OULUN AMMATTIKORKEAKOULU

Tomi Lahdenperä

Programming automation for power supply unit controller

Programming automation for power supply unit controller

Tomi Lahdenperä
Final projects
Autumn 2022
Bachelor of Engineering, Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology, Option of Device and Product Design

---

Author: Tomi Lahdenperä
Title of thesis: Programming automation for power supply unit controller
Supervisor: Teemu Leppänen
Term and year when the thesis was submitted: Autumn of 2022
Number of pages: 25

Objective of this thesis was to develop an automated Power Supply Unit (PSU) controller programming software for programming Infineon XDPP1100 PSU controllers in production. The project was commissioned by Nokia Solutions and Networks Oy.

In this thesis, the software and hardware tools used in the project as well as the Test automation in general are discussed. The thesis also introduces Inter-Integrated Circuit (I2C) and Power Management Bus (PMBus) protocols.

The result of this thesis project is a fully automated software providing a simple and functional use for production. Also, the project improved author's skills in many different areas.

Keywords: Infineon, XDPP1100, OpenTap

CONTENTS

VOCABULARY

5G	(Fifth Generation network standard)
ACK	Acknowledge bit
API	(Application Programming Interface) Interface that allows two applications to talk to each other.
C#	(C Sharp programming language) Used to develop computer programs, developed by Microsoft.
COO	(Chief operating officer) High-ranking executive position
CPU	(Central processing unit)
FAE	(Field applications engineering) Technical support given by a company to customers who use its products.
HW	(Hardware) Computer hardware, physical parts.
I2C	(Inter-Integrated Circuit)
IDE	(Integrated Development Environment)
LSB	(Least significant byte)
MN	(Mobile networks) Communication network
NACK	Not Acknowledge bit
OTP	(One Time Programmable Memory) Type of memory that permits data to be written only once.
PC	(Personal computer) A microcomputer designed to be used by one person at a time.
PMBus	(Power Management Bus)
PSU	(Power supply unit)

	Supplies power to the components.
RF	(Radio frequency)
SCL	(Serial clock line)
SCRUM	(Software development framework) Way of working that consists of daily meetings.
SDA	(Serial data line)
Supply Chain	The network of all the individuals, organizations, resources, activities, and technology involved in the creation and sale of a product.
SW	(Software) Computer program.

1 INTRODUCTION

This thesis is part of studies at Oulu University of Applied Sciences. The project was provided by Nokia, and it was carried out during 2022. Nokia is a global telecommunication, information technology, and consumer electronics company founded in Finland in 1865. Nokia is a leading 5G provider which has around 90,000 employees globally.

Nokia Mobile Networks (MN) Chief Operating Officer (COO) Supply Chain is agile, resilient, and high-performing supply chain enabling MN business success in a volatile market. It is responsible for building supply chain strategies, creating delivery capability, and running factories including logistics and sales. At Test Software (SW) Platforms, we are responsible for creating production capable testing solutions, drivers and features for users and customers. Our team uses scrum as a way of agile software development.

Use of components have grown exponentially during the last decades and is approximately doubling every two years. We have felt the impact by seeing the prices skyrocket and seeing shortages in almost every industry. Nowadays many companies hardly develop their own components. This often results in use of 3rd party components, as they are usually cheaper and versatile for different types of applications.

Many chips that are used today are digital, meaning that they can be programmed to accomplish different types of tasks. Microcontrollers and Central Processing Units (CPU) are well known examples of digital chips. Unfortunately, the manufacturers hardly provide any easy-to-use solutions for every type of projects, and they need to build their own software solutions.

The goal of this thesis was to develop automation software to program XDPP1100 PSU controller chips provided by Infineon. These chips can be used in a Radio frequency (RF) product Hardware (HW) solution. The purpose of this automation is to eliminate the need to use the software provided by Infineon so that the programming could be done effortlessly in the production environment.

2 BACKGROUND

The Infineon XDPP1100 PSU controllers are currently programmed using the Infineon graphical user interface (GUI). It is challenging to use and requires training. When using the GUI, the user needs to manually browse and choose the two programmable files for each controller, which makes the programming process a complicated and time-consuming task. First, there is a patch file, which is the Firmware for the controller. Second, there is the configuration file, of which purpose is to change settings of the controller.

Firmware is a software that provides basic machine instructions allowing the hardware to function and communicate with other software running on a device. Firmware provides low-level control for a device's hardware (HW). (11)

The production file used in the process must be created with the Infineon GUI, following these steps:

- Start the GUI
- Load PMBus Spreadsheet which fits to the firmware patch
- Open Config file (.pcd)
- Go to the Multi-Device-Programmer
- Click on "Save Production File"

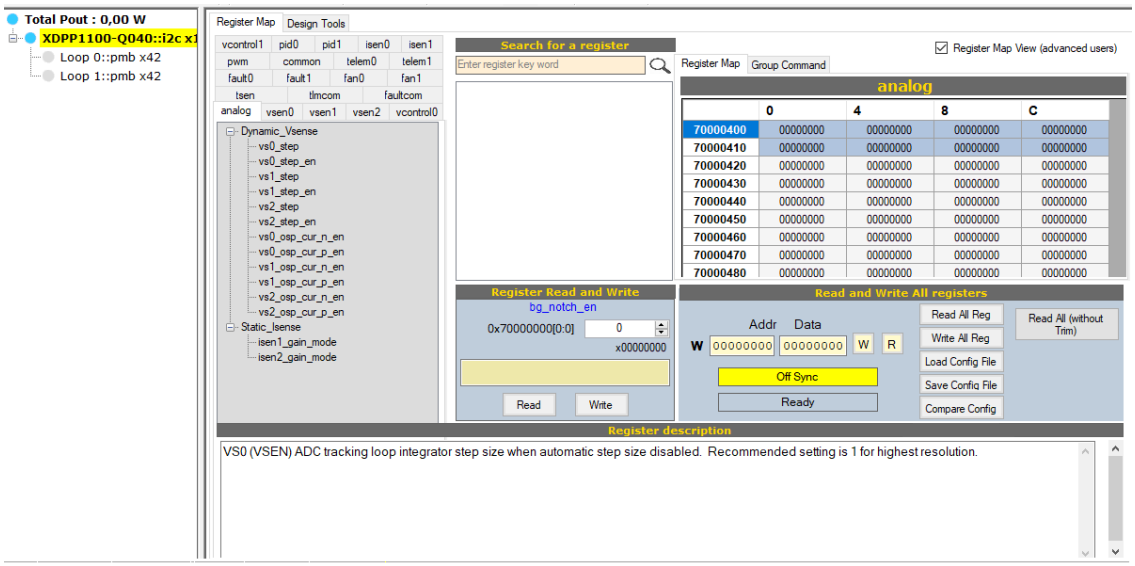

FIGURE 1. Screenshot of the Infineon GUI, showing one recognized XDP1100 PSU controller

The plan is to automate the GUI's features into a simple Test Automation SW. This means that user would not have to choose the needed files each time; the test step would then take care of the file processing and programming. Specifically, the objectives of this project are listed in Table 1.

Project objectives
• Find, study, and install all the software needed
• Study all the documentation provided by Infineon
• Contact the Infineon Field applications engineering (FAE)
• Create a parser method for the files
• Program the parsed data to the Infineon Controllers
• Verify using the Infineon GUI
• Programming process must be as fast as possible
• Create a finished Test Step for OpenTap

TABLE 1. Project objectives

2.1 Test automation

Test automation is a popular approach to testing. It is the process of using automation tools, scripts, and software to maintain test data, execute tests, and analyse test results to improve software quality. Automated testing can save a lot of time and ~~a lot of~~ money. (9)

A test sequencer is used in test automation to perform different types of tests for example on HW. Different test operations include for example verification, calibration, and programming. Test sequencers are usually used by operators who can monitor and control the performed tests. Results from tests can then be gathered for comparing and evaluation to see if the process was successful.

This project will allow the programming process to be fully automatic, which helps to reduce production costs and requires zero-to-none experience from a new user.

2.2 Tools for test automation

In Figure 2, the test system architecture is presented. The tester setup used in this project contains a Personal Computer (PC) and a PSU. The PC is needed for running OpenTAP, and PSU is used for powering the Infineon XDPP1100 PSU controller chip.

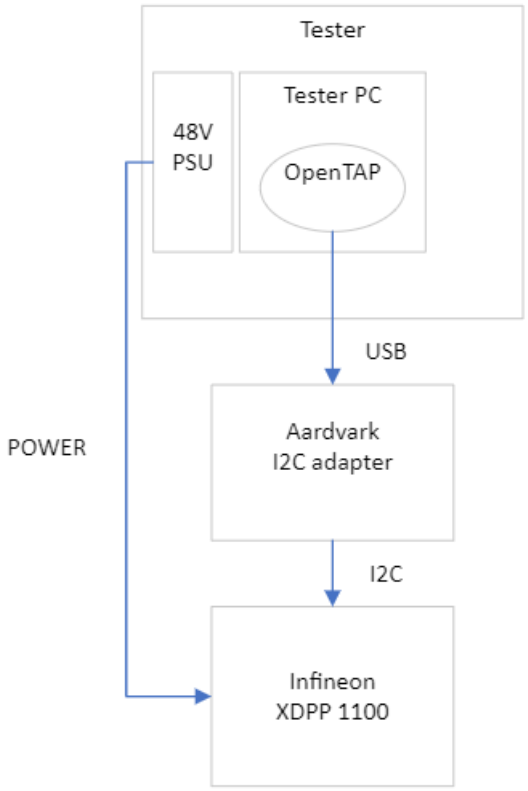

FIGURE 2. System Diagram

Test automation platform used in this project is the Keysight OpenTAP. It is an Open-Source software automation platform for effortless automation development. It is built on the .NET platform. It provides wide range of test sequencing functionality and infrastructure for different types of automation needs. OpenTap provides capability for implementing test steps, instrument plugins, device under test (DUT) plugins and result listeners. (1)

FIGURE 3. Aardvark I2C/SPI Host Adapter

OpenTAP communicates with Aardvark using Universal Serial Bus (USB). USB is a plug and play interface that allows a computer to communicate with peripheral and other devices. USB connectors come in different shapes and sizes. (10) Aardvark uses USB-B to USB-A type connector.

In turn, Aardvark is used as the I2C adapter for communication between the Test Sequencer and the XDPP1100 PSU controller. Aardvark controls the Infineon XDPP1100 controller using the dongle, which communicates with I2C.

The Aardvark I2C/SPI Host Adapter is a fast and powerful I2C bus and serial peripheral interface (SPI) bus host adapter through USB. It allows a developer to transfer serial messages using the I2C and SPI protocols. It allows I2C bus speed up to 800 kHz.

Infineon XDPP1100 GUI can be used to program XDPP1100 PSU controllers with Patch and Configuration files. In this SW project; however, the goal is to make the use of the GUI unnecessary. This SW will be executed using C# as a programming language building the software on top of OpenTAP Test Automation Software and using OpenTap Application programming interface (API). The basic principle of the software will be simply parsing the required Files (2 for each PSU controller) and then writing the data using I2C and PMbus.

Visual studio will be the main tool for programming the Test Steps. It is an integrated development environment (IDE) developed by Microsoft. It is used to develop computer programs. It supports 36 different programming languages, and it provides several different versions for all customers. (2)

2.3 Communicating with devices

The Inter-integrated Circuit is a simple-to-use serial protocol for two-wire interfaces between master and slave electronic devices. I2C uses Serial Clock (SCL) and Serial Data (SDA) for the communication. Each slave device has its own unique address. (6)

On the I2C bus, the data is transferred as messages. These messages can include start and stop, address, data and acknowledge bits, as shown in Table 2 and Figure 4. I2C can also be used to read from the device, for example register values.

I2C Message content
• Communication begins with master device generating Start condition bit, switching SDA from high voltage level to low voltage level.
• Address Frame includes unique address for the identifying the slave device.
• Read / Write Bit is a single bit specifying if the master is sending or receiving data.
• After each byte of data has been sent, acknowledge bit is received to confirm if the data was successfully received by slave. ACK stands for successful transfer; NACK stands for not successful transfer.
• Data bits, some type of messages send from master to slave.
• Communication stops with Stop condition bit, SDA switching from low voltage to high level voltage.

Table 2. I2C Message structure

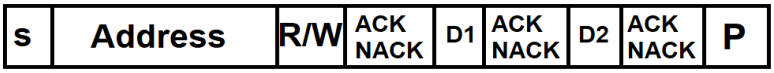

FIGURE 4. I2C Protocol (6)

The Power Management Bus is a simple open standard digital power-management protocol. It allows system developers to configure the power supply to perform a specific task and to monitor the status of the power system. It is built from the I2C protocol. PMBus supports a maximum bus speed of 400 kHz, and because of the built-in timeouts, all PMBus devices support a minimum bus speed of 10 kHz. (7)

2.4 Programming the PSU

Infineon XDPP1100 GUI can be used to program XDPP1100 PSU controllers with Patch and Configuration files, as shown in Figure 5. It uses USB212C dongle provided by Infineon to communicate with the chip. It can also be used to verify if the programming was successful.

The Patch file is a Binary file containing bytes. Binary files can be read as a text, but they are usually used to store raw data used by computers. The Patch file contains parameters that are used by the controllers. These parameters consist of different information and features used by the controller. Such as version number, register address and resolution of output current~~, etc~~.

The configuration file contains PMBus commands that are used to change the settings of the PSU controller, as shown in Table 2. These commands can include, for example, Read Frequency, Fault limits, Fan configurations and Power modes.

Loop 0 PMBus Configuration:
% 0B 6 013 2 00 00
% 0B 6 01C 2 01 70
% 0B 6 01D 2 02 15

Table 3. Sample from the Configuration File

Verification of the controllers were performed using the GUI's feature to compare the written configs on the controllers. This was only used in the development phase to verify the programmed controller chips.

FIGURE 5. Screenshot of the Infineon GUI, showing the buttons to perform Configuration Comparison

3 IMPLEMENTATION

After getting familiar with the tools and software needed for the project, the implementation phase began by building the test automation software. The operation of the software is based on building a parser that process the data and commands from the programmable files so that the needed commands can be sent in subsequent order.

By sending the data and commands, the configuration of the PSU controllers can be altered, making them operate as desired. In this chapter, the steps for parsing the files and programming the controllers will be discussed.

1. Before programming, the PMBus command to enable offset resistors is sent including Power Good to normal command.
2. Next, the OTP partition I2C commands are sent, if needed. Along with PMBus commands to store the configuration.
3. After that, the I2C command to send parsed patch file data is send, 4 bytes at a time.
4. After every 1kb of data has been sent, PMBus commands to store the data from Scratch Pad to OTP are sent.
5. Before moving on to production file, the System Reset PMBus command is sent.
6. Moving on to Production file programming, for register configuration, the register configuration data along with the register address is sent via I2C.
7. Finally, for the PMBus commands, the commands are sent to the controller via PMBus.

3.1 Initialization

Before moving on to programming phase, the offset resistors must be enabled and the power good polarity needs to be changed from inverted to normal because in our application we use several controllers, and initially they all share the same I2C address. Also, the controllers use One-Time-Programmable (OTP) type memory, which means that the Patch file must be programmed correctly on the first run because no changes can be made after.

For programming the configuration, it is recommended by the Infineon FAE to use the GUI for creating the production file containing the register configuration for more simple execution.

Pmbus address	XDPP1100 PMbus address
PMbus command:	FW_CONFIG_PMBUS
Data:	I2C_addr_offset: enable Power_good: normal

Also, the OTP memory sections on each controller must be checked if the default Trim Data configuration is in correct format for our application. If the memory sections are invalid, the correct Trim Data needs to be stored to OTP.

I2C address	XDPP1100 I2C address
I2c register	OTP Data partition register
Data:	OTP Partition size

Pmbus address	XDPP1100 PMbus address
PMbus command:	MFR_FIRMWARE_COM- MAND
Data:	STORE_TRIM

These commands will invalidate the old Trim Data and write the new Trim Data to OTP.

3.2 Patch-file

Patch file parsing step began with reading through the document provided by Infineon.
What needed to be done was to read the patch file data in 1kb chunks and write it to the I2C register 4 bytes at the time.

	0	4	8	C
20061800	00000010	10001000	5BFD1D7F	10024000
20061810	100245FD	5C843EFA	5971BBBB	4604B510
20061820	F0004620	4903FA87	460A4620	F84EF000
20061830	0000BD10	00030D40	4806B510	28006800
20061840	6901D002	47882004	49032001	60080440
20061850	0000BD10	20063A04	E000E280	4806B510
20061860	28006840	6901D002	47882005	49032001
20061870	60080480	0000BD10	20063A04	E000E280
20061880	2800B538	4C0ED10E	28006820	68C1D00B
20061890	47882004	90006960	68856820	34082004
200618A0	47A8CC0E	4807BD38	20006020	480460A0
200618B0	30082301	60417003	49036082	E7E960C1
200618C0	20063A04	100054CC	10024039	2800B538

FIGURE 6. XDPP1100 Controller scratch pad data

As shown in Figure 6, starting from the register address 2006180, each register contains 4 columns, each column containing 4 bytes of patch file data. The first 1kb of patch file data would need to be first written to the scratch pad, and then moved to the OTP memory using PMBUS commands.

After the first 1kb of data has been moved to the OTP memory, the program must start writing the second 1kb chunk to the same register starting address.

After parsing the patch file data, the data must be written to the controller in format shown below:

I2C address	XDPP1100 I2C address
I2c register	Patch register address
Data:	4 bytes of parsed data from the patch file

After the full 1kb of data has been written to the controller, the following PMBUS commands are executed:

Pmbus address	XDPP1100 PMbus address
PMbus command:	MFR_FIRMWARE_COM-MAND_DATA
Data:	Partition number = 1, header is in-cluded, amount of data to be stored

Pmbus address	XDPP1100 PMbus address
PMbus command:	MFR_FIRMWARE_COMMAND
Data:	STORE_FW_PATCH

On the last 1kb of data, the following PMBUS commands are executed:

Pmbus address	XDPP1100 PMbus address
PMbus command:	MFR_FIRMWARE_COM-MAND_DATA
Data:	Partition number = 1, header not included amount of data to be stored

Pmbus address	XDPP1100 PMbus address
PMbus command:	MFR_FIRMWARE_COMMAND
Data:	STORE_FW_PATCH

Pmbus address	XDPP1100 PMbus address
PMbus command:	MFR_FIRMWARE_COMMAND
Data:	SYSTEM_RESET

After system reset, the patch file data is successfully stored to OTP memory. It cannot be altered after storing;instead it would have to be invalidated before updating. However, it was decided to abandon this feature, as the need for an update at this stage would be very unlikely. If needed in the future, this can be done using the GUI.

3.3 Production file

The production file contains several different types of commands, and must be parsed, to write the config to the controller. All lines in the file starting with % character, must be written to the XDPP1100.

3.3.1 Register configuration

For example: % 16 6 37 8 70000400 0000A409

Where:

- 16 – Command Length, field that specifies the number of characters.
- 6 – Type, 1 character field that specifies the data type.
- 37 – Checksum, 2 hex digits, represents the sum of all the characters on the line, excluding the checksum itself.
- 8 70000400 – Register address, first character being the character length of the address.
- 0000A409 – Data, contains the executable code, memory-loadable data, or descriptive information to be transferred.

To write the Register configuration, 8 bytes must be written to the I2C address of the XDPP1100 controller for every line, first the address bytes, then the data bytes, Least Significant Byte (LSB) first. For the example line above, you would write to the I2C address of XDPP1100:

I2C address	XDPP1100 I2C address
I2c register	Configuration register address
Data:	Configuration data

3.3.2 PMBus Commands

Examples:

% 0C 6 36 4 27 E810

% 10 6 31 8 C4 0022001D

% 19 6 23 10 1B 0000000000000000

For the PMBus command lines, there is a slight difference compared to the register configuration command presented above.

The first example, where:

- 0C – Command Length, field that specifies the number of characters.
- 6 – Type, 1 character field that specifies the data type.
- 36 – Checksum, 2 hex digits, represents the sum of all the characters on the line, excluding the checksum itself.
- 4 – Length of the command data. PMBus command 0x27 expects 4 characters of data, so the value is 4.
- 0x27 – PMBus command code, the PMBus command code is always 2 characters.
- E810 – PMBus command data, contains data to be written by the PMBus command.

For the PMBus commands, depending on the length of the data, there are different rules that need to be taken in attention.

For the first example above, it would be a "Write Word" transaction, so you would write to the PMBus address of XDPP1100:

Pmbus address	XDPP1100 PMbus address
PMbus command:	VOUT_TRANSITION_RATE
Data:	VOUT_TRANSITION_RATE_DATA

The second example is for "Block Write", so you would write one byte containing the number of data bytes followed by the data bytes:

Pmbus address	XDPP1100 PMbus address
PMbus command:	DATA COUNT, FW_CONFIG_PWM
Data:	FW_CONFIG_PWM_DATA

There is one more exception in case of the SMBALERT_MASK command (0x1B). If there are any bits set, you must separate the line to several commands. For every byte, you must write a separate command consisting of the Command Code (0x1B), the Command Code of the Status Byte you want to mask (0x7A to 0x81), and the mask byte from the production file.

To write the SMBALERT_MASK, you must use Write Word, with the command code of the status you want to write as low byte and the value as high byte.

In the final example, all masks are set to 0x00, so you would send the following commands via PMBus.

Pmbus address	XDPP1100 PMbus address
PMbus command:	SMBALERT_MASK
Data:	STATUS BYTE, MASK BYTE

In total, 8 commands must be written to change the masks for all the 8 status bytes.

3.4 Controller programming test

After programming the controllers, the success of the operation can be checked by debugging using the Infineon GUI.

After storing the first Patch to the controller, it was noted, that the patch was not successfully stored to the OTP memory. The PMbus commands had a small typo, which caused the entire programming process to fail several times. After careful analysis of the logs, the problem was noticed and fixed.

Another issue was detected when programming the Production file to the controllers; the Infineon GUI had a bug when creating the file causing the programming to fail. The Infineon FAE was however contacted, and the problem was solved quickly.

4 EVALUATION

In the end, all the project objectives were successfully achieved.

The project started by installing all the required SW for the project, including OpenTap, Infineon GUI, and Visual Studio. I was also part of another project, where OpenTap and Visual Studio were used so all that had to be studied was Infineon GUI.

After becoming familiar with the SW, the documentation provided by Infineon was studied thoroughly.
Also, the Infineon FAE was contacted through email regarding the programming process.

Software development for creating parser method was successfully completed, along with programming the parsed data to the Controllers. Infineon GUI was used to verify that the data was successfully programmed.

Created Software methods were finally compiled into fully working & automated Test Steps for OpenTap.

5 CONCLUSION

The objective of this thesis was to develop a fully automated, suitable for production software, to program the Infineon XDPP1100 PSU Controllers. The program using aardvark dismisses the need to use any 3rd party software or hardware. It allows the operators to run the process quickly and effortlessly, saving production time, with no need for special training. The test steps made are used in production at Nokia Oulu Factory. Project was planned to be finished during spring 2022.

The thesis project provided a perfect possibility to learn software development in a professional way. Many previously unfamiliar tools were used in the project.

For volume production, this method can also be further developed into more reliable and faster method.

6 REFERENCES

1. Keysight OpenTap Test Automation Platform. Date: 29.6.2022.
   https://opentap.io/
   https://doc.opentap.io/

2. Microsoft Visual Studio 2019. Date: 29.6.2022.
   https://visualstudio.microsoft.com/

3. Infineon XDPP1100 programming instructions. Date: 29.6.2022.
   https://www.infineon.com/dgdl/Infineon-DC_DC_converter_XDP_digital_power_controller_XDPP1100_programming_instructions-UserManual-v03_00-EN.pdf?fileId=5546d46279cccfdb0179efee1a370425

4. Aardvark I2C/SPI Host Adapter User Manual. Date: 29.6.2022
   https://www.totalphase.com/support/articles/200468316-Aardvark-I2C-SPI-Host-Adapter-User-Manual

5. Test execution engine. Date: 29.6.2022.
   https://en.wikipedia.org/wiki/Test_execution_engine

6. I2C information. Date: 11.11.2022.
   https://i2c.info/
7. Basics of the I2C communication protocol. Date: 11.11.2022.
   https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/

8. What is PMBus. Date: 27.11.2022.
   https://www.totalphase.com/blog/2021/02/what-is-pmbus/

9. What is test automation. Date: 12.12.2022.
   https://www.perfecto.io/blog/what-is-test-automation

10. USB explained. Date: 12.12.2022.
    https://www.computerhope.com/jargon/u/usb.htm

11. Firmware definition. Date: 15.12.2022.
    https://www.techtarget.com/whatis/definition/firmware