCP server runs on top of the openSUSE operating system. When the platform is initially connected to the server via the PoE capable switch, it will automatically ask for an IP address from the DHCP server (running on the desktop PC). This IP address is assigned based upon the MAC address that is established by the "TFTPboot" program (written previously by Javier Lara Peinado – see Section 2.1.2). This network boot loader not only implements the DHCP client, but it will make TFTP requests to retrieve the program, and saves the received program in the flash memory.

The DHCP server is installed with YaST[31], an management tool for openSUSE. In addition to installing the DHCP server, we also need to configure it correctly. This means that we need to make an entry in the DHCP server's configuration file configured for the specific device that we want to connect. This means that we make a host specific entry in the configuration file using the same MAC address that we have programmed into the platform when installing the TFTPboot loader in both block of the flash memory. As noted in Javier Lara Peinado's thesis we use an address from the Locally Administered Address Range x2-xx-xx-xx-xx-xx, specifically from 02-00-00-xx-xx-xx.

3.4.2 TFTP server

One of the biggest advantages of a TFTP server is that is simplifies providing programs to embedded platforms. This project will take advantage of the TFTP server installed on the desktop PC. The TFTP server was also installed and configured using YaST. However, before we could use this to load our network interface testing programs we first had to overcome the problem of downloading programs via TFTP and storing them in the flash memory. The details of how this problem was solved are given in Section *4.2*.

4 Analysis

This chapter will explain the methods used to accomplish the goals stated in Chapter 3. This involved programming the MSP430 MCU and making a series of measurement. All of the source code and additional documents are publicly available via the Github repository "https://github.com/kekovski/MSP430", whose structure is explained in ***Appendix A***.

4.1 Network topology

Figure 4-1 illustrates the network topology of the test environment. The motherboard is connected via an Ethernet cable to the HP ProCurve Switch 2626. Another Ethernet cable connects the PC (runs a DHCP and a TFTP server).

When the motherboard is initially connected to the network it negotiates with the DHCP server to obtain an IP address. Detailed information about this process is given in Section *4.1 "Verifying the network connection"* of Rafid Karim and Haidara Al-Fakhri's bachelor's thesis[32].After the DHCP server has assigned the motherboard an IP address and provided some configuration (specifically the name of the program to be loaded and the IP address of the TFTP server), the TFTPboot program[33]starts running. Unfortunately, this boot program initially did not do what it supposed to do. Details of this are given in the next subsection.

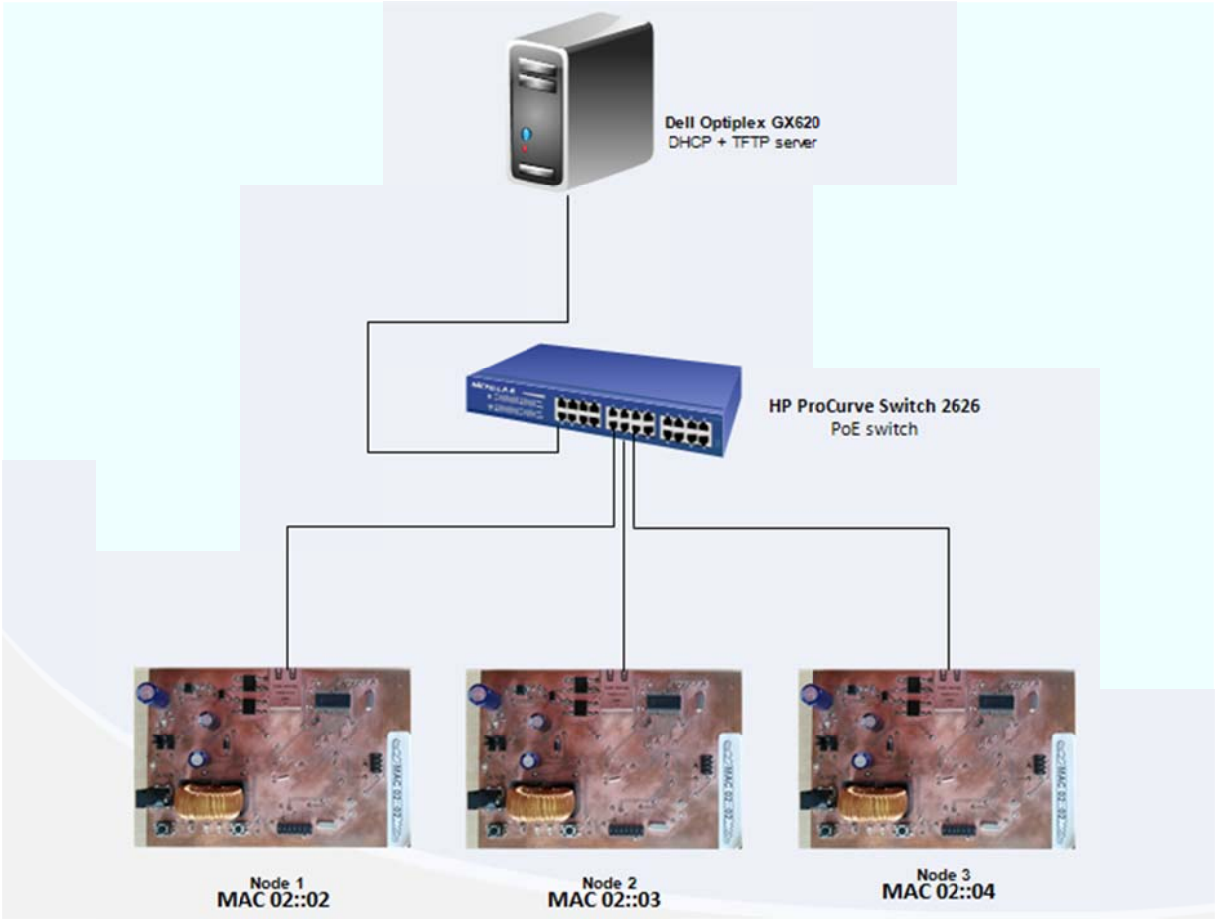


Figure 4-1: Network topology

4.2 TFTP loading problem

This section discusses the process of diagnosing and the fixing the problems with TFTP boot loading.

4.2.1 Symptom

First step was to determine the source of the problem. To do this I captured with Wireshark the packets sent and received via the Ethernet NIC of the PC that was connected to the PoE switch. Figure 4-2shows the packets captured when the motherboard attempted to boot. This series of packets show that the motherboard successfully obtained an IP address from the DHCP server, but not the required boot file from the TFTP server. More precisely, the TFTP server got the correct RRQ and started by sending the first data packet. However, the server did not get an ACK message back from the motherboard. Thus, the symptom of the problem was that the motherboard did not send an ACK message back to the server, hence a timeout would occur and the server would resend the first data packet.

No.	Time	Source	Destination	Protocol	Length	Info
4	0.939209000	0.0.0.0	255.255.255.255	DHCP	342	DHCP Discover - Transaction ID 0x12233456
5	0.000241000	192.168.1.1	255.255.255.255	DHCP	342	DHCP Offer - Transaction ID 0x12233456
6	0.010964000	0.0.0.0	255.255.255.255	DHCP	342	DHCP Request - Transaction ID 0x12233456
7	0.000212000	192.168.1.1	255.255.255.255	DHCP	342	DHCP ACK - Transaction ID 0x12233456
10	0.002888000	192.168.1.7	192.168.1.1	TFTP	71	Read Request, File: frequencyScanner.txt, Transfer type: octet
11	0.000991000	192.168.1.1	192.168.1.7	TFTP	558	Data Packet, Block: 1
12	1.001426000	192.168.1.1	192.168.1.7	TFTP	558	Data Packet, Block: 1
14	1.964087000	192.168.1.1	192.168.1.7	TFTP	558	Data Packet, Block: 1
17	0.011083000	Hewlett-_80:5f:40	HP_00:00:67	HP	93	HP Switch Protocol
21	0.002026000	192.168.1.1	192.168.1.7	TFTP	558	Data Packet, Block: 1

Figure 4-2: Wireshark capture of the failed TFTP process

4.2.2 Causes of the problem& fixes

It was obvious that the problem was in the motherboard since the motherboard received a data packet, but never acknowledged it. The first step was to debug the TFTPboot code, which was loaded in the MSP430. Using the MSP430-USB-Debug-Interface MSP-FET430UIF the MCU was debugged to resolve the problem(s).

4.2.2.1 RAM allocation problem

The first problem encountered during debugging was that the TFTPboot program only allocated 160 bytes of RAM memory for use as a stack and another 160 bytes of RAM memory for use by the program as the heap (despite the MCU having 16 KB of RAM). This program buffered the received TFTP DATA packet in RAM memory, however not enough memory had been allocated to store a TFTP data packet - which has a maximum length of 558 bytes. As soon as the 160 bytes were exhausted, the program crashed. This was the main cause for the TFTPboot program not being able to send back an ACK message. In summary the MCU received the DATA packet, started to process it, but as soon as the allocated RAM was full, the program crashed before sending an ACK message. This problem was solved by changing properties in the linker. The *"-heap_size"* and *"-stack_size"* flags were changed from 160 to 1024. The value 1024 was chosen because this would allocate more than enough space in the RAM to store a DATA packet and give some margin for future extensions to the program. These linker settings are changed in CCS via *Project → Properties → Build → MSP430 Linker → Basic Options*.

4.2.2.2 Overwrite problem

Unfortunately, after successfully buffering the first DATA packet the program crashed again. After hours of debugging it turned out that each time the first bytes were about to be written to the flash memory the program stopped working. To understand why this error occurs is it necessary to understand the format of a TI-TXT file [34].

The TI-TXT file format is a ASCII hexadecimal file containing a MSP430 program. CCS creates a TI-TXT file when you build a project. By default when a program is generated for the MSP430F5437A the linker indicates that the program should start from the first address in the flash memory. Figure 4-3 shows the memory map of the MSP430F5437A. In this case, the linker generates a file to be loaded at 5C00h. However, the TFTPboot program is also located in flash memory and starts at 5C00h. Thus, each time the TFTPboot program wanted to write a byte to the flash memory it tried to overwrite itself. The result of this was that the program crashed.

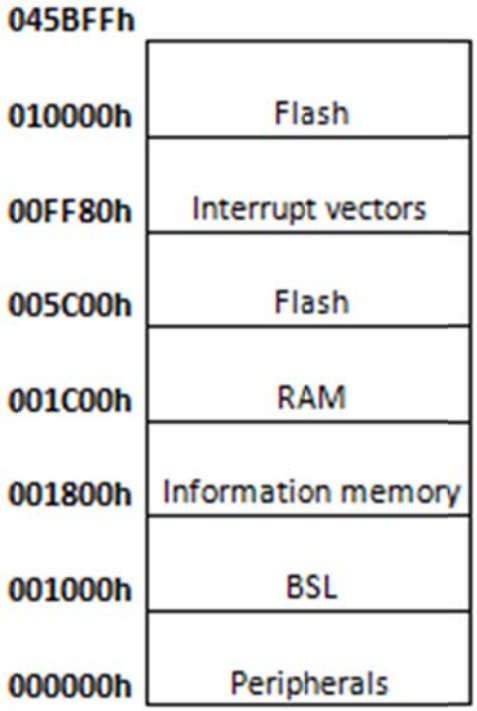


Figure 4-3: Memory map of MSP430F5437A

The solution is to find the last used address by the TFTPboot in the flash memory and change the starting address of the program to be loaded into flash memory via the bootloader- to the next free available memory address in the flash memory. To do this, the values of the system memory map in the Linker Command File had to be changed. The TFTPboot program is 14 KB is size. Hence with simple math is it possible to calculate the address where we should start to loaded the new program. Adding 14 KB (3800h) to 5C00h indicates where the downloaded program should be loaded. Therefore, 9400h was chosen as the starting address to avoid conflicts with the boot loader. In the Linker Command File on line 63 the start address for FLASH has to be changed from 0x5C00 to 0x9400. This tells the program to skip the first 14 KB of flash memory and start loading the program from address 9400h. The size of the flash memory is on the same line as the starting address and must also be changed as the amount of available Flash memory shrinks, hence 3800h is subtracted from the previous amount of flash memory giving 6B80h.Figure 4-4 shows the memory map after these steps.

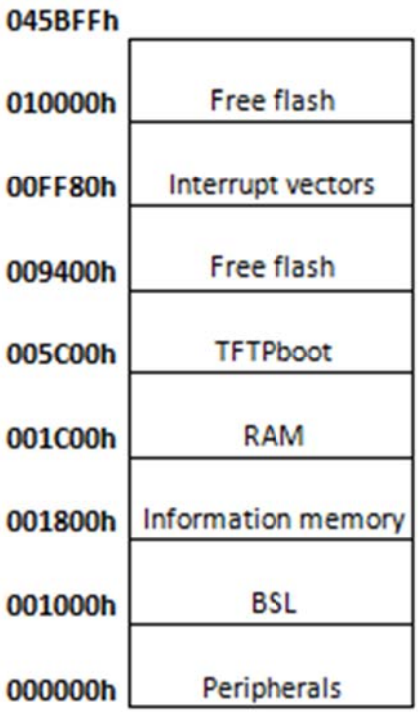


Figure 4-4: Memory map of MSP430F5437A after flashing TFTPboot

4.2.2.3 Small fixes

Not everything was fixed after solving the RAM allocation and overwriting problems. Several small errors in the code were detected. The first one was a bad loop in the parser logic. Namely an equal sign was forgotten on line 143 of Parser.c, which caused a "*WRONG_FORMAT_ERROR*". Subsequently an unnecessary reset of pointer was executed in the *void Flash_segmentErase (unsigned intbaseAddress, unsigned char *Flash_ptr)* method. This line of code is located on line 74 of flash.c. This reset caused also a crash of the program.

4.2.3 Result

After all these steps was it possible to download a TI-TXT file from the TFTP server to the MSP430F5437A's flash memory. Figure 4-5displays the DATA packets sent and the ACK messages received for each of them. Furthermore, Figure 4-7 and Figure 4-8 shows the memory contents in the MSP430's flash memory before and after downloading the program.

No.	Time	Source	Destination	Protocol	Length	Info
178	0.000028000	192.168.1.1	192.168.1.7	TFTP	558	Data Packet, Block: 47
179	0.051919000	192.168.1.7	192.168.1.1	TFTP	60	Acknowledgement, Block: 47
180	0.000027000	192.168.1.1	192.168.1.7	TFTP	558	Data Packet, Block: 48
181	0.052004000	192.168.1.7	192.168.1.1	TFTP	60	Acknowledgement, Block: 48
182	0.000028000	192.168.1.1	192.168.1.7	TFTP	558	Data Packet, Block: 49
183	0.052003000	192.168.1.7	192.168.1.1	TFTP	60	Acknowledgement, Block: 49
184	0.000093000	192.168.1.1	192.168.1.7	TFTP	558	Data Packet, Block: 50
185	0.051955000	192.168.1.7	192.168.1.1	TFTP	60	Acknowledgement, Block: 50
186	0.000089000	192.168.1.1	192.168.1.7	TFTP	558	Data Packet, Block: 51
187	0.051890000	192.168.1.7	192.168.1.1	TFTP	60	Acknowledgement, Block: 51
188	0.000036000	192.168.1.1	192.168.1.7	TFTP	558	Data Packet, Block: 52
189	0.052011000	192.168.1.7	192.168.1.1	TFTP	60	Acknowledgement, Block: 52
190	0.000029000	192.168.1.1	192.168.1.7	TFTP	558	Data Packet, Block: 53
191	0.052003000	192.168.1.7	192.168.1.1	TFTP	60	Acknowledgement, Block: 53
192	0.000028000	192.168.1.1	192.168.1.7	TFTP	558	Data Packet, Block: 54
193	0.051989000	192.168.1.7	192.168.1.1	TFTP	60	Acknowledgement, Block: 54
194	0.000029000	192.168.1.1	192.168.1.7	TFTP	558	Data Packet, Block: 55
195	0.052043000	192.168.1.7	192.168.1.1	TFTP	60	Acknowledgement, Block: 55
196	0.000029000	192.168.1.1	192.168.1.7	TFTP	558	Data Packet, Block: 56
197	0.051955000	192.168.1.7	192.168.1.1	TFTP	60	Acknowledgement, Block: 56
198	0.000036000	192.168.1.1	192.168.1.7	TFTP	558	Data Packet, Block: 57
199	0.052025000	192.168.1.7	192.168.1.1	TFTP	60	Acknowledgement, Block: 57
200	0.000027000	192.168.1.1	192.168.1.7	TFTP	558	Data Packet, Block: 58
201	0.052004000	192.168.1.7	192.168.1.1	TFTP	60	Acknowledgement, Block: 58
202	0.000073000	192.168.1.1	192.168.1.7	TFTP	558	Data Packet, Block: 59
203	0.051892000	192.168.1.7	192.168.1.1	TFTP	60	Acknowledgement, Block: 59
204	0.000093000	192.168.1.1	192.168.1.7	TFTP	558	Data Packet, Block: 60
205	0.051888000	192.168.1.7	192.168.1.1	TFTP	60	Acknowledgement, Block: 60
206	0.000131000	192.168.1.1	192.168.1.7	TFTP	558	Data Packet, Block: 61
207	0.051880000	192.168.1.7	192.168.1.1	TFTP	60	Acknowledgement, Block: 61
208	0.000096000	192.168.1.1	192.168.1.7	TFTP	558	Data Packet, Block: 62
209	0.051901000	192.168.1.7	192.168.1.1	TFTP	60	Acknowledgement, Block: 62
210	0.000042000	192.168.1.1	192.168.1.7	TFTP	558	Data Packet, Block: 63
211	0.051938000	192.168.1.7	192.168.1.1	TFTP	60	Acknowledgement, Block: 63
212	0.000043000	192.168.1.1	192.168.1.7	TFTP	331	Data Packet, Block: 64 (last)
213	0.018426000	192.168.1.7	192.168.1.1	TFTP	60	Acknowledgement, Block: 64

Figure 4-5: Success in sending TI-TXT file to the motherboard

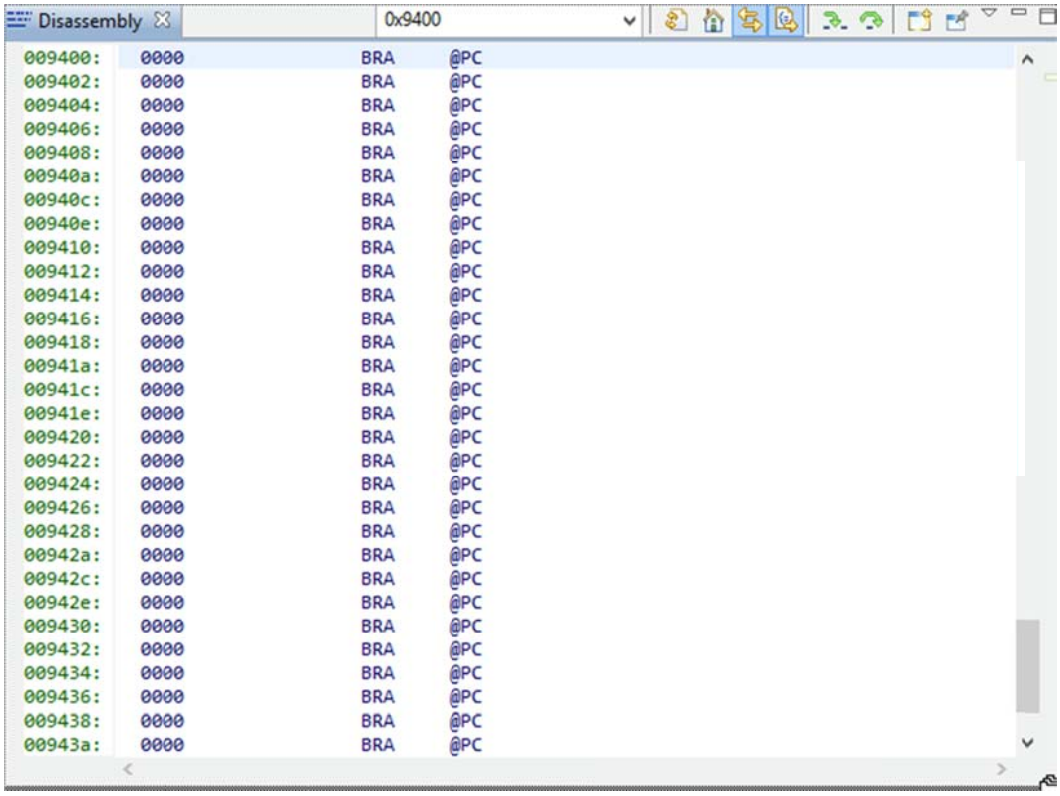


Figure 4-6: Memory before loading TI-TXT file

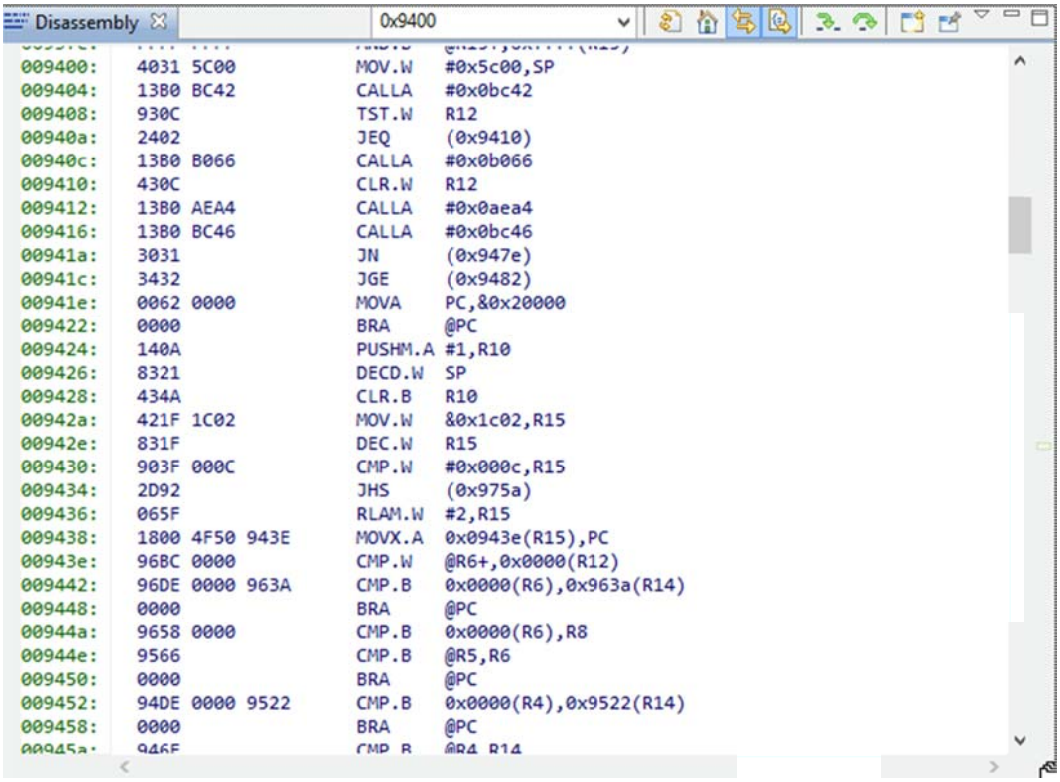


Figure 4-7: Memory after loading TI-TXT file

CCS's free version does not allow us to load more than 16 KB of code. However, the new bootloader does **not** have this limitation, thus avoiding this restriction of CCS. However, the user loses 14 KB of flash memory to the TFTPboot application. Note that the TFTPboot application includes the DHCP, UDP, IP, and Ethernet controller interfacing functionality. If one were really pressed for flash memory space it would be possible to expose the entry points to the subroutines in the TFTPboot application so that the networking related code already included in the boot loader could be reused (rather than including it yet again in the image of the program being downloaded).

4.3 IP stack evaluation

The major goal of this project was to evaluate the Microchip TCP/IP stack when adapted for the MSP430F5437A and running on this motherboard. This section will describe this evaluation and give an overview of the measurements that were made. To make these measurements a test program called "Analyze" was written for the MCU*. The Analyze program contains a code snippet from another performance testing program (see [35] and [36]).

4.3.1 UDP Packet sending from MCU to PC

The first test measures how fast the MCU can create UDP packets and send them to the PC's Ethernet port. This measurement tests the transmit performance of the Microchip TCP/IP stack's UDP module. The results from these tests will give application developers for this platform a good overview how fast the MCU can send UDP packets, i.e., what is the maximum UDP data rate they can expect. The program sends 1024 UDP packet with a given

*This program is available from the Github repository.

sized UDP payload. For this measurement, the payload sizes were 8, 16, 32, 64, 128, 256, 512, and 1024 bytes. Using Wireshark the duration of the UDP transfers for each sized payload were determined and the transmit rate of the stack is calculated. This measurement includes how long it takes to open a socket, transmit one packet, and close the socket in a loop. Figure 4-8 illustrates the flowchart of the *Analyze* program for sending packets from the MCU to the Ethernet controller. From the flowchart we can see that what can be observed from the Wireshark capture is the time from one UDP packet being sent to the next UDP packet being sent.

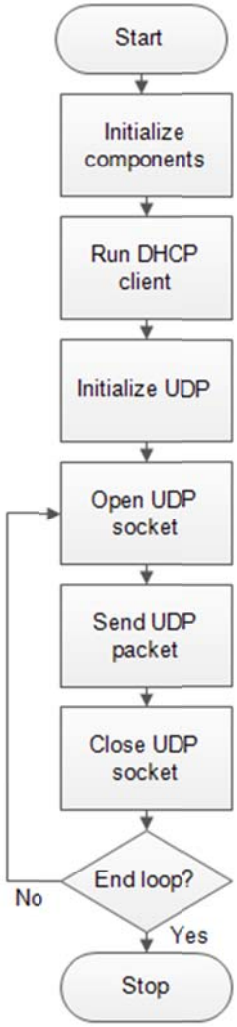
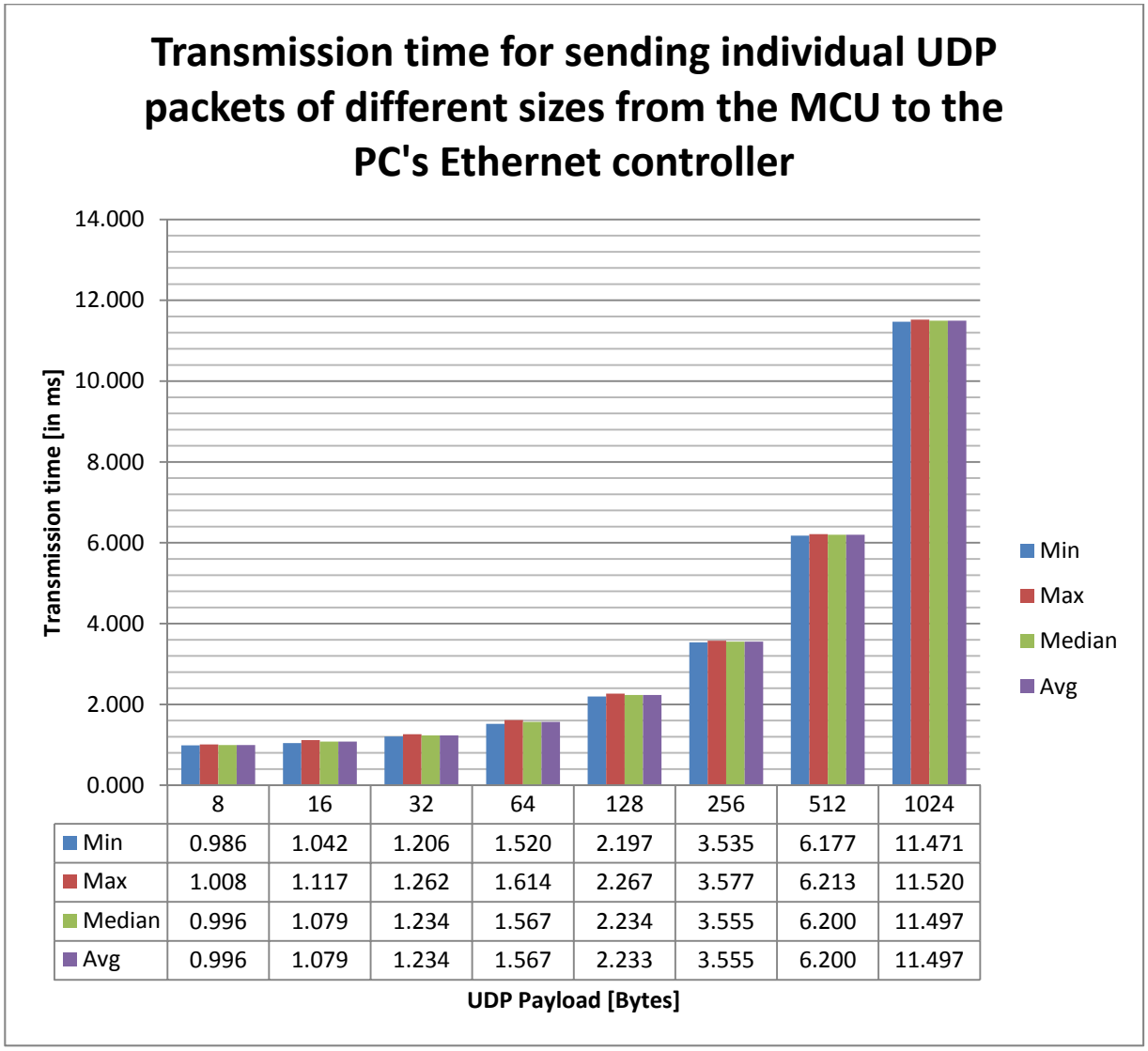


Figure 4-8: Flowchart of the Analyze program for sending UDP packets from the MCU

4.3.1.1 Measurements

As mentioned earlier different sized UDP packets were sent during the test. Wireshark captured each UDP packets received by the PC's Ethernet controller. Based on these packet captures it is possible to compute some simply statistics minimum, maximum, median, average throughput, and the standard deviation. Figure 4-9 shows the statistics for sending 1024 UDP packets with different sized payloads. In addition to these statistics is it also useful to know the standard deviation, shown in Figure 4-10. It should be noted that the Ethernet interface is operating in 10Mbps mode.



	8	16	32	64	128	256	512	1024
Min	0.986	1.042	1.206	1.520	2.197	3.535	6.177	11.471
Max	1.008	1.117	1.262	1.614	2.267	3.577	6.213	11.520
Median	0.996	1.079	1.234	1.567	2.234	3.555	6.200	11.497
Avg	0.996	1.079	1.234	1.567	2.233	3.555	6.200	11.497

Figure 4-9: Transmission time for sending individual UDP packets of different sizes from the MCU to the PC's Ethernet controller

- **Minimum and maximum time**: The minimum and maximum transmission times increases as the payload size is increased. However, the difference between the minimum and maximum value is relatively small. This small difference is not surprising as the MCU is only executing the loop of "Analyze" test program shown in Figure 4-8.
- **Median time**: These statistics clearly show an exponential pattern. We did a regression analysis of this data to compute the base time to do the socket opening and closing as well as the time to invoke the packet sending process, then we found the coefficient of the term representing the size of the packet. This coefficient gives us the time per byte of payload.

Regression analysis (times given in ms)

Regression Statistics	
Multiple R	0.999999482
R Square	0.999998963
Adjusted R Square	0.99999879
Standard Error	0.004026769
Observations	8

ANOVA

	df	*SS*	*MS*	*F*
Regression	1	93.83381421	93.83381421	5786899.254
Residual	6	9.72892E-05	1.62149E-05	
Total	7	93.8339115		

	Coefficients	*Standard Error*	*t Stat*	*P-value*
Intercept	0.908904725	0.001796636	505.8924481	4.0265E-15
UDP payload	0.010338609	4.29773E-06	2405.597484	3.48309E-19

Now is it possible to compose a formula to compute the time to send a packet with some amount of payload.

Time to send a packet = fixed cost + transmission per byte of payload * x

fixed cost = intercept coefficient

transmission per byte of payload = UDP payload coefficient

x = UDP payload size

Because of the extremely small P-value is it possible to predict almost 100% exactly how much time it will take to transmit a UDP packet containing some amount of payload. The formula above is applied and Table 4-1 and shows the comparison between the estimated and measured results. Note that the fixed cost of sending a zero byte sized payload UDP packet is 0.908904725 ms.This time represents the time required to open and close the socket as well as the time to issue the command to send the packet buffered in the Ethernet controller.

Table 4-1: Estimated transmission time of a single UDP packet based on the regression analysis

UDP Payload	Estimated transmission time (ms)	Measured transmission time(ms)
8	0.9916136	0.996000
16	1.0743225	1.079000
32	1.2397402	1.234000
64	1.5705757	1.567000
128	2.2322467	2.234000
256	3.5555886	3.555000
512	6.2022725	6.200000
1024	11.4956403	11.497000

- **Average time**: The small difference between average and median speed indicates that there are not that many exceptional cases (i.e. very fast or very slow).
- **Standard deviation**: Standard deviation is very small. This means that the transmission time is almost the same for every packet; however, there is a clear dependence of the standard deviation on the packet size – as would be expected since any variance in the performance of the operations inside the loop should be dependent upon the size of the payload – since all of the other operations (opening and closing the socket) should not depend upon the payload size.

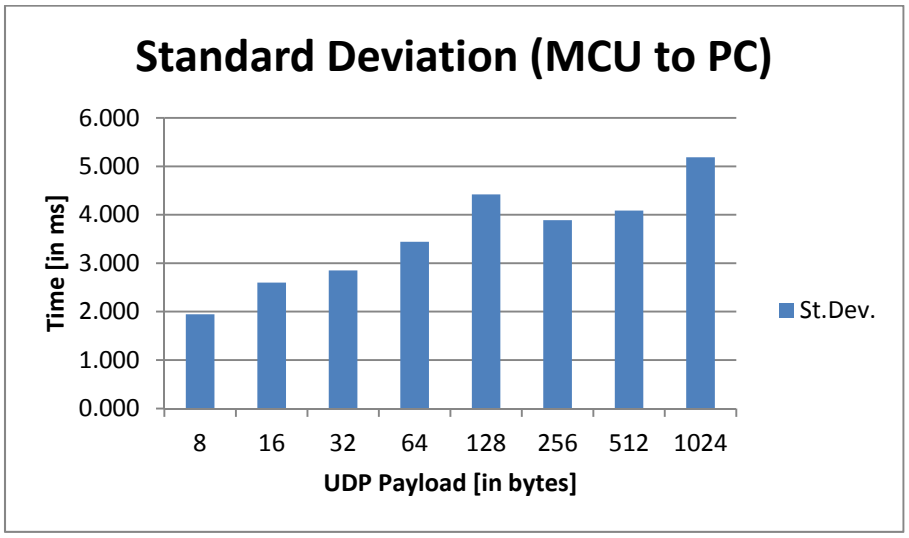
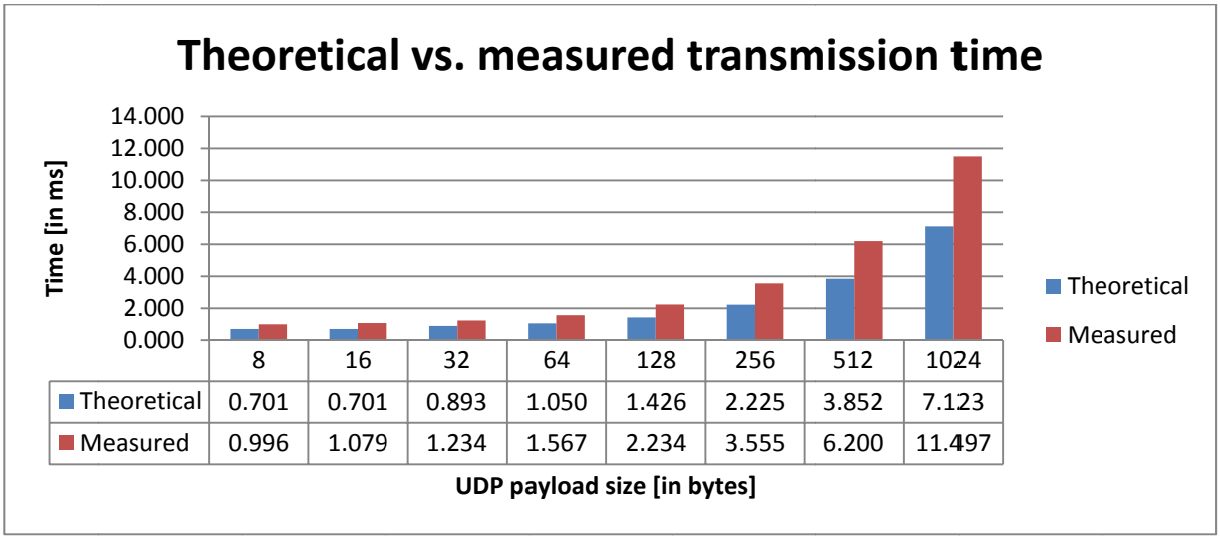


Figure 4-10: Standard deviation of the transmission times shown in the previous figure (MCU to PC)

4.3.1.2 Theoretical vs measured transmission time

It is useful to know how much the overhead is for each UDP payload size. Therefore, the theoretical time to send a UDP packet over an unloaded 10Mbps Ethernet with minimum Interframe Space was calculated to compare it with the measured results. Figure 4-11 illustrates the difference between the theoretical and measured transmission time to send a UDP packet. Also the values are given in the table below the figure. The difference between the differences is exponential. This indicates the larger the UDP packet, the higher the overhead is. For more detailed measurements see the Theoretical tab of Analysis/Statistics.xlsx file in the GitHub repository.



	8	16	32	64	128	256	512	1024
Theoretical	0.701	0.701	0.893	1.050	1.426	2.225	3.852	7.123
Measured	0.996	1.079	1.234	1.567	2.234	3.555	6.200	11.497

Figure 4-11: Theoretical vs. measured transmission time to send a UDP packet of the indicated size

4.3.2 UDP Packet sending from ENC28J60 buffer to PC

Not only is the transmit time of packets departing from MCU interesting, but the time to transfer bytes to/from the MCU to the ENC28J60 Ethernet controller's buffer is also interesting. In this measurement rather than creating a new UDP packet each time, we simply resend the packet that is already available in the Ethernet controller's buffer. The expected result was a faster transmission time from the ENC28J60 to the PC's Ethernet controller than in the previous measurements. Figure 4-12illustrates the logic of the *Analyze* program for sending packets already contained in the ENC28J60's buffer to the PC's Ethernet controller.

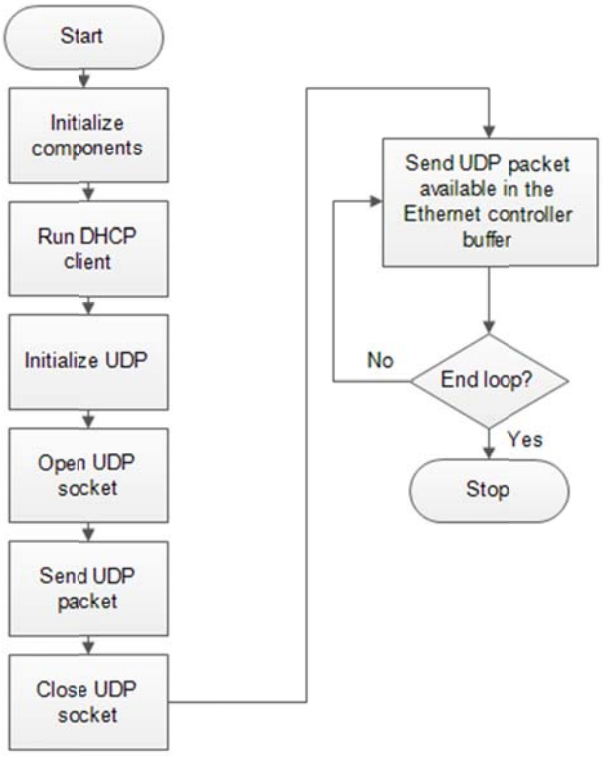
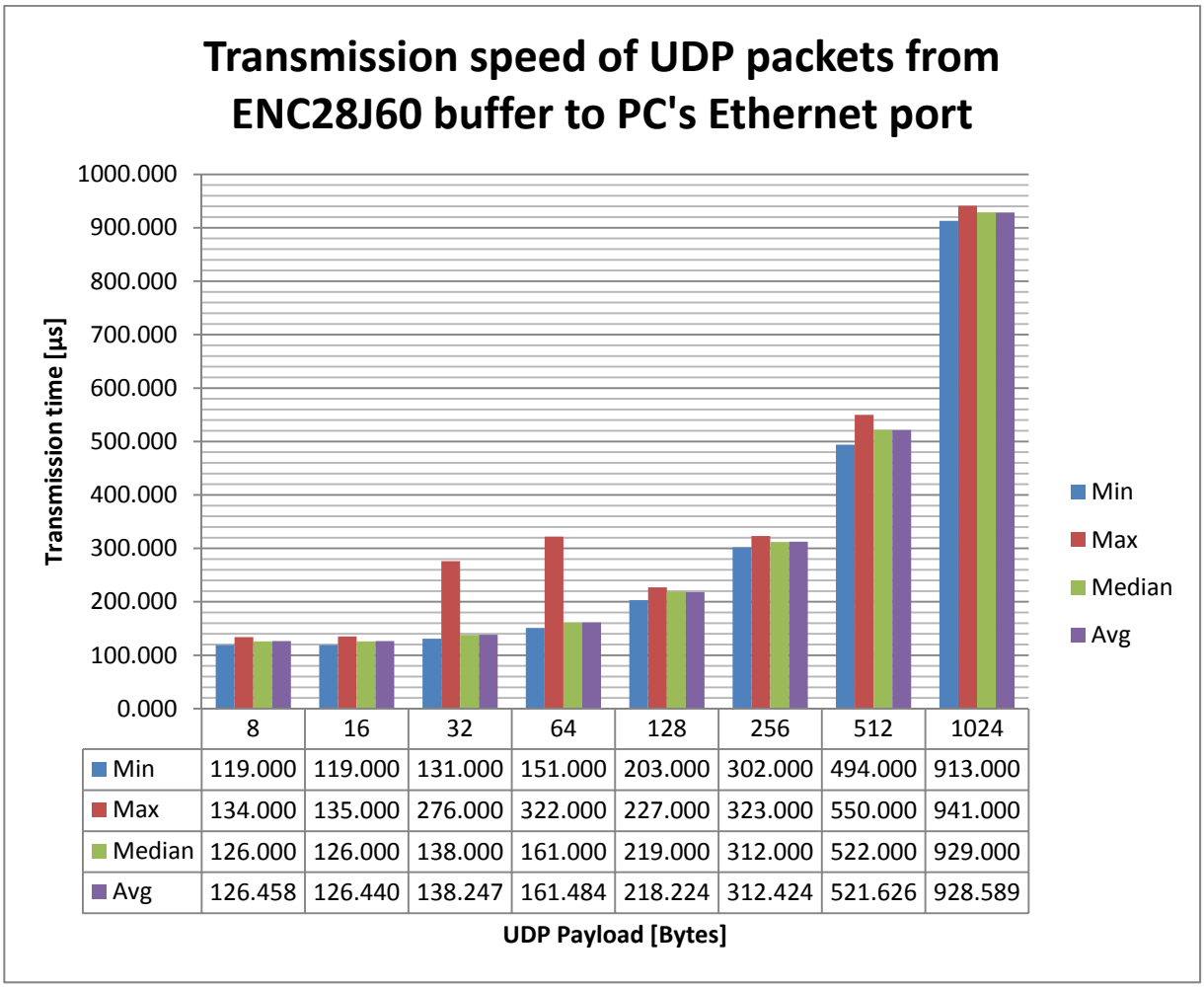


Figure 4-12: Flowchart of the Analyze program (sending existing UDP packets from the ENC28J60's buffer)

4.3.2.1 Measurements

As stated earlier this set of measurements is based upon sending the UDP packet already available in the ENC28J60 Ethernet controller's buffer. As a rest there is no need for a transfer of *data* from the MCU to the Ethernet controller's buffer – hence only commands are being sent over the SPI from the MCU to the Ethernet controller to send this buffered packet. This packet is sent1024 times in a loop. Wireshark is again used to capture packets. Again, the minimum, maximum, median, average transmission times and standard deviation values calculated. Figure 4-13 shows the statistics for sending UDP packets with different payload sizes from the Ethernet controller to the PC's Ethernet controller. The standard deviations are shown in Figure 4-14.



	8	16	32	64	128	256	512	1024
Min	119.000	119.000	131.000	151.000	203.000	302.000	494.000	913.000
Max	134.000	135.000	276.000	322.000	227.000	323.000	550.000	941.000
Median	126.000	126.000	138.000	161.000	219.000	312.000	522.000	929.000
Avg	126.458	126.440	138.247	161.484	218.224	312.424	521.626	928.589

Figure 4-13: Transmission time for individual UDP packets of the indicated sizes (i.e., transmission time of an existing packet in the ENC28J60's buffer to PC)

Note that the times to send 8 and 16 bytes of UDP payload should be the same since the minimum network payload size of a 10 Mbps Ethernet frame is 46 bytes. After subtracting 20 bytes for the IP header and another 8 bytes for the UDP header, we have 46-28=18 bytes and this is larger than both an 8 byte and a 16 byte UDP payload.

- **Minimum and maximum transmission time**: As the first measurement the minimum and maximum speed increases as the payload gets bigger. However, the difference between the minimum and maximum value is relatively small. Except 32 and 64 bytes payload packets have an exceptional high maximum value. These

high values appeared only one time in their categories. The reason for these outliers is probably a failure in one of steps in the loop to send the data in the buffer. Possibly the code retake the loop step and sends it again. This causes an almost double so high transmission speed.

- **Median transmission time**: This statistic shows clearly an exponential pattern as did the previous measurements. However, the transmission time is roughly 11 to 12 times smaller than the time measured when the MCU also has to transfer the payload of the packet across the SPI to the Ethernet controller's buffer. This allows us to compute the time required to transfer the data of the UDP packets via the 8MHz SPI from the processor, see Table 4-2.

Table 4-2: SPI processing speed

UDP Payload (bytes)	UDP Packet size (bytes)	UDP Packet size (bits)	Transmission via 8MHz SPI - Theoretical(µs)
8	**64**	512	64
16	**64**	512	64
32	**78**	624	78
64	**110**	880	110
128	**174**	1392	174
256	**302**	2416	302
512	**558**	4464	558
1024	**1070**	8560	1070

- **Average transmission time**: As for the earlier measurements, the small difference between average and median speed indicates that there are not many exceptional cases (i.e., very fast or very slow).
- **Standard deviation**: Standard deviation indicates a very small difference between the transmission time for each of the different sized packets.

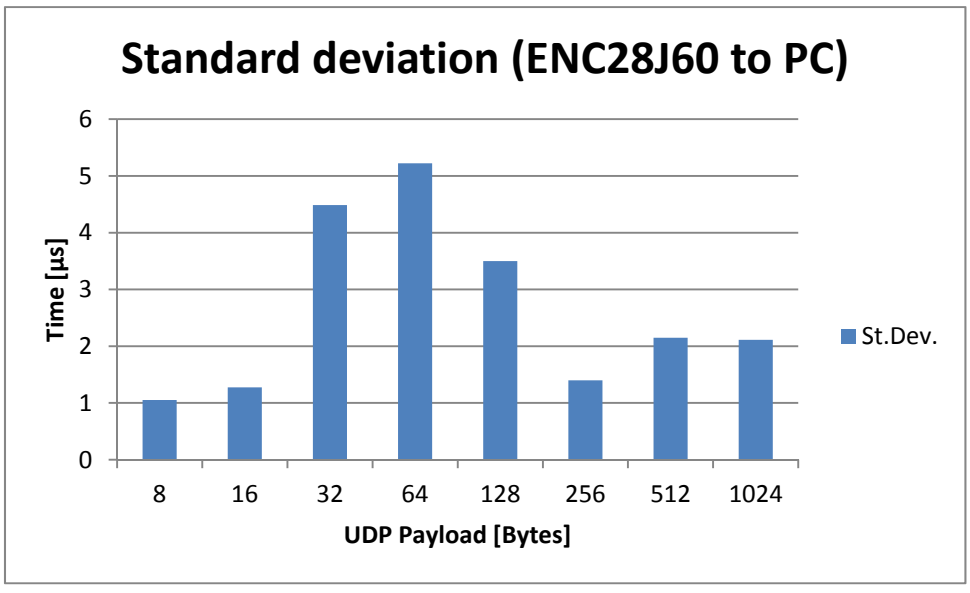


Figure 4-14: Standard deviation (ENC28J60 to PC)

Considering the case of a 1024 byte UDP payload we see that it should take about 1 ms to transfer the payload from the MCU to the Ethernet controller's buffer and about 0.9 ms to transmit the packet – so the fact that it takes 11.497 ms to transmit a 1024 byte UDP payload packet from the MCU to the PC's Ethernet interface means that there is a lot of unexplained time (11.497-(1.066-0.929) = ~9.5 ms).

4.3.3 Analysis of TFTP processing

It may also useful be useful for the developer to know how much time is needed to perform the TFTP and flash programming for a given sized program. Therefore, the Analyze program's TI-TXT file is loaded into MSP430's Flash memory by means of TFTPboot program while this action is captured by Wireshark.

4.3.3.1 TFTP transfer byterate

Wireshark captured every step of the TFTP process. This section examines how quickly the RRQ, DATA, and ACK packets are processed. Figure 4-15shows how fast the TFTP process runs. The peaks are clearly visible when DATA and ACK packets are sent. The first peak is the RRQ packet bit rate where the short peak at the end is the last DATA packet that is less than 512 bytes long. Using this measurement was possible to calculate a down load bit rate of11.4817514 KBps.

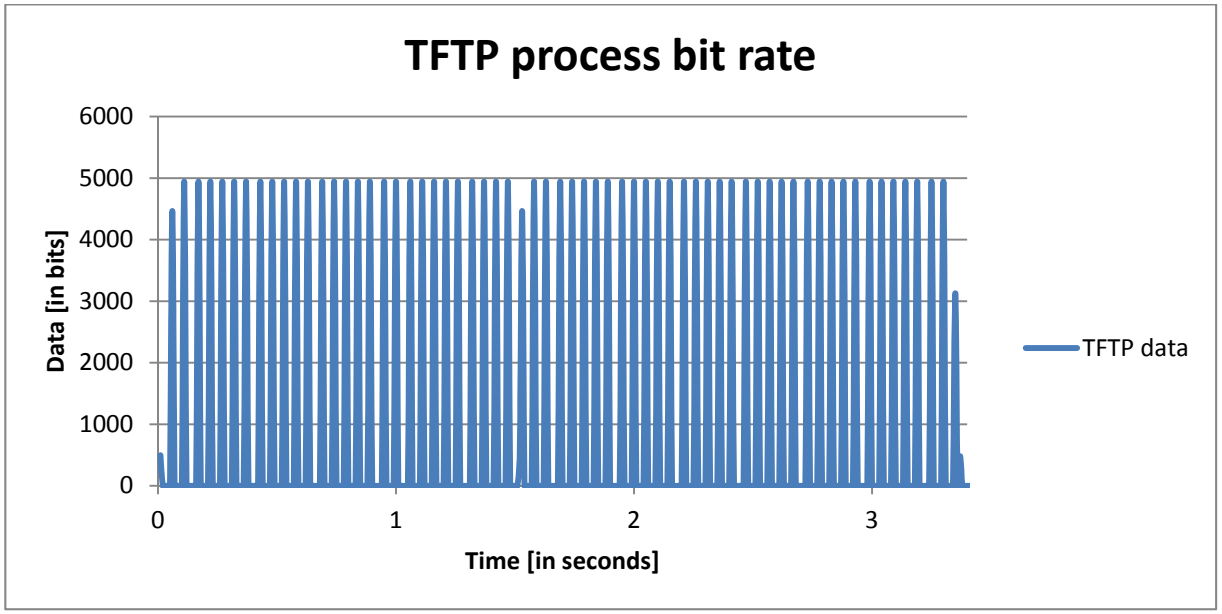


Figure 4-15: TFTP processing bit rate

4.3.3.2 TFTP processing speed

Many programs can be written for the MSP430, hence the size of the programs can differ. Since the bit rate for the TFTP and flash programming is known, we can compute the time required to download and store any sized file. Table 4-3 shows this for different sized files starting from 1 KB to 230 KB (this is the upper bound because a maximum of 229 KB are available in the Flash memory as some of the space is taken by the TFTP boot program). Figure 4-16 shows this data as chart. Note that this table and figure were computed assuming that the time for TFTP and programming the flash is linear increase in the file size.

Table 4-3: TFTP download and flash programming times for different sized files

Boot file size (KB)	Processing time (seconds)	Boot file size (KB)	Processing time (seconds)
1	0.087	**205**	17.854
5	0.435	**210**	18.290
10	0.871	**215**	18.725
15	1.306	**220**	19.161
20	1.742	**225**	19.596
25	2.177	**230**	20.032
30	2.613		
35	3.048		
40	3.484		
45	3.919		
50	4.355		
55	4.790		
60	5.226		
65	5.661		
70	6.097		
75	6.532		
80	6.968		
85	7.403		
90	7.839		
95	8.274		
100	8.709		
105	9.145		
110	9.580		
115	10.016		
120	10.451		
125	10.887		
130	11.322		
135	11.758		
140	12.193		
145	12.629		
150	13.064		
155	13.500		
160	13.935		
165	14.371		
170	14.806		
175	15.242		
180	15.677		
185	16.113		
190	16.548		
195	16.983		
200	17.419		
205	17.854		

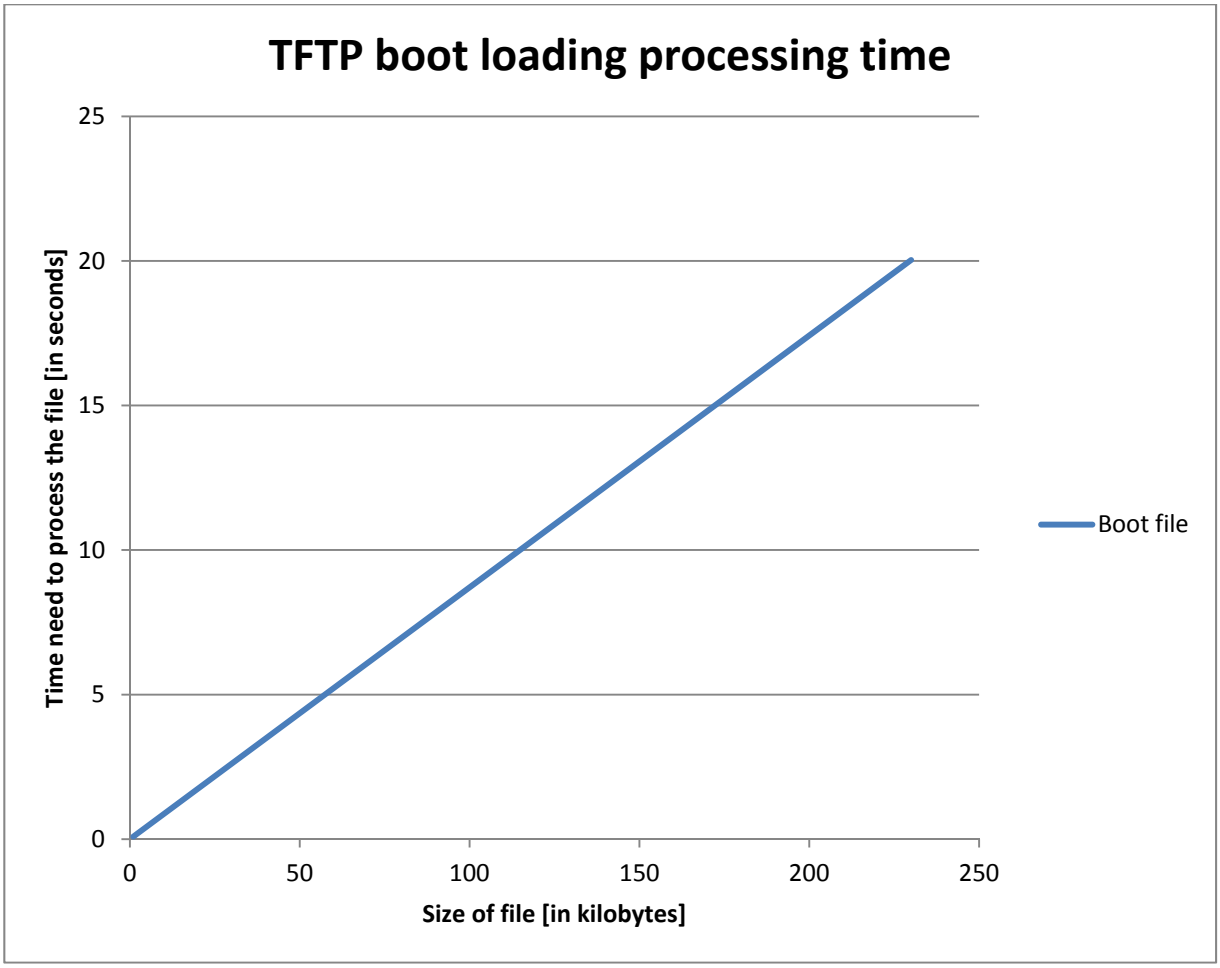


Figure 4-16: TFTP boot loading processing time as a function of file size

4.3.4 Conclusion

It is logical that the transmission times for different sized UDP payloads are different. The first test consisted of a loop, which creates a new UDP packet and sends it. This obviously requires more time than simply retransmitting a packet that is already in the buffer of the Ethernet controller. These measurements give developers the ability to predict the performance of a UDP application on this platform. The differential measurement (i.e., using the data from the two sets of UDP measurements) enables us to compute the per byte transfer time across the SPI bus, the time to perform a socket open, close, and send a UDP packet. These measurements provide programmers with a good overview of the maximum transmission rates they can expect from this platform.

5 Conclusions and future work

This chapter presents conclusions based on the performed tasks and analysis. The initial goals of the project and our achievements are compared. Furthermore, some suggestions are made of future work. Finally, several economic, social, environmental, and ethical reflections are given.

5.1 General conclusions

The main objectives of this project were to fix the TFTP boot loading problem and to evaluate the Microchip TCP/IP stack when using a TI MSP430 MCU and a Microchip ENC28J60 Ethernet controller connected via SPI. The first goal, namely TFTP boot loading now works. The TFTP boot loader program correctly downloads a program into flash memory from a TFTP server. However, some additional improvements could have been made. Since the board has two programmable buttons and one is used for resetting the motherboard. Pressing this button resets the motherboard and downloads an updated version of the file provided by the TFTP server -if there is such a file available on the TFTP server. The second button could have been configured as a soft reset button. When this second button is pushed, the motherboard could simply restart the already loaded program. Unfortunately, I was not able to realize this functionality. A hard reset can be done by simply unplugging and plugging the Ethernet cable. As I had already spent a lot of time to solve the basic boot loading problem I decided not to spend more time introducing this new functionality.

The second goal of this project was to evaluate the Microchip TCP/IP stack's performance. Several measurements and calculations were done. However, it is possible to do more measurements and calculations. I focused on UDP transmission from the MCU to the PC's Ethernet controller. Additionally, I calculated how fast a TFTP boot file is processed by the embedded board. In the future, it would be interesting to measure the Transmission Control Protocol (TCP) protocols throughput. Since I was close to the deadline for this project, I decided to skip measurements of TCP.

This was my first experience with hardware, thus I learned a lot about microcontrollers and how to program them. Before this project, my knowledge of the C programming language was very limited. This lead to many struggles when programming the MCU. However, by reading datasheets I gained insight into the MSP430 MCU family. Furthermore, the extensive use of Wireshark motivated me to learn how to use this handy tool much better than I could before this project. Additionally, I inspected every detail of the TFTP protocol and learned how to create UDP packets in the MCU.

If I was to do this project all over again, I would probably start with the IP stack evaluation instead of fixing the TFTP issue. I feel that I could have done a much more extensive evaluation of the IP stack. Unfortunately, I lost a lot of time trying to fix the TFTP problem. However, I would probably not have learned as much as I did about how to program microcontrollers and how the IP stack works at a low level.

My advice to future contributors to this project is to make much more extensive use of the Ethernet controller's buffer when possible. According to my measurements it is possible to decrease the time needed to generate and send UDP packets if one were to make fuller use of the capabilities of this Ethernet controller. Of course, a clear task is to evaluate the TCP module of this IP stack. Additionally, carefully reading the data sheets and documentation can save a lot of time and annoyance.

5.2 Future work

As discussed earlier, evaluation of the TCP functionality of the IP stack has not been done, but clearly should be done. Additionally, it is possible to do many interesting calculations based on the existing observations. In this way the capabilities of the motherboard will be much better documented for future developers.

Another area of tests and measurements is to determine the maximum rate at which data contained in UDP packets can be transferred from the PC to the MCU. As the current measurements only consider traffic going in the other direction.

Moreover, the programmable buttons can be used to better effect. It is recommended that both hard and soft reset functionalities be implemented.

A potentially interesting idea is to develop a monitoring program for the network, which uses this embedded platform. A program could monitor a number of these boards to see if they are active or not and to detect failures as soon as possible. A benchmark tool could be implementing to help developers measure and analyze the throughput and latency of different nodes in the network. Furthermore, the TFTP boot loader has an important place in facilitating future tests and measurement. In summary it is it up to future developers to exploit this Swiss knife of a lower power PoE networked computing platform.

5.3 Required reflections

This project reduces costs by exploiting the TFTP boot loading functionality. Future users can develop programs with a code size larger than 16 KB. This can save a developer a lot of money, considering that a node locked single user license of CCS costs US$495.00. This can be a big gain for an organization that needs, for example, to support 50 developers which would otherwise cost US$19,994.00*.

Avoiding these costs raises the question if it is ethical to do this or not. Some people would say that this is ethical because the user does not load program code larger than 16 KB via the IDE. While others might say it is not because the user should pay the company for the use of the IDE. It is up to the user to decide if the usage of TFTPboot loading system is ethical or not. It should also be noted that the TFTPboot loading system can be used with code compiled using other development tools.

While carrying out this project no environmental or sustainability issues were encountered. However, in retrospect the use of the TFTP boot loader does contribute to sustainability as it allows the same hardware to easily be reprogrammed for many different uses – in fact, the same hardware can be potentially dynamically used for different purposes at different times (the maximum number of times that the flash memory can be reprogrammed will set an upper limit on this reuse). Furthermore, this project does not seem to have a positive or negative effect on society, although it facilitates the development of new applications that might have positive or negative effects on society.

*Given the current price for CCS at http://www.ti.com/tool/ccstudio.

References

[1] A. López and F. J. Sánchez, “Exploiting Wireless Sensors: A gateway for 868 MHz sensors,” Master’s thesis, KTH Royal Institute of Technology, School of Information and Communication Technology, Stockholm, Sweden, 2012.

[2] “TI MSP430F5437A Datasheet.” .

[3] “ENC28J60 Data Sheet.” .

[4] J. Lara Peinado, “Minding the spectrum gaps: First steps toward developing a distributed white space sensor grid for cognitive radios,” Master’s thesis, KTH Royal Institute of Technology, School of Information and Communication Technology, Stockholm, Sweden, 2013.

[5] “SPI Block Guide V03.06.” .

[6] P. M. in D. Mah, “What does the new Power over Ethernet standard mean for IT pros?,” *TechRepublic*. [Online]. Available: http://www.techrepublic.com/blog/data-center/what-does-the-new-power-over-ethernet-standard-mean-for-it-pros/. [Accessed: 23-May-2014].

[7] “IEEE SA - 802.3af-2003.” [Online]. Available: http://standards.ieee.org/findstds/standard/802.3af-2003.html. [Accessed: 25-May-2014].

[8] S. Alexander and R. Droms, “DHCP Options and BOOTP Vendor Extensions,” *Internet Request for Comments*, vol. RFC 2132 (Draft Standard), Mar. 1997.

[9] J. Alba Tormo Peiró,, “Spectrum sensing based on specialized microcontroller based white space sensors: Measuring spectrum occupancy using a distributed sensor grid,” Master’s thesis, KTH, School of Information and Communication Technology, Communication Systems, Stockholm, Sweden, 2013.

[10] R. Karim and H. Al-Fakhri, “Smart Door Lock : A first prototype of a networked power lock controller with an NFC interface.,” Bachelor’s Thesis, KTH, School of Information and Communication Technology, Communication Systems, Stockholm, Sweden, 2013.

[11] R. Droms, “Dynamic Host Configuration Protocol,” *Internet Request for Comments*, vol. RFC 2131 (Draft Standard), Mar. 1997.

[12] J. Postel, “Internet Protocol,” *Internet Request for Comments*, vol. RFC 791 (INTERNET STANDARD), Sep. 1981.

[13] J. Postel, “User Datagram Protocol,” *Internet Request for Comments*, vol. RFC 768 (INTERNET STANDARD), Aug. 1980.

[14] K. R. Sollins, “TFTP Protocol (revision 2),” *Internet Request for Comments*, vol. RFC 783, Jun. 1981.

[15] K. R. Sollins, “The TFTP Protocol.” [Online]. Available: http://www.rfc-editor.org/ien/ien133.txt. [Accessed: 23-May-2014].

[16] K. Sollins, “The TFTP Protocol (Revision 2),” *Internet Request for Comments*, vol. RFC 1350 (INTERNET STANDARD), Jul. 1992.

[17] “TL2575HV-ADJ | Step-Down (Buck) Converter | Converter (Integrated Switch) | Description & parametrics.” [Online]. Available: http://www.ti.com/product/tl2575hv-adj. [Accessed: 23-May-2014].

[18] “TPS2375 | Powered Device | Power Over Ethernet (PoE)/LAN Solutions | Description & parametrics.” [Online]. Available: http://www.ti.com/product/tps2375. [Accessed: 23-May-2014].

[19] “IEEE-SA -IEEE Get 802 Program - 802.3: Ethernet.” [Online]. Available: http://standards.ieee.org/about/get/802/802.3.html. [Accessed: 23-May-2014].

[20] “Introduction to JTAG | Embedded.” [Online]. Available: http://www.embedded.com/electronics-blogs/beginner-s-corner/4024466/Introduction-to-JTAG. [Accessed: 23-May-2014].

[21] “Texas Instruments MSP430 JTAG header pinout.” [Online]. Available: http://www.jtagtest.com/pinouts/msp430. [Accessed: 23-May-2014].
[22] “ProCurve Switch 2626 (J4900B) specifications - HP Products and Services Products.” [Online]. Available: http://h10010.www1.hp.com/wwpc/ca/en/sm/WF06b/12136296-12136298-12136298-12136298-12136316-12136318-31539227.html?dnr=2. [Accessed: 23-May-2014].
[23] “Dell OptiPlex GX620.” [Online]. Available: http://www.dell.com/support/drivers/us/en/19/Product/optiplex-gx620. [Accessed: 23-May-2014].
[24] “openSUSE.” [Online]. Available: http://en.opensuse.org/Main_Page. [Accessed: 23-May-2014].
[25] “Realtek RTL8139 Ethernet controller.” [Online]. Available: http://www.realtek.com.tw/products/productsView.aspx?Langid=1&PFid=6&Level=5&Conn=4&ProdID=16. [Accessed: 28-May-2014].
[26] “MSP430 USB Debugging Interface - MSP-FET430UIF - TI Software Folder.” [Online]. Available: http://www.ti.com/tool/msp-fet430uif. [Accessed: 23-May-2014].
[27] “Wireshark · Go Deep.” [Online]. Available: http://www.wireshark.org/. [Accessed: 23-May-2014].
[28] “Tcpdump/Libpcap public repository.” [Online]. Available: http://www.tcpdump.org/. [Accessed: 23-May-2014].
[29] “Eclipse - The Eclipse Foundation open source community website.” [Online]. Available: http://www.eclipse.org/. [Accessed: 23-May-2014].
[30] “CCS - Texas Instruments Wiki.” [Online]. Available: http://processors.wiki.ti.com/index.php/Download_CCS. [Accessed: 23-May-2014].
[31] “Portal:YaST - openSUSE.” [Online]. Available: http://en.opensuse.org/Portal:YaST. [Accessed: 23-May-2014].
[32] R. Karim and H. Al-Fakhri, “Smart Door Lock : A first prototype of a networked power lock controller with an NFC interface.,” Bachelor’s Thesis, KTH, School of Information and Communication Technology, Communication Systems, Stockholm, Sweden, 2013.
[33] J. L. Peinado, “Mind-the-gaps GitHub,” *GitHub*. [Online]. Available: https://github.com/cazulu/mind-the-gaps. [Accessed: 23-May-2014].
[34] “TI-TXT file format -srec_ti_txt Linux man page.” [Online]. Available: http://linux.die.net/man/5/srec_ti_txt. [Accessed: 23-May-2014].
[35] H. Schlunder, “UDP Performance Test microcontrollers.” [Online]. Available: https://github.com/exosite-garage/mcp_dv102412_cloud. [Accessed: 23-May-2014].
[36] “Microchip TCP/IP Stack Help.” .

Appendix A

GitHub repository

All the source code and related documents of this project are publicly available on a GitHub repository. The link to this repository is https://github.com/kekovski/MSP430 and consists of several folders:

- Analysis: Calculations and charts based on the measurements
- Captures: All the Wireshark captures
- The source code is divided in two parts
  - Analyze: Program for IP stack evaluation
  - Updated version of a TFTP program loader for MSP430
- Figures: All of the used figures in the final report