Computer Engineering

Mekelweg 4,

2628 CD Delft

The Netherlands

http://ce.ewi.tudelft.nl/

CE-MS-2014-04

Delft University of Technology

April 2014

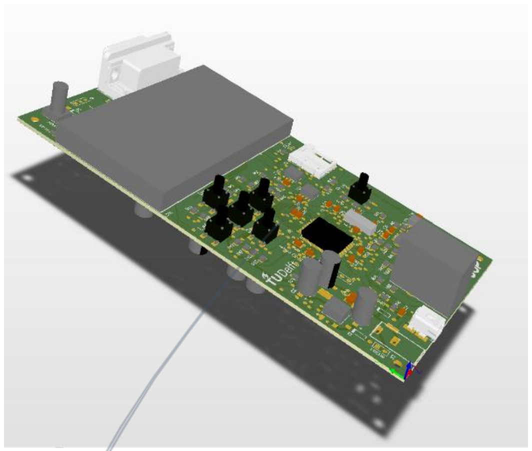
MSc Thesis

Secure Standalone Programmer (SSP)

Farhad Irannejad

Supervisor

Dr. ir. A.J. van Genderen, CE, TU Delft



***Abstract**. The ability to write a firmware to an embedded device in a secure way is becoming an essential task as electronic design aspects are migrating from hardware to software. The secure stand-alone programmer (SSP) is a microcontroller programmer that is able to program a target microcontroller without the intervention of a PC. This programmer and its PC tool use cryptographic methods to protect the HEX file from intentional abuse. The programmer is also able to communicate with GSM, Ethernet and USB which allows it to be used for home automation and also for remote reprogramming and debugging a device.*

Faculty of Electrical Engineering, Mathematics and Computer Science

CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENT

I would like to thank Arjan van Genderen for being the supervisor for this master’s thesis and the feedback and the time he has spent on reading and reviewing my report. I would like to thank members of computer engineering group in EEMCS faculty for their support for my project. Also I would like to thank the members of circuit and system group for letting me use their lab facilities. I would like to thank Sorin Cotofana and Rene van Leuken for being my thesis committee members.

Farhad Irannejad
Delft, Netherland
April, 2014

1 INTRODUCTION

Nowadays more and more aspects of an electronic design are migrating from hardware to software (e.g. filtering, control loops, etc.). So the software part of a design needs to be protected as it is the intellectual property of the designers. Otherwise a company might copy the software that is developed by other parties, thus avoiding paying for the development costs which enables the company to sell the same product more cheaply. This might lead to higher market shares for the competitor and an overall loss of profit for the original developers.

Hopefully most microcontrollers come with protection to prevent the read of flash or EEPROM memory. But a Firmware (normally a HEX file[3]) needs to be loaded to a target microcontroller after fabricating the hardware. To that end the manufacturer of the electronic board needs to have the firmware to program the devices. Normally manufacturers fabricate electronic boards for many other companies and it is less likely that designers and manufacturer are working for the same company. As the manufacturer makes the actual device and as they have access to the firmware, the manufacturer can easily sell the device to other parties without the designers noticing.

1.1 BACKGROUND

Not much academic research has been done in this field. For high-end products like mobile phones a secure bootloader[1] is being used. In this method first a bootloader is programmed to the device, then this bootloader uses cryptographic methods to write the firmware in to the device itself. As these devices normally have a big storage size, the crypto bootloader is not much overhead for them. But for low end devices like 8 bit microcontrollers with 16KB of flash memory it is not rational to use 50% of the memory for a secure bootloader and of course it will increase the overall price of the product if a microcontroller with a higher storage capacity is chosen. Also, in this method the bootloader itself also has to be written to the target in a secure way.

Another way of protecting the firmware is to decrypt the firmware in the programmer and not in the device itself, then there will not be an additional overhead to the memory for solving this issue. There are companies like Softlog[2] that are offering tools for protecting the firmware but they do not offer much information about the implementation of their design, so one cannot assure the level of the security of their design.

1.2 THESIS OBJECTIVES

The purpose of this thesis assignment is to develop an open source programmer that can solve the above-mentioned problem by using cryptographic methods. The SSP is planned to be more than a simple programmer, it should be able to program target devices without intervention of a computer with just a single button for the user convenient. Also, it should be able to store a number of firmwares, in order to let the operator choose between firmwares. The SSP should be able connected to the internet via an Ethernet socket and also to the target device via UART, in order to connect the target device to a host faraway for monitoring, controlling and debugging purposes.

SSP is planned to use in for three main situations:

1) Production programming
2) Consumer electronics after sales service
3) Research and development of the devices which have electronic boards

1.2.1 Production programming

After PCB assembly it is necessary to load a firmware to the board. In order to ensure that no one can extract or modify the firmware (SSP file) without having the keys for the encryption, it is required to encrypt and sign the firmware, this can be done in a PC application. In this way, firmware could easily be distributed to the suppliers over the internet. The keys for encryption must be pre-shared between the SSP programmer and the PC application, so the firmware is generated for a specific SSP programmer and it cannot be used by any other SSP programmer.

The SSP should be able to store a number of SSP files (encrypted firmware) in its SD memory, so it can be used for different target devices without reconnecting it to a PC. An Operator should be able to see the information of the firmware on the SSP's LCD and choose the right firmware to load to the board. It should also program target devices without the intervention of a PC (stand-alone), so that the operator doesn't need to worry about the extra configuration and connection to the PC.

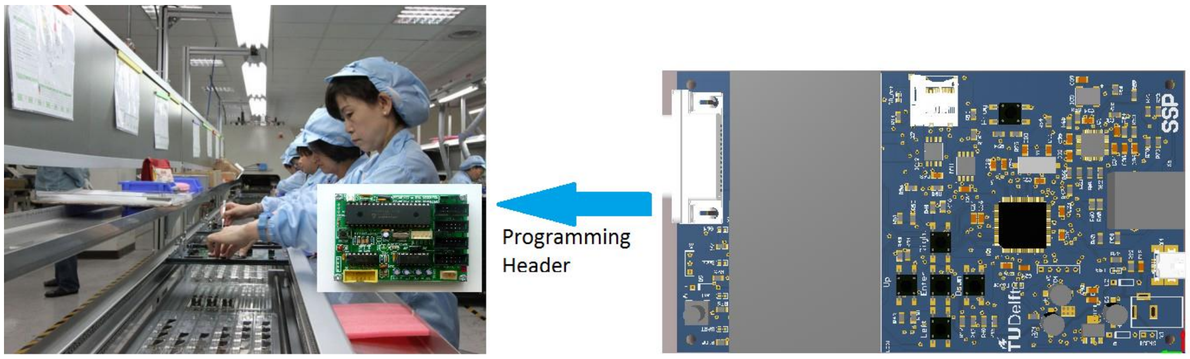


Figure 1 Production Programming – SSP programmer on the right, Production line on the left

1.2.2 Consumer electronics after sales service

It might be that there is a flaw in the loaded firmware for a device. In this case, the SSP can be used to reprogram an electronic board in the consumer's place. This avoids a lot of time being wasted due to transportation and if also any damage that might be caused when the device has gone for a repair cycle. As with the production line, the SSP benefits from storing multiple firmware and loading firmware on its own.

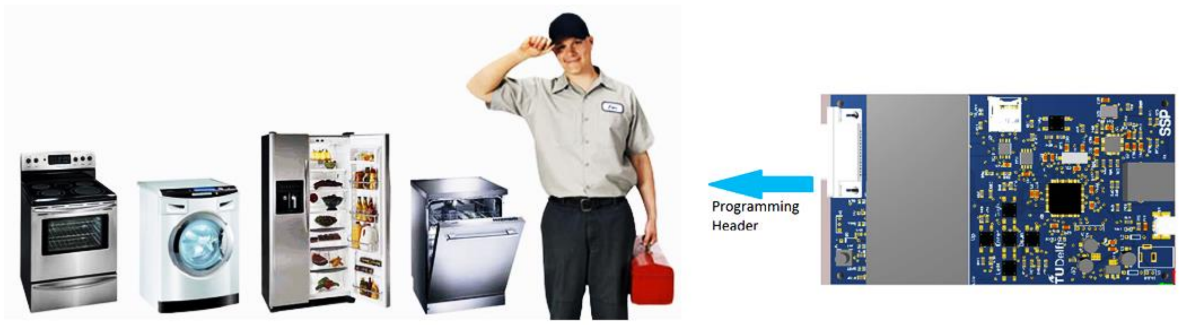


Figure 2 after sales service – SSP programmer on the right, after sales service-man on the left

1.2.3 Research and development of the devices which have electronic boards

The SSP could offer much more than a programmer. With an addition of Ethernet connection and GSM module it could be able to communicate with the outside world. One example of its use would be a major help in designing or improving the efficiency of a new product based on its consumer usage. For instance, if a refrigerator manufacturer is planning to launch a new product. After testing the product within its laboratories it is necessary to optimize the control system or test the product base on customers usage in different cities or even different parts of the world. The SSP will allow to do this without the need for moving the whole setup to the consumer's location by having a remote connection to the target device.

In this way, not only they can have bidirectional communication with the target board over the internet, but they can also reprogram the target remotely and optimize the control parameters.

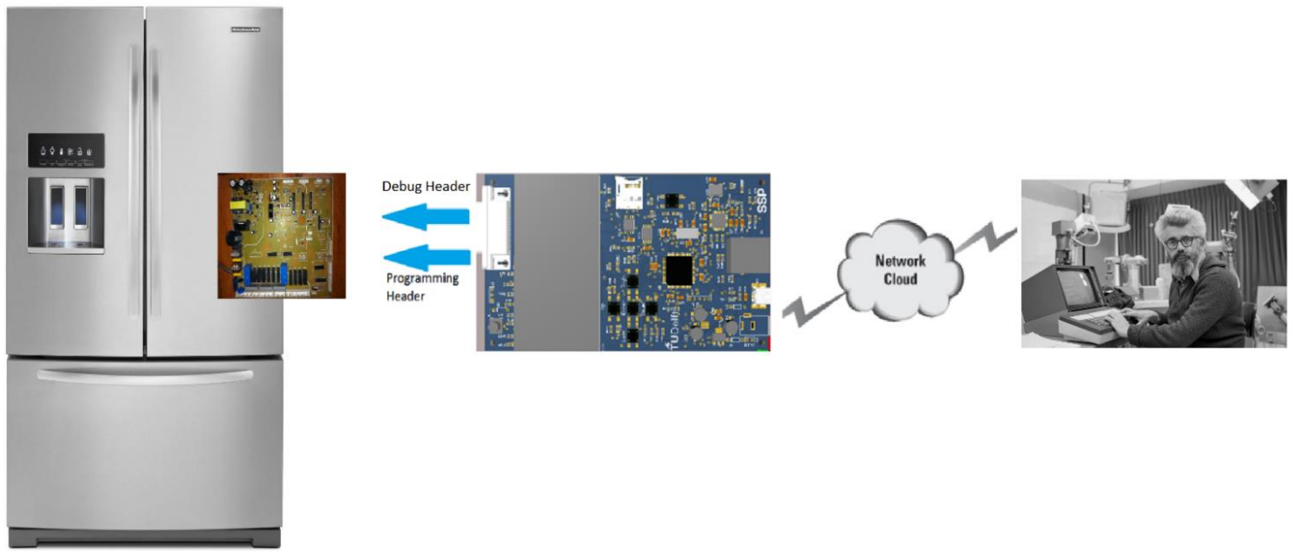


Figure 3 Research and development of a product

1.3 Thesis structure

This thesis assignment is about designing the SSP and its PC tool. The report focuses on the actual design and implementation of the three parts of the design naming: 1) The SSP hardware 2) The SSP software 3) The PC tool, in chapter 2. Followed by the outcome of the project and comparisons with other methods in chapter 3. Finally possible future expansion of the design is given in chapter 4.

2 METHOD

For a simple programmer, a HEX file which is the representation of data in memory location is used to program a target device. This HEX file has a plain text file format. In order to have the required extra security on the programming method a new file format (.SSP) has to be introduced. This file should include encrypted data and also information about the firmware and the target device. The encrypted data in the file should only be available in presence of the key for the file. The key for the encryption has to be shared between the programmer and the PC application.

The general idea of the design is represented in Figure 4. The idea is to encrypt the firmware which avoids regenerating the original hex file from SSP file without having the encryption key. The file should also include a unique identifier for the file which allows to control the number of successful programming to the targets by checking if the file has been used by the programmer before. In this way designers can have more control over the production. This unique identifier has to be protected with a signature which avoids modifying the file. With this signature the programmer can check if the file is modified and check the validity of the source of the file.

The design consist of two parts, one of which is the PC application and the other is the programmer hardware and its software. A specific file format (.SSP) is introduced to allow the programmer and the PC application to talk to each other.

A PC application is developed to encrypt and sign the firmware and generate the SSP file and the programmer is designed to verify and decrypt the SSP files and extract the embedded HEX file and information and, in the final stage, load it on to the target device.

The keys for the encryption have to be stored in the PC and the programmer. The key file in the PC should be encrypted with a password phrase, so that it can be stored in a safe manner. For the programmer the key has to be stored in the flash memory of the programmer's microcontroller and the configuration of the microcontroller has to be set for avoiding the read of flash memory.

As the generated SSP file has encrypted data, it could be transferred through an unsecured internet connection by means of email or other file sharing systems.

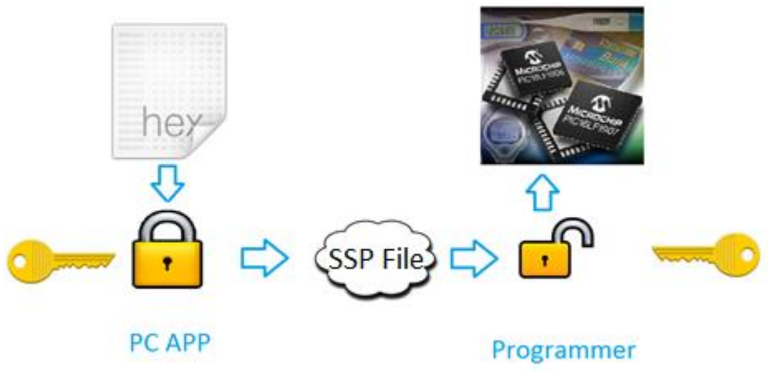


Figure 4 the Idea

2.1 SSP FILE

The SSP file format is introduced to the system for the connection between a PC and the programmer.

The SSP file is comprised of three parts.

1) Header

2) Compressed and encrypted HEX file

3) Signature

2.1.1 Header

Extra information about the target device and the firmware is added to the file as a header. This header is concatenated to the top of the encrypted firmware. To make the header readable in the SSP file with a simple text editor, it consists of a number of null terminating strings and these strings are not encrypted. The data in the header are as follows:

1) Version of the header
2) Name of the firmware
3) Version of the firmware
4) Creation date of the file
5) Unique ID for the file
6) Target microcontroller
7) Number of HEX records in the file

Each of the strings in the header starts with a specific identifier string. These strings are represented in Table 1. As an example, if the target device is chosen to be "PIC16F1936" then, its corresponding string would be "MCU: PIC16F1936". In this way the programmer can easily recognize the string in the case of there being different header versions.

Header Information	Identifier String
Version of the header	`HVersion:`
Name of the firmware	`Name:`
Version of the firmware	`Version:`
Creation date of the file	`Creation Time:`
Unique ID for the file	`UID:`
Target microcontroller	`MCU:`

Table 1 Header identifier strings

2.1.1.1 Version of the header

It might be necessary to change the Header format to expand the configurability of the programmer. A header version identifier is added to the header which allows the design to be compatible with the future versions of the programmer.

2.1.1.2 Name of the firmware

It is necessary for the programmer operator to know which file he wants to load to the target device. This name has to be in the file itself and protected by a signature, in order to avoid any possible mistakes. This string is shown on the SSP's LCD while programming.

2.1.1.3 Version of the firmware

Normally firmware is changed in time, this can be for upgrades or for solving flaws in the software or for compatibility with a specific hardware version. To track the files properly a version is added to the header.

2.1.1.4 Creation date of the file

When using the PC tool to generate the SSP file, a timestamp is added to the header. The operator can check to see if the file is newly generated and the designer has not sent any wrong files by mistake.

2.1.1.5 Unique ID for the file

The SSP file needs to have a unique ID. Then the programmer can identify whether the file has been used before and for how many successful programming count. This ID cannot be changed in the file as it is protected by the designer's signature.

2.1.1.6 Target microcontroller

Because different microcontrollers are programmed in different ways it is necessary to include the microcontroller's model in the file.

2.1.1.7 Number of HEX records in the file

To calculate the message digest to validate the designer's signature, the programmer needs to know when the file finishes, to do this a 32bit integer number, representing the number of the HEX record in the file, is added to the end of the header. Before this number an escape character (0x13) is added to the header to find this number easier.

2.1.2 Compressed and encrypted HEX file

A HEX[3] file is a representation of the data in memory locations. The file format is standardized by the Intel Corporation. This file format is used for the binary information of almost all devices with memory storage (e.g., Microcontrollers, EEPROM, Flash memories).

Each line of text in a HEX file is called HEX record. Each record starts with a ":" character and it is followed by the data below:

1) One byte for data count in the record
2) A 16 bit address of the data
3) One byte for Record Type
4) Actual data to be written in memory
5) One byte checksum
6) End of line character/s

Below one see an example of a HEX record from Wikipedia[18]:

```
:10010000214601360121470136007EFE09D2190140
```

- 🟨 Start code
- 🟩 Byte count
- 🟪 Address
- 🟥 Record type
- 🟦 Data
- ▨ Checksum

Most HEX records consist of 16 bytes of data. In order to make the access of a record convenient and faster, smaller records are padded with zero. In this case all the records will be 21 byte sized and there will be no need for ":" to identify the start of a record. Addresses that are multiples of 21 represent the start of a record.

A HEX file has a plain text format and data is shown in hexadecimal notation. This means that to show each byte there will be 2 ASCII characters. To decrease the amount of data to be encrypted and decrypted that is directly proportional to the processing time, the HEX file is compressed by storing the binary value of each byte instead of 2 ASCII characters. In this way the output file will be half the size of the input file.

The record's address format is big endian format but, 'int' type data in the PIC32 microcontroller is in little endian format[4]. In order to avoid extra conversion in the programmer's microcontroller, the endianness of the addresses is reversed.

The compressed HEX file is encrypted using a symmetric encryption method. In this method both encryptor and decryptor use the same key and this key is pre-shared between them. AES with a key size of 128 bits in CBC mode is used for symmetrical encryption as it is known to be unbreakable at this time and it is widely used in the internet and in banking and in many other secure systems[5].

The formatted header is added to the encrypted message and then the whole message is signed to avoid changes in the SSP file and to make sure of the source of the file in the programmer.

2.1.3 Signature

To avoid intentional changes in the SSP file; it is signed in the PC application and then it is verified on the programmer. In order to sign a message or a file, first the message digest of the file has to be calculated. Then this digest must be encrypted using an asymmetrical encryption method. Asymmetrical encryption means that the encryption and decryption keys are not the same, so that it is possible to have the verifier and the signer with two different keys, one is privately hold by the signer and the other is distributed publicly. The encrypted message digest is decrypted in the verifier side with the public key of the signer. This extracted message digest can be compared with the calculated digest of the received file and if both of them are the same it means the file is not modified.

Secure Hash Algorithm (SHA) which is a cryptographic hash function is used for calculating the message digest. It is called secure hash because the message cannot be changed, as opposed to CRC hash functions where the message could be modified but the hash result can still be the same. SHA256 is used in this file format and at this time this algorithm is known to be unbreakable.

This message digest is encrypted with 1024bit RSA private key and the result is added to the end of the SSP file. The verifier can verify the file integrity by decrypting the signature with the public key of the signer and comparing the result to the received message digest. This method is widely used in the internet when accessing a web service using a Transport Layer Security(TLS)[6].

2.2 SSP HARDWARE

The SSP hardware is designed to be more than a simple programmer and it can be used for multiple purposes. Figure 5 demonstrates different modules for the hardware in blocks. Brown blocks are sockets and interfaces of the programmer to the outside and green blocks are internal controllers in the programmer. The hardware consists of different modules:

1) Programming header for connecting to the target microcontroller
2) USB connection for loading firmware to the programmer in a convenient way
3) Ethernet controller and socket to connect to the internet
4) SD memory reader, to be able to load firmware or log data
5) EEPROM to buffer the firmware
6) LCD and push buttons for user interfacing
7) Internal control for generating required voltage levels for programming
8) GSM module for having internet wirelessly or receiving a command via SMS

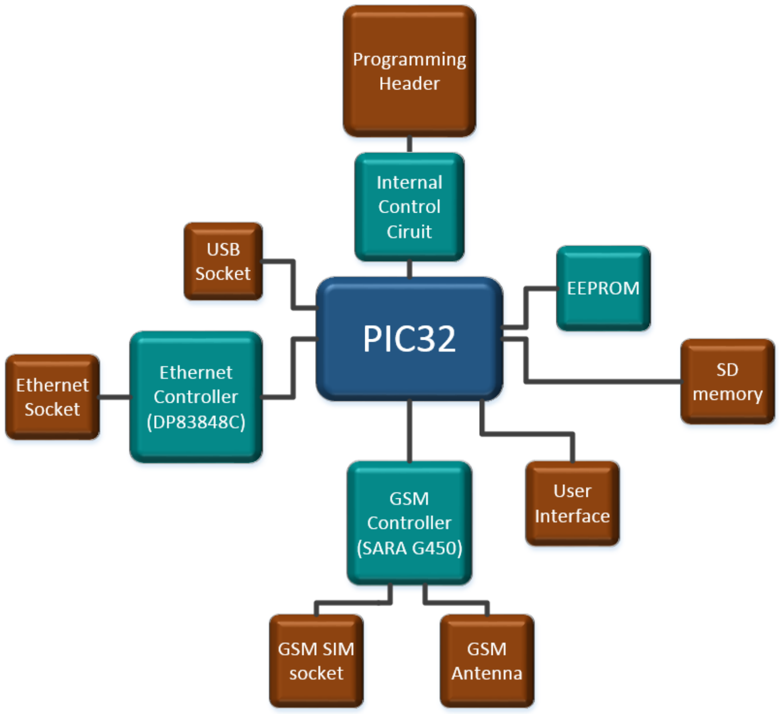


Figure 5 SSP hardware block diagram

The PCB has 4 layers and components are situated on both sides of the PCB. Figure 6 shows the different parts of the design. Each part is explained in detail in its respective section.

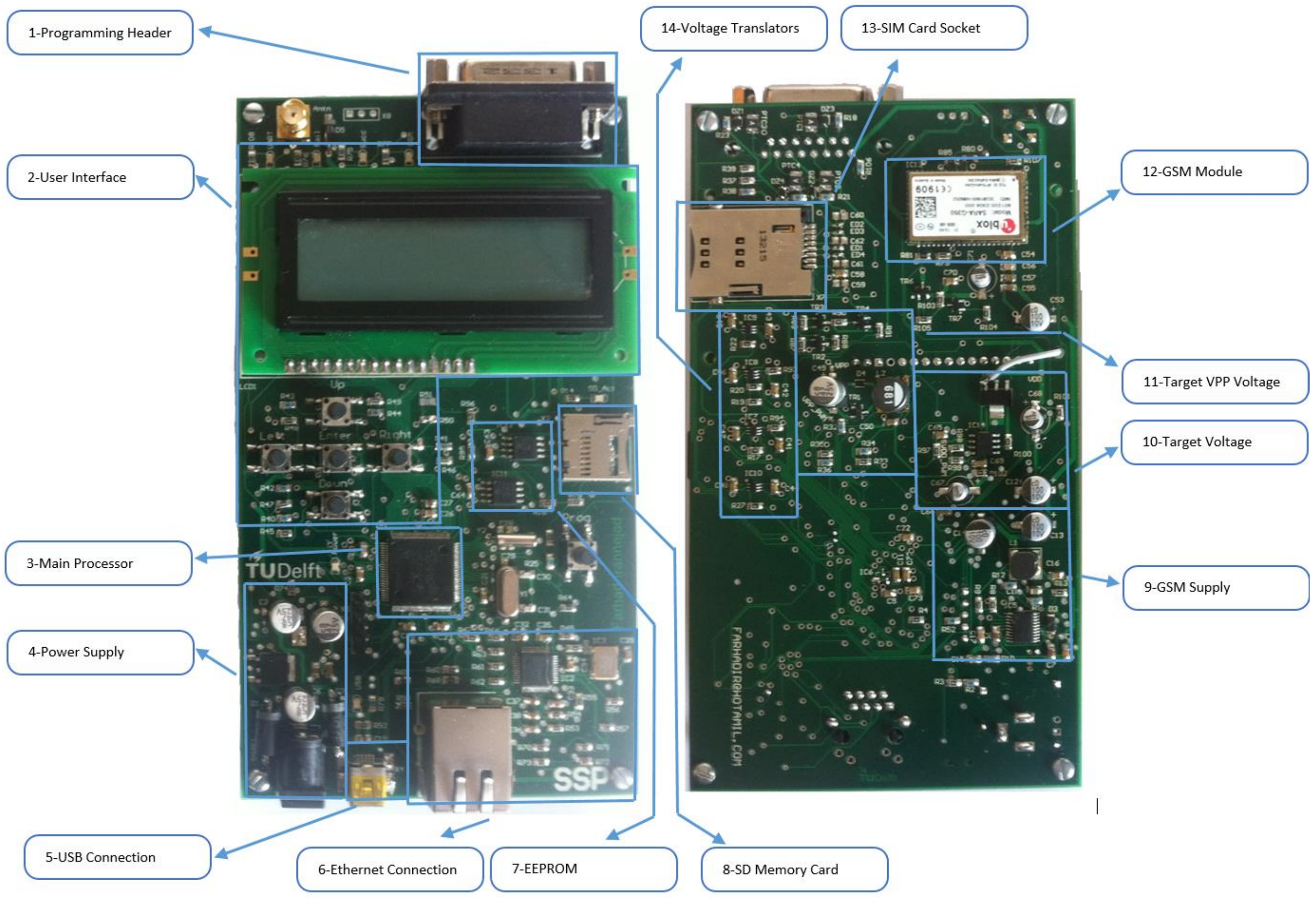


Figure 6 SSP Hardware – Top and bottom side of the board

2.2.1 Programming header

The Programming header carries In Circuit Serial Programming (ICSP)[7] signals and also a bidirectional UART connection, in order to send and receive information to/from the target device.

In Circuit Serial Programming (ICSP) is a technique in which a programmable device is programmed after the device is placed in a circuit board.

To program a PIC microcontroller with ICSP, 5 signals are necessary[7]. See Figure 7:

1) Programming voltage (normally 9 volts) – Target_VPP
2) Supply voltage (normally 3.3 or 5 volts) – Target_VDD
3) Ground reference
4) Programming clock – Target_CLK
5) Programming data – Target_Data

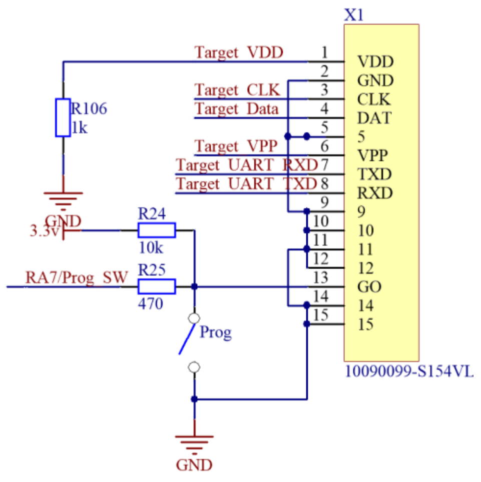


Figure 7 Programming Header

There is also possibility to connect an extra program key to the programming header (Pin number 13) to make the operator's job easier.

2.2.2 User interface

The user can communicate with the programmer with a couple of push buttons and a character LCD. A conventional 2*16 LCD[8] integrated with the HD44780 controller is used for this programmer. Five push buttons are used as joystick for the user to surf menus, and there is a special button for programming. There are a couple of LEDs for showing the status of the programmer, including: 1) USB 2) Power 3) Programming Failed 4) Programming Passed 5) UART 6) GSM 7) SD memory action

2.2.3 Main Processor

PIC32MX795[4] is selected for this design because of its rich set of peripherals. There are stacks for USB and Ethernet connection for this processor, which makes it easy to connect this board to the outside world. This microcontroller has a powerful MIPS4K processor inside, which gives it the performance of 1.65 DMIPS/MHz or 3.28 Coremarks™ MHz[9]. There are also many design examples, application notes and development kits available for Microchip's microcontrollers which make the design process faster.

2.2.4 Power Supply

The programmer can be powered via a USB cable or an external power jack. Barrier voltage diodes are used to choose the maximum voltage between the USB and the external power source, which makes it possible to power the board with both connections simultaneously. The input voltage is stepped down by a 3.3 volts linear voltage regulator to power the components on the board. Two layers of the board are dedicated to the power supply, one for 3.3 volts and the other for ground.

2.2.5 USB Connection

The PIC32 is integrated with a USB host and device controllers. This microcontroller can be connected to the USB socket directory. USB connection is used to transfer data from PC to SD memory within the programmer in a fast and reliable way. Microchip has provided a stack for full USB support and many experimental setups, in order the programmer become familiar with the stack.

2.2.6 Ethernet Connection

Ethernet connection is added to the design, for the design to be able to connect to the internet. Not only to upgrade its firmware from a server that is not locally connected to the programmer, but also to be able to do some extra controlling and monitoring to the connected target. This extra functionality can be used to debug the device that the target is designed for or the target itself. PIC32MX795L is integrated with Media Access Controller (MAC) [4]. With just a phy chip and an Ethernet socket with coupling transformers for hardware [16] and use of TCP/IP which Microchip has provided [9], data transmission through the internet from and to the device is deployed easily.

2.2.7 EEPROM

Two 1M bits EEPROM chips are used to store the encrypted firmware. These EEPROMs are used as a buffer that makes the programmer much faster compared to reading information directly from the SD memory card. EEPROMs are connected to the PIC32 by a SPI bus.

2.2.8 SD Memory Card

Micro SD memory is used to store firmware and in some cases log data. SD memory has a SD and SPI interface. SPI is used in this design because not much bandwidth is necessary and it requires lesser number of connection wires. Microchip's MDD stack [9] is used to access the formatted data stored in the SD memory.

2.2.9 GSM Supply

SARA G350 GSM module [10] is a high power demanding device. It cannot be powered by linear voltage regulator, because it would waists a lot of power. As the manufacturer of the GSM module suggested, the LM20343 switching voltage regulator [11] is used. The design example of the chip's datasheet is used for this design.

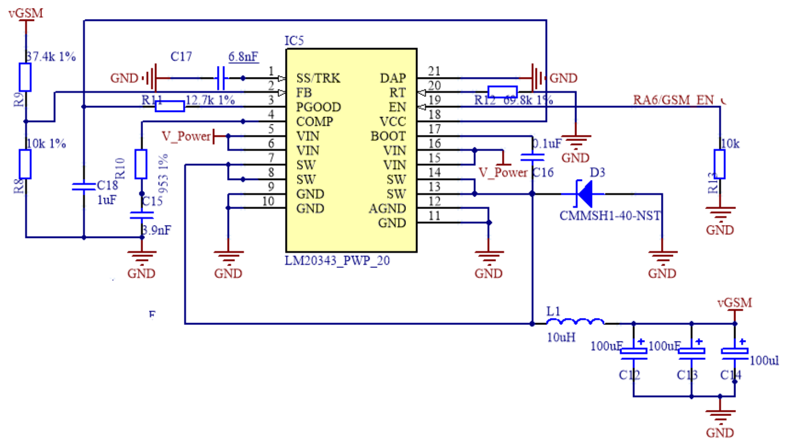


Figure 8 GSM Power Supply Schematic

2.2.10 Target Voltage Regulator

The Programmer is able to program microcontrollers with different voltage levels and it can also power the target board. Circuit below is designed as a variable DC power supply. The circuit is controlled by a PWM output of the programmer's microcontroller (RD2). This PWM signal is low pass filtered by a simple resistor capacitor network to generate a pure DC reference voltage, then this reference voltage is compared to half of the output voltage (TargetV)[12]. In this way this power supply can generate voltages from zero up to twice the input voltage that is standard 3.3 volts and the maximum of the input voltage. This output voltage is then read by microcontroller's ADC to adjust the PWM duty cycle, in order to maintain output voltage fixed and compensate nonlinearities.

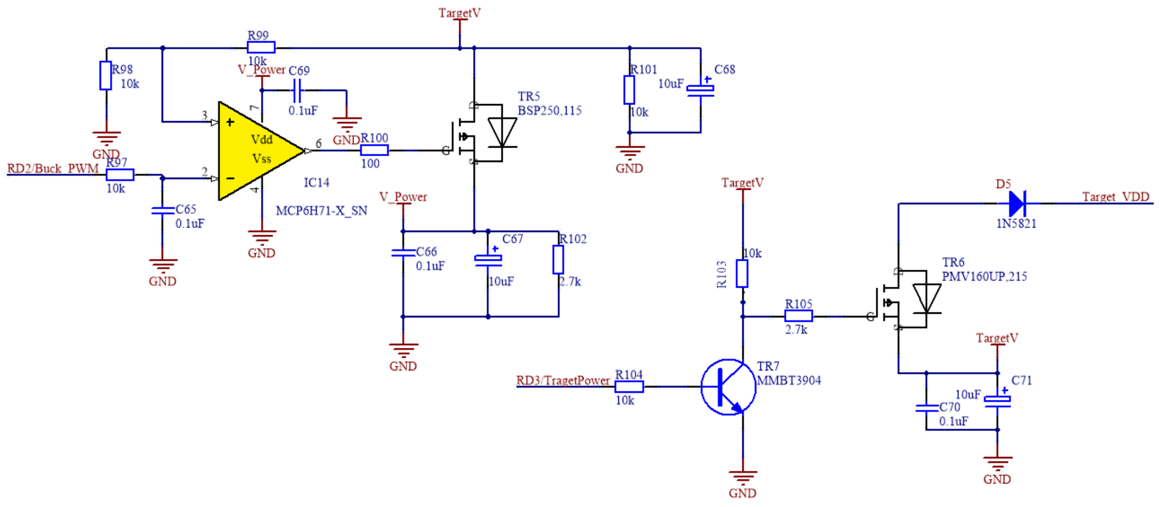


Figure 9 Power supply for programming

This internally generate voltage can supply the target device by TR6 a P-Channel mosfet. There is a blocking diode 'D5' in the case if the target board is already powered.

2.2.11 Target Vpp Voltage

For programming a PIC microcontroller, a 9 volts programming voltage is required. Because there is no voltage higher than input voltage (normally 5 volts) available in the programmer, it is necessary to have a boost power supply for generating this voltage [12].

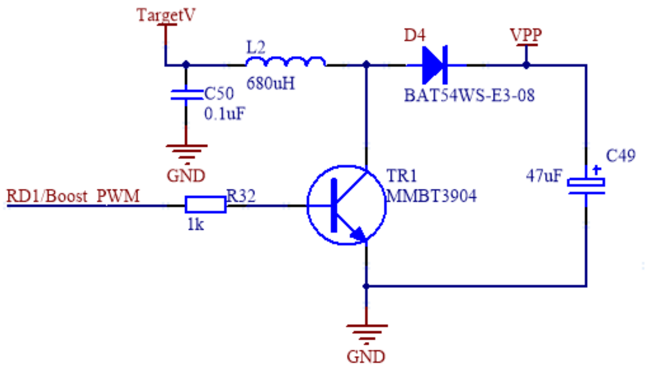


Figure 10 Boost power supply for generating 13 volts

Figure 10 demonstrates the boost voltage generator used by this programmer. The circuit is controlled by a PWM signal from the microcontroller. By adjusting the duty cycle of PWM signal it is possible to change the amount of stored energy in the inductor (L2) in each PWM cycle. The inductor's current is proportional to its energy and the current goes through the capacitor C49 and the capacitor integrates the current and generates the output voltage VPP. Diode D4 avoids the current to flow back from the capacitor. In this way it is possible to generate higher voltage than the input voltage. The output voltage (VPP) lowered by a passive voltage divider then it is sampled by the microcontroller's ADC to control the input PWM duty cycle in order to maintain the output voltage at the desired voltage [12].

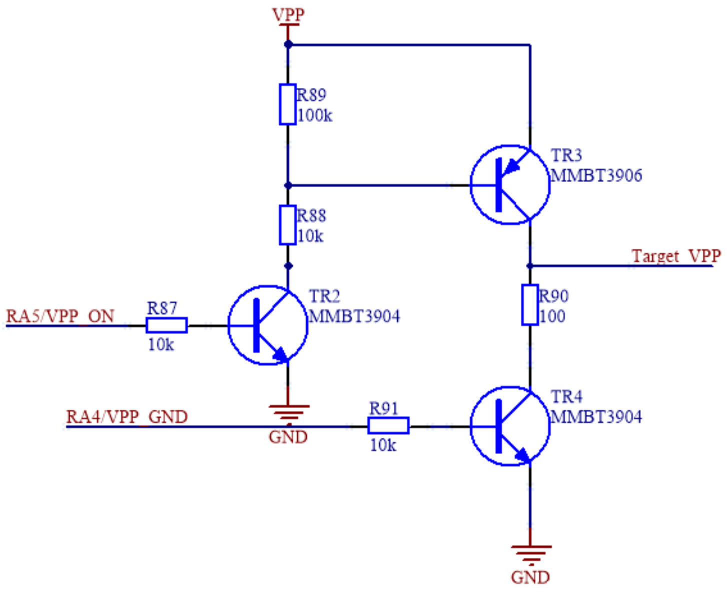


Figure 11 Vpp switch circuit

The MCLR pin of the target device or the programming voltage of the programmer has to able to be in 3 states, naming: 9 volts (programming mode), ground, and floating (disconnected). The circuit in Figure 11 is designed to switch the target's MCLR pin to one of the required states [13].

2.2.12 GSM module

A GSM module is added to the design to make the programmer be able to connect to the internet, even when there is not a proper infrastructure like Ethernet network available. The GSM module that is used for this project is SARA-G350[10], which is designed specifically for machine to machine (M2M) interface purposes. This module supports microphone and speaker connection, but they are not used for this design. Connection to this module is via the PIC32 UART module. Because the voltage level in GSM module is different form the voltage level used for the PIC32, a simple resistor voltage dividers are used for voltage level translation.

Right now the module only supports 2G, but manufacturer has announced that, this year they will introduce a new module with support for 3G with a complete compatibility to the 2G version.

2.2.14 Voltage Translators

The programming method of the PIC microcontroller is serial. For communicating with the PIC memory controller two signal paths are necessary; one is the clock signal and the other is a bidirectional data signal[12].

The clock signal is a one directional signal from programmer to the target device. Because the target voltage might be different from the programmer's internal voltage, a voltage translator is necessary. A bipolar voltage translator IC (74LCV1T45) is used for this purpose that is also able to make the output signal floating. The output of the circuit is protected by a PTC and a Zener diode to avoid over current and voltage situations. The input of the circuit (RF13) is directly connected to the PIC32 pin and the output (Target_CLK) is directly connected to the programming header. Output can be enabled by RA9.

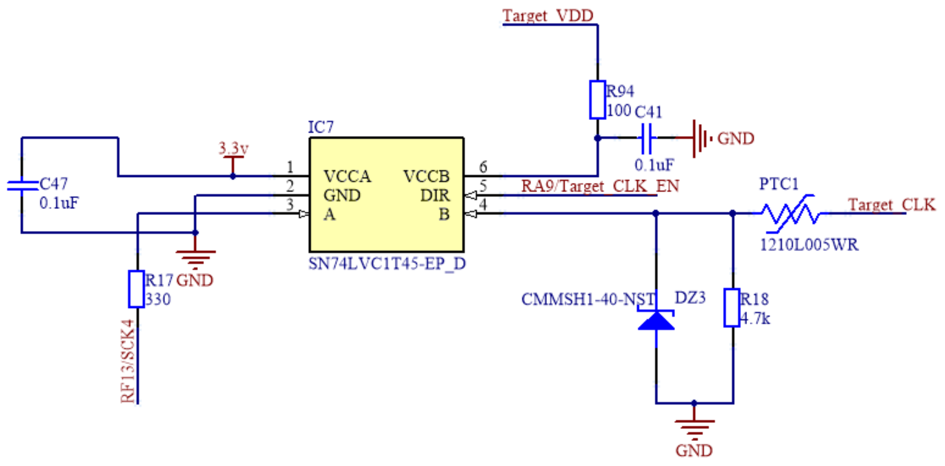


Figure 12 Programming clock circuit

Data signal driver (Figure 13) is almost the same as the clock driver circuit, except that the data signal is bidirectional and data has to flow from programmer to target and vice versa[12]. Data flow direction is controlled by 'DIR' pin of the 74LCV1T45 IC[14]. If the 'DIR' pin is low the 'Target_Data' value can be read by RF4 pin on the PIC32 and if the 'DIR' pin is high the 'Target_Data' value can be written by RF5 pin on the PIC32.

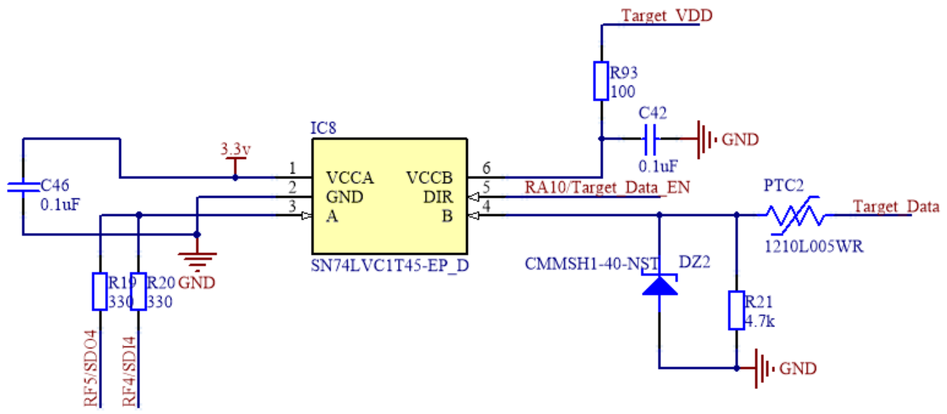


Figure 13 Programming data circuit

In the programmer socket an extra UART connection is added to let the programmer connect to the target microcontroller and gather/send some data from/to the target device and translate this information to TCP/IP packets. This extra connection path can be used for remote controlling or debugging.

2.3 SSP SOFTWARE

Like many other Embedded Systems, the software for SSP which is running on the PIC32 is written in C. Microchip offers MPLAB X IDE and XC32 compiler for developing software for PIC32 for free [9]. Also TCP/IP and USB stacks can be downloaded from Microchip's website for free [9].

The software part of SSP is divided in five parts:

1) User interface
2) Programming
3) Internal control
4) Ethernet controller
5) USB controller

2.3.1 User interface

Software reads push buttons states and shows the status of the programmer on a couple of LEDs. It also controls a 2*16 HD44780 character LCD via a parallel port. User can select a specific firmware by pressing "Enter" key and choose the right firmware according to the information showing on the LCD. By holding the "Enter" key for more than 2 seconds the user is able to load the firmware from SD memory to the external EEPROM on the board. By pressing the program key the programming sequence starts. After programming, a red LED will illuminate if the programmer failed to load the firmware to the target or a green LED illuminates meaning a successful write to the target.

When the USB cable is connected, the USB led will start blink with the frequency of one hertz.

2.3.2 Programming

For the start of programming the data first needs to be copied to the SD card of the programmer. There are a number of ways to do so:

1) Detaching the SD card from the programmer and connecting it directly to a PC and copying the SSP file to the card and then plugging the SD card back in the programmer
2) Connecting via programmer's USB connection and copy the SSP file to the programmer. After connection, the PC will recognize the programmer as a memory storage device
3) Copying data using FTP or a custom TCP/IP protocol. In this case it is necessary for the programmer to be connected to Ethernet connection

2.3.2.1 Copying Data to the EEPROM

After loading the SSP file to the programmer, programmer has to verify if the SSP file is not modified and file's publisher is known. To do so, the signature part of the file is extracted from the file and it is decrypted using the public RSA key. The result is the message digest that has to be the same as the file's digest which is calculated in the programmer. If both digests are the same, the header and encrypted file can be stored in the EEPROM, if not the file is deleted. The RSA key is stored in the PIC32's internal flash memory, and the configuration bits of the PIC32 are configured in a way that avoid any read of the data in its flash memory. Data is copied to an EEPROM IC because of its faster access time and simplicities. In this way programmer will be able to continue programming even without the SD card attached. The message digest is also copied to the EEPROM in order to check the integrity of the data afterwards. The data in the EEPROM is still encrypted because the EEPROM communication bus can be monitored and the written data could be extracted. Figure 14 demonstrates the process of copying data from the SD card to EEPROM.

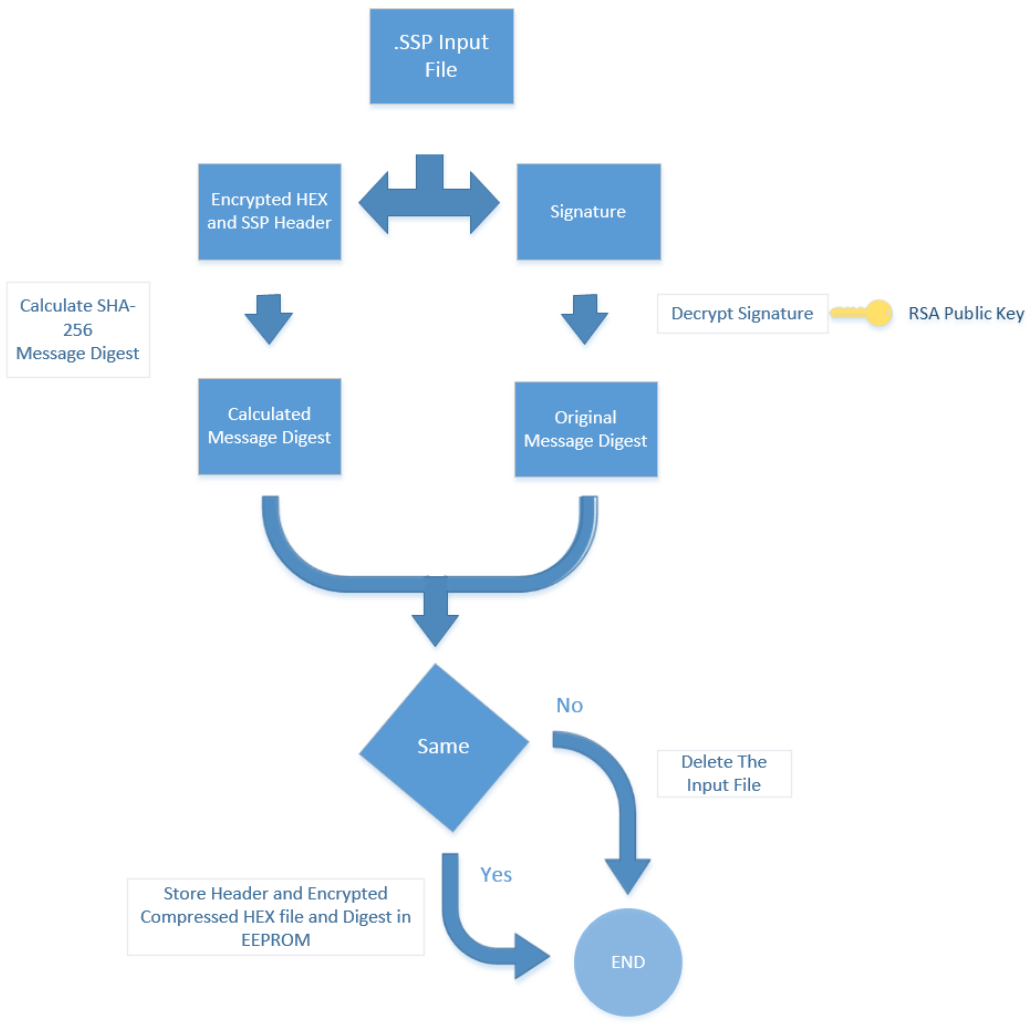

.SSP Input File
Encrypted HEX and SSP Header
Signature
Calculate SHA-256 Message Digest
Decrypt Signature
RSA Public Key
Calculated Message Digest
Original Message Digest
Same
No
Delete The Input File
Yes
Store Header and Encrypted Compressed HEX file and Digest in EEPROM
END

Figure 14 Signature verification and loading a SSP file to the programmer process

2.3.2.2 Reading data from the EEPROM

For programming the target device, data is loaded from the external EEPROM chip to the microcontroller in blocks, because there might not be enough space in PIC32's RAM, which is 128KB, to store the whole firmware for programming chips with high amount of flash memory capacities (in MB range in some cases). Each HEX record has the size of 21 bytes and the AES encryption block size is 16 bytes. In order to avoid extra calculation for identifying a record start, each time the least common multiple of the two numbers (336 = 16*21) bytes are read from the EEPROM. Then this block is decrypted and buffered locally in PIC32. The key for decryption is stored in the PIC's flash memory and configuration bits of the PIC32 are set in a way that it avoids a read of the flash memory. Then by each call of the higher level software a HEX record is checked and is returned if the checksum is correct. When there are no more buffered records, again the microcontroller will access the external EEPROM chip to refill the buffer with a new block. This process continues until the end of the file is reached. Figure 15 shows the flow chart of reading data from the EEPROM.

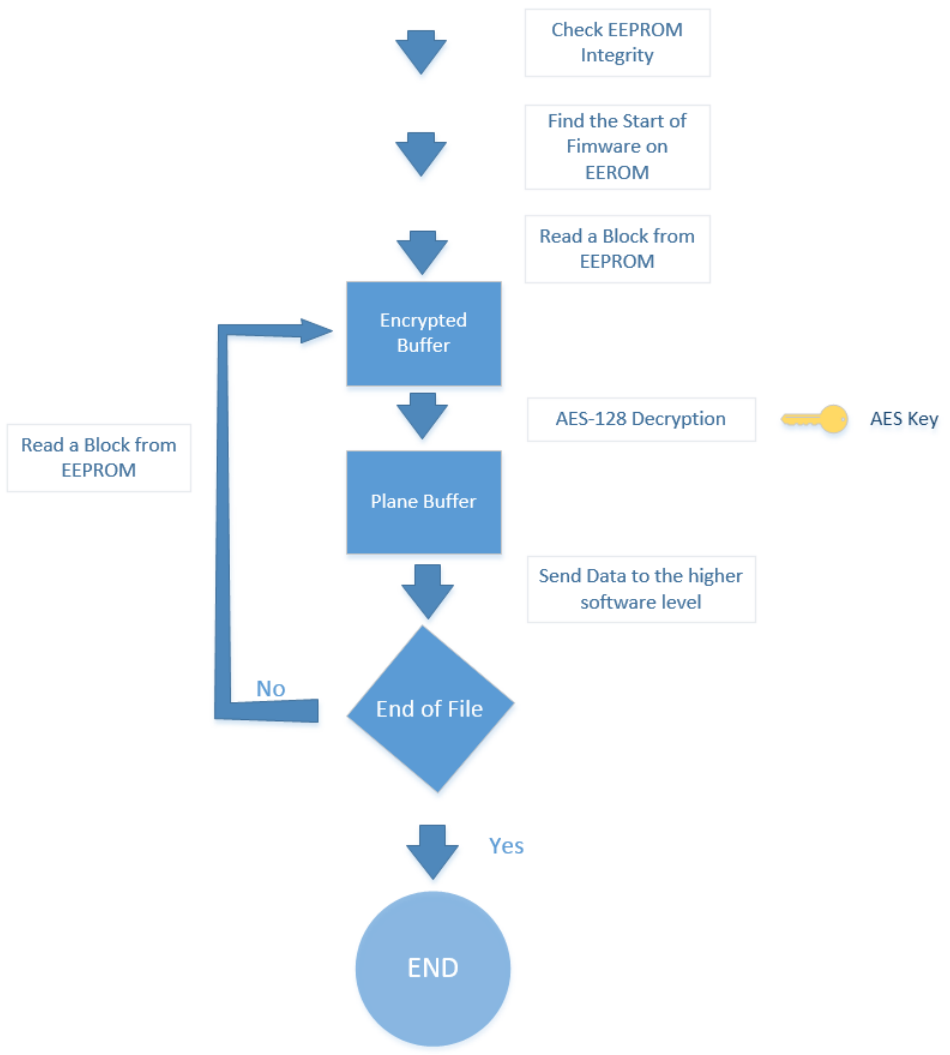


Figure 15 Decryption and reading a record process

2.3.2.3 Programming Sequence

The first stage of programming is to check the target device part number. This number is read from the target and compared with the value in the header of SSP file. If the target device does not match the SSP file the program sequence is aborted, if they match the target is erased for making the target ready for writing data. The software is implemented in a way that with each call to a function, successive data in the memory location is returned from the local buffer. After erasing the target, microcontroller's FLASH and EEPROM memories data are written to the target device. Then the written data is verified and after that configuration bits (including Flash write protect) are written and verified[13].

The integrity of the EEPROM memory is checked with SHA algorithm two times, in order to make errors as low as possible. First time is at the start of the programming, before writing, and the second one is for the verification stage. Also the check sum of each HEX record is checked before returning the data to the higher software layer (programming layer).

2.3.3 Internal control

The software is able to read voltage levels of Input, Target voltage, generated target voltage and programming voltage. The PIC32's ADC is configured to read voltage levels automatically (one after the other and automatic sampling). The software generates an interrupt when all the voltages are read. The maximum allowed voltage for the PIC32's ADC is 3.3 volts, in order to be able to measure higher voltages a passive voltage divider is used.

For the power supplies, in order to make the generated target voltage and the programming voltage at the required level, the duty cycle of the PWM is modified in the interrupt service routine of ADC. Figure 16 show the conversion and control block diagram used in software for the programming voltage circuit and also for the target supply voltage controller.

A Simple proportional controllers with saturation value are used to control the output voltages.

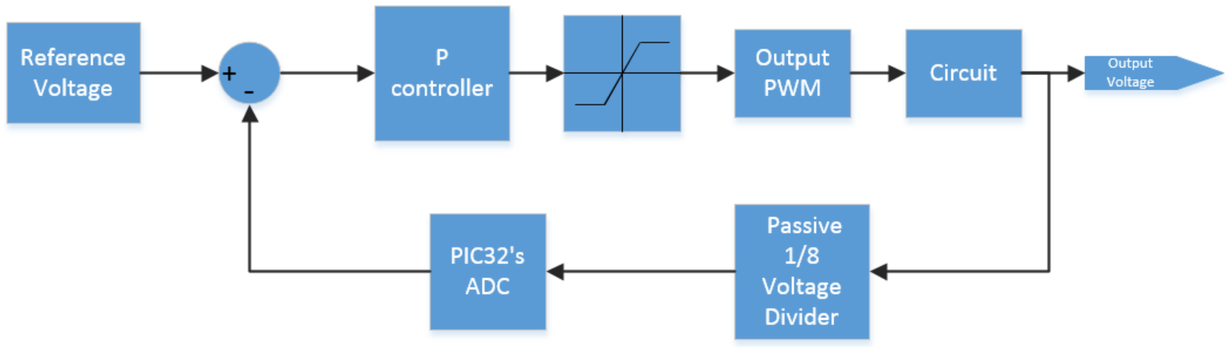


Figure 16 Voltage controller

2.3.4 Ethernet controller

Microchip has provided a full TCP/IP stack for PIC32 for free [9]. This stack has to be configured to meet the specific design requirement. The Stack is running a DHCP client to gain an IP address from a router or a server. For the user to be able to see some information and the status of the programmer via computer terminal, SSP also runs a telnet server. A TCP-UART bridge server is also running on the board which can be used to debug and monitor the target device remotely over the internet.

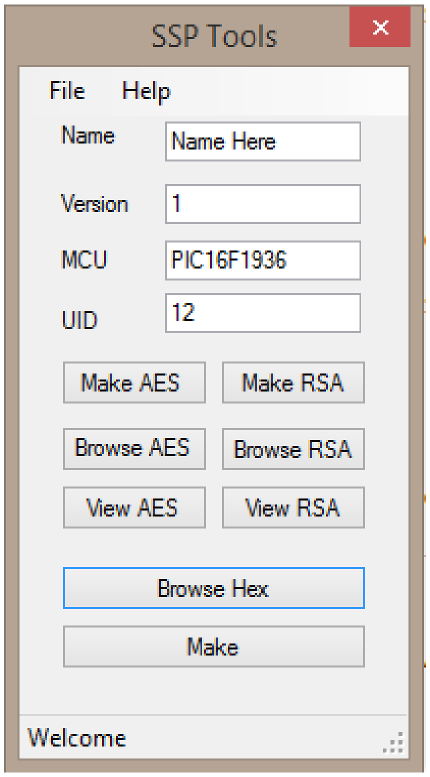
2.3.5 USB Controller

Likewise Ethernet Microchip has provided a full USB stack for PIC32 for free [9]. This stack is configured to act as a memory storage device when it is connected to a computer. By this way a PC recognizes the SSP as a memory device and the user can easily copy and paste SSP files to the SD memory of programmer without the need for any special device driver.

2.4 PC APPLICATION

A PC application is developed to encrypt a HEX file and sign it, in order to generate the output SSP file. This application is developed for a Windows machine with the aid of Microsoft's .Net framework in C# language. Figure 17 shows the main application's form.

In the PC application a form is filled by the user about the name of the file, version, target microcontroller, etc.; this information is used to make the header for the SSP file. User needs to load the HEX file and also two keys for generating the SSP file. The keys are stored on a disk encrypted using a password and for loading the keys the correct password has to be entered by the user.

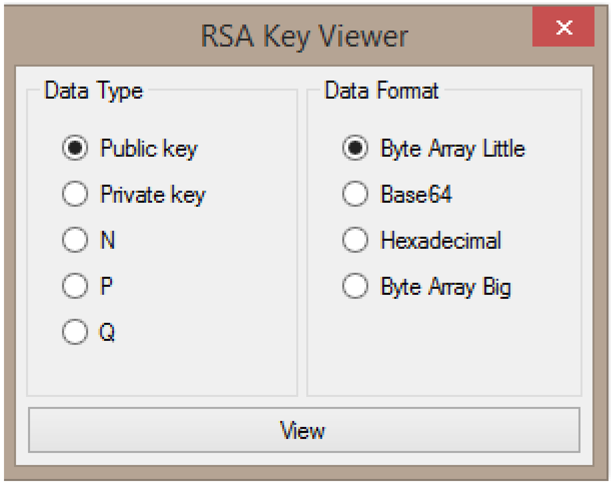


Name: Name Here
Version: 1
MCU: PIC16F1936
UID: 12
Welcome

Figure 17 PC Application's main form

2.4.1 Key management

The PC application is able to generate the AES and RSA keys. These keys can be viewed in a number of ways including hexadecimal integer, little endian byte array, big endian byte array and base64 string. These keys are included in the SSP software at the compile time as a byte arrays. Keys for the PC application are encrypted and signed with the same method that is used for SSP file; keys are encrypted with the AES algorithm, then the header is added and then the message is signed with the application's private key. The AES encryption key is derived from a text string that the user has chosen as the password for the key. A string is used as password for the user's convenience. The password string is converted to a binary byte array using RFC2898 protocol [15] and the output is used as AES key for symmetrical encryption. Keys can be stored in hard drive to be loaded later in to the application.


RSA Key Viewer

Data Type: [✓ Public key] Private key N P Q

Data Format: [✓ Byte Array Little] Base64 Hexadecimal Byte Array Big

Figure 18 Key viewing form

2.4.2 SSP file generation process

After the keys for the file generation are loaded and the form is filled, a HEX file needs to be loaded to the application. Figure 19 shows the overview of the process to generate the SSP file. First the HEX file is compressed, using the method mentioned in SSP file format. Then it is encrypted using AES-128 and after that the formatted header is added to the encrypted HEX file, next the message digest is calculated and it is encrypted by a private RSA key to make the signature of the whole file which is then added to the end of the file.

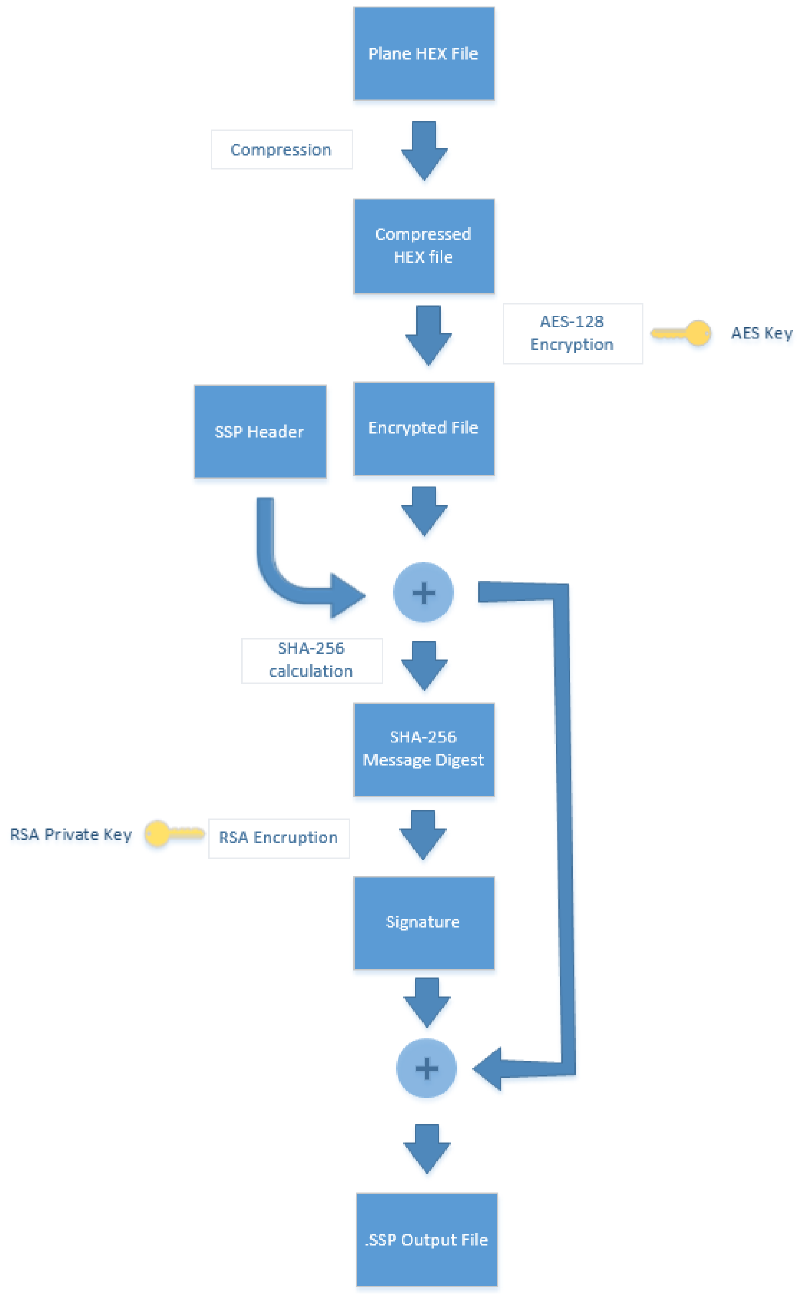


Figure 19 SSP file generation process

3 DISCUSSION

There are many IC programmers in the market but the special thing with this programmer is the ability to write the firmware to the target in a secure way which means the firmware file generated for the target device can only be decrypted with the programmer holding the key for the file. Another big advantage of this programmer is its standalone feature which makes it able to program the target device without the intervention of a PC, which makes it suitable for production and field programming. It is also able to hold a number of firmware in its storage to program different target devices.

All of the information required to program the device is stored in the SSP file, so there is no need for the operator to configure anything and with a single button the target device could be programmed.

The SSP targets low end Microcontrollers. These microcontrollers have the highest annual production rate compared to other processors and microcontrollers. Low end microcontroller are designed to have as less price as possible.

The encryption involved does not add much over head to the programming time. 300 milliseconds for the total of 2.3 seconds programming time of 5K bytes of a PIC microcontrollers flash memory.

3.1 PROBLEMS IN THIS SYSTEM AND COMPARISON TO OTHER METHODS

The method used in the SSP will be able to offer security to some extent and it is to avoid simple misuse of the firmware. As in the last stage of the programming (connecting to the target device) there is no security involved and the programming bus can be monitored and data could be extracted. There is no way for avoiding this, as a security method must be developed by the chip manufacturers.

For more advanced systems like mobile devices, designers have solved this problem by using a secure bootloader[1]. In this case a bootloader with a special key is loaded to the device and later a user can simply upgrade the firmware of the mobile device with encrypted and signed firmware via the bootloader. Even in this case the bootloader has to be written to the target in a secure way.

4 Future Work And recommendations

The SSP has many different modules that may be even used for other purposes other than programming as a development platform.

Below a couple of ways for improving and expanding the usage of this programmer are discussed.

4.1 Programming counter

One of the main issues that could not be solved during this project is the programming count. In order to fully monitor the usage of the programmer it is useful to maintain information about the number of successful programming counts for each firmware and somehow limit it. The SSP file includes a unique identifier and this number is protected by the signature of the designers. The programmer can check if it has program the specific firmware before and the programming count it is not higher than the value allowed by the designers. For that, it is necessary to store information on a memory. All non-volatile memory units have a life cycle and they can be reprogrammed for a finite number of times. For a counter like this, which changes often, it is necessary to devise an elegant implementation for memory write. If not, the programmer would have a low life cycle which means in the middle of the production the programmer might suddenly stop working and because the memory cells are destroyed, the programmer needs to be changed.

Three methods were found during this thesis assignment for solving this problem, each of them has its own benefits and drawbacks.

4.1.1 Writing on a non-volatile memory only in the case of power loss

It is possible to use a voltage detector circuit that finds the power out situation and interrupts the main processor. Then the processor can start writing the context information on the non-volatile memory unit while the supply's voltage is decreasing before the black out. Then again the processor can read the data from the non-volatile memory and store and modify them later in the local RAM, which has no life cycle issue.

In this way there will not be a life cycle issue as the memory cells only will be programmed in power out situations. In a very pessimistic case, repowering the device for 10 times a day, with an EEPROM with a life cycle of 100K writes, the programmer can live for more than 27 years. But the problem with this scenario is that if the power drops suddenly the processor will not have the enough time to write the data on the memory. This means if the user shorts circuits the power supply line on the board before unplugging it, the programmer cannot save the programming counts.

4.1.2 Authenticating with a server before the start of programming

A more advanced way of monitoring the programmer is to have an online programming. In this way each time before starting to program a target device, the SSP has to ask permission from a server that is maintained by designers of the firmware. This way, the programmer can only program targets when it is allowed.

SSP software is running a TCP/IP stack. It also includes a SSL layer for communications. It is possible to write a custom TCP/IP application for the SSP and also a service that runs on the above mentioned server that can communicate with each other in a secure way.

But the problem with this method is, if the in middle of production internet access for the SSP is lost then programmer fails to program, which is disastrous for a production line.

It might be a wise choice to combine this method with one of the other memory based methods.

4.1.3 Increasing the non-volatile memory life cycle by software

There are a number of methods like wear-leveling[17] that change the locations of data or decrease the number of erase cycles by writing bits instead of bytes. Another way is not to store all increments in the counter and for example only increase in number of 10. These methods will increase the life cycle of the programmer to some extent, but if the programmer is going to be used for mass production, like 10K a day, none of these method will grantee a life cycle of more than a year for the programmer.

4.2 Sharing Key between Designer and Supplier

In this design the keys for the decryption are written to the SSP's flash memory at the compile time. So the keys cannot be changed later. If necessary, one can find a secure way to change the keys remotely.

4.3 Add support for other devices

Right now this programmer only supports Microchip's enhanced midrange microcontrollers. With the same hardware it is possible to add support for all of Microchip's microcontrollers by just changing the software. It is also possible to change the programming part of the hardware design to add support for other Microcontrollers or any device with memory.

4.4 Developing an USB driver for PC for SSP

For now the only possible way for programming the target device is to load the firmware into the SD card of the programmer. For expanding the usage of the programmer it is possible to write a custom USB driver for this programmer that could make it compatible with MPLAB X IDE and that can be used directly by the IDE itself. Then there will not be a need for an extra PC application. Extra configurations and maybe status of the programmer could also be written to/read from the programmer.

It is also possible to write a driver to write the SSP file directly to the EEPROM of the programmer and omit the SD card memory as a mid-step for programming.

4.5 Internet connection through GSM network

The SSP includes a GSM module. Because the focus for this assignment was the programmer itself, it was decided not to diverge the project and invest time in optimizing the programming function of the device instead of having extra functionality for it. This module benefits from internal TCP/IP stack. It is possible to combine GSM and Microchip's TCP/IP stack together, then programmer could be connected to the internet even wirelessly. Being able to have wireless connection, then the programming could be monitored more reliable and also the programmer could be used for data logging and monitoring without the need of an Ethernet network infrastructure at the place where the programmer is being used.

4.6 Microcontroller with a crypto bootloader in hardware

One of the most secure ways to write a firmware to a target device is to have a bootloader in the hardware of the target chip. It is possible to add this crypto bootloader when you can modify the hardware of the chip as an example to an open source soft core in a FPGA.

5 Conclusion

The SSP is designed to write a firmware or any other type of data to a device with memory in a secure way. The SSP can be used for writing data in the target's memory and protect the design's intellectual property by using cryptographic methods. In this proposed method of programming, in order to avoid changes in the file, the firmware is also signed. The cryptographic methods that is used in this system (AES and RSA) are known to be the most secure methods at the time of the design.

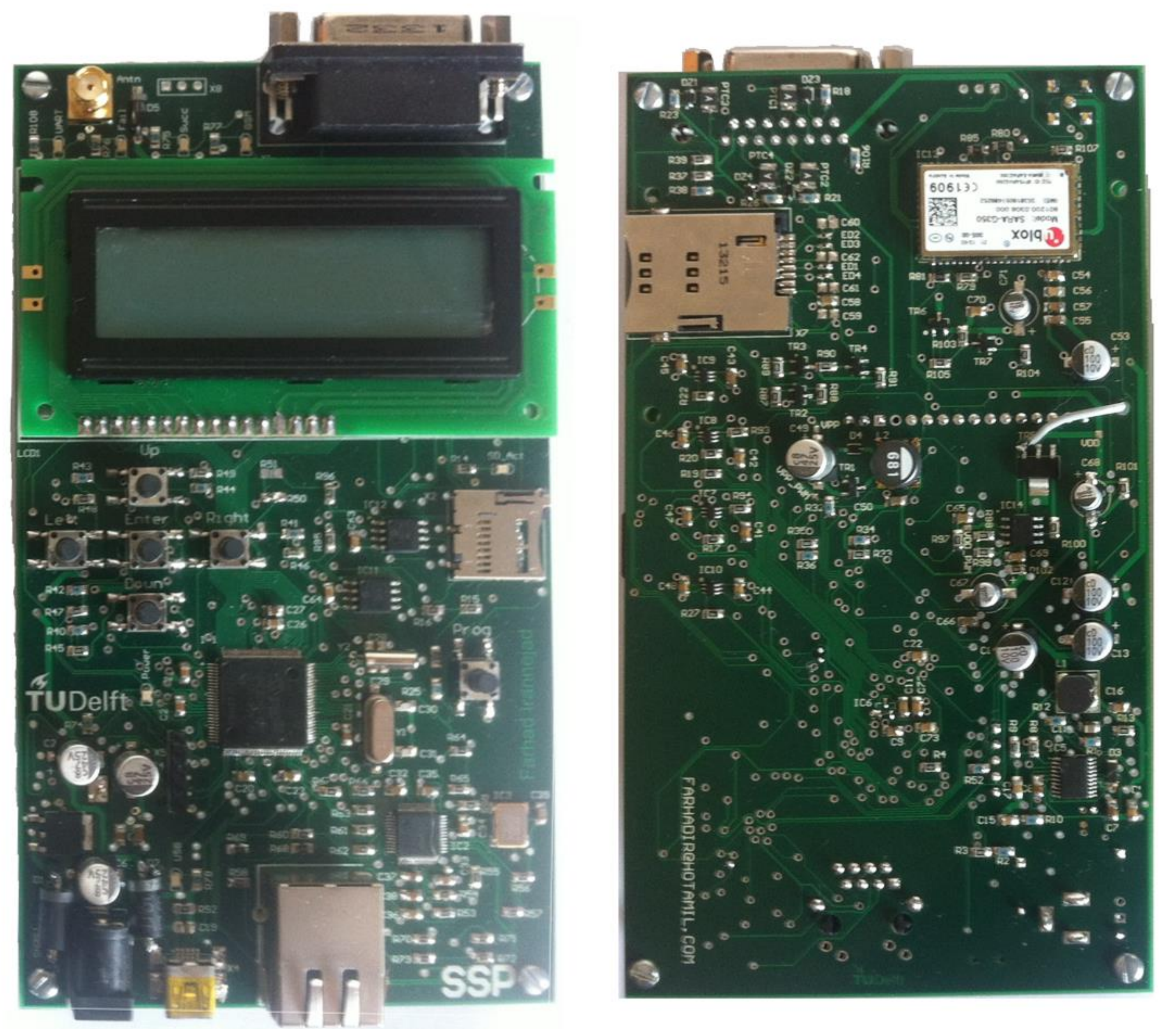
The method offers protection of the software to some extent, for simple designs it is a valid product. It can also be used for remote debugging and development of a product.

6 References

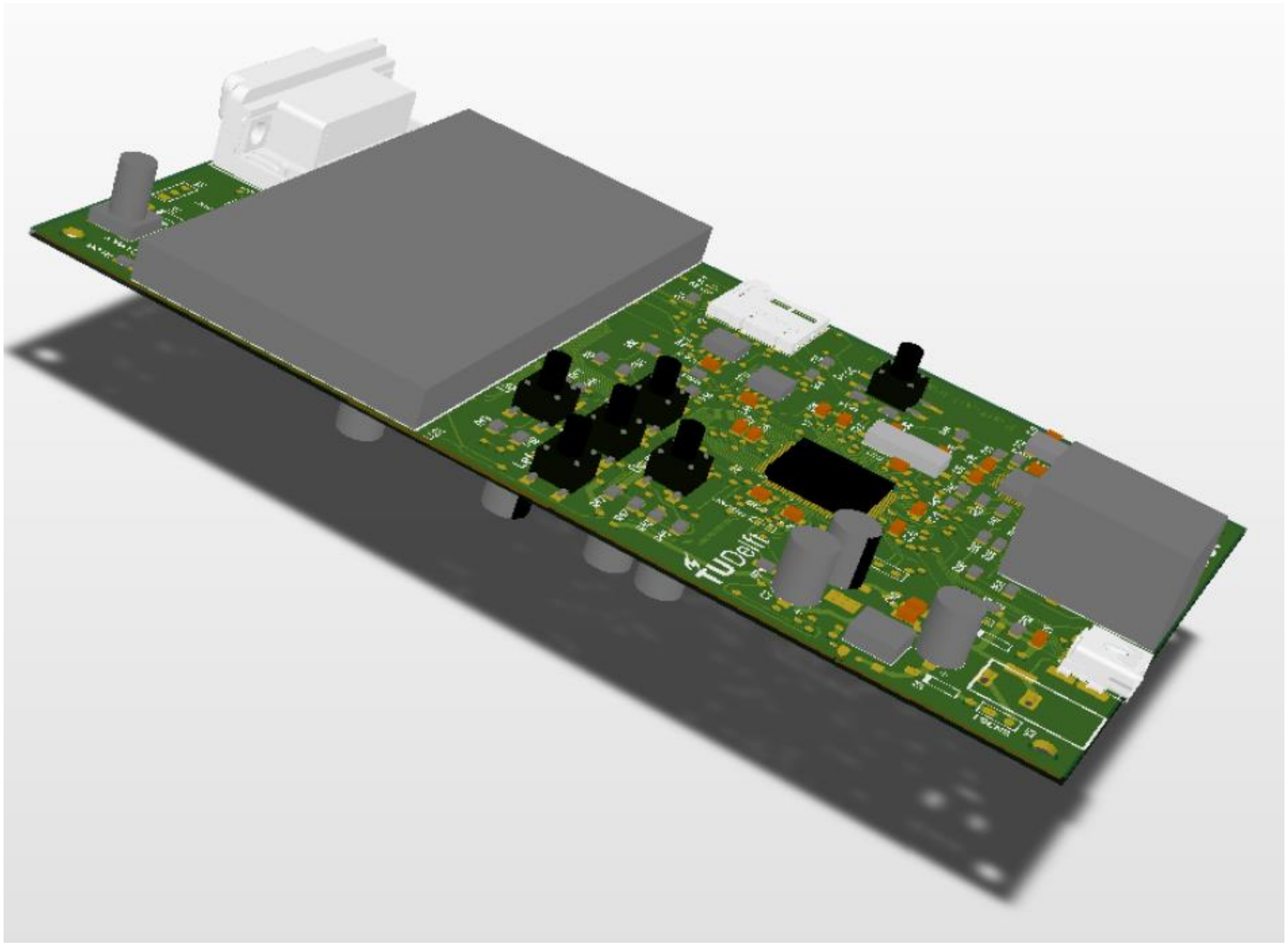
1- Atmel, AVR231: AES Bootloader, March 2012
2- www.softlog.com
3- Intel Hexadecimal Object File Format Specification Revision A, June 1988
4- PIC32MX795F512L Data sheet
5- How secure is AES against brute force attacks? Mohit Arora, Sr. Systems Engineer & Security Architect, Freescale Semiconductor
6- The Transport Layer Security (TLS) Protocol Version 1.2
7- Microchip In-Circuit Serial Programming™ (ICSP™) Guide
8- HDM16216H-B 2*16 character LCD data sheet
9- www.microchip.com
10- SARA-G3 series GSM/GPRS modules Data Sheet
11- LM20343 36V, 3A Adjustable Frequency Synchronous Buck Regulator Datasheet
12- Microchip PICkit™ 3 Programmer/Debugger User's Guide
13- Microchip PIC16F193X/LF193X Memory Programming Specification
14- Texas Instrument SN74LVC1T45 Datasheet
15- Password-Based Cryptography Specification Version 2.0 RFC 2898
16- PIC32 Ethernet Starter Kit User's Guide
17- Atmel, AVR116: Wear Leveling on DataFlash , July 2012
18- http://en.wikipedia.org/wiki/Intel_HEX

7 APPENDICES

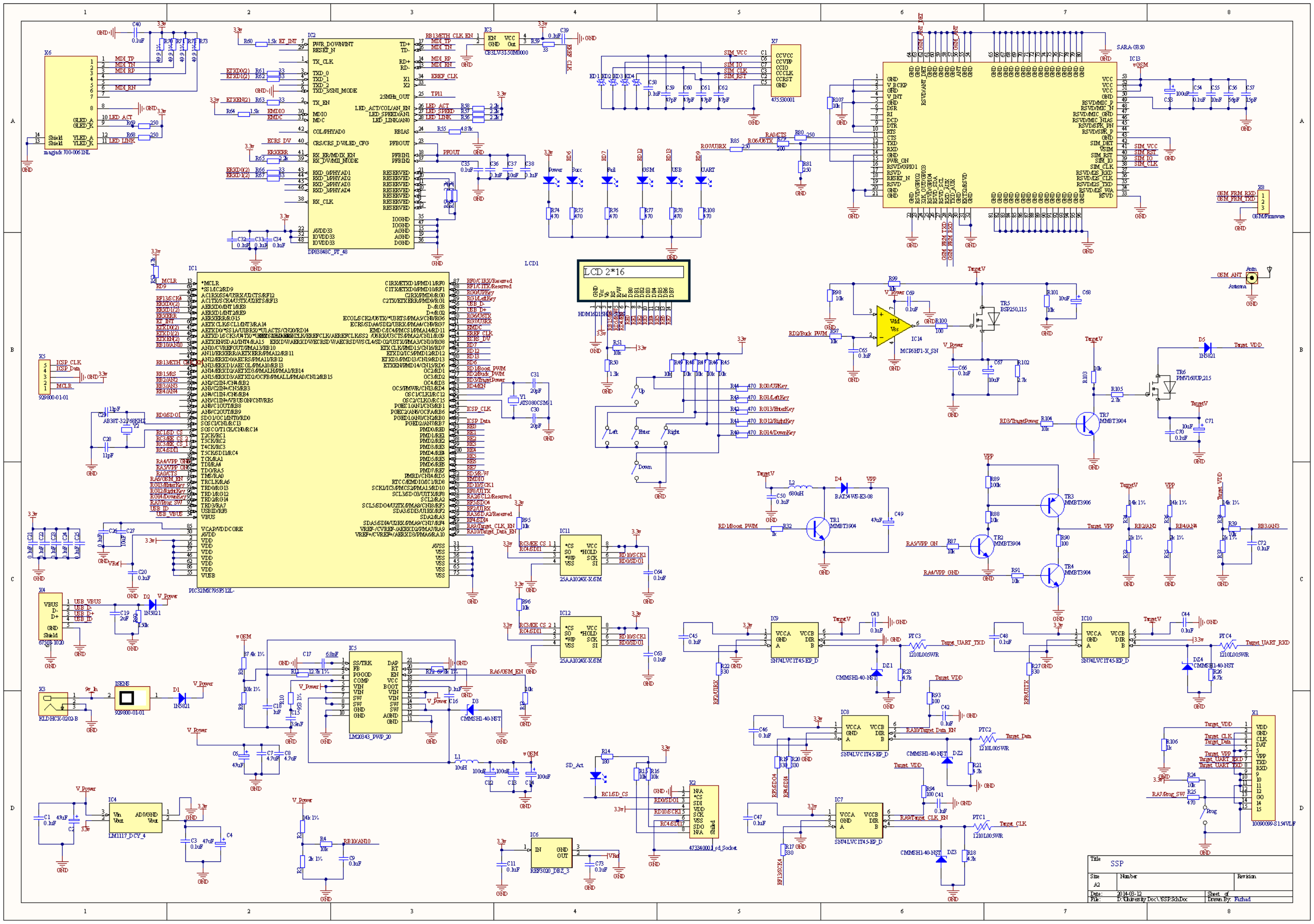
7.1 SSP PROTOTYPE



7.2 SSP 3D MODEL

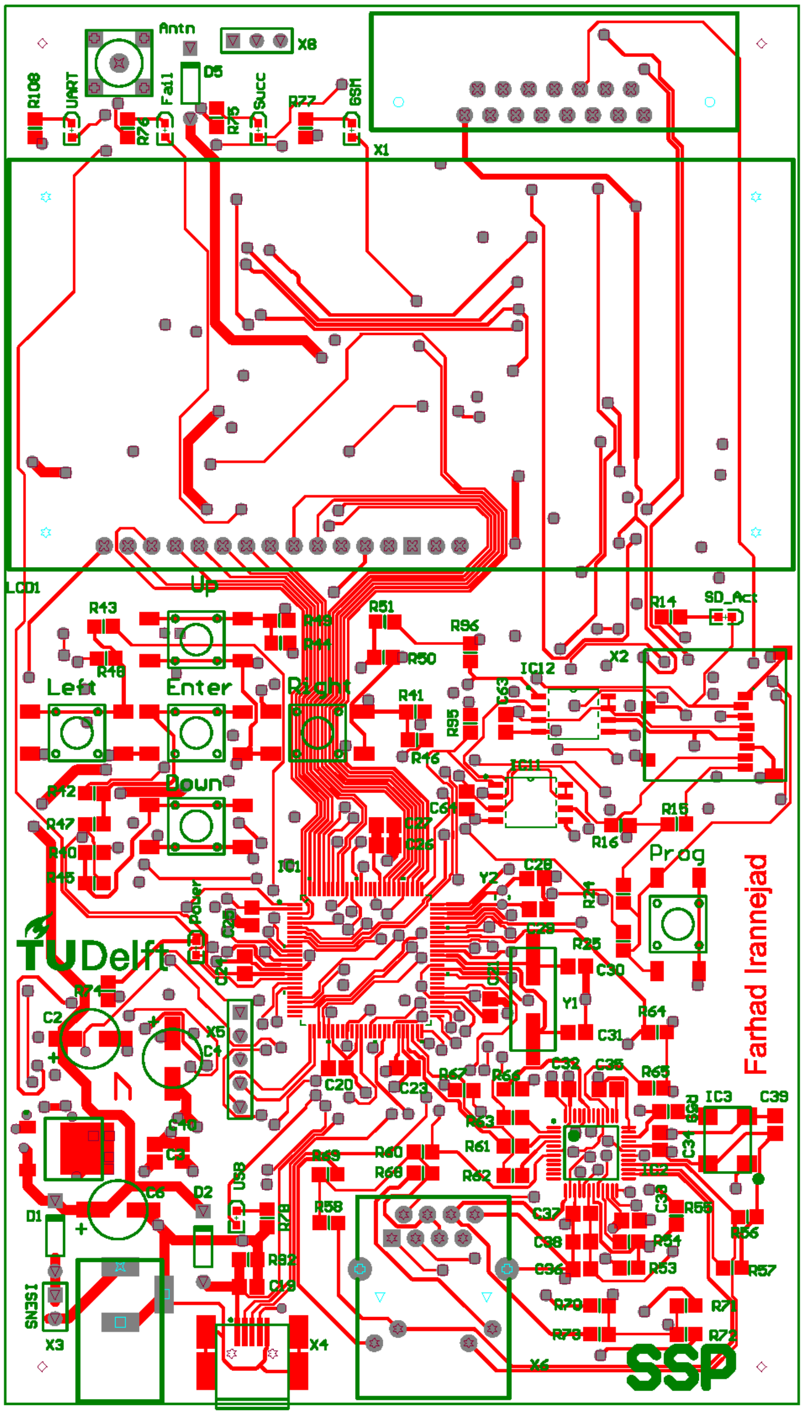


7.3 SSP Hardware Schematic

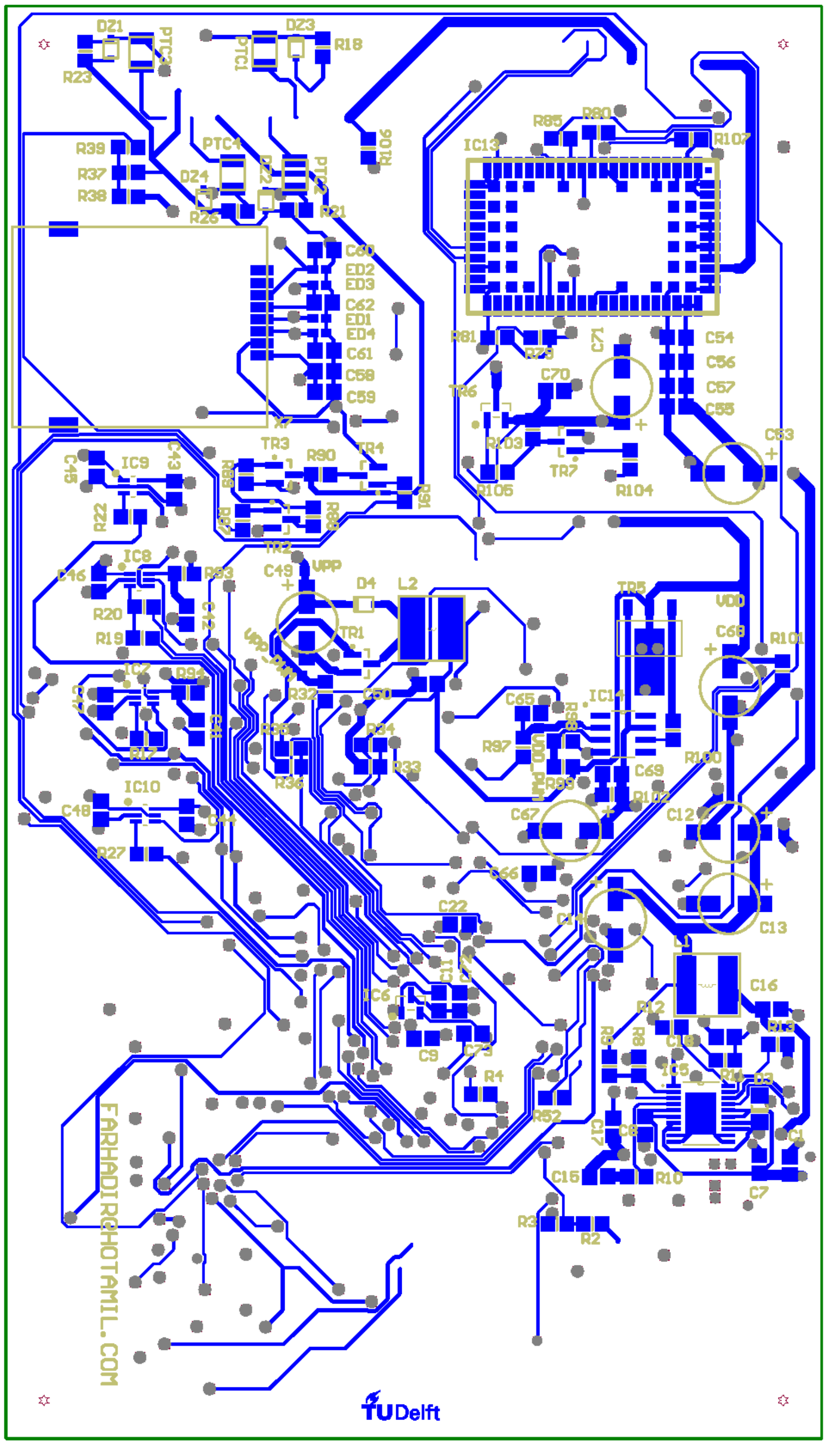


7.4 SSP PCB SKETCH

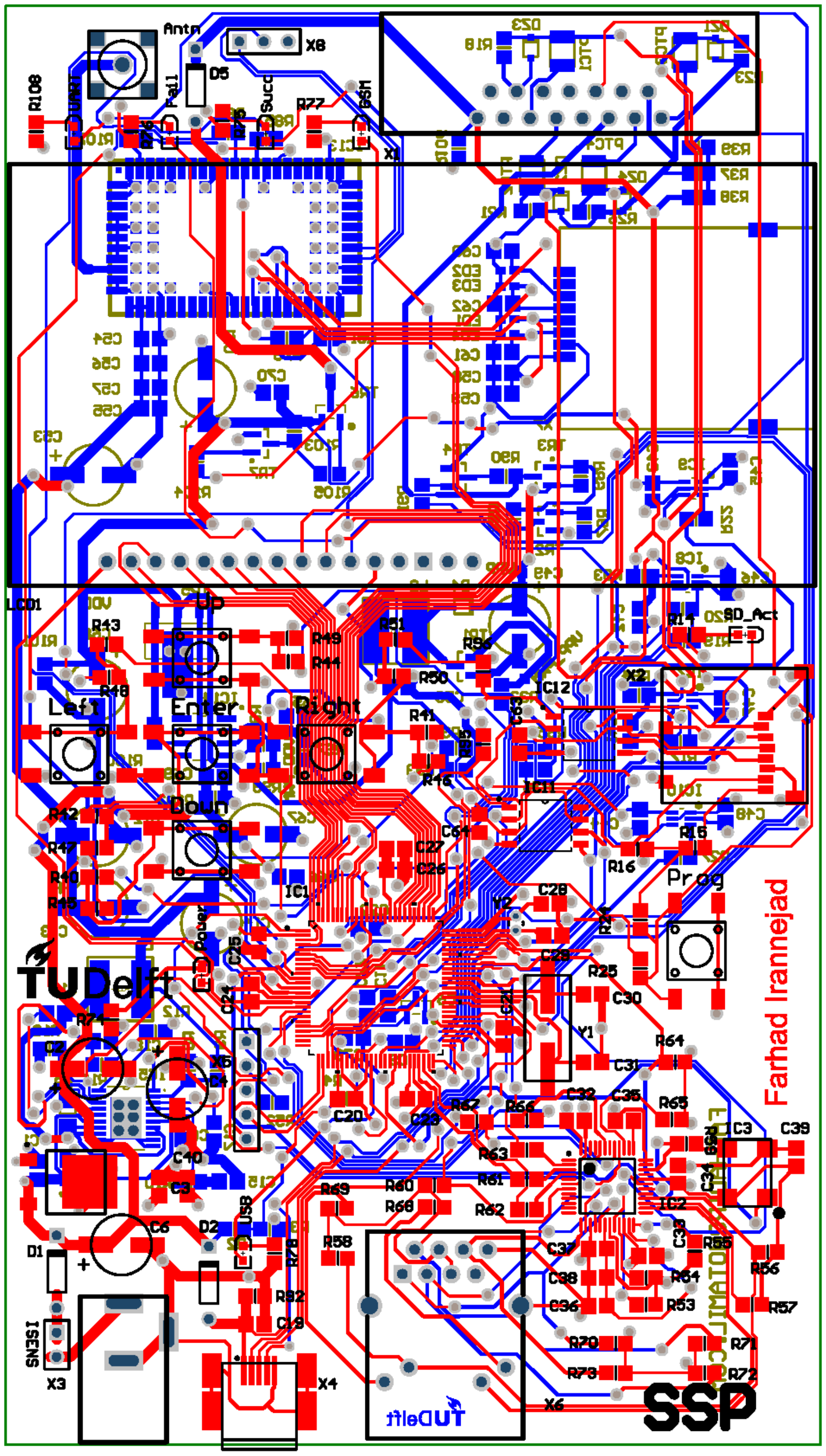
7.4.1 Top



7.4.2 Bottom



7.4.3 All Layers



7.5 Hardware Errata

7.5.1 Ethernet Controller

In IC2, pin number 37 has to be connected to the net label ”PFOUT” or pin number 18.

In the prototype it is connected with an external jumper wire.

7.5.2 ADC voltage reference

The voltage reference IC (REF3020) for ADC that has reference voltage of 2.048, is lower than 2.5; the minimum allowed in the PIC32's datasheet. In the prototype this IC is omitted and the supply for ADC is connected directly to the 3.3 volts line.

7.5.3 Character LCD

The LCD chosen for this design needs 5 volts supply. But the microcontroller works with 3.3 volts. In the prototype the supply voltage for LCD is directly connected to the input voltage of the board. So, for powering the board using a power socket, the only input voltage allowed for the board is 5 volts. The contrast pin of the LCD is directly connected to the ground to meet the required voltage difference between Vdd and Vo mentioned in the LCD's datasheet (4.1 < <4.7 volts). R51 is omitted and R50 is short circuited.