Advanced 3D Cell Culturing and Monitoring System

Dissertation

for obtaining the academic degree

Doctor rerum naturalium
(Dr. rer. nat.)

submitted to the
Department of Mathematics and Natural Sciences
Ilmenau University of Technology

by

Ing. Martin Bača

1. Examiner: Prof. Dr. rer. nat. habil. A. Schober
2. Examiner: PD Dr. phil. nat. I. Oehme
3. Examiner: Prof. Dr. R. O'Connor

Submission date: 21.03.2023
Defense date: 05.09.2023

DOI: 10.22032/dbt.59056
URN: urn:nbn:de:gbv:ilm1-2023000277

Zusammenfassung

Die vorgelegte Doktorarbeit stellt ein 3D-Zellkultursystem mit einem vollautomatisierten analytischen, biochemischen Assay und vollautomatisierter Kultivierung mit Mediumwechsel vor. Dieses integrierte Kultivierungs- und Analysesystem wurde in dieser Arbeit mit 3D-Hepatozytenkulturen in Polycarbonat-MatriGrid®-Gerüsten für das 3D-Wachstum von Zellkulturen prototypisiert. Das System perfundiert MatriGrid-Kulturen kontinuierlich mit Wirkstoff-ergänztem Medium und führt bei Bedarf eine Bewertung der Wirkstofftoxizität durch Beobachtung und Messung der Konzentration eines Indikators, des Biomarkers Albumin, durch. Das System kann die MatriGrid-Kultur mit unterschiedlichen Flussraten perfundieren, automatisierte Medienwechsel durchführen und mit dem mitgelieferten ELISA-Modul Proben des zu analysierenden Kulturmediums nach Bedarf untersuchen. Das System unterstützt die parallele Kultivierung von Zellen in mehreren Bioreaktoren. Das Fluidnetzwerk wurde aus Materialien konstruiert, die wenig Proteine und kleine Moleküle binden, absorbieren oder adsorbieren, um seine Anwendung für niedrige Biomarkerkonzentrationen und Langzeitexperimente zu erweitern. Die Doktorarbeit beschreibt das Systemdesign, den Aufbau, das Testen und die Verifikation unter Verwendung von 3D-gewachsenen HepaRG-Zellkulturen. Die zeitabhängige Wirkung von APAP auf die Albuminsekretion wurde über 96 h untersucht, wobei sowohl mit dem neu entwickelten System als auch konventionell in Mikrotiterplatten, gemessen wurde. Es zeigte sich, dass die Ergebnisse vergleichbar sind. Dieses Resultat belegt die Verwendung des Systems als eigenständiges Gerät, das in Echtzeit arbeitet und in der Lage ist, gleichzeitig Zellkultur- und Mediumanalyse in mehreren Bioreaktoren durchzuführen, mit erhöhter Zuverlässigkeit der 3D-Kultivierung, in einfacher Handhabung und Messung. Auf diese Weise soll das neu entwickelte 3D-Zellkultivierungs- und Analysesystem 3D-Zellkultivierungstechniken und -experimente für weitere Forschungsgruppen bekannt machen.

Abstract

This thesis presents a 3D cell culturing system with a fully automated analytic biochemistry assay and fully automated culturing with medium change. This integrated culturing and analytic system was prototyped in this work with 3D hepatocyte cultures in polycarbonate MatriGrid® scaffolds for 3D growth of cell cultures. The system continuously perfuses MatriGrid cultures with drug supplemented medium and performs, on demand, drug toxicity evaluation by observing and measuring the amount of an indicator, the biomarker albumin. The system can perfuse the MatriGrid cultures using different flow rates, performs automated medium changes and can make on-demand samples of the culture medium to be analyzed with the included ELISA module. The system supports parallel culturing of cells in multiple bioreactors. The fluidic network was constructed from low protein and small molecules binding, absorbing or adsorbing materials to extend its application for low biomarker concentration and long-term experiments. The thesis describes the system design, construction, testing, and verification using 3D-grown HepaRG cell cultures. The time-dependent effect of APAP on albumin secretion over 96 h, measured with newly developed system and conventional microtitre plates was measured and the results are comparable. These results confirm the use the system as a standalone device that works in real time and is capable of simultaneous cell culture and medium analysis in multiple bioreactors, with increased reliability of 3D-culturing, ease of handling and measurement. This way the newly developed 3D cell culturing and analysis system is aimed to promote 3D cell culturing techniques and experimentation to more research groups.

Table of Contents

Abbreviations and acronyms

1D	one-dimensional
2D	two-dimensional
3D	three-dimensional
3D-ACAD	automatic 3D-culturing and analysis device
ABTS	2,2'-azino-di [3-ethylbenzthiazoline] sulfonate
AC	alternating current
ADC	analog to digital converter
ADHP	10-Acetyl-3,7-dihydroxyphenoxazine
ALF	acute liver failure
ANSI	American National Standards Institute
APAP	N-acetyl-para-aminophenol
ARM	advanced RISC machines
BSA	bovine serum albumin
BW	bandwidth
CAN	controller area network
CDT	C/C++ development tooling
CFA	cyst fluid antigens
CK-MB	creatine kinase myocardial band
CMOS	complementary metal–oxide–semiconductor
CPU	central processor unit
CYP	cytochrome P450 family
DC	direct current
DI	de-ionized
DMSO	dimethyl sulfoxide
DPSS	diode-pumped solid state
EC_{50}	half maximal effective concentration
ECM	extracellular matrix
EEPROM	electrically erasable programmable read-only memory
ELISA	enzyme-linked immunosorbent assay
EMA	European Medicines Agency
EMI	electromagnetic interference

FBS	fetal bovine serum
FCS	fetal calf serum
FDA	Food and Drug Administration
FEP	poly-(tetrafluoroethylene-co-hexafluoropropylene)
FMIA	flow-through membrane immunoassay
FTA	flow-through assay
GBW	gain–bandwidth product
GDB	GNU debugger
GNU	extensive collection of free software
GSH	glutathione
GST-α	glutathione-S-transferase alpha
HepaRG	immortalized human hepatic cell line
HepG2	human hepatocyte carcinoma cell line
HPA	hydroxyphenylacetic acid
HPLC	high-performance liquid chromatography
HPPA	3-p-hydroxyphenylproprionic acid
HRP	horseradish peroxidase
I^2C	inter-integrated circuit
IDE	integrated development environment
iPSC-CM	induced pluripotent stem cell–derived cardiomyocytes
JTAG	Joint Test Action Group
KTP	potassium titanyl phosphate
LDH	lactate dehydrogenase
LED	light-emitting diode
LQFP	low profile quad flat package
LSB	least significant bit
MIT	Massachusetts Institute of Technology
MTP	microtiter plate
$Na:YVO_4$	neodymium-doped yttrium orthovanadate
NAPQI	N-acetyl-p-benzoquinone imine
NEMA	National Electrical Manufacturers Association
OEM	original equipment manufacturer
OPD	o-phenylenediamine

PBS	phosphate-buffered saline
PC	polycarbonate
PC	personal computer
PCB	printed circuit board
PDMS	polydimethylsiloxane
PEEK	polyether ether ketone
PEG	polyethylene glycol
PMT	photomultiplier tube
POF	plastic optical fiber
PS	polystyrene
PVA	polyvinyl alcohol
PVC	polyvinyl chloride
R&D	research and development
RISC	reduced instruction set computer
RMS	root mean square
RTC	real time clock
SD	secure digital
SDS	sodium dodecyl sulfate
SMD	surface-mount device
SPI	serial peripheral interface
TFT	thin-film transistor
TIA	transimpedance amplifier
TMB	3,3',5,5' tetramethylbenzidine
TRIS	tris(hydroxymethyl)aminomethane
UART	universal asynchronous receiver-transmitter
USB	universal serial bus
WCA	whole cyst antigens
WME	Williams′ medium E

1 Introduction

YOU, WHO are blessed with shade as well as light, you, who are gifted with two eyes, endowed with a knowledge of perspective, and charmed with the enjoyment of various colours, you, who can actually see an angle, and contemplate the complete circumference of a Circle in the happy region of the Three Dimensions.

(Edwin Abbott, *Flatland,* 1884)

1.1 Motivation

2D cell culture schemes grow cells on flat surfaces. Such surfaces are coated with material to promote the adherence of cells, growth, and the spread of the culture. Having been used successfully for decades in labs, 2D culturing remains the primary method used in most cell culturing applications. Certainly, the technology and techniques involved in 2D culturing are straight-forward and comparatively inexpensive, and the long-standing use of 2D cultures has generated a wealth of available literature. However, the strongest argument in favor of 2D systems is the ease with which cells can be observed and measured. 2D cell schemes are characteristically simpler to evaluate compared to the widely varying and non-standard 3D cell culture systems which are currently available.

This latter point, the ease of observation and measurement, is the primary issue. The advantages of 3D culturing are certainly numerous and can be listed in extraordinary detail, but this does not change the fact that the observation and measurement of cells in the majority of new 3D culturing systems is more complex, requiring more intricate 3D-specific culturing laboratories. This, regrettably, inhibits the use of 3D culturing systems for many laboratories which would require a complete retooling of equipment and training for staff to be specific to the new 3D system. The ideal situation would therefore be a 3D culturing system, which is reproducible and simple to use for interested laboratories, and which *increase* the ease of observation and measurement, using the

automation and system integration techniques. The importance of development of new scientific techniques was also recognized by Nobel Prize laureate Sydney Brenner: “Progress in science depends on new techniques, new discoveries and new ideas, probably in that order.”[1]

1.2 Thesis Aims and Objectives

At first glance, an extensive study of commercially or otherwise available automated systems for cell culture management and analysis revealed a wide range of possibilities. What until now is not available, however, is a device platform that combines these two claims. In the case of 3D cell culture, such a link is necessary, since the inherent properties of 3D cell culture make it necessary in the most cases for analysis to interrupt the experiment. From this point of view, a combined solution of automatic cell culture management and directly linked analytics is of great scientific interest. This is exactly what in this thesis should be done.

The aim of this thesis is therefore to develop a new cell culturing system suitable for 3D cell cultures which would be easy to work with and encourage more research laboratories to transition towards 3D cell cultures. The new culturing tool must be robust, easy to handle and provide high reliability of culturing. Additionally, reliable way of monitoring the cell culture state by measuring the concentration of selected biomarkers should be provided. This analysis should be performed in an automated way and on-demand to support the concept of easy-to-use tools. The 3D culturing part and the analytical part should be compatible and create an integral 3D culturing and analytical system. Further objective is to design whole system in a compact and portable form, which could be used anywhere needed. Such 3D culturing system could be advantageously in routine preparation of 3D cell cultures or to perform toxicity assessment of various drugs towards the 3D cultured cells. It is known that 3D cultures resemble more closely *in-vivo* environment and thus toxicity tests performed on 3D cultures provide more relevant results [2, 3]. It is obvious that both scientific aspects from the life sciences field and engineering considerations play a role in answering such a question and play a role in the construction of such a device platform.

The functionality of the developed system should be verified on real 3D cell culture and the analytical measurement results should be verified by comparing with the results measured by conventional analytical methods.

The prototype of culturing device should preferably support the micro-bioreactor and MatriGrid® polycarbonate scaffolds - the tools previously developed at Ilmenau University of Technology [4].

The automated analytic part of the system should be preferably based on the enzyme-linked immunosorbent assay (ELISA). ELISA is a method which uses optical readout and antibodies to detect a molecule of interest in, for example, the cell medium. More specifically, ELISA is based on the detection of this molecule of interest in a liquid environment by way of a liquid reagent, a reagent which creates a series of biochemical reactions and yields a final indicator, an indicator which can be easily observed and measured to give the amount of this molecule of interest in the liquid environment.

The problem posed for this thesis should be therefore approached from two sides, with the life science question taking the lead and the engineering side working as an assistant for the problems to be solved.

1.3 Thesis layout

Chapter 2, which follows this introduction reviews available 3D culturing systems with respect to the possibility of biomarker level measurements and degree of automation. Second part of this chapter reviews existing automated ELISA systems. **Chapter 3** lies out the concept of the analytical module, discuss the selection of its building components and describes the assay protocol in detail. This is followed by the detailed description of the readout system and its integration to the analytical module. The last part of chapter 3 is dedicated to the control unit design and functional explanation of underlying electronics. Short description of associated software and corresponding software development tools is also included. The 3D culturing unit prototype design is described in the **Chapter 4**. The topology and functionality of its fluidic network is explained in detail. This chapter also include description of the MatriGrid® scaffolds and the micro-bioreactor tools. The initial testing and functional evaluation of the 3D culturing and

analytical modules are described in the **Chapter 5**. This includes the testing of the culturing unit fluidics, the analytical module fluidics and readout performance evaluation. Finally, the complete albumin ELISA was conducted by the analytical module. Few problems were identified, which required design adjustment. **Chapter 6** describes the identification of the functionality flaws pointed out in the previous chapter and their solving. This include the cleaning procedure improvement and the fluidic manifold redesign. The performance improvement is demonstrated by the standard curve measurement in the low concentration range, as well as by comparing the measurements results with the conventional ELISA method results. The usability of the complete 3D culturing and analytical system is demonstrated in **Chapter 7**. The toxicity of APAP on the 3D grown and perfused HepaRG cell culture was evaluated over 96 hours. Again, the results provided by the prototype system were compared to conventional ELISA protocol. **Chapter 8** discuss the drawbacks of the newly developed system and proposes possibilities for improvement during the further development. Additionally, finished and tested extension of the 3D culturing system, which allows parallel operation of 8 units is described. In the **Chapter 9**, potential applications of the culturing and analytic systems are discussed. The last part of the thesis is the conclusion provided in **Chapter 10**.

2 State of the art

2.1 Cell culturing systems

In general, although simpler for observation and measurement, 2D cell cultures are not demonstrative of the real situation of cells in real 3D *in-vivo* environment. Mounting cells on a flat surface is certainly a fast method to visualize growth and differentiation, however the resulting biological system is perhaps not the most optimal to comprehend function in the human body. For simple experiments to answer simple questions, the complexity of 3D cultures is likely unwarranted, but it is very clear that cells in 3D behave quite differently compared to cells which are surrounded by other cells in 3D.

This latter point is important in the context of predictivity. 2D cell systems are currently used in numerous drug and therapy screening applications but if the system is not demonstrative of the real situation, can the system be trusted to always be predictive? By slightly increasing the complexity of the technology and system into 3D, in the long-term this could decrease the price and failure-rate in clinical trials for novel drug discovery (which succeeded in and pre-clinical study at the 2D level). Indeed, large pharma companies devote extraordinary monetary resources each year for trials of novel drug therapies, with the majority of such trials ending in failure. 3D culture systems could provide a greater degree of predictivity at the pre-clinical level.

At the experimental level, a typical result of cell growth and differentiation is that cells consume molecules form the cell media and correspondingly release waste. This media must necessarily be changed, in almost all cases manually by the user. This disrupts the cell culture, the hermeticity, and, due to the manual nature of the process makes large-scale testing of the culture (for example in screening or therapeutic applications) an impossibility. This means, *a culture* should be 3D to increase predictivity, but *the culturing* should be automated to increase through-put and applicability to large-scale pre-clinical testing.

Following the previous sentence in detail, there are numerous 3D cell culture and culturing products available on the market today. However, the automation of culturing in these products is still in its infancy. In Table 2.1, essentially all relevant 3D culturing

technologies, commercially or semi-commercially available, are summarized. In the context of "Fully Automated Culturing", this means that the cells of interest can be seeded and the device will provide all necessary interim steps without manual user intervention. In the context of "Fully Automated Measurement", this means that the molecule or molecules of interest for measurement can be sampled by the system without manual user intervention. There is not a single instance of a system meeting both these requirements. In extreme contrast, as will be presented in this text, the 3D-Automated Culturing and Analysis Device (3D-ACAD) does meet both requirements.

Table 2.1 Overview of 3D culturing systems and their automation capabilities.

Company	3D area of expertise	Fully Automated Culturing	Fully Automated Measurement
3D Biomatrix	(2010, spin-off, U. of Michigan) Hanging drop plates for the generation of 3D spheroids.	No	No
3D Biotek	(2007) Inserts of various materials designed to turn 2D culture plates into 3D culture environments. Also, 3D inserts for bioreactors.	No	No
Biontex Laboratories	(1998) 3D cell cultures on solid substrates, optimized for hydrogel substrates.	No	No
CellASIC	(2005, 2012 acquired by Merck) MiCA (Microfluidic Cell Array) for 3D culture, a perfused plate for hepatocytes.	No	No
Cellec Biotek	(spin-off, U. of Basel) Bioreactors ("U-cup"s) for 3D cell culture and tissue generation.	Yes	No
Cellendes	(2009, spin-off, U. of Tubingen) 3D hydrogel kits/components (PEG-link, CD-link, Maleimide-PVA set, Maleimide-Dextran set), adhesion peptides. BSA-based gels.	No	No
CELLnTEC	(2002) products to improve isolation and proliferation of undifferentiated cells, or encourage complete differentiation in 2D or 3D cultures, using 3D (and 2D) epithelium models.	No	No
Cosmo Bio	(1978) Mebiol Gel 3D, an atelocollagen-coated scaffold for 3D culturing.	No	No
Epithelix Sari	(2006, U. of Geneva) MucilAir-HF: 3D human airway epithelia reconstituted in vitro by a co-culture of epithelia with human fibroblasts.	No	No
Geistlich Pharma AG	(1851) Orthoss, Chondro-Gide, technologies for regeneration of bone and cartilage.	No	No
Hamilton	BioLevitator, a bench-top incubator and bioreactor hybrid utilizing magnetic Global Eukaryotic Microcarrier (GEM) technology.	Yes	No
InSphero	(2009, spin-off, U. of Zurich) GravityPLUS plates for spheroids (scaffold-free 3D microtissues) organotypic, for biomimetic drug testing with embryonic stem cells.	No	No
Invitrogen	Reagent, cell, and kit supplier. AlgiMatrix, Geltrex, primary cells, stem cells.	No	No
Irisbiosciences	3D cell culturing products, biomaterials, Xeno-free polysaccharide-based hydrogels (mimsys G, heteropolysaccharide based on Gellan Gum); mimsys U (sulfated heteropolysaccharide from Ulva Lactuca).	No	No
Kirkstall Ltd	(2006, U. of Sheffield) Quasi-Vivo, system for co-	Yes	No

	culture of several cell types under controlled media flow		
Kiyatec	(2005, U. of Clemson) 3D co-culture. 3DKUBE configurations: independent chambers, segregated co-culture, cell migration.	Yes	No
MatTek Corporation	(1985, MIT) Produces in vitro human cell-derived tissue equivalents for use in product development/efficacy. EpiDerm-FT, a full thickness skin model, EpiVaginal, an ectocervico-vaginal model, and EpiOral, an oral (buccal) model.	No	No
Medical Device Company Ltd	Medical hydrogel compression devices.	No	No
Microtissues Inc	3D cell culture devices. Autoclavable, reusable micromolds allowing casting 3D Petri dishes from agarose.	No	No
Nanofiber Solutions	(2009, Ohio State U.) culture plates and scaffolds for bioreactors, scaffolds for in vivo tissue engineering made from aligned (NanoAligned) or randomly oriented (NanoECM) polycaprolactone electrospun nanofibers.	No	No
QuinXell Technologies Ltd.	(2011) TisXell biaxial spherical bioreactors for regeneration medicine and tissue engineering. Supports various scaffolds.	No	No
regenHU	3D bioprinters and biomaterials. Bioreactors and microbioreactors. Bioink: universal matrix for 3D tissue printings; Biofactory: bioprinters for tissue engineering; Biomanufacturing: 3D optical biopsy unit and tissue modeling software.	Yes	No
Reinnervate	Alvetex products. Polystyrene scaffold inserts for microplates.	No	No
SCIVAX corporation	NanoCulture Plates (NCP), conventional clear bottomed plate with an engineered micropatterned base that encourages 3D growth and formation of spheroids.	No	No
Stemmatters	Hydrogel systems based on polysaccharides.	No	No
TAP Biosystems	RAFT, a system to automate compression of hydrogels in microtiter plate formats.	No	No
Tecan Group Ltd	Automation of fluid and plasticware handling. Automation of Alvetexand and RAFT collagen scaffold-based 3D culture, automation of Hydrogel-based 3D culture.	Yes	No
TEDD Competence Centre	(U. of Zurich) 3D cell and tissue models, assays, imaging technologies, automation, molecular reporter systems, biomimetic scaffold substances, bioprinting.	No	No
Vitrocell Systems	(2007) Vitrocell, equipment designed to accept 3D cell inserts for culturing	No	No
ZenBio	(1995) ZenComplete, ZenSkin, donor-specific tissue acquisition to full thickness skin testing or 3D skin equivalents.	No	No
Zyoxel	(2009, spin-off, U. of Oxford) LiverChip, a multi-well plate platform enabling maintenance of 3D liver tissue cultures under constant perfusion. TissueFlex, 3D perfused cell culture under microchemostat conditions.	Yes	No

In contrast to many 3D culturing systems listed in Table 2.1, the 3D-ACAD design strictly avoids the use of hydrogels or construction materials based on PDMS. Hydrogels based on natural or synthetic polymers are commonly used in commercial 3D culturing systems as the support matrix for cell culture. (Biontex Labotatories, Cellendes, Irisbiosciences, Stemmaters, etc.) However, the use of hydrogel in general slows down the diffusion (or even traps) of the biomarkers and other molecules from cells to the medium [5]. Moreover, PDMS-based technology strongly interferes with the detection of soluble protein metabolites or markers due to absorption and adsorption [6]. To avoid these problems, the 3D-ACAD system uses the scaffold mediated 3D culture.

2.2 Automated ELISA systems

The enzyme-linked immunosorbent assay (ELISA) is an analytical measurement method which has existed in various forms for more than 40 years. During that time, iterations on the method have developed ELISA into the immunoassay with the highest sensitivity and highest specificity, leaving ELISA the clear and obvious choice to measure molecules of interest, in particular in complex liquid environments with numerous other molecules. As such, ELISA was the obvious choice for an automated system analyzing cell medium from 3D cultures. The work presented in this text demonstrates the full automation of the ELISA method, taking cures from classic automated flow systems developed previously by Lund University in Sweden [7].

In the previous work of Lund, a fully automated immunoassay was developed by joining flow-injection analysis with ELISA resulting in a fast flow-ELISA system. The system utilized competitive binding between antibodies, antigens, and fixed amounts of enzyme-labelled antigens, all in the liquid environment. Although novel for the time, the method unfortunately relies on antibodies to be immobilized to a solid support and then to be placed in a small column of the flow system. This does not allow for multiple assays and multiple concentrations as the solid support is necessarily manually replaced by the user leaving the system essentially dedicated to a single assay. Although protein interactions were used instead of real immunochemical interactions, the system did at least show that *flow-ELISA could be suitable for on-line monitoring* of biological macromolecules.

In the classic follow-up from Lund process integration using fermentation was studied with on-line process monitoring of the molecule of interest, alpha-Amylase [8]. A column of crosslinked starch was used to adsorb the a-amylase before and after specific fermentation steps so that the concentration of a-amylase could be continuously monitored using flow-injection and immunochemical measurement. This system, although not technically more advanced than their previous system, did show that *flow-ELISA could be suitable for living cells* (being from the fermentation process). However, due to the industrial nature of the fermenter, only simple measurements of the concentration were performed. Multiple parameters were not investigated and modified.

In contrast to the work of Lund, two key papers have recently claimed to have fluidic methods which outperform ELISA and therefore both of these papers deserve a brief comment. In the first such study, the flow-through membrane immunoassay (FMIA) platform was utilized as an alternative to ELISA in fast high-throughput scheme [9]. However, the FMIA utilizes a 96-well vacuum plate in which the molecule of interest and other reagents are drawn through a fixed nitrocellulose membrane. This already limits the design of the system as the nitrocellulose the membrane must necessarily be covered in advance with the capture molecules (concentrations therefore remain invariable). Additionally, the capture molecules in this study are gold nanoparticle-labeled antibodies which are then utilized for the visible assay signals. Gold-nanoparticles are toxic to cells/cultures [10], immediately limiting the technology as the possibility of contamination, in particular with complex (3D) cultures is too great to risk using metal nanoparticles. On the positive side, the FMIA does provide rapid results (<30 min), but the authors further claim to require fewer user steps than ELISA which is simply not true if the individual steps of the FMIA are counted. Finally, the final three arguments for FMIA over ELISA are that FMIA: 1) “provides multiple assay results (including controls) for each sample”, and 2) “uses reagents that can be stored in stable dry form”, and “generates visible spots that can be quantified by a camera or a flatbed scanner”. Standard ELISA meets these criteria, and the system presented in this thesis exceeds the FMIA parameters in all aspects.

In the second such study, a flow through assay (FTA) was developed on cellulose acetate membranes for the cysticercosis, a parasitic tissue infection caused by larval cysts of the tapeworm [11]. FTA was claimed to be as good as ELISA in such a situation. Trying both

cyst fluid antigens (CFA) and whole cyst antigens (WCA) in the FTA, the assay consisted of (very similar to the previous alternative method) an antigen coated onto a membrane and then the membrane being mounted on a flow-through device. The membrane is again the assay capture matrix, with the criticisms from the previous paragraph still valid in this case. Although not as toxic as nano-particles, a colloidal gold conjugate was used as the antigen-antibody reagent for detection – again, possibly not ideal for cell cultures. The authors showed that between CFA and WCA, that results in the FTA were better with CFA (96.0% sensitivity; 96.0% specificity) compared to WCA (92.0% sensitivity; 96.0% specificity). These results were then compared to tests performed using ELISA. The ELISA showed 96 per cent sensitivity with both the antigens. This demonstrates that FTA is not superior to ELISA, the FTA simply has a sensitivity and specificity which agrees closely with the results of the ELISA, and only under certain conditions. Indeed, as the authors say, "The highest diagnostic accuracy (96%) was obtained with CFA-FTA and CFA-ELISA". However, the FTA is by no means fully automated as the work presented here.

More recently, four proof of concept studies were published, using the electrochemical sensor as the means of readout. Lebogang at all (2017) uses the Separose™ beads filled into micro column for the quantification of microcystin-LR [12]. The setup is based on flow-ELISA with amperometric sensor and 2,2'-azinobis-(3-ethylbenzothiazoline-sulfonic acid) (ABTS) as the substrate. The run time of automated sequential flow assay is 20min. The setup does not provide means for parallel measurements and it can be regarded as single channel system. The amperometric sensor is connected to laboratory potentiostat and can perform up to 6 measurements without significant accuracy degradation. The setup is based on standard laboratory equipment and it is not portable.

Riahi at all (2016) uses flow-ELISA in combination with PDMS microfluidic chips to measure transferrin and albumin levels in hepatocyte culture medium [13]. Disposable magnetic beads are used as the solid phase and electrochemical amperometric sensor connected to the potentiostat as the detector. The system comprises microfluidic bioreactor for perfusing 2D hepatocyte culture. Similarly, to previously described system, the setup was designed as single channel quantitation tool. The supporting equipment is not integrated with the fluidic part.

The works presented by Shin at all (2017) and Zhang at all (2017) use the same electrochemical sensor design with functionalized surface by antibodies to achieve specific selectivity for detection of desired biomarkers [14, 15]. The sensor uses the electrochemical impedance measurement as the means of detection. Other common features are automated flow-ELISA architecture and the use of microfluidic chips based on PDMS material. The system presented by Shin was designed to measure the concentration of albumin and glutathione-S-transferase-alpha (GST-α) in samples from hepatic culture bioreactor. The multiparameter sensing capability was extended in work presented by Zhang, where additional measurement of cardiac biomarker creatine kinase MB (CK-MB) was included, together with three environmental sensors (temperature, pH and dissolved O_2). Separate, gel-based 3D hepatocyte HepG2 culture and cardiomyocyte culture (iPSC-CMs) were grown in micro bioreactors and perfused in one common loop. The functionalized immunosensors saturate after several measurements and require invasive *in-situ* regeneration process where the thin layer surface of gold electrode is etched out. As the consequence, the sensor lifetime (number of regeneration cycles) is reduced. However, it was demonstrated that 25 regeneration cycles cause no significant loss of functionality. Both systems, despite being capable of multiparameter sensing, provide no more than single channel, meaning no support for parallel sample measurement. The complexity and handling requirements of those systems are extensive.

The use of PDMS material is not optimal for cell culture systems due to problems associated with adsorption and absorption of hydrophobic small molecules and drugs [6, 15, 16]. This becomes the limiting factor during long experiments and low biomarker concentrations. For long term 3D cell culture experiments, a reliable and robust culture and analysis platform is needed, which is hard to achieve using PDMS technology. Moreover, contamination is a major problem that arises in most modern culture and analysis devices due to complex handling of units during cell seeding/harvesting, medium exchange and sale collection [17]. The immuno-electrochemical sensors technology, despite the excellent sensitivity they provide, is not matured and the stability of those sensors is not sufficient for use in measurement equipment where the reliability and robustness is the priority. The principal disadvantage of single channel system is the lack of concurrent calibration. The sensor must be calibrated prior to sample measurement and recalibrated in regular intervals to avoid the loose of accuracy. This becomes even more important for sensors with reduced stability.

In summary, the presented work here aimed to develop a fully automated and robust device (the 3D-ACAD) which combines 3D cell culture with a fully automated perfusion, medium change, ability for repeated drug applications, sampling, and followed by an automated flow-ELISA for detection of cell-derived albumin for the assessment of hepatotoxicity. In comparison to other works, this device was focused on developing a scaffold-based 3D-culture-and analysis system which allows good accessibility of the drug to the cells, minimizing adsorption and absorption of small molecules, drugs, and biomolecules inside the closed system. Additionally, the readout system is based on robust optical sensing principle designed for multi-channel operation, with inherent calibration. In contrast to some of the problems and methods discussed above, the ELISA analyzer module in this work has been designed in such a way that almost any commercially available ELISA assay kit can be used with this system and therefore made available to a wide range of users rather than using custom membranes and custom assays.

3 Designing the prototype of the ELISA analytical device

3.1 Introduction

This chapter describes the development and design of the prototype of the automated ELISA analytical device, including the associated electronic control unit. The purpose of the prototype device is to verify the possibility of automated ELISA assay in a flow-through configuration. The prototype device was preferentially built from commercially available components, rather than custom components, allowing for rapid improvements and modifications in trial versions of the system. The following text describes the design of the fluidic part along with the sandwich ELISA details and the design of the integrated fluorimeter as the means of readout. The associated control unit schematics, layout and the embedded software are also briefly described.

3.2 Fluidics

The design of the fluidics must support all fluidic operations required by the typical ELISA procedure. This includes sequential transfer of the capture antibody, the analyte, the labeled antibody, and the substrate into the reaction chamber. Thorough washing of the fluidic paths is very important. The fluidic designs must enable efficient washing to prevent cross contamination of the fluidic paths by different regents.

3.2.1 Selected Elisa Assay

For the detection of albumin in the cell culture medium the commercial Human Albumin Quantitation Set (Bethyl, E80-129) was selected. The analyzer device is supposed to replicate the protocol recommended by the manufacturer. It is based on the sandwich ELISA assay (Figure 3.1) using the Goat anti-Human Albumin Coating Antibody, human reference serum and the HRP labelled Goat anti-Human Albumin Detection Antibody. Useful quantitation range lies inside the concentration interval from 6,25 ng/ml to 400 ng/ml of albumin.

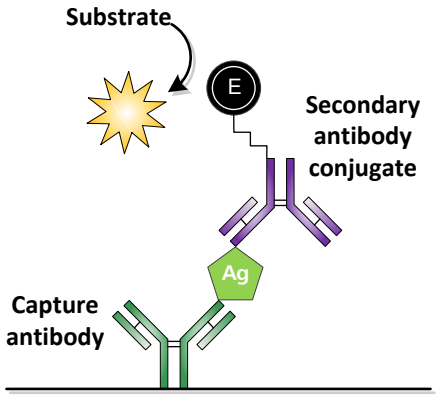

Figure 3.1 Sandwich ELISA principle.

The assay protocol as recommended by the manufacturer comprises the following steps:

1. Coat the plate using the primary (capture) antibody
2. Incubate the plate at room temperature 1 hour
3. Wash the plate five times
4. Coat the plate using the blocking solution
5. Incubate 30min at room temperature
6. Wash the plate five times
7. Add samples and standards to plate wells
8. Incubate 1hour at room temperature
9. Wash the plate five times
10. Add HRP conjugated (detection) antibody to the plate wells
11. Incubate 1 hour at room temperature
12. Wash the plate five times
13. Add the substrate solution to the plate wells
14. Develop the plate 15 min
15. Stop the reaction adding the stop solution to the plate wells
16. Measure the absorbance or fluorescence (depending on the used substrate) using plate reader.

This protocol uses MTP as the solid phase reaction carrier assuming the fluid operations are performed by the pipetting operations. The protocol needs to be adapted for the flow-through configuration to address the different means of liquid handling. On the other side, the fluidic components should be selected with regard to assay protocol requirements.

In order to automate the assay, one could choose to use pipetting robots, complex machines optimized for using MTPs and automated pipetting. However, pipetting robots tend to be large in size and quite expensive, and trained personnel are necessary to operate such robots. Furthermore, many biological experiments simply require the monitoring of some metabolic indicator in the cell culture medium at regular intervals and for such a trivial task, the use of pipetting robots would be a unnecessarily expensive investment and not economically justified. Therefore, there is a need for a smaller, simpler device which could provide repeated and automated analysis using e.g. ELISA as the analysis method. One way to achieve this goal is to utilize a flow-through configuration. The pipetting operations are eliminated this way and replaced by the pumps and valves as the means of liquid handling according to required protocol. This configuration is rarely utilized because it creates new challenges – e.g. to avoid cross contamination of the commonly used fluidic paths or cleaning the fluidics after the end of essay. Careful choice of the fluidics material, components and washing procedures is very important to successfully implement of selected assay in the flow-through configuration analytic device.

3.2.2 Selection of the fluidic components

In the first step of the ELISA protocol, the capture antibody needs to be bound to a solid surface (also referred to as solid phase). Various materials have been used for this purpose – polystyrene (PS) and polyvinyl chloride (PVC) are among the most commonly used [18]. For the implemented flow-through configuration the 20µl size PVC capillary (SC-Sanguia Counting, Type 100024) was chosen as the solid phase component. PVC and PS are reported to have a high protein binding capacity. The selected volume is sufficiently high to enable the measurement by the optical readout device, but reasonably small to save expensive reagents. The selected PVC capillary had internal diameter of 0.96 mm, outside diameter 1.8 mm and are approximately 28 mm long.

3.2.2.1 Tubing selection

The appropriate tubing material for this application needs to have a low protein binding capacity. The ELISA protocols usually use highly diluted antibody solutions. It is therefore important that binding of antibodies to the tubing walls be minimized to prevent

the depletion of reagent solution and also to prevent unwanted reactions inside the tubing. The tubing material should be also chemically inert to withstand cleaning and protein desorption operations. Optimal material properties for this application has the flexible polymer with a trade name C-Flex manufactured by the Saint Gobain Performance plastics [19]. This polymeric compound is based on hydrogenated styrene/isoprene-butadiene/styrene block copolymer and belongs to the thermoplastic elastomers group. It is highly biocompatible, has low protein binding capacity, and sufficient chemical resistance. Moreover, it is also compatible with the selected pinch solenoid valves of the system designed here. Another material used in the manifold construction is the Teflon FEP. It is characterized by excellent chemical resistance, high temperature processability and low protein binding capacity compared to other standard materials. The tubing internal diameter should be small to minimize dead volume on one side, but of sufficient size to prevent clogging by eventual precipitates. The internal diameter of 0.58 mm seemed to be a good compromise and was therefore chosen for this application.

3.2.2.2 Active fluidic components

An analytical flow-through device is expected to use a rather higher number of fluidic paths. The switching between different fluidic paths as required by the ELISA protocol is realized by use of two-way and three-way valves. For practical reasons, only solenoid (electromechanically operated) isolation valves were considered. Important requirements for our selected valves were low dead volume and biocompatibility of all the wetted parts. A low footprint area and a low power operation are of secondary importance. The use of pinch valves over traditional seat or membrane valves is preferred for truly zero dead volume and very simple washing and maintenance. The pinch valves manufactured by Bio-Chem Fluidics Inc. perfectly fit all these requirements. The 19 mm diameter valves optimized for 0.5mm internal diameter C-Flex tubing were selected: the type 075P2NC12-23B is normally closed two-way valve and the type 075P3MP12-23B is the three-way valve (Figure 3.2). The valves require 12 V / 240 mA (2.9 W) for switching, after that the power can be reduced to 5 V / 100 mA (0.5 W) to hold the valve in the switched position. The tubing can be easily inserted to or removed from the valve head.

Figure 3.2 Selected types of the 2-way (left) and 3-way (right) solenoid valves

The pump selection is primarily dictated by the flexibility. It is required to operate in a wide flow-rate range (10 µl/min to 1000 µl/min), the precise (accuracy 10% of pumped volume is sufficient) and predictable dosing must be possible and reverse operation is also required. Similarly, as in the case of the valves, all wetted parts must be biocompatible and have sufficient chemical resistance. The low protein binding property is not required for the pump. The syringe pumps and the peristaltic pump are principally suitable for this application, however the peristaltic pump is preferred over the syringe pump because the pumped volume can be unlimited and does not depend on the syringe size. Syringe pumps are more precise and have more uniform flow-rate compared to peristaltic pumps, which makes the very suitable for segmented flow and microfluidic applications. The pumping accuracy of the peristaltic pump is mainly dependent on the accuracy of controlling its rotor speed and position. For that reason, peristaltic pumps using a DC motor or asynchronous AC motor are not suitable. The preferred motor for controlling the peristaltic pump suitable for the ELISA analyzer is a DC stepper motor, because a stepper motor is synchronous, the rotor position depends only on the number of step pulses issued by the motor controlled and is independent of the mechanical load (to a certain maximum limit). Suitable stepper motor controlled peristaltic pump heads are manufactured by the company Boxer GmbH and the type 61131.000 was selected for use in the analyzer (Figure 3.3). It is powered by NEMA23 size stepper motors with 200 steps per revolution. The pump head uses the pharmed tubing with internal diameter of 0.5 mm arranged in four independent channels. The measured pumped volume is 12.5 µl per revolution. The calculated motor speed for the flow-rate 10 µl/min is 0.8 rpm or 160

steps/min and the calculated motor speed for the maximum flow-rate 1000 µl/min is 80 rpm or 16000 steps/min.

Figure 3.3 Selected stepper motor driven peristaltic pump

3.2.3 Fluidic Topology

Figure 3.4 shows schematically the complete fluidic topology of the prototype analyzer device. It accomplishes all steps of the sandwich ELISA protocol for seven samples and/or analyte standards. Special attention was paid to the design of the washing and cleaning operations. The whole fluidic subsystem is composed of the following components:

- Two peristaltic pumps (type 61131.0000, Boxer GmbH)
- 19 two-way normally closed solenoid pinch valves (075P2NC12-23B, Bio-Chem Fluidics Inc.)
- One three-way solenoid pinch valve (075P3MP12-23B, Bio-Chem Fluidics Inc.)
- Four nine-port manifolds (P-191, IDEX Health & Science)
- Two 50ml polypropylene containers – one for the washing buffer and one for the waste.
- Eleven 2ml polypropylene containers – seven for the samples or analyte standards and four for sandwich ELISA reagents.
- Eight replaceable 20µl PVC capillaries (Type 100024, SC-Sanguia Counting)
- C-Flex tubing, internal diameter 0.58mm (type 10025-23B, Bio-Chem Fluidics Inc.)

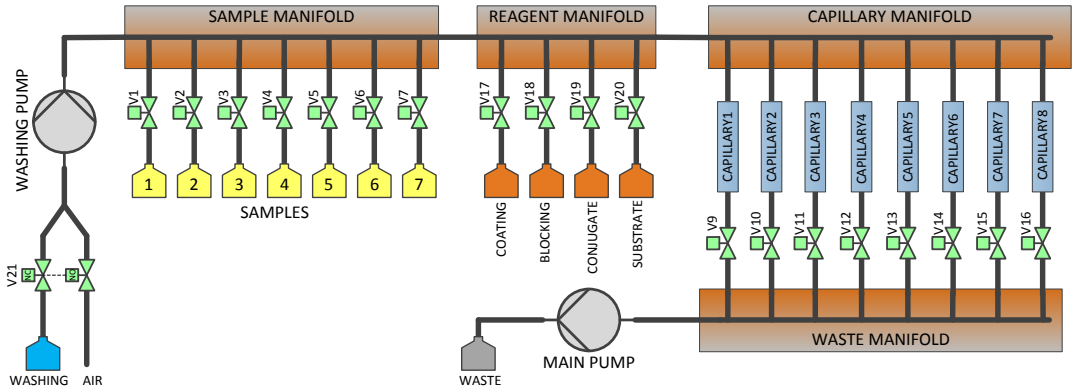

Figure 3.4 The fluidic topology of the analyzer based on the ELISA protocol

The operation of this fluidic network is relatively simple. The valves V17 to V20 select which reagent will be transferred to the selected capillary. The valves V1 to V7 select which sample or analyte standard will be transferred to the selected capillary. Normally, only one valve from the groups V1 to V7 and V17 to V20 will be open at the same time. The valves V9 to V16 select the destination capillary into which the fluid will be pumped. The main Pump is the pump used for the transfer of selected sample or reagent into selected capillary. The washing pump is stopped during this operation. The washing pump is used to wash all the fluidic paths and capillaries using the washing buffer. When some capillary needs to be washed, one of the valves V9 to V16 corresponding to selected capillary will be opened and both the main pump and the washing pump will be started at the same speed. By opening the valve V21 during the washing procedure, it is possible to empty the whole fluidic network. Capillary 8 has a special purpose as it is used as the fluidic bypass during the washing operations – in case capillaries 1 to 7 are filled with reagents according to the protocol and in case it is also necessary to wash the manifold common fluidic paths. Then capillary 8 is used to drain the washing solution to the waste. This way the washing of the common fluidic paths can be realized without disturbing the content of the other capillaries.

For example, if the protocol requires to pump a conjugated antibody into capillary 2, the following sequence will be executed: V19 will be open to connect the conjugate antibody to the fluidic network, V10 will be also open to enable the flow through the capillary 2 and the Main Pump will be instructed to pump a calculated amount of reagent so that capillary 2 will be filled with it. All remaining valves are in the closed position and the washing pump is stopped.

Because this prototype analyzer version contains only seven capillaries available for the measurement, it is possible to measure only seven points in one run. Those seven points comprise the actual samples as well as the concentration standards. A typical configuration is to use the 5 capillaries to measure points of the standard curve and the remaining two capillaries can be used to measure samples automatically taken from the two culturing units (described in the **Chapter 4**)

The following section describes the complete sandwich ELISA protocol optimized for this flow-through configuration in detail. The specific flow-rates and volumes used in the protocol are dependent on the fluidic network physical size, taking into account the dead volume of the fluidic paths, and preventing a significant pressure drop over the fluidic network.

3.2.4 Flow-through optimized sandwich ELISA protocol

3.2.4.1 Step 1 – Coating the capillaries with the capture antibody.

The first step of the human albumin ELISA protocol coats the walls of capillaries (solid phase) with the capture antibody. The capture antibody as purchased (A80-129A-11, Bethyl Inc.) was diluted with the coating buffer in the ratio 1:100. The coating buffer has the following composition:

Coating Buffer:

- 50 mM Na_2CO_3
- pH adjusted to 9.6 using HCl or NaOH

During step 1 the diluted capture antibody is pumped to capillaries 1 to 7 using the following sequence:

- Valves V16 and V17 will open, all other valves remain closed (unpowered)
- The main pump will be activated to pump 300 µl at the speed of 350 µl/min of diluted capture antibody solution through the reagent and capillary manifolds to capillary 8. The purpose is to fill the fluidic paths and manifolds with the reagent.

- The valve V16 will be closed again and the valve V9 will open
- The main pump will be activated to pump 50 µl at the speed of 100 µl/min of diluted capture antibody solution to capillary 1. The pumping speed is reduced to prevent turbulent flow through the capillary
- The previous step will be repeated six more times to fill the capillaries 2 to 7 in the serial sequence. The corresponding valves V10 to V15 will be used instead of the valve V9.

The flow path of step 1 for filling capillary 1 is illustrated in Figure 3.5. During this step all the capillaries are filled with the capture antibody solution. The total duration of step 1 is 4 minutes and 28 seconds. 700 µl of the capture antibody reagent is consumed in the first step.

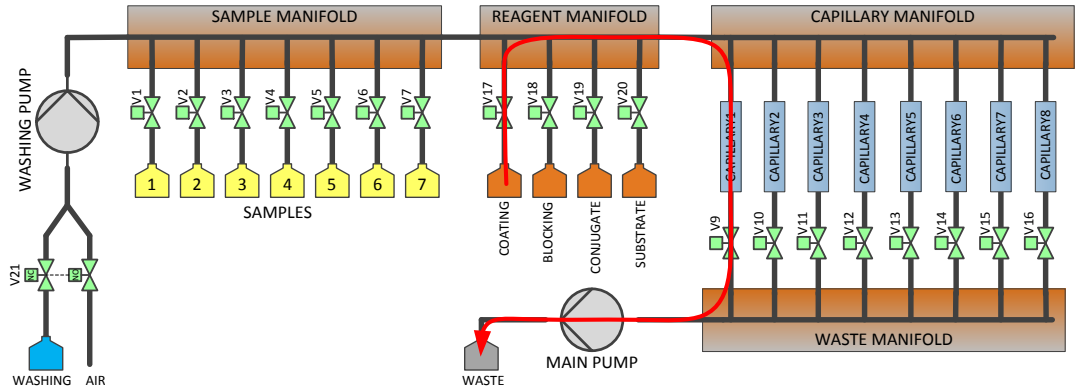

Figure 3.5 Flow path for filling capillary 1 with the capture antibody during step 1. Capillaries 2 to 7 are subsequently filled using the valves V10 to V15 instead of V9.

3.2.4.2 Step 2 – Incubation of the capture antibody.

During step 2, the dissolved capture antibodies in the capillaries are adsorbed on the surface of the PVC capillaries. The incubation period is 15 min and takes place at room temperature. The schematic picture of the ELISA after step 2 is shown in Figure 3.6.

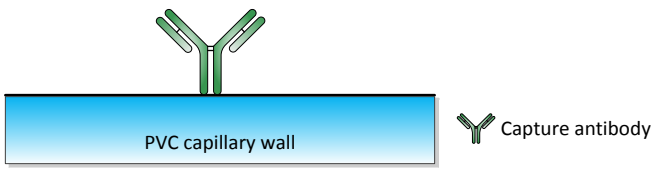

Figure 3.6 The first step of sandwich ELISA - the capture antibody was attached to the solid phase.

The incubation time remains constant for all capillaries despite the serial way of filling the capillaries and the non-negligible pumping time. This is achieved by keeping the capillary filling speed the same for the steps before and after the incubation period. The incubation period is only slightly extended by the time required to fill the manifold and drain capillary 8 of the liquid, which is the duration of the first part of the sequence following the incubation (usually 75 seconds). The incubation time can be easily corrected for this increase if needed. The timing of the first three steps of the ELISE sequence: the coating with capture antibody, the incubation and the washing is shown in Figure 3.7. The same principle is used also in the following parts of the ELISA sequence.

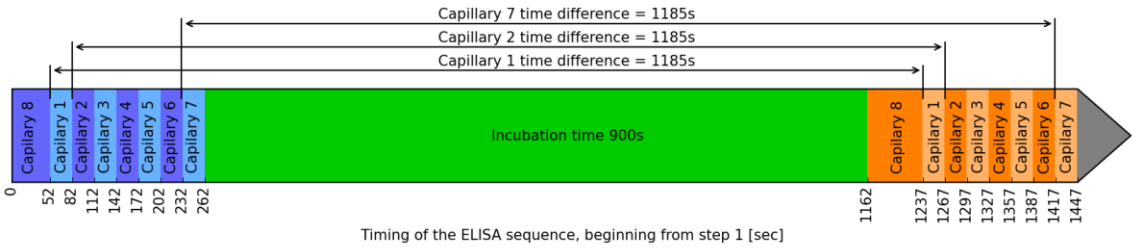

Figure 3.7 The beginning of the ELISA sequence – capture antibody coating (blue), incubation period (green) and the first part of the washing (orange). The time between the start of coating and the capillary with antibody to the washing of the same capillary is constant for all channels.

3.2.4.3 Step 3 – Washing

After the capture antibody was attached to the solid phase, the fluidic paths and the capillaries need to be washed to remove all residuals of the capture antibody reagent. The cleaning is performed using the washing buffer of the following composition:

Washing Buffer:

- 50 mM Tris(hydroxymethyl)aminomethane (TRIS)
- 140 mM NaCl
- pH adjusted to 8.0 using HCl or NaOH

During this washing step the washing buffer is pumped to all capillaries using the following sequence:

- The valve V16 will open and the valve V21 will switch to the washing container fluidic path, all other valves remain closed (unpowered)
- The main pump and the washing pump will be simultaneously activated to pump 500 µl at the same speed of 400 µl/min. This will wash the sample manifold, the reagent manifold and the capillary manifold. The washing buffer will be pumped through capillary 8.
- The valve V16 will be closed again and the valve V9 will open
- The main pump and the washing pump will be simultaneously activated to pump 50 µl at the same speed of 100 µl/min. This will wash capillary 1 at low speed to not desorb the attached antibodies.
- The previous step will be repeated six more times to wash capillaries 2 to 7 in the serial sequence. The corresponding valves V10 to V15 will be used instead of the valve V9.

This washing procedure is repeated two times during the step 3. The total time required to complete step 3 is 9 minutes and 42 seconds. 1700 µl of washing buffer is consumed during this time. The Figure 3.8 shows the washing flow path.

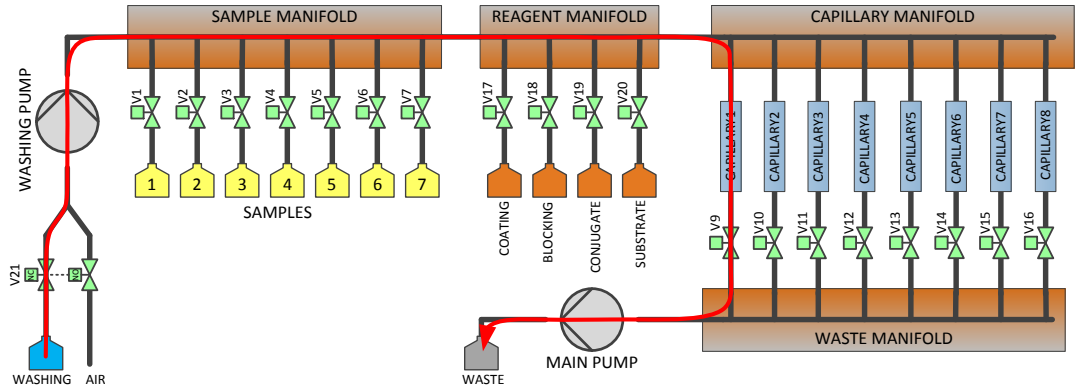

Figure 3.8 Flow path used for washing the capillary 1. Capillaries 2 to 7 are washed in the same manner using the valves V10 to V15 instead of V9.

3.2.4.4 Step 4 - Blocking the capillaries with the bovine serum albumin (BSA)

The coverage of the solid phase with the capture antibody performed in the steps 1 and 2 leaves a lot of binding sites on the surface free. The purpose of blocking step is to saturate those free sites with the BSA which is inactive in the assay. This greatly reduces the

interference and lowers the background reading. The blocking solution has the following composition:

Blocking buffer:

- 50mM Tris(hydroxymethyl)aminomethane (TRIS)
- 150mM NaCl
- 1% Bovine serum albumin (BSA)
- pH adjusted to 8,0 using HCl or NaOH

The fluidic operation to complete step 4 is very similar to the first step. It differs in using the blocking reagent instead of the capture antibody reagent. The exact sequence of the step 4 is following:

- Valves V16 and V18 will open, all other valves remain closed (unpowered)
- The main pump will be activated to pump 300 µl at the speed of 350 µl/min of the blocking buffer through the reagent and capillary manifolds to the capillary 8.
- The valve V16 will be closed again and the valve V9 will open
- The main pump will be activated to pump 50 µl at the speed of 100 µl/min of the buffer solution to the capillary 1.
- The previous step will be repeated six more times to fill the capillaries 2 to 7 in the serial sequence. The corresponding valves V10 to V15 will be used instead of the valve V9.

The step 4 flow path for filling the capillary 1 is illustrated on the Figure 3.9. During this step all the capillaries are filled with the blocking buffer solution. The total duration of step 1 is 4 minutes and 28 seconds. 700 µl of the blocking reagent is consumed in the step 4.

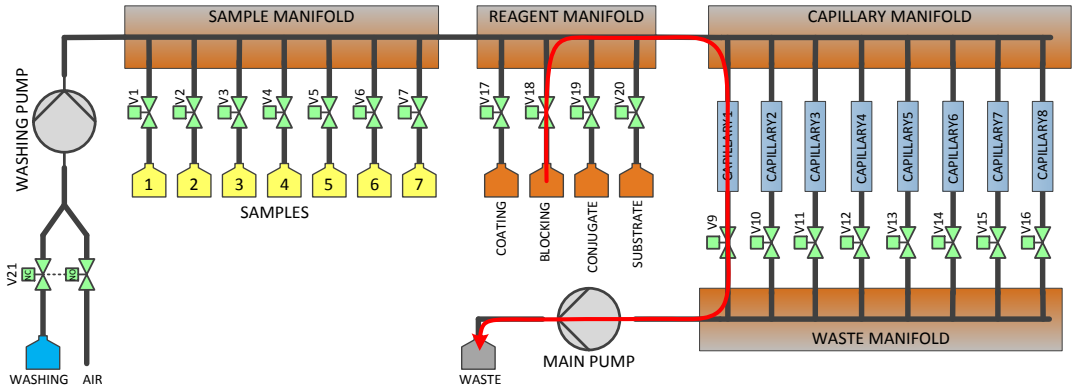

Figure 3.9 Flow path for filling the capillary 1 with the blocking buffer. Capillaries 2 to 7 are subsequently filled using the valves V10 to V15 instead of V9.

3.2.4.5 Step 5 – Blocking Incubation

During incubation time the BSA will saturate the surface of the solid phase and it will block all free binding places not occupied by the capture antibody. The blocking incubation time is 30 min and the operation is also made at room temperature. The schematic picture of the ELISA after step 5 is shown on the Figure 3.10.

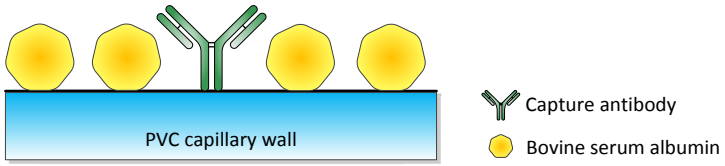

Figure 3.10 Situation at the capillary surface after blocking step. No free place is left on the surface for binding more proteins.

3.2.4.6 Step 6 – Washing

After the completion of the blocking step, the washing is needed again to clean the fluidic paths and to remove the blocking solution from the capillaries. This washing step is exactly the same as the washing described in the step 3. Again, it requires 9 minutes and 42 seconds and 1700 µl of washing buffer to complete.

3.2.4.7 Step 7 – Filling the capillaries with human serum albumin samples

During this step the analyte samples held in the containers 1 to 7 will be transferred one by one to respective capillaries 1 to 7. Until now the procedure was the same for all 7 capillaries. Starting from this step each capillary will have different protein compositions on the surface, depending on the analyte composition.

The fluidic topology as designed contains 8 capillary channels. The capillary 8 is not used for the measurement and serves as the draining the fluid through manifolds while not affecting the capillaries 1 to 7, which are the actual measurement channels. The typical ELISA procedure includes the standard curve measurement along with the samples containing unknown concentration of the analyte. The standard curve serves as way of concentration calibration. The standards curve is created by measurement of a series of standards with different but known concentration of analyte. The seven available channels of the proposed device can be allocated either to the sample or standard measurement. The configurations providing reasonable measurement accuracy are: 6 standards + 1 sample, 5 standards + 2 samples or 4 standards + 3 samples.

The sample source

There are two ways in which the sample solution can be interfaced to this analytic device: The sample can be place in a small container directly in the device. This is the way the concentration standards are connected to the device. Alternatively, and preferably the fluidic connection is used to interface the culturing unit (see the **Chapter 4**) with the selected sample valve (V1 to V7). This will enable the automated sample feeding from the culturing unit to the analytic device. In this case the volume of pumped sample is increased to compensate for the dead volume of the inter-module fluidic connection.

The step 7 will start with the transfer of the sample 1 to capillary 1 according this sequence (Figure 3.11):

- Valves V1 and V16 will open, all other valves remain closed (unpowered)
- The main pump will be activated to pump 300 µl at the speed of 350 µl/min of the first sample solution through the reagent and capillary manifolds to the capillary 8.

- The valve V16 will be closed again and the valve V9 will open
- The main pump will be activated to pump 50 µl of the sample 1 solution at the speed of 100 µl/min of to the capillary 1.
- Valve V9 will close

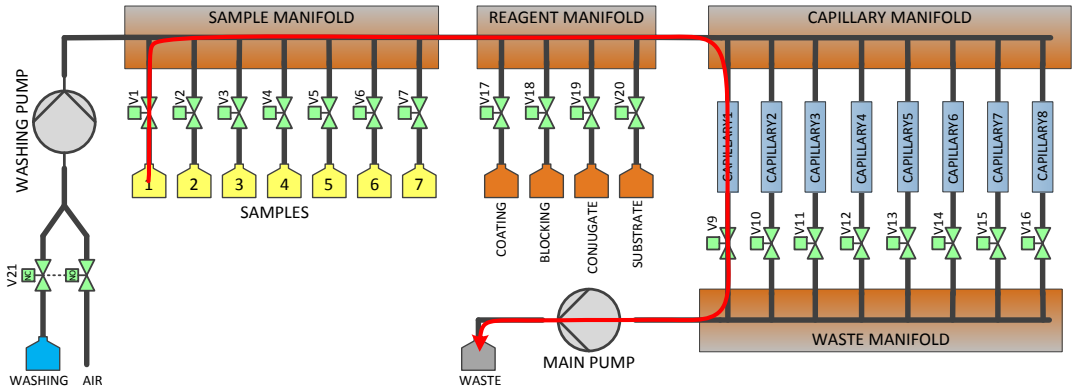

Figure 3.11 Fluidic path for transferring the sample 1 into the corresponding capillary 1

After the sample 1 has been transferred to the capillary 1 and before the transfer of sample 2 to corresponding capillary 2 the fluidic paths must be washed to avoid the cross contamination of the samples. This "washing after sample" procedure is performed after each sample transfer and has the following sequence:

- The valve V16 remains open after the previous sample transfer. All other valves stay closed.
- The main pump and the washing pump will be simultaneously activated to pump 350 µl at the same speed of 350 µl/min. This will empty the fluidic path inside the sample manifold, reagent manifold and capillary manifold. The liquid will be drained through the capillary 8.
- The valve V16 will close and the valve V9 will open
- The washing pump will be activated to pump 3 µl at the speed of 100 µl/min. This will make small air gap inside the capillary manifold from the common fluidic path towards the fluidic path of the capillary 1.
- The valve V9 will close and the valve V1 will open
- The washing pump will be activated to pump 50 µl at the speed of 100 µl/min. This will make small air gap inside the sample manifold from the common fluidic path towards the fluidic path of the sample 1.

- The valve V1 will close and the valves V16 and V21 will open
- The main pump and the washing pump will be simultaneously activated to pump 500 µl at the same speed of 350 µl/min. This will wash the sample manifold, the reagent manifold and the capillary manifold common fluidic paths. The washing buffer will be pumped through the capillary 8.
- The valve V21 will close

In this moment the transfer of the sample 1 to the capillary 1 has been finished and the fluidic paths were washed and transfer of the sample 2 to the capillary 2 can take place. The sequence of emptying the flow path is depicted on Figure 3.12.

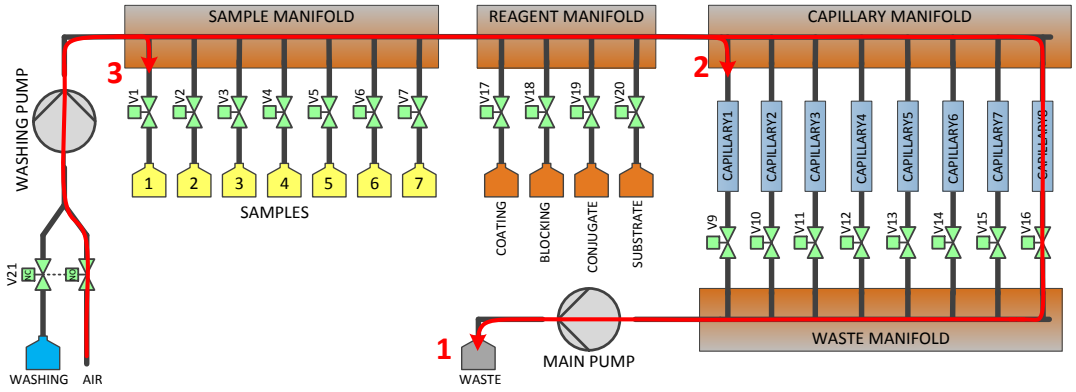

Figure 3.12 After the sample 1 has been transferred to the corresponding capillary 1, the fluidic path will be emptied in three steps: 1st – the fluid is pumped out of all manifolds through the capillary 8, 2nd – the side arm of the capillary manifold is emptied by introducing a small air gap, 3rd – the side arm of the sample manifold is also emptied. Subsequent washing of the common fluidic path (marked as red "1" on the figure) completes the "washing after sample" sequence.

The step 7 then continues with transferring of sample 2 to the capillary 2 followed by the "washing after sample" sequence again. This scheme is repeated until all seven samples are transferred to their corresponding capillaries. The step 7 consumes 350 µl of each sample and 3500 µl of washing buffer. The total time required for completion of this sequence step is 30 minutes and 55 seconds.

3.2.4.8 Step 8 – Sample Incubation

During this incubation time the human serum albumin contained in the sample bonds to the capture antibody. The number of the antibody-albumin pairs will be different in each

capillary depending on the albumin concentration in the samples. The samples must be sufficiently diluted so that even for the maximum sample albumin concentration the antibody binding capacity will be not exceeded at the end of the incubation time, which is set to 15 minutes. The incubation takes place at room temperature. The schematic picture of the ELISA after step 8 is shown on the Figure 3.13.

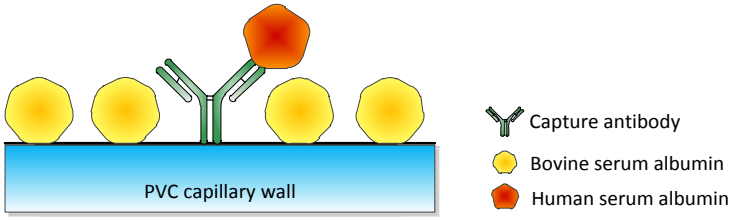

Figure 3.13 Situation at the capillary surface at the end of sample incubation time. The human serum albumin is selectively bonded to the capture antibody. Ideally there is no other possibility for the albumin to bond.

3.2.4.9 Step 9 – Washing

The sample incubation is followed by washing again. This washing uses the sequence described the step 3. This washing step requires 9 minutes and 42 seconds and 1700 µl of washing buffer to complete – same values are used in the step 3.

3.2.4.10 Step 10 – Filling the capillaries with enzyme labeled secondary antibody.

The secondary antibody binds specifically to the human serum albumin. The amount of bonded antibody will be therefore directly proportional to the amount of human serum albumin already bonded to the capture antibody. If the sample contained no human serum albumin, no secondary antibody will be bonded and it will be washed away in the following steps. This secondary antibody has covalently bonded HRP enzyme which later allows substrate conversion to a colored dye. The secondary antibody as purchased (A80-129P-30, Bethyl Inc.) was diluted with the blocking buffer (described in the step 4) in the ratio $1:10^5$.

During step 10 the diluted secondary antibody is pumped to capillaries 1 to 7 using the following sequence:

- Valves V16 and V19 will open, all other valves remain closed (unpowered)
- The main pump will be activated to pump 300 µl at the speed of 350 µl/min of diluted secondary antibody solution through the reagent and capillary manifolds to the capillary 8.
- The valve V16 will be closed again and the valve V9 will open
- The main pump will be activated to pump 50 µl at the speed of 100 µl/min of diluted capture antibody solution to the capillary 1. Low pumping speed prevents disturbing the surface layer.
- The previous step will be repeated six more times to fill the capillaries 2 to 7 in the serial sequence. The corresponding valves V10 to V15 will be used instead of the valve V9.

The step 10 flow path for filling the capillary 1 is illustrated on the Figure 3.14. During this step all the capillaries are filled with the secondary antibody solution. The total duration of step 10 is 4 minutes and 28 seconds. 700 µl of the secondary antibody reagent is consumed.

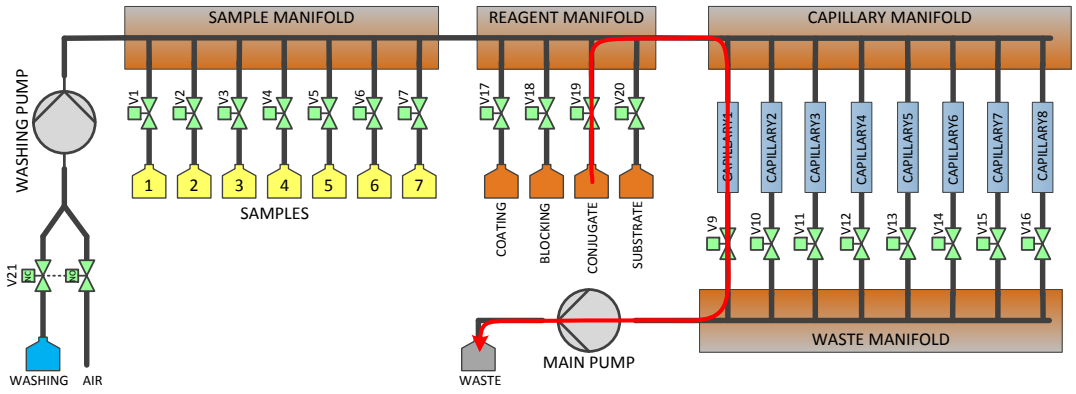

Figure 3.14 Flow path for filling the capillary 1 with the secondary antibody solution. Capillaries 2 to 7 are subsequently filled using the valves V10 to V15 instead of V9.

The secondary antibody is covalently bonded to the HRP enzyme, which converts the substrate to colored compound. This reaction is supposed to take place inside capillary where the enzyme is attached to the capillary walls as a part of the sandwich ELISA stack. Even traces of HRP enzyme inside the fluidic paths preceding the capillary start the substrate conversion prematurely and result in the increased signal background. It is therefore very important to carefully clean the common fluidic paths to remove traces of

HRP enzyme before substrate enters the flow paths. For that reason, a special cleanup sequence is executed at the end of the step 10.

This cleanup sequence comprises following operations:

- The valve V16 will open while all the other valves remain closed.
- The main pump and the washing pump will be simultaneously activated to pump 350 µl at the same speed of 350 µl/min. This will empty the fluidic path inside the sample manifold, reagent manifold and capillary manifold. The liquid will be drained through the capillary 8.
- The valve V16 will close and the valve V17 will open
- The washing pump will be activated to pump 50 µl at the speed of 100 µl/min. This will remove the fluid still remained in the coating reagent side flow path of the reagent manifold.
- The previous operation will be repeated two more times to remove fluid remaining in the blocking and conjugated flow paths of the regent manifold. The valves V18 and V19 will be used instead of the valve V17.
- The valves V16 and V20 will open while all the other valves remain closed.
- The main pump will be activated to pump 25 µl at the speed of 200 µl/min. This will extract any secondary antibody solution possibly present in the substrate side flow path of the reagent manifold.
- The valves V20 will close and the valves V16 and V21 will open
- The main pump and the washing pump will be simultaneously activated to pump 500 µl at the same speed of 350 µl/min. This will wash the sample manifold, the reagent manifold and the capillary manifold common fluidic paths. The washing buffer will be pumped through the capillary 8.
- The valves V16 will close and the valves V17 will open
- The washing pump will be activated to pump 50 µl at the speed of 350 µl/min. This will pump the washing buffer into the coating reagent side flow path of the reagent manifold.
- The previous operation will be repeated two more times to pump the washing buffer into the blocking and conjugated flow paths of the regent manifold. The valves V18 and V19 will be used instead of the valve V17.

This special reagent manifold cleanup requires 4 minutes, 38 seconds and 650 µl of washing buffer to complete.

3.2.4.11 Step 11 - Incubation of the secondary antibody

This incubation time provides sufficient time for the conjugated secondary antibody to bind to the human serum albumin which was possibly (depending on the albumin content of the sample) present on the capillary surface layer. The secondary antibody contains covalently attached label which later convert the substrate to detectable substance. The amount of the enzyme attached to the protein stack during the incubation period is directly proportional to amount of immobilized albumin on the capillary surface. The incubation time is set to 15 minutes. The operation is performed at room temperature. The schematic picture of the ELISA after step 11 is shown on the Figure 3.15.

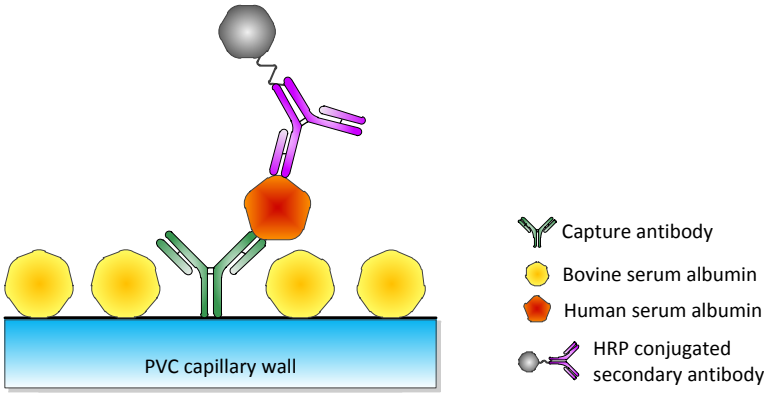

Figure 3.15 The complete sandwich ELISA stack at the end of step 11. If the sample contained human serum albumin, the HRP conjugated secondary antibody will be attached to it.

3.2.4.12 Step 12 – Washing

A regular washing procedure is introduced at the end of the incubation time. The sequence already described in the step 3 is repeated three times to address increased requirements for cleanliness before the substrate pumping step. This step requires 14 minutes and 33 seconds to complete. Additionally 2550 µl of washing buffer is consumed.

3.2.4.13 Step 13 – Transferring substrate to the capillaries

The substrate is generally colorless solution which can be converted by the action of enzyme to colored, fluorescent or chemiluminescent compound depending on the substrate type. The concentration of resulting compound can be measured by suitable optical detector. The substrate chosen for this albumin assay is QuantaRed™ Enhanced Chemifluorescent HRP Substrate (Number 15159, Thermo Scientific). For more details see the **Chapter 3.3**.

The substrate solution is prepared according to manufacturer instructions and has the following composition:

- 50 parts of QuantaRed™ Stable Peroxide Solution
- 50 parts of QuantaRed™ Enhancer Solution
- 1 part of QuantaRed™ ADHP Concentrate

This substrate solution is pumped to capillaries 1 to 7 using the following sequence:

- Valves V16 and V20 will open, all other valves remain closed (unpowered)
- The main pump will be activated to pump 300 µl at the speed of 350 µl/min of the substrate solution through the reagent and capillary manifolds to the capillary 8.
- The valve V16 will be closed again and the valve V9 will open
- The main pump will be activated to pump 50 µl at the speed of 100 µl/min of the substrate solution to the capillary 1.
- The previous step will be repeated six more times to fill the capillaries 2 to 7 in the serial sequence. The corresponding valves V10 to V15 will be used instead of the valve V9.

The corresponding flow path for filling the capillary 1 is illustrated on the Figure 3.16. During this step all the capillaries are filled with the substrate solution. The total duration of step 13 is 4 minutes and 28 seconds. 700 µl of the substrate reagent is consumed.

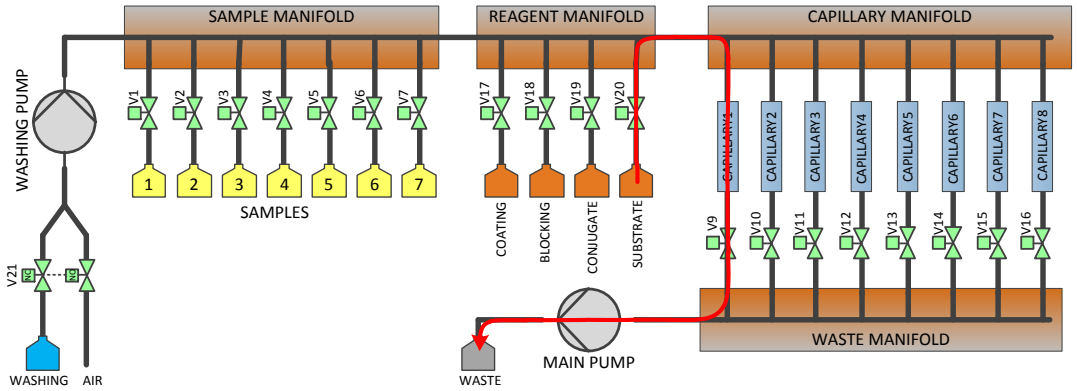

Figure 3.16 Flow path for filling the capillary 1 with the substrate solution. Capillaries 2 to 7 are subsequently filled using the valves V10 to V15 instead of V9.

3.2.4.14 Step 14 – Substrate incubation and readout

Immediately after the substrate solution enters the capillary, the immobilized enzyme starts to convert the optically inactive substrate into a detectable compound. In this specific case the non-fluorescent ADHP substrate is converted in the presence of hydrogen peroxide into resorufin - a highly fluorescent compound (Figure 3.19). The fluorescence of the substrate solution is measured by the integrated fluorimeter after exactly measured incubation time (same incubation time is used for each capillary). More details about the readout system can be found in the **Chapter 3.3**. The situation inside the capillary during this phase is depicted on the figure Figure 3.17.

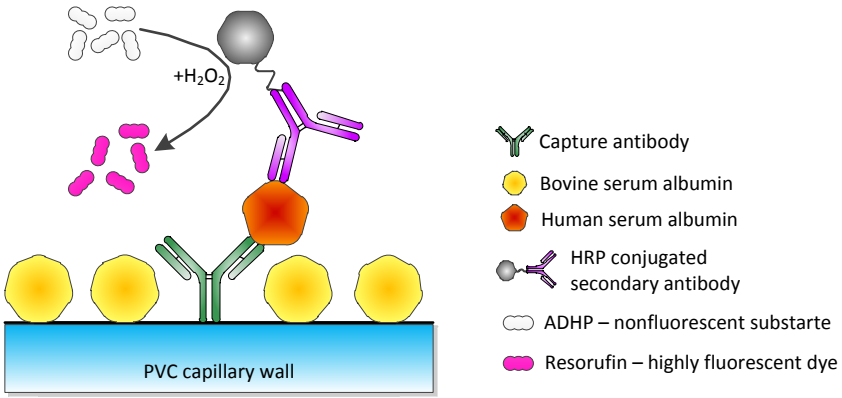

Figure 3.17 Conversion of the non-fluorescent ADHP substrate to fluorescent resorufin dye by the action of immobilized HRP enzyme during the step 14.

The fluorescence measurement is performed at fixed time points of 7, 14, and 21 minutes for each capillary to provide possibility to measure reaction kinetics. The duration of this step is 21 minutes and 26 seconds.

Step 14 completes the whole ELISA protocol with the total execution time of 3 hours, 13 minutes and 26 seconds.

3.3 Readout system

The selection of the readout system is based on the used enzyme label. The horseradish peroxidase (HRP) is widely used because the molecule is relatively small, does not cause hindrance problems, and is robust and inexpensive. A variety of substrates are available for HRP and they can be divided into the following categories:

- Colorimetric substrates
- Fluorescent substrates
- Luminescent substrates

Colorimetric substrates provide directly visible colored product which absorbs light in the visible range. The absorbance is proportional to the analyte concentration and it is measured photometrically. The usual dynamic range is about two orders of magnitude. The representative example of colorimetric substrate is TMB (3,3',5,5' tetramethylbenzidine), which produces a blue dye which can be measured at a wavelength of 650 nm. Other colorimetric substrates compatible with HRP are ABTS (2,2'-azino-di [3-ethylbenzthiazoline] sulfonate), and OPD (o-phenylenediamine).
In the fluorimetric assay a non-fluorescent substrate is converted to fluorescent dye by the action of the enzyme. The produced dye fluoresces when excited by the light of suitable wavelength. The intensity of the fluorescence is proportional to the analyte concentration. Compared to the colorimetric substrates, the fluorescent substrates benefit from higher sensitivity and broader dynamic range. On the other side the instrumentation is more complicated than for the absorbance measurement. Commonly used fluorimetric substrates include Amplex Red™, HPA (hydroxyphenylacetic acid) and HPPA (3-p-hydroxyphenylproprionic acid).

In a luminescent assay the enzyme converts the substrate to a chemical compound which emits photons of visible light instead of producing a colored product. Enhanced luminescent assays provide the highest sensitivity and dynamic range. The intensity of the produced light is proportional to the analyte concentration. The drawback is the stability of the luminescent light emission which is transient in its nature. The produced light must be intense, since it is not accumulated over time like in the case of color or fluorescence. Examples of luminescent substrates suitable for HRP comprise the luminol, luciferin, and some polyphenols.

The choice of the fluorimetric substrate for this prototype device is preferred over the other options. It provides the advantage of the higher dynamic range over the photometric assay (four orders of magnitude required) and the stability of the output signal. After reviewing the markets substrates for HRP, the use of QuantaRed™ Enhanced Chemifluorescent substrate (15159, Thermo Fisher Scientific) was selected as the most suitable for this application. The kit contains the ADHP (10-Acetyl-3,7-dihydroxyphenoxazine) non-fluorescent compound which is converted by the action of HRP in the presence of hydrogen peroxide to highly fluorescent dye resorufin (Figure 3.19). The sensitivity of this enhanced substrate is comparable with luminescent substrates. The development of the colored resorufin allows also the colorimetric measurement if needed.

The fluorimeter creates an integral part of the analyzer system. Its mechanical, optical, and electrical properties should be specifically designed to support the resorufin spectral properties. This will provide the advantage or easier automation and optimal price to performance ratio.

The design of the optics is driven by the properties of the fluorophore and the physical size of the measurement cell. The fluorophore used in the selected albumin assay is resorufin (Figure 3.18), which is created by the deacetylation and oxidation of the QuantaRed™ substrate catalyzed by HRP enzyme.

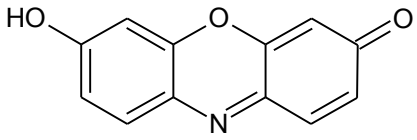

Figure 3.18 The chemical structure of the resorufin fluorescent dye

Resorufin has the peak excitation wavelength 571 nm and the peak emission wavelength 585 nm. Figure 3.20 shows the corresponding spectra. The Stokes shift is about 14 nm and the corresponding spectra partly overlap. For accurate fluorescence measurements it is therefore necessary to use optical filters with an optical density of 5 or higher to prevent even traces of the excitation light to reach the detector. Otherwise the sensitivity and the dynamic range of the fluorimeter would be decreased.

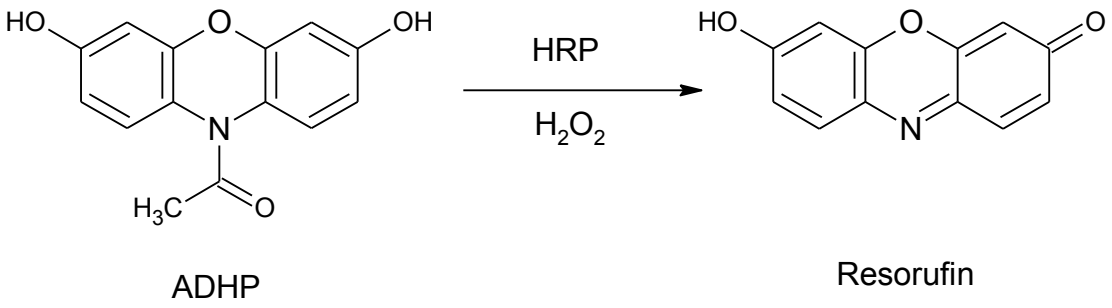

Figure 3.19 ADHP substrate reaction. Non-fluorescent ADHP compound is converted by the action of HRP enzyme in the presence of H2O2 into highly fluorescent resorufin dye.

In a typical configuration the short pass excitation filter is used to block the higher wavelength part of the excitation light, which would otherwise pass through the emission filter and reach the detector. Similarly, the emission filter blocks the filtered excitation light to reach the detector. Obviously, the excitation filter cut-off wavelength should be lower than the emission filter cut-off wavelength. Ideally the excitation light source should have maximum radiation energy close to the 571 nm and the emission filter allows as much as possible of the emission light to reach the detector.

3.3.1 Excitation light source selection

During the initial testing both the Green LED (SSL-LX5093SGC/B, Lumex) and green DPSS laser (DJ532-10, Thorlabs) were evaluated as the possible excitation sources for the resorufin. Green high intensity LEDs with peak wavelengths between 525 nm to 565 nm are available on the market. Typical spectral width for these LEDs is about 40 nm to 50 nm, which requires use of the excitation filter to limit the excitation spectrum at the upper end.

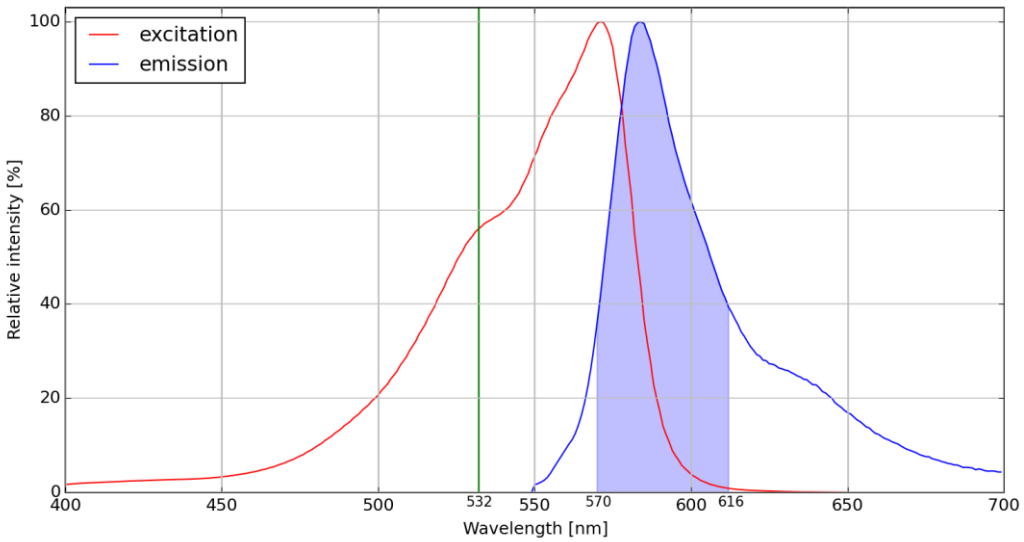

Figure 3.20 The excitation and emission spectrum of resorufin fluorescent dye with overlaid emission filter passband (blue area) and excitation laser line (green line)

During the initial test it was found very problematic to focus the LED generated beam precisely without significant loss of intensity. The tested packages were T-1¾ (5 mm diameter through-hole LED) with standard hemispherical ends and also with the flat end. The other advantages of the LED source are low price, simple driver circuit, longer lifetime, and good temperature stability.

The other option was to use a green diode-pumped solid state (DPSS) laser. With a 532 nm spectral line the absorption by the fluorophore is only 56% of the maximum. This disadvantage is more than compensated by the very intense light compared to LED. Also, thanks to very narrow spectrum and higher distance from the emission spectra, the use of the excitation filter is not necessary. On the other side laser source is more expensive, requires precise driver, and has a relatively narrow operating temperature range.

After considering the advantages and disadvantages for both the LED and laser solutions, the 10 mW green DPSS laser (DJ532-10, Thorlabs) was selected as the excitation light source for the fluorimeter design.

3.3.2 Detector selection

Many fluorimeters use the photomultiplier tube (PMT) or the photodiode as the detector. PMTs are special sort of vacuum tubes with high internal gain (up to several millions). Photomultiplier tubes are used in application which require high sensitivity and low noise

operation. Single photon counting mode can be used if required. PMTs are not affected by the Johnson (thermal) noise, which is another great advantage. The operation of PMTs requires using of high voltages (low KV range), they are generally sensitive to electrostatic and magnetic fields. Long term stability is affected by the diffusion of the helium from the surrounding atmosphere into the tube through the glass walls. The price is relatively high (hundreds to thousands €).

Photodiodes have generally lower sensitivity compared to PMTs, they have no internal gain (with the exception of avalanche photodiodes) and are affected by thermal noise. Photodiodes are low voltage devices, small physical size and insensitivity to electromagnetic interference makes them easy to integrate into portable devices. The photodiodes are very cheap when compared to prices of photomultiplier tubes.

Because the volume of the measured sample is relatively high (about 10 µl) and the concentration of the resorufin in a typical assay sample is also reasonably high (nM to µM range) the selection of the photomultiplier tube as the detector for this application is not justified. The use of difficult to integrate, environmentally sensitive and expensive device is not outweighed by the requirement for higher sensitivity.

Use of the silicon photodiode has proven to be satisfactory for this application during the initial tests. The preferred type should have radiant sensitive area of several square millimeters, high quantum yield, low capacitance, low noise equivalent power and metal housing. After reviewing datasheets of several potentially suitable photodiodes (S1223, OSD5-5T, BPX61, BPW21R) the BPX61 and S1223 types have very low capacitance, dark current and good noise performance. The BPX61 type (OSRAM) was selected as the fluorimeter detector because of better pricing and availability while the performance is similar to the S1223 type.

The traditional way ELISA reactions take place in the wells of a microtiter plate. For a flow-through setup a kind of "flow cell" is needed. In this application the transparent PVC capillary tubes (20 µl, SC-Sanguia Counting 100024) normally used for the blood sampling. The capillary has the outer diameter 1.8 mm, the inner diameter 0.9 mm and length approximately 27 mm. It is inside this capillary where the resorufin fluorophore will be produced by the oxidation of ADHP catalyzed by HRP enzyme. The fluorimeter should be therefore adapted to excite and sense the emission light from this cylindrical shaped space.

3.3.3 Fluorimeter configuration

Fluorimeters are usually constructed is such way that the emission sensing axis is at the right angle with the excitation beam axis. This minimizes the amount of excitation light entering the sensing path thus improving the signal to noise ratio. The fluorimeter of the ELISA analyzer uses the same configuration. The cross section of the fluorimeter with the depicted excitation and emission optical pathways is shown on the Figure 3.21. The fluorimeter uses construction components of the ½ inch lens tube system marketed by Thorlabs.

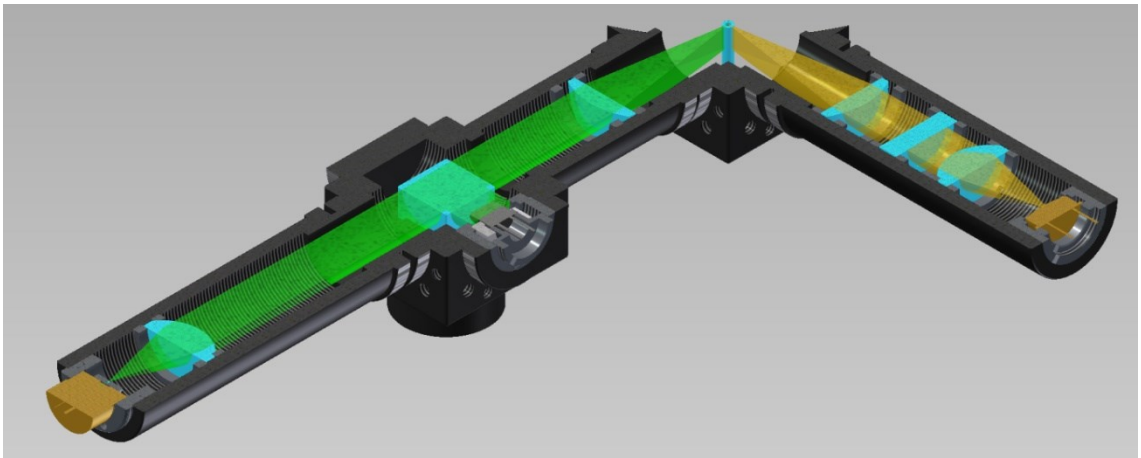

Figure 3.21 Cross section of the fluorimeter showing the excitation beam (green) and emission pathway (yellow)

3.3.4 Excitation optical path description

The DJ532-10 laser output beam diameter is approximately 50-60 µm. The beam is first expanded using the sapphire ball lens (diameter 0.5 mm, type 46117, Edmund Optics) and collimated into parallel beam using the plano-convex lens (LA1540, Thorlabs). The beam is then passed through the 50:50 beam splitter (BS010, Thorlabs) which diverts the portion of the beam to the feedback photodiode (BPW34, Vishay) of the laser driver circuit. The other portion of the excitation beam is focused by the cylindrical lens (type 46194, Edmund Optics) to the center of the modified cage cube (SC6W, Thorlabs) where the capillary will be positioned by the rotary capillary holder. The rotary holder is a motor powered double disc which has eight capillaries mounted on its circumference. The cylindrical surface (with the diameter equal to the diameter of the rotary capillary holder) was milled at the diagonal plane of the cube cage, which allows the capillary to be aligned

with the center of the cube and the focal line of the excitation beam (Figure 3.22). The length of the capillary interior which is illuminated by the excitation beam is about 8 mm.

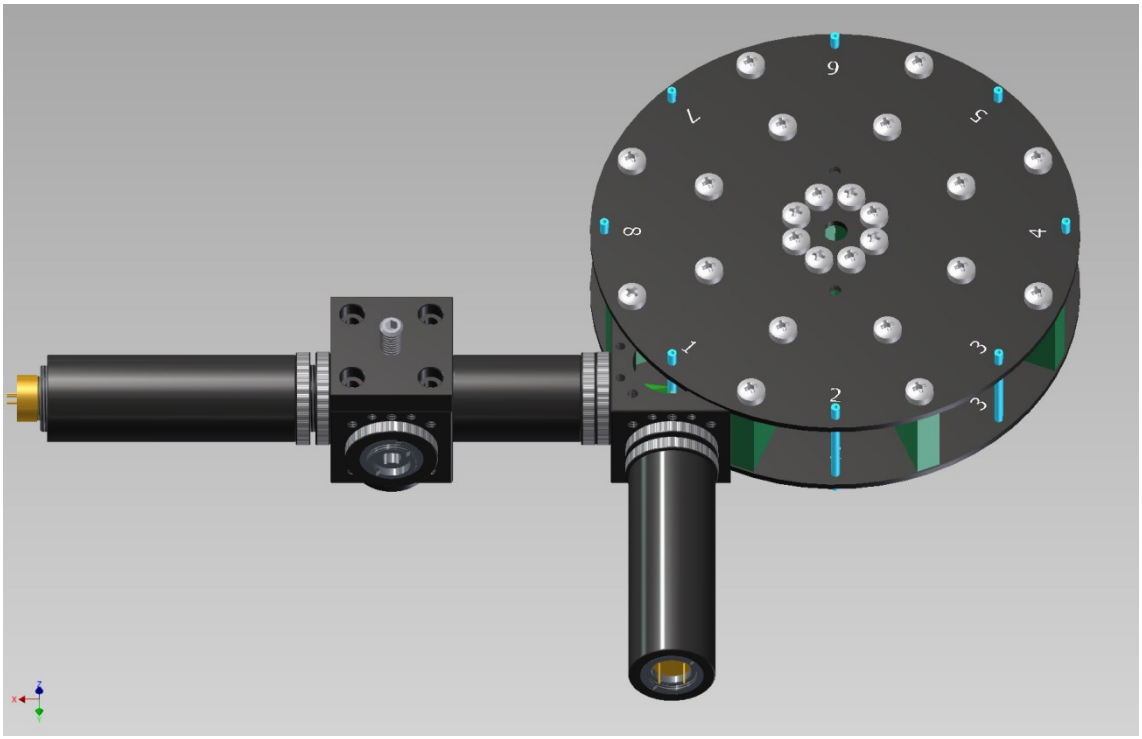

Figure 3.22 The Fluirimeter assembly including the rotary holder with capillaries (blue color)

3.3.5 Emission optical path description

A portion of the resorufin emitted light is collimated into a parallel beam using the cylindrical lens (type 46194, Edmund Optics), which is passed through the suitable optical filter (type 67020, Edmund Optics). The filter is a bandpass filter with center line 591.5 nm, bandwidth of 43 nm and OD >6 blocking in the stopband. Although the use of longpass filter would allow emission light with longer wavelengths to reach the detector and slightly increase the sensitivity, the bandpass filter provides better signal to noise ratio by filtering out the stray light with longer wavelengths. The transmission profile of the filter is shown on the Figure 3.23. The filtered light is focused by the plano-convex lens (LA1540, Thorlabs) to the radiant sensitive area of the photodiode (BPX61).

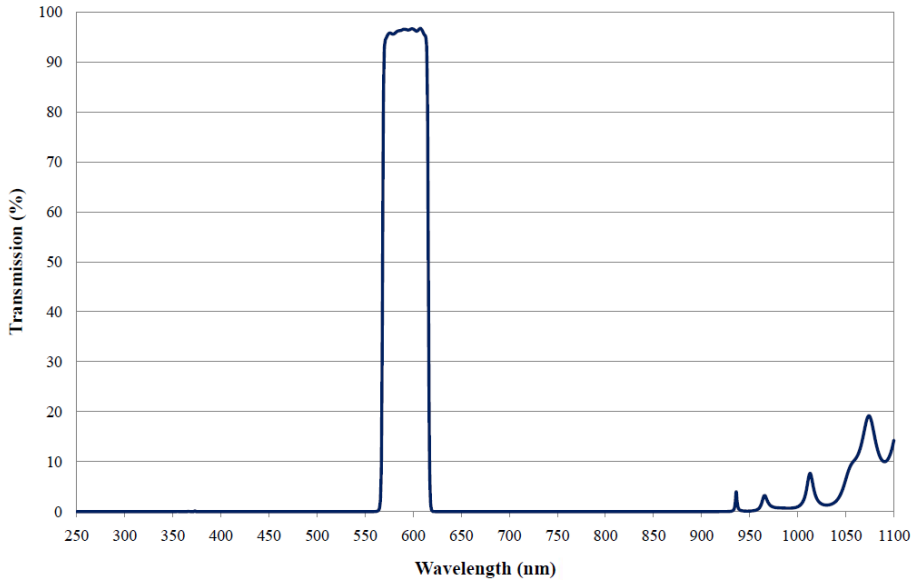

Figure 3.23 Transmission profile of the bandpass filter, type 67020 (Edmund Optics, center wavelength 591,5nm, bandwidth 43nm)

3.3.6 DPSS Laser driver circuit

It is important to keep the excitation light optical power constant in order to provide stable, reproducible measurements. If the required dynamic range of the fluorimeter is four orders of magnitude, the excitation power density must not fluctuate more than 0.01%. Used DPSS laser contains the monitoring photodiode. This photodiode senses the optical power of the 808 nm pumping laser diode and not the power of the 532 nm output beam produced by the Nd:YVO_4 and KTP crystals. The performance of those crystals is temperature dependent. Stabilizing the pumping diode optical power therefore does not provide stable output power of the green light. The heat dissipated by the pumping diode heats the crystals and the output power will have significant drift. It was necessary to use a separate photodiode sensing the portion of the excitation beam to stabilize the output optical power.

The diver circuit is based on a dedicated integrated circuit (iC-WKN, iC-Haus). The functional schematic is shown on the Figure 3.24. iC-WKN is designed to drive laser diodes in the continuous mode. It contains multiple protection circuits for the laser diode and requires only a few external components to operate. The adjustable resistor **RP1** serves for adjusting the laser output power (up to 10mW). The laser beam intensity is sensed by the **PD1** photodiode ad is fed back to the driver **U2**. All capacitors in this circuit work like bypass or filtration capacitors. The input voltage in the range 3 to 5V is

connected to XC5. The current consumption is in the range 120 to 200 mA depending on the adjusted optical output power.

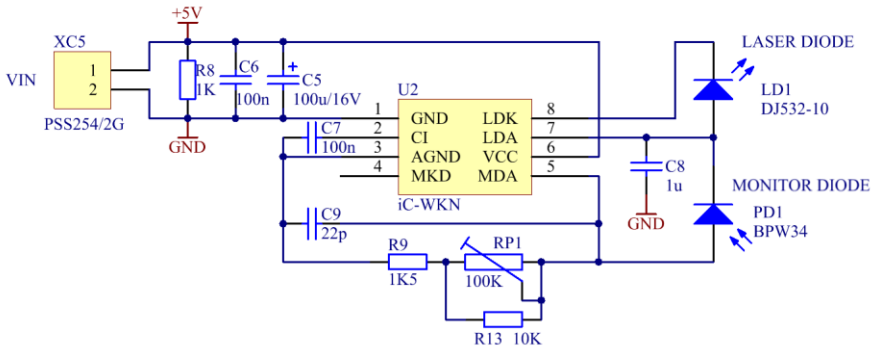

Figure 3.24 Laser diode driving circuit schematic

3.3.7 The photodiode front end amplifier

The fluorimeter photodiode usually needs to measure very low light intensities. It requires an amplifier with the gain of several millions to provide voltage suitable to interface to analog to digital converter (ADC). The photodiode BPX61 works in photovoltaic mode and it is connected to the transimpedance amplifier, which provides many performance advantages. The schematic of the photodiode amplifier is shown on Figure 3.25. The zero bias means no dark current, the internal diode capacitance stays at constant (zero) potential, therefore its influence on the detector speed is minimal. The **R2** defines the gain of the transimpedance amplifier, which is 10^7 in this case. **C1** limits the bandwidth of the detector to reduce the noise. Another noise reduction occurs at the output filter **R1 C2**. The corner frequency is set to 1.59 KHz. Operational amplifier **U1** (LTC6244, Linear Technology) is a low noise CMOS type with low input bias current. The reference voltage is set to 2.5V and it is generated by the **U3**.

The voltage output vs. incident light characteristic has negative slope. The output voltage is highest at 2.5V in the dark (no light reaching the photodiode) and it is decreasing with the increasing light level. The minimum output voltage (maximum light reaching the photodiode) is defined by the low saturation voltage of the **U1**, which is about 25 mV. This negative slope was necessary to use in order to interface to the ADC in the environment with the unsymmetrical power supply.

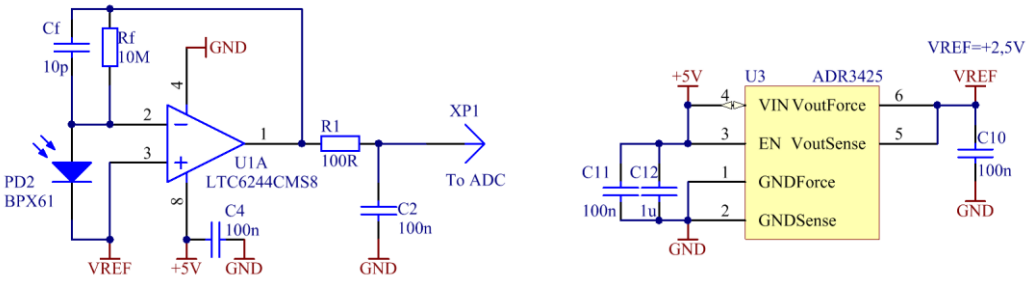

Figure 3.25 The functional schematic of the photodiode amplifier

3.3.8 The Analog to Digital (ADC) interface

The functional schematic of the ADC part is shown in Figure 3.26. The AD7794 AD converter is configured in the differential input mode, internal gain set to unity and update rate to maximum (470Hz). According to AD7794 datasheet, the converter has effective noise free resolution 16 bits in this configuration. Using only one half of the input differential range (+2.5V down to 0V) the final noise free resolution is 15 bits. And the ADC input voltage resolution will be:

$$V_{1LSB} = \frac{V_{ref}}{2^{15}} = 76.3\mu V \qquad (3.1)$$

Where:

V_{ref} is the ADC reference voltage

The maximum photodiode current for the full-scale ADC reading is given by the following equation:

$$I_{PD\max} = \frac{V_{ref}}{R_f} = 250nA \qquad (3.2)$$

Where:

V_{ref} is the ADC reference voltage,

R_f is the transimpedance amplifier feedback resistance

On the other side, the minimum photodiode current needed for ADC reading of one (1LSB) will be calculated as follows:

$$I_{PD\min} = \frac{V_{ref}}{R_f \cdot 2^{15}} = 7.63pA \qquad (3.3)$$

Where:

V_{ref} is the ADC reference voltage,

R_f is the transimpedance amplifier feedback resistance

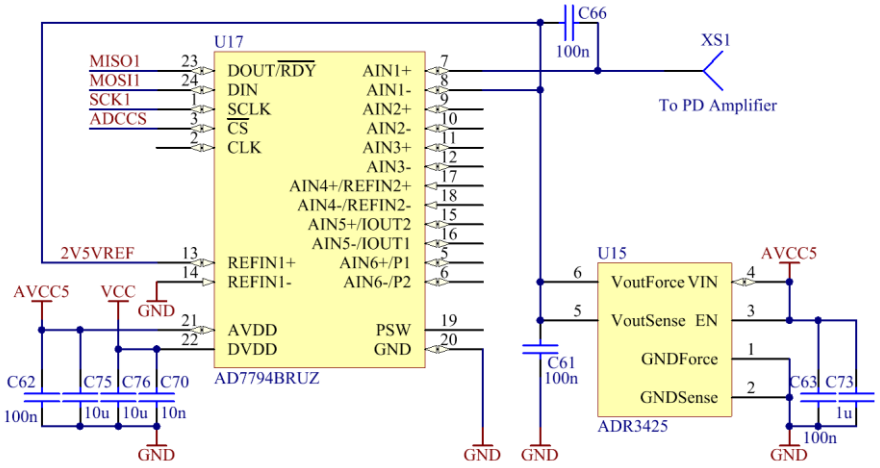

Figure 3.26 The functional schematic of the fluorimeter ADC converter

In order to maintain the noise free resolution of 15 bits, the photodiode and transimpedance amplifier total peak-to-peak noise must be kept below 76.3 µV. Typically noise calcullations use the root-mean-square noise (RMS). The conversion between the peak-to-peak noise and RMS noise uses the fact that both thermal noise and shot noise have the Gaussian probability distribution. By specifiing the noise amplitude interval normalized to the standard deviation σ it is possible to estimate the probability of occurrence of the amplitude. Using the Gauss error function for the 6.6σ interval provides the 99.9% probability of occurrence [20]. In this way the 76.3 µV peak to peak noise corresponds to the 11.56 µV RMS noise using the factor 6.6. The following part the electrical noise background of the fluorimeter will be investigated.

3.3.9 The fluorimeter detector and amplifier noise estimation

The photodiode noise is calculated as the sum of three components: the thermal (Johnson) noise i_{th} of the photodiode internal shunt resistor R_{sh} , the dark current shot noise i_{sd} and the photocurrent shot noise i_{sl} [21]. The internal shunt resistor R_{sh} of the BPX61 photodiode was calculated to be 2 GΩ using the dark current and the reverse bias voltage values provided in the datasheet. Because the photodiode works in the photovoltaic mode, the bias voltage is zero (see Figure 3.25), the dark current is also zero and the dark current

shot noise will not contribute to the total photodiode noise. In the following noise calculations the temperature of 300 K is assumed. The thermal noise density of the photodiode is calculated as follows:

$$i_{th} = \sqrt{\frac{4K_bT}{R_{sh}}} = 2.88fAHz^{-1/2} \quad (3.4)$$

Where:

K_b is the Boltzmann constant,

R_{sh} is the photodiode shunt resistance,

T is the temperature in K

The noise density due to photocurrent shot noise is calculated according to equation (3.5). It is calculated for the minimum photocurrent because this case will be affected the most by noise and the signal to noise ratio (SNR) will be the worst.

$$i_{sl} = \sqrt{2qI_{PD\min}} = 1.56fAHz^{-1/2} \quad (3.5)$$

Where:

q is the elementary electric charge,

I_{PDmin} is the photocurrent corresponding to the one quantization step of the ADC

Total RMS noise at the output of the TIA amplifier originating from the photodiode only and contained within defined bandwith BW is calculated using the following formula:

$$V_{pd} = R_f\sqrt{(i_{th}^2 + i_{sd}^2 + i_{sl}^2) \cdot BW} = 1.64\mu V \quad (3.6)$$

Where:

i_{th} is the thermal photodiode noise density,

I_{sd} is the photodiode noise density originating from the dark current

I_{sl} is the photodiode noise density originating from the photocurrent

R_f is the transimpedance amplifier feedback resistance

BW is the noise bandwidth

The corresponding bandwidth is defined as the brick-wall equivalent of the TIA low pass filter and it is calculated as follows:

$$BW = \frac{1}{4R_fC_f} = 2500Hz \quad (3.7)$$

Where:

R_f is the transimpedance amplifier feedback resistance

C_f is the transimpedance amplifier feedback capacitance

This way, the total noise voltage originating from the photodiode is estimated to be 1.64 μV (RMS value). TIA limits the photodiode noise bandwidth to 2.5 KHz.

The TIA noise must be also evaluated. Usually its noise contribution is the most significant. The Amplifier noise can be separated into three components: the thermal noise of the feedback resistor R_f, the input voltage noise of the operational amplifier and the input current noise of the operational amplifier [22]. The photodiode internal capacitance **Ci** (72 pF for BPX61 and zero bias) at in input of the amplifier causes the amplifier noise gain peaking at higher frequencies which has negative impact on the noise performance of the detector. The noise gain begins to increase at the F_z frequency:

$$F_z = \frac{1}{2\pi R_f (C_i + C_f)} = 194Hz \qquad (3.8)$$

Where:

R_f is the transimpedance amplifier feedback resistance

C_f is the transimpedance amplifier feedback capacitance

C_i is the photodiode internal capacitance

The thermal noise originating from R_f has limited bandwidth with corner frequency F_p:

$$F_p = \frac{1}{2\pi R_f C_f} = 1.59KHz \qquad (3.9)$$

Where:

R_f is the transimpedance amplifier feedback resistance

C_f is the transimpedance amplifier feedback capacitance

The total thermal noise contained within this bandwith will be:

$$V_{th} = \sqrt{4K_b T R_f \frac{\pi . GBW}{2} \cdot \frac{F_p}{F_p + GBW}} = 20.35\mu V \qquad (3.10)$$

Where:

K_b is the Boltzmann constant,

R_f is the transimpedance amplifier feedback resistance**,**

T is the temperature in K,

GBW is the gain bandwidth product of used operational amplifier (50 MHz for LTC6244)

The noise contribution by the operational amplifier input current noise is calculated according the following formula:

$$V_{in} = i_n R_f \sqrt{\frac{\pi \cdot GBW}{2} \cdot \frac{F_p}{F_p + GBW}} = 0.28\mu V \qquad (3.11)$$

Where:

i_n is the input noise current density of used operational amplifier (0.56 fA/√Hz for LTC6244),

R_f is the transimpedance amplifier feedback resistance,

GBW is the gain bandwidth product of used operational amplifier (50 MHz for LTC6244)

The noise contribution by the operational amplifier input voltage noise is calculated according the following formula:

$$V_{en} = e_n \sqrt{\frac{\pi \cdot GBW}{2} \cdot \frac{F_p(GBW + F_z)}{F_z(GBW + F_p)}} = 202.92\mu V \qquad (3.12)$$

Where:

e_n is the input noise voltage density of used operational amplifier (8 nV/√Hz for LTC6244),

R_f is the transimpedance amplifier feedback resistance,

GBW is the gain bandwidth product of used operational amplifier (50 MHz for LTC6244)

It can be seen that this last noise contribution alone is one order of magnitude higher than other noise sources. This is caused by high bandwidth of the LTC6244 amplifier. To reduce this noise contribution, a simple RC low pass filter needs to be connected to the output of the transimpedance amplifier. This filter is represented on the Figure 3.25 by the components **R1** and **C2**. The corner frequency of this filter will be:

$$F_{lp} = \frac{1}{2\pi R_1 C_2} = 15.9KHz \qquad (3.13)$$

The noise contribution by the operational amplifier input voltage noise after the low pass filter is calculated according the following formula:

$$V_{enf} = e_n \sqrt{\frac{1}{4R_1C_2}} = 1.26\mu V \qquad (3.14)$$

Where:

e_n is the input noise voltage density of used operational amplifier (8 nV/√Hz for LTC6244),

The ***Vin*** and ***Vth*** noise component contributions will be unaffected by the filter, because their bandwidth is within the passband of the output filter. The total noise of the transimpedance amplifier will be calculates as a square root of the sum of squared components:

$$V_{amp} = \sqrt{V_{enf}^2 + V_{in}^2 + V_{th}^2} = 20.39\mu V \qquad (3.15)$$

This calculation does not include the flicker noise of the operational amplifier and the thermal noise of the otput filter resistor **R1**. Their contribution is neglible. The Figure 3.27 shows the PSPICE noise simulation of the TIA with and without the output filter. The total noise density and integrated noise are shown. The simulated total noise of the TIA is 22.14 µV, which is in good agreement with the previous calculations.

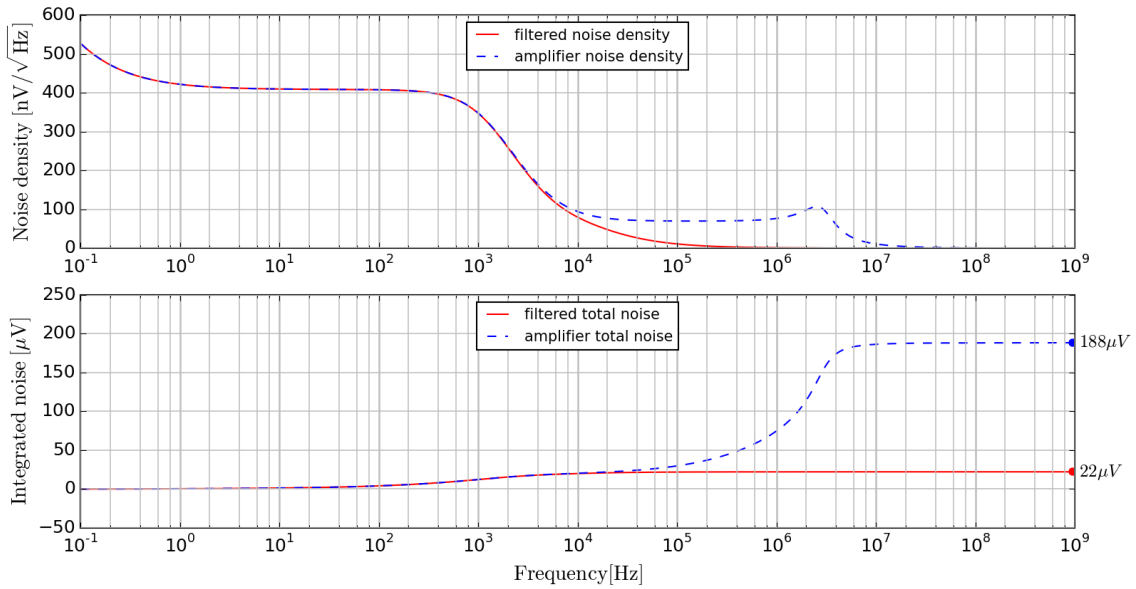

Figure 3.27 The simulated noise density and integrated noise of the transimpedance amplifier

The ADC quantization noise also significantly contributes to the overall noise performance. It is calculated according the following equation [23]:

$$V_{nADC} = \frac{V_{1LSB}}{\sqrt{12}} = 22.03\mu V \quad (3.16)$$

The resulting noise is calculated by combining the total photodiode noise V_{pd}, the total amplifier noise V_{amp} and the ADC quantization noise V_{nADC} together:

$$V_n = \sqrt{V_{amp}^2 + V_{pd}^2 + V_{nADC}^2} = 30.06\mu V \quad (3.17)$$

The total noise is still higher than required RMS maximum 11.56 µV for the flicker-free ADC conversion. A further possibility to decrease noise is to use signal processing in the digital domain. Provided that the signal is almost stationary and the noise is not correlated to the signal the averaging of N ADC samples will reduce the noise by factor $\sqrt{N}$, while the signal remains unaffected [23, 24]. In this case the averaging of 16 samples is used to decrease the noise four times:

$$\bar{V_n} = \frac{V_n}{\sqrt{N}} = 7.52\mu V \quad (3.18)$$

Where N is the number of averages

This resulting RMS noise 7.52 µV should provide the flicker-free digital representation of the photodiode detected emission light intensity.

3.4 Control Unit

There is an obvious need to electrically drive all the active fluidic components of the analyzer and culturing unit in a specific preprogrammed way to enable its proper functionality. During the early testing phase, it is likely that some changes or adjustments in the fluidics and its active components will be made. The control unit should therefore provide sufficient flexibility to support the varying count of the active fluidic components (valves, pumps) or ability to interface sensors with various interfaces. Widely used approach to control such systems is to use a commercially available modular system (e.g. LabVIEW) using the personal computer (PC) as the central control element. Although this may be practical approach for many laboratory experimental setups, the control unit for the analyzer and culturing modules described in this work benefits from the advantages which provides the customized embedded technology. Among the most important is the real-time operation and high degree of integration which provides the

timing precision and reliability required by this application. The additional benefits are independence on the control PC, low power consumption, portability and cost effectiveness. The control unit described here has the ability to control six stepper motors in microstepping mode, 32 solenoid valves and two constant current sources capable of driving low power laser diodes. Additionally, the control unit provides direct interfaces for various sensors (spectrometer head, fluorimetric and photometric sensors, pressure sensor, capacitive sensors, analog voltage inputs and others). The core of the control unit makes use of the 32-bit microcontroller STM32F103ZET [25] which is based on the ARM CORTEX-M3 architecture. Interfacing to potential host system is possible through the USB or serial interfaces. Graphical LCD display and two rotary encoders serve as the user interface. The corresponding printed circuit board (PCB) was designed using four electrical layers and has the size 160 x 200 mm. The whole system is powered by single 12V source. The schematic and layout design were made using the Altium Designer software [26], version 10.391.

3.4.1 Control unit schematics

This section describes the function of important schematic parts of the control unit in more detail. The complete schematics can be found in the Appendix 1.

3.4.1.1 Microcontroller and communication interfaces

The central element of the control unit is the 32-bit microcontroller STM32F103ZET [25] (STMicroelectronics, **U6**) in the 144-pin LQFP package which offers sufficient I/O ports to interface all used peripherals. The controlling program is stored in the internal 512KB nonvolatile memory. The schematic part covering the microcontroller and related system and interface circuitry is shown on the Figure 3.28. The system clock is generated by internal oscillator stabilized by the 8 MHz crystal **X2**. The microcontroller core is clocked by the 72 MHz clock signal internally multiplied and derived from the 8 MHz oscillator. A low frequency oscillator synchronized by **X1** serves as the clock source for internal real-time-clock (RTC). A small lithium battery **BT1** keeps the RTC running during the time the control unit is unpowered. The microcontroller is powered by single supply voltage **VCC** (3.3V). Capacitors **C13-C23**, **C31**, **C37**, **C47**, **C50** and **C51** serve as power supply decoupling. In the normal state the jumpers **J1** and **J2** are switched to ground, so

the microcontroller always boots from the internal flash memory. The control unit contains two USB interfaces. Both are configured as the "USB Device" and support the standard USB2.0 Full Speed. The connection USB1 (**XC2**) serves for sending control commands by the host system, while the USB2 connection (**XC1**) is intended for hi-speed debugging. The control unit also provides two serial interfaces without hardware handshake function supporting maximum transfer rate 115200 Baud. These allow connection of various slave integrated sensors or modules. Alternatively, it can be used for low-speed debugging. CAN bus interface (**XC4**) allows direct interfacing of the CETONI neMESYS [27] syringe pumps.

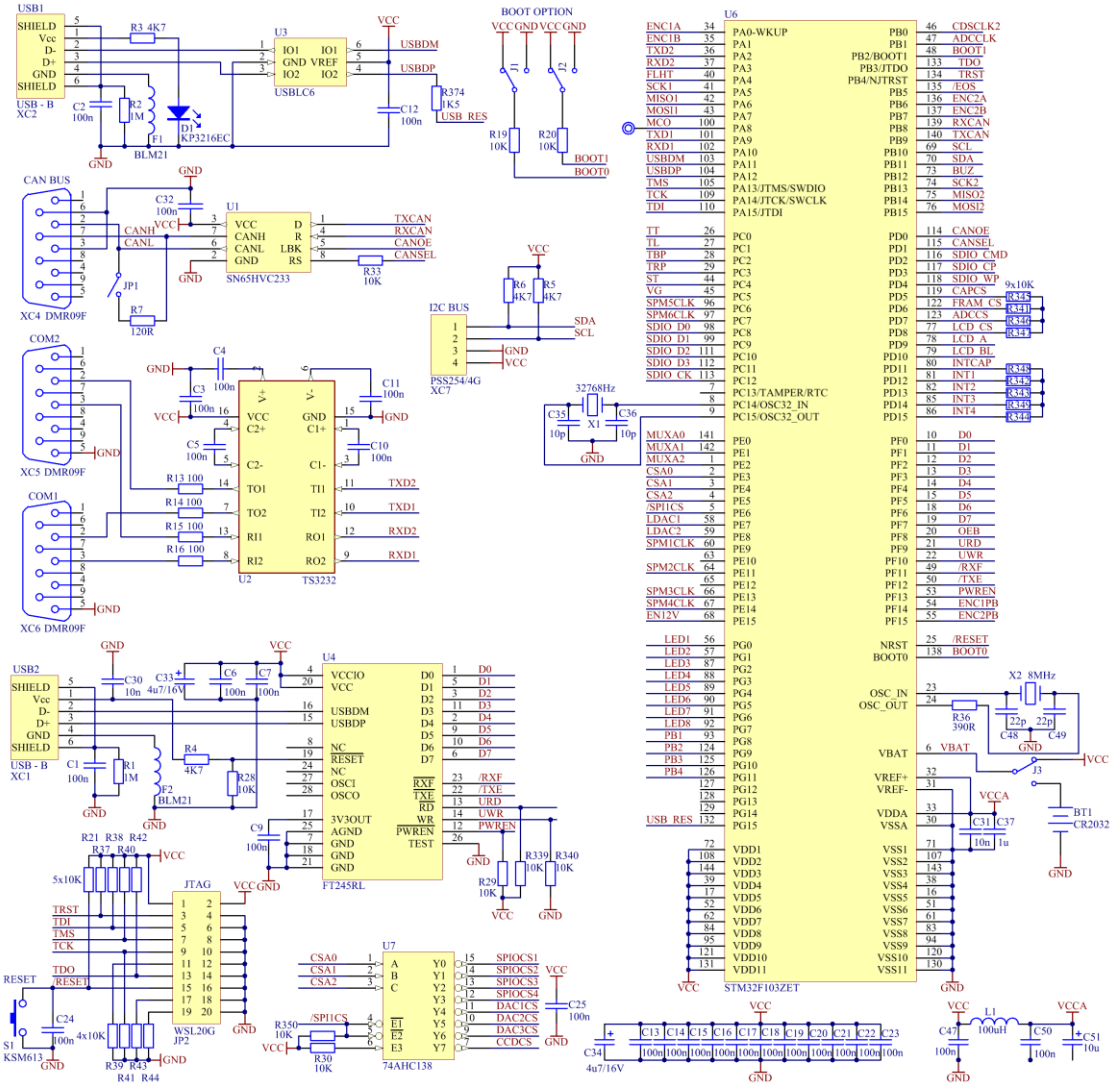

Figure 3.28 Schematic of the microcontroller part and communication interfaces

Additionally, the powered connector with I^{2}C bus is available for connecting low-speed OEM sensor modules. The JTAG interface (**JP2**) allows programming of the internal

memory of the microcontroller and the low-level debugging. The address decoder (**U7**) provides chip select signal demultiplexing for all peripheral chips controlled by the SPI bus.

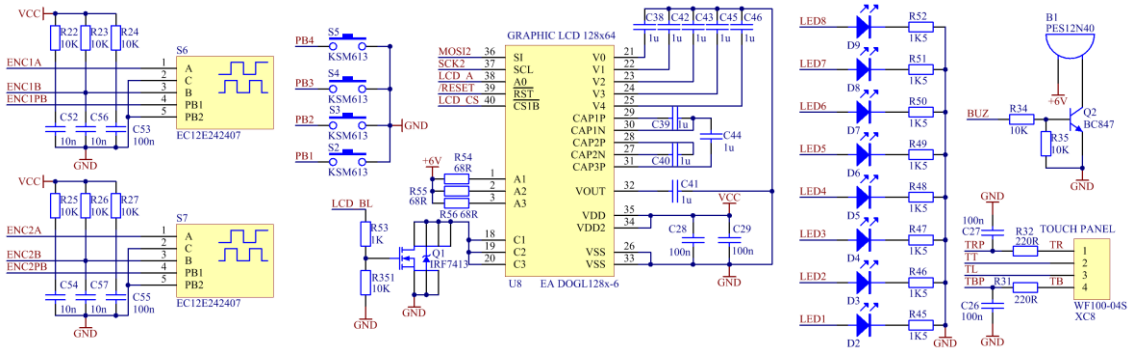

Figure 3.29 Schematic of the human interaction interfaces

3.4.1.2 Human control interfaces

For the interaction with user, the control unit contains a graphical display (**U8**) with resolution of 64 x 128 pixels and resistive touchscreen connected through the connector **XC8**. Display is controlled over a serial SPI bus. The touchscreen is connected directly to the microcontroller and is read using the internal AD converters. Additionally, two rotary encoders (**S6, S7**) simplify the numerical input by the user. Quadrature signals are internally decoded by the microcontroller. Acoustic signalization is provided by small piezo transducer **B1**. Eight directly controlled LEDs (**D2 - D9**) and four pushbuttons (**S2 - S5**) are intended for software debugging purposes. The schematic of the human control interface part is shown on the Figure 3.29.

3.4.1.3 External memory interface

The control unit of the standalone system should have the possibility to store the system configuration and the measurement results on the internal or removable media. For this purpose, the control unit has one Secure Digital (SD) memory card interface and one internal ferroelectric random-access memory (F-RAM, **U5**) for storing up to 32KB of configuration data (Figure 3.30). Both devices can communicate with the microcontroller over the SPI bus. The SD card interface additionally supports faster 4-bit data bus mode.

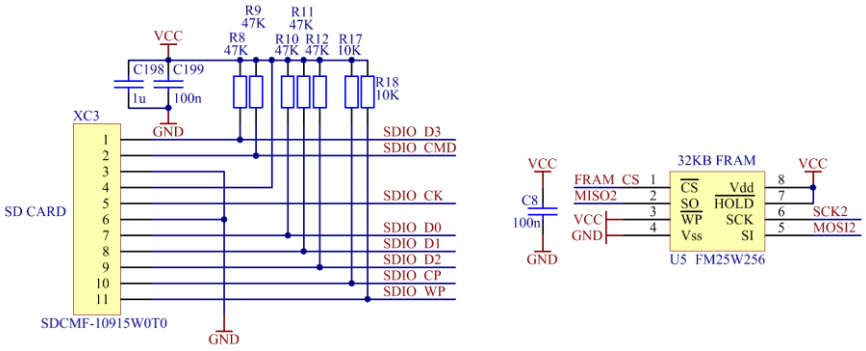

Figure 3.30 Schematic of the memory interfaces

3.4.1.4 Power supply

The control unit is powered by the single DC source of 12V with maximum current 3A. However, typical current consumption is much lower – 1A to 1,5A so the whole system could be powered by larger accumulator if needed. The schematic of the power supply circuitry is shown on the Figure 3.31. Connector **XC30** serves as the 12V power supply input. The fuse **F3** serves as the overcurrent protection, while the diodes **D56** and **D59** provide overvoltage and reverse polarity protection. The 12V power supply for stepper motor drivers **12VMOT** is derived from the main power over the EMI filter (**L3**, **C174**, **C176**, **C183** and **C185**). Powering of the solenoid valves comprises two phases: during the active switching phase the voltage is set to 12V, while during the standby phase (the solenoid valve is in the steady ON state) the driving voltage is lowered to approximately 5V. During the active phase the solenoid power supply **12VSOL** is connected to the main power supply node **VIN** through the EMI filter (**L2**, **C172**, **C173**, **C182** and **C184**) and transistor **Q11**. The signal **EN12V** controls switching of the **Q11**. When the **Q11** is switched off, the **12VSOL** node is powered from the **+6V** source through decoupling diode **D62**. This way the power needed to keep the solenoid valve in the switched-on position is reduced from 240mW to 100mW. In the situation where multiple solenoids are switched on, the power saving is significant. The power supply for all the digital circuitry is provided by the switching step-down regulator **U38**, providing stable voltage output of 3.3V at the node **VCC**. Most of the analog circuitry requires stable +5V power supply with low ripple voltage. This is derived from the main 12V supply using the two-stage regulator. The first stage is switched step down regulator generating output voltage **+6V**. To remove the switching ripple a second linear low-drop regulator **U37** downregulates

the **+6V** input voltage to the required ripple-free +5V output **AVCC5**. The switching regulators operate with high efficiency of 70-90% which helps to save power and reduce thermal losses at higher load currents.

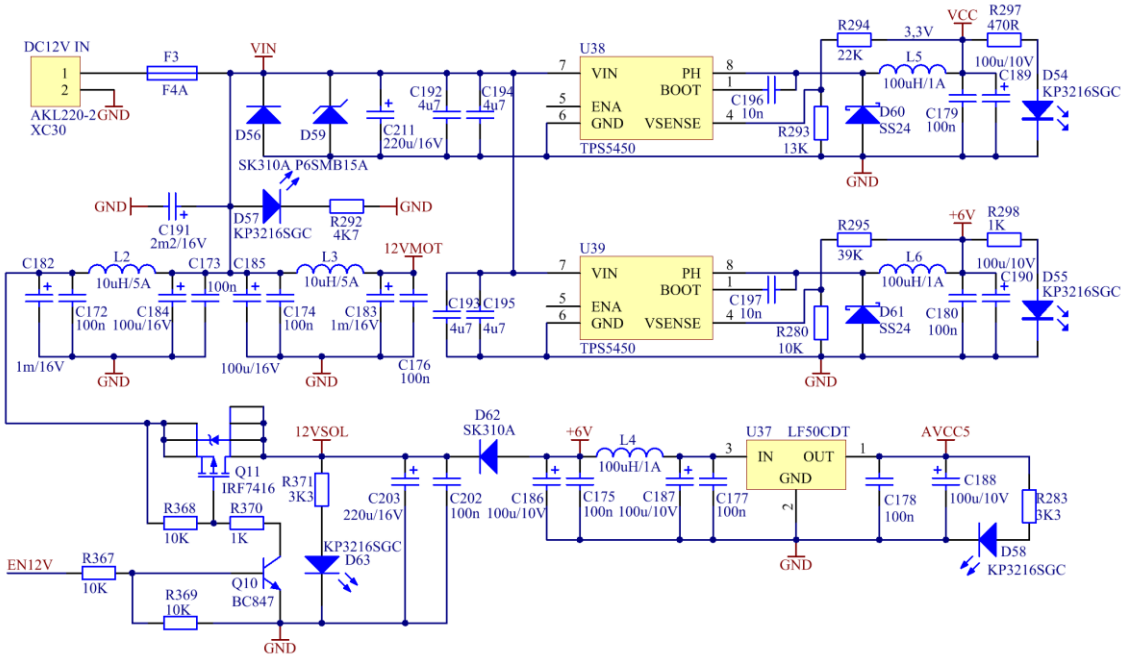

Figure 3.31 Schematic of the control unit power supplies

3.4.1.5 Spectrophotometer and combined photometric/fluorimetric sensor interfaces

The control unit was designed to accommodate a commercial spectrophotometer head based on the Hamamatsu S8378 CMOS linear sensor chip (**U40**, Figure 3.32)[28]. The spectral range is 316nm to 1210nm with a spectral resolution of 3,5nm/pixel. Optical input is provided via SMA connector attached optical cable. After the light exposure the pixels values are sequentially clocked out of the chip to the video output. Because the sensor chip works with 5V level signals, a voltage level converter (**U36**) is necessary to interface to the 3.3V signal level of the microcontroller. The analog pixel voltage is digitized using the 16-bit imaging signal processor (**U35**). The signal processor has three analog input channels. Only one channel is used by the spectrometer part. Additionally, two four-channel combined photometric/fluorimetric sensors can be connected to the control unit. Each channel of the sensor comprises one excitation LED and two photodiodes with integrated preamplifiers. The amplified signals provided by the

photometric channels are connected to the inputs **S1A** – **S8A** of the multiplexer **U41**. Similarly, amplified signals of the fluorimetric channels are connected to the inputs **S1B** – **S8B** of the same integrated circuit. The outputs of the multiplexer are connected to the remaining two channels of the signal processor **U35** where they are digitized and read by the microcontroller. The LED diodes of the sensor are driven by the adjustable constant current source created by operational amplifiers **U32**, **U33**. Those are connected as the differential amplifiers regulating the voltage drop across the output 50 Ω resistor according to the input voltage provided by the D/A converter **U42**.

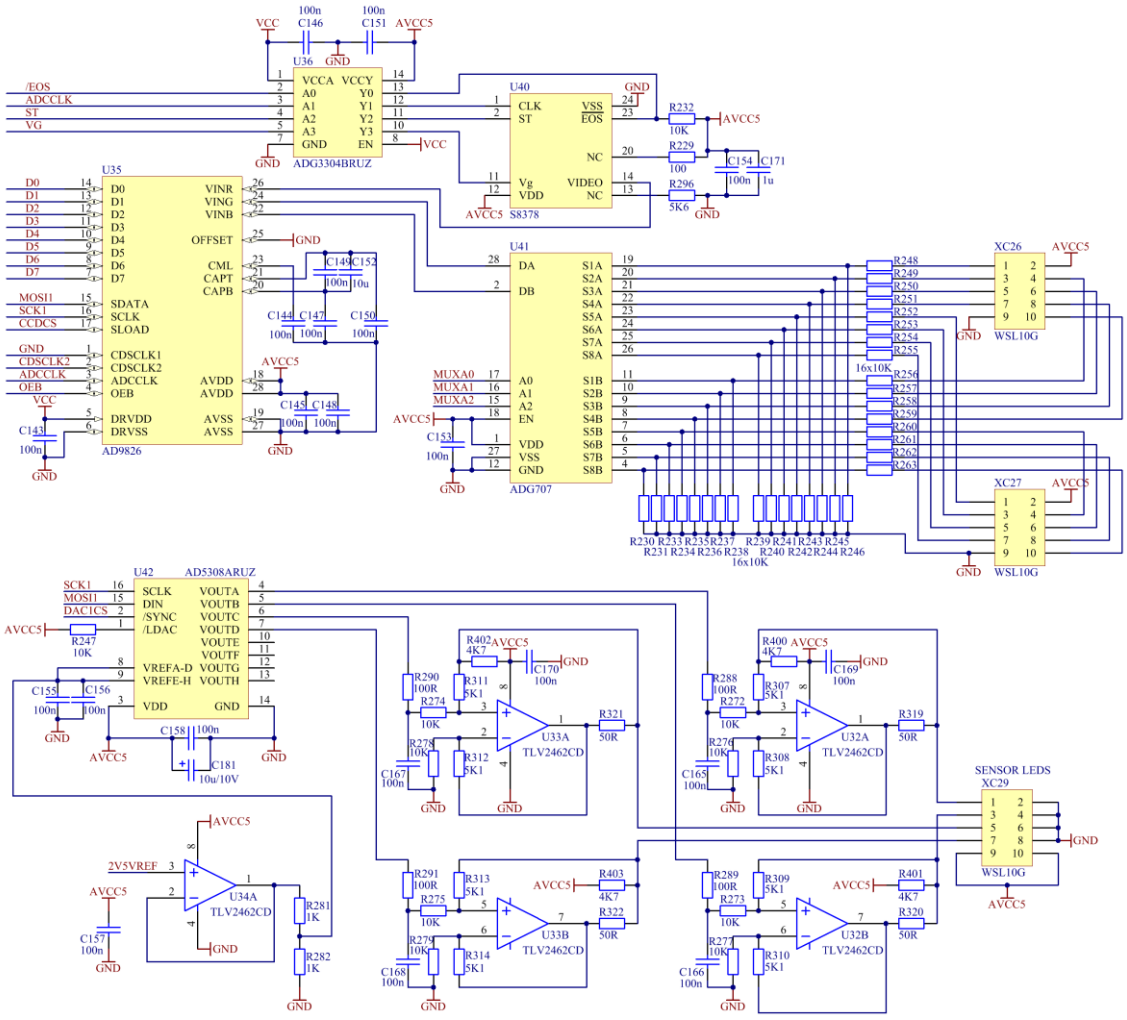

Figure 3.32 Schematic of the spectrometer interface and photometric/fluorimetric sensors interface

The differential amplifier has internal gain of 1/2. Therefore, for the input voltage of 1V, voltage drop of 0.5V across the output 50 Ω resistor will be created, which means 10 mA LED current. The A/D converter **U42** has internal gain 2 and using the reference voltage of 1.25V the maximum output voltage will be 2.5V and the maximum LED current will

be 25 mA. Figure 3.32 shows the constant current sources for the first sensor only. The identical constant current sources for the second sensor are connected to the respective outputs **VOUTE – VOUTH** of the D/A converter **U42**.

3.4.1.6 Solenoid valve drivers

The control unit allows connection up to 32 solenoid valves. Each valve is directly switched by the Darlington transistor inside the integrated circuit **U11**, **U12**. The maximum driver output current is 500 mA. The drivers **U11** and **U12** contain integrated free-wheeling diodes, therefore are capable of switching inductive loads. The Darlington drivers are controlled by the I/O extension circuit **U9** controlled by the SPI bus. The power source **12VSOL** is set to 12V during the switching period, after which it is decreased to 5V to reduce power consumption.

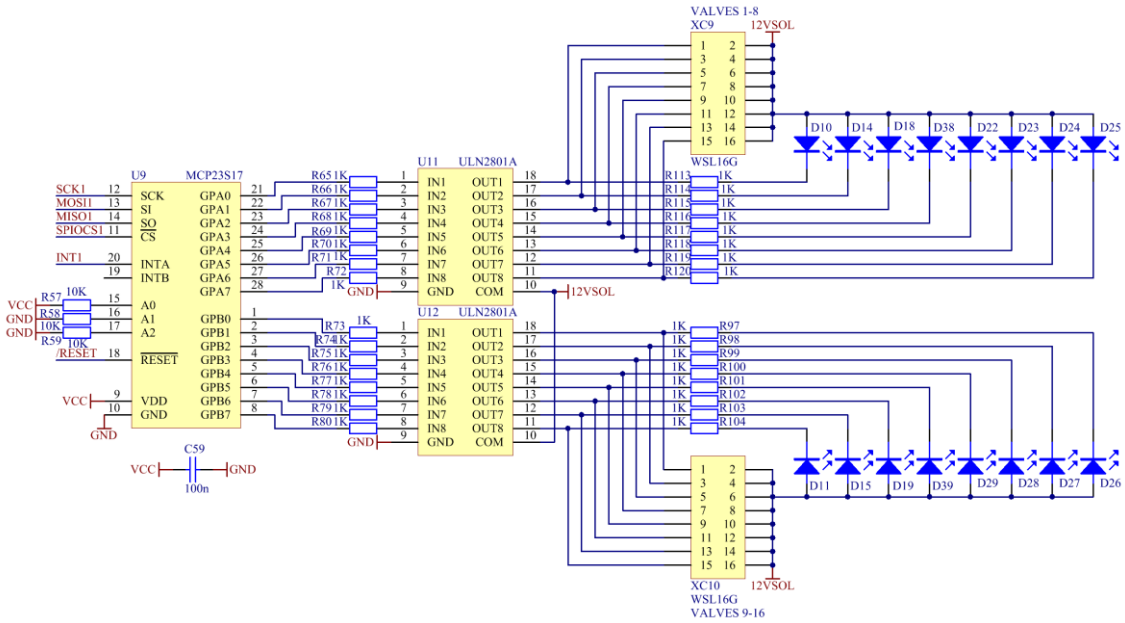

Figure 3.33 Schematic of the solenoid valves driver

This switching is realized using the signal EN12V as described in the power supply section. The indication LED diodes are connected in parallel to the solenoid valves. The driver circuitry for the first 16 valves is shown on the Figure 3.33. The remaining valve drivers are of identical design.

3.4.1.7 Stepper motor drives

The control unit can independently operate up to six two-phase bipolar stepper motors in the microstepping mode. The driver circuitry is based on the integrated stepper motor driver L6208 (STMicroelectronics)[29]. Figure 3.34 shows the driving circuitry for one stepper motor. The I/O expanders **U19**, **U20** and the D/A converters **U21**, **U22** are shared by several stepper motor drivers L6208 (**U24** – **U29**). The L6208 driver has built in pulse width modulated (PWM) output stages with adjustable current limit. L6208 does not natively support microstepping operation. However, by varying the maximum phase current limit for both phases independently the microstepping can be realized. The software implements the microstepping operation with up to 32 microsteps for smooth motor operation. The dynamic current limit setting is provided by the D/A converter **U22**. The decay mode, direction of operation, driver enable signal and stepping mode control signal are controlled by the I/O extension chip **U20**.

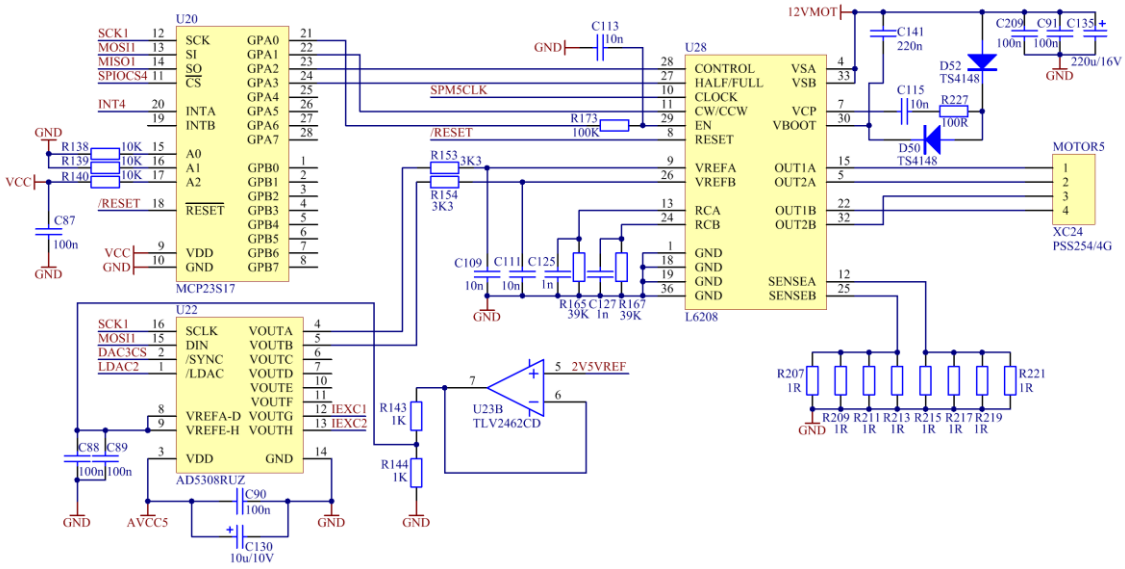

Figure 3.34 Schematic of the stepper motor driver

The stepping clock input is routed directly to the timer module of the microcontroller. The maximum phase current, which is possible to set by the D/A converter is 5A, however the peristaltic pump motors operate with the phase current 3A or less, depending on the motor speed. The maximum supported motor speed is 160 rpm, which corresponds to stepping frequency of 17066 microsteps per second using the 32 microstepping mode.

3.4.1.8 Pressure sensor and capacitive sensors

The integrated differential pressure sensor with the maximum pressure range of 1000KPa (**U18**, MPXV5100DP, NXP Semiconductors)[30] was added to the design of the control unit to help diagnose possible problems in the fluidic part of the system by measuring pressure profiles during fluidic operations. The sensor has linear analog output in the range 0.2 – 4.7 V with a slope of 4.5 mV/KPa. The sensor is connected to the input of 24-bit A/D converter **U17**. For the correct operation with the sensor, the reference inputs 2 are selected by the software (**REFIN2+**, **REFIN2-** inputs of the **U17**). The reference voltage source **U15** is connected to the primary reference inputs of the **U17** (**REFIN1+**, **REFIN1-**) and is needed for the fluorimeter readout, as the A/D converter **U17** is also shared with this peripheral. The schematic part relevant to the pressure sensor and the capacitive sensors is shown on the Figure 3.35.

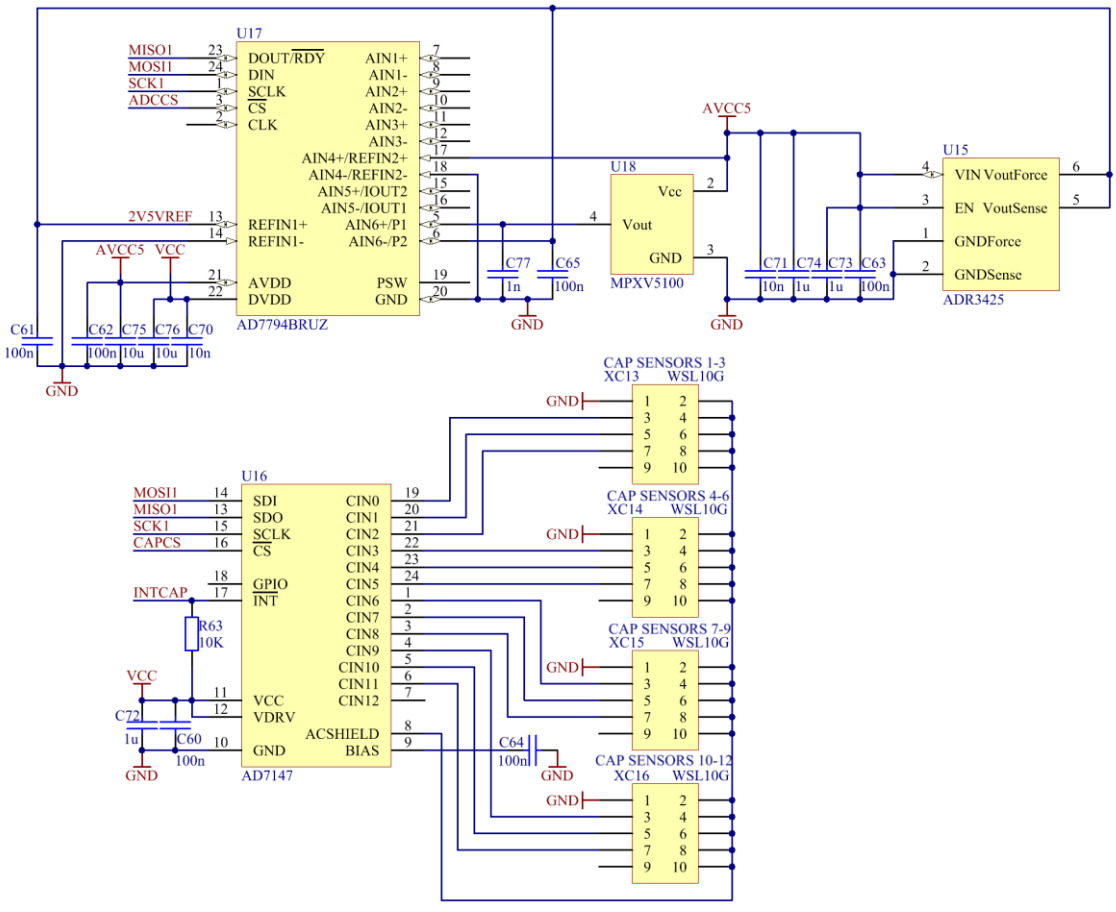

Figure 3.35 Schematic of the pressure sensor and capacitive sensors

The capacitive sensors are intended to be used for detection of a fluid inside the tubing. Total of twelve sensors are supported and the interface uses the capacitance-to-digital

converter chip AD7147 (**U16**, Analog Devices)[31]. The AD7147 chip contains all necessary circuitry for the intended functionality, so the corresponding schematic part contains minimum of external components. The chip communicates with the microcontroller using the SPI bus.

3.4.2 Printed circuit board (PCB) design for the control unit

Design of the PCB is of crucial importance for good overall performance of the control unit. Sensitive analog circuitry intended for measuring of low-level signals are combined with the high-speed digital integrated circuits on the same board. The layout was designed with focus to physically and electrically decouple those parts using proper shielding techniques. Together with the effort to keep the board size as small as possible, the layout design resulted in a four-layer PCB with the dimensions of 200 mm x 160 mm and standard thickness of 1.6 mm. The layer stack details can be found in the Table 3.1. The layout was designed with the minimum track width 0.2 mm and minimum clearance between different tracks 0.2 mm. The minimum plated through-hole diameter is 0.4 mm. The top and bottom layers were protected by the green solder mask. The assembly is combined using mostly surface mounted devices (SMD), but some through-hole components are also used. The prototype board contains 783 components in total, with 566 components assembled on the top side and 217 components on the bottom side. The prototype board was assembled manually. The assembly plan and layout of all layers can be found in the Appendix 2 and Appendix 3. Figure 3.36 shows assembled control unit PCB.

Table 3.1 The layer arrangement of the control unit printed circuit board

Layer	Layer designator	Copper Thickness	Purpose
Top Layer	L1	35µm	top signal layer
Internal Layer 1	L2	35µm	ground plane
Internal Layer 2	L3	35µm	power supply plane
Bottom Layer	L4	35µm	bottom signal layer

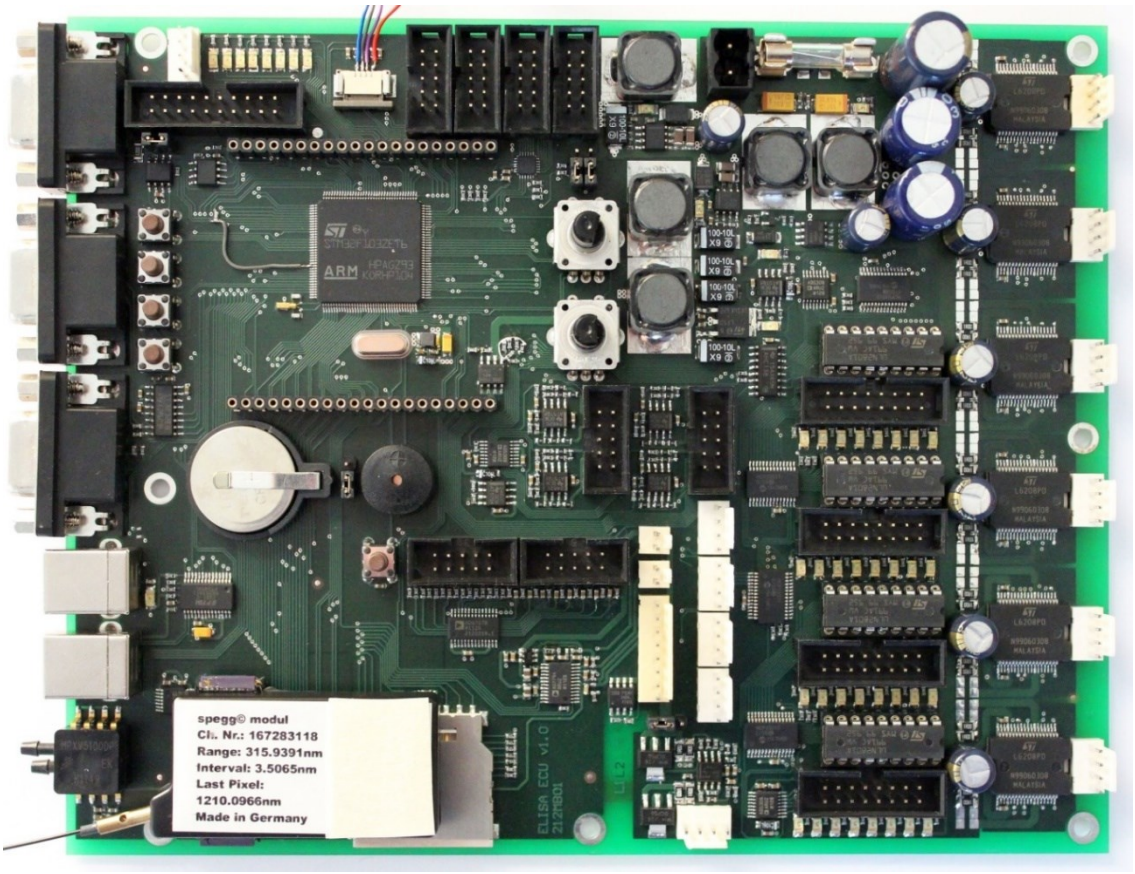

Figure 3.36 Assembled top side of the control unit PCB including the spectrometer module

3.4.3 Control unit embedded code overview

The code for the control unit was written in ANSI C programming language and it was compiled for the ARM CORTEX-M3 architecture. The latest source code version v1.09 contains approximately 10 thousand lines of code excluding used libraries. Additionally, the open source bootloader OpenBLT [32] was used to simplify the firmware update procedure. The embedded code makes use of two external libraries provided by the microcontroller manufacturer: STM32F10x Standard peripheral library v3.5.0 and STM32F10x USB-FS-Device Driver v3.3.0. Additionally a ported code for formatted output (printf.c) is also used [33]. Table 3.2 lists the shortly described source code files. The hierarchical order of the embedded code modules is shown on the Figure 3.37.

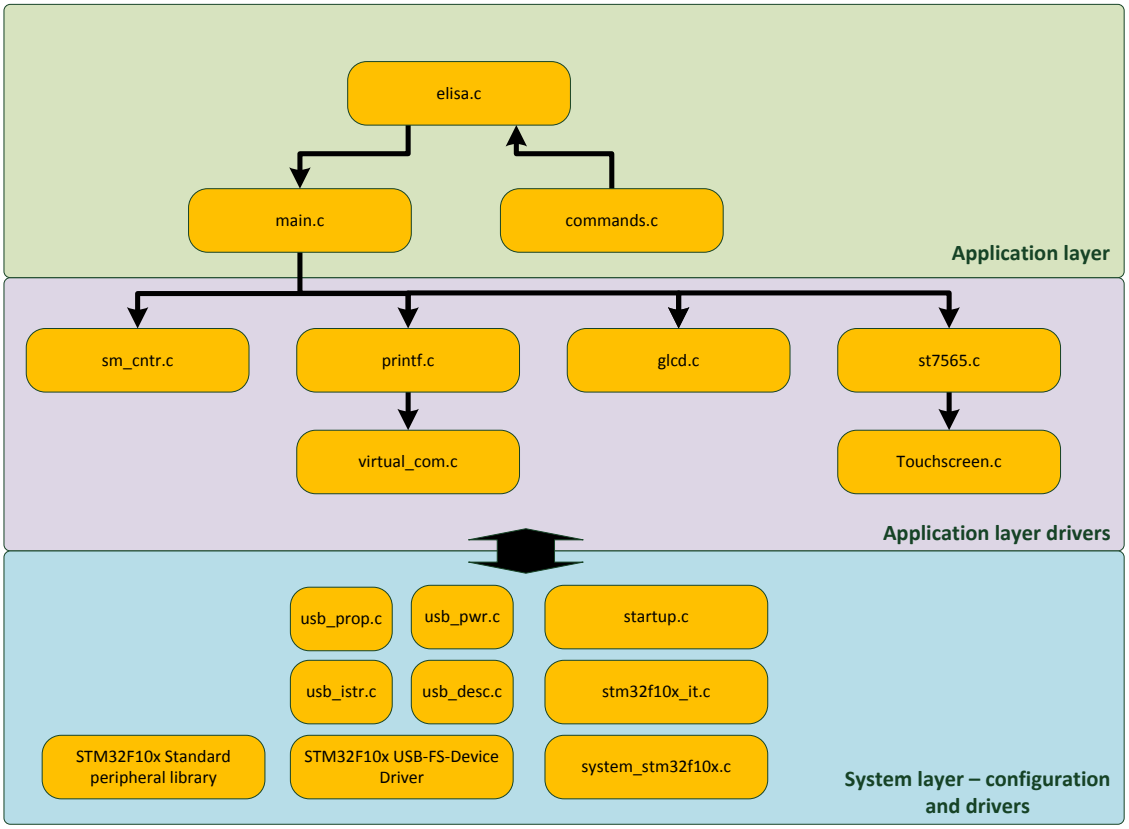

Figure 3.37 the hierarchical order of the control unit code modules

The control unit embedded code in the recent version allows the user to control whole system using a host computer. The communication between the computer and control unit is realized over the USB bus. The control unit behaves as the USB device class with implemented virtual serial communication interface. The host computer uses arbitrary terminal program to facilitate the serial communication. The control unit has implemented a set of commands for controlling and debugging all analyzer and culturing unit functions. Table 3.3 lists the implemented command set including the command parameter description.

Bootloader implementation

The OpenBLT bootloader simplifies the embedded code updating. The bootloader is located at the beginning of the embedded flash memory of the microcontroller. After the microcontroller system reset the bootloader is always executes as the first. The bootloader checks if a valid application code is present in the flash memory using the checksum mechanism. If valid application has been found, the bootloader exits and the application

code starts to execute. Otherwise the bootloader will wait for connection with the host computer in order to program new application code into the flash memory.

Table 3.2 The list of embedded code files with corresponding description

File name	Description
elisa.c	High-level routines for the sandwich ELISA protocol and culturing unit
main.c	Low-lever routines for the peripheral chips and sensors
commands.c	Definition of host control and debug commands
Touchscreen.c	Touchscreen driver code
glcd.c	Graphical LCD driver code
st7565.c	High-level routines for graphical LCD
sm_cntr.c	Stepper motor driver code
printf.c	High-level formatted output routines
startup.c	System initialization code, interrupt vectors definition
stm32f10x_it.c	Interrupt handler routines
system_stm32f10x.c	System routines, system clock management
virtual_com.c	Virtual communication port high-level routines
usb_desc.c	USB descriptor definition
usb_endp.c	USB endpoint routines
usb_istr.c	USB interrupt routines
usb_prop.c	Virtual communication port low-level routines
usb_pwr.c	USB power handling routines
global.h	I/O port definitions, peripheral chip registers definitions
elisa.h	Header file for elisa.c, definition of parameters for fluidics
main.h	Header file for main.c
commands.h	Header file for commands.c
Touchscreen.h	Header file for Touchscreen.c, configuration parameters for the touchscreen
glcd.h	Header file for glcd.c
st7565.h	Header file for glcd.c, configuration parameters for LCD
sm_cntr.h	Header file for sm_cntr.c, configuration parameters for stepper motors
printf.h	Header file for printf.c
font5x7.h	Small font definition for LCD
fontgr.h	Large font definition for LCD
mnlogo.h	Definition of MN logo for LCD
stm32f10x_it.h	Header file for stm32f10x_it.c
stm32f10x_conf.h	STM32F10X Peripheral library configuration file
virtual_com.h	Header file for virtual_com.c, configuration parameters for USB
usb_desc.h	Header file for usb_desc.c, configuration parameters for USB
usb_conf.h	USB endpoint configuration file
usb_istr.h	Header file for usb_istr.c
usb_prop.h	Header file for usb_prop.c
usb_pwr.h	Header file for usb_pwr.c

The uploading utility named “MicroBoot” is part of the OpenBLT project [32], and communicates with the microcontroller over a USB interface. The complied code to be uploaded should be in the Motorola S-record format (*.srec).

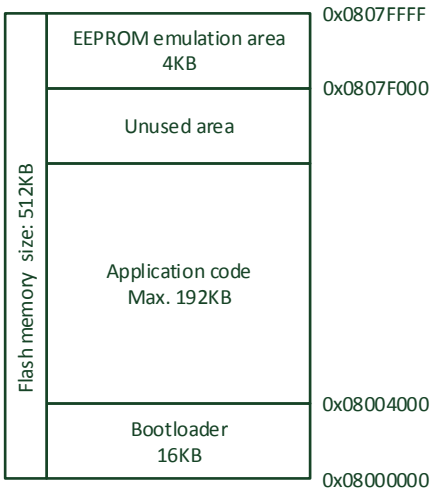

Figure 3.38 the microcontroller flash memory map

The application code must be compiled with starting address of 0x08004000. The microcontroller flash memory map is shown on the Figure 3.38. The 4KB area at the top of the flash is reserved for emulation of EEPROM memory to store the application configuration data. The bootloader can be called directly from the application code using the command “callbootloader”.

Software development tools

For developing the embedded code, the open source development tools were used exclusively. The compiler, debugger and integrated development environment (IDE) was running on personal computer using Windows as the operating system. The compiler was based on the YAGARTO GNU ARM toolchain, version v2.22 [34]. Eclipse was used as the IDE, version 4.2.1 [35]. Additional C/C++ Development Tooling (CDT) plugin was installed into Eclipse IDE. The JTAG interface was used for code debugging. The used OpenOCD debugger tool consists from the software part and the hardware debugging tool [36]. The software part (used version v0.9.0) enables the remote debugging and communicates with the GNU GDB tool. The hardware part (JTAG adapter) is of own design and it is based on the original OpenOCD FTDI2232 interface [37]. The schematic and layout of this JTAG interface can be found in the Appendix 8 and Appendix 9.

Table 3.3 the list of the control commands for the control unit including the syntax and description. The commands are marked in blue and the command parameters are marked in red.

Main menu commands	Parameter: (valid range) - parameter description	Command description
setled *led_index new_state*	led_index: (0 - 7) - selects LED to be controlled	controls the eight debug LEDs
	new_state: (0 or 1) - new LED state 0=OFF, 1=ON	
laser *new_state*	new_state: (0 or 1) - 0= turn laser OFF, 1= turn laser ON	switch ON or OFF the fluorimeter laser
changer home		move the sample changer to home (the first) position
changer move *position*	position: (1 - 8) - new position the changer will move to	move the sample changer to selected position
flmeasure		enter to the fluorescence measurement menu
elisa		enter to the ELISA menu
start pump *pump_index volume speed*	pump_index: (main | wash | br1) - selects the pump. Multiple selection is possible.	Start selected pump(s) to pump selected volume at selected speed
	volume: (-20000 - 20000) - pumped volume in µl, negative number means opposite direction	
	speed: (1 - 5000) - pumping speed in µl/min	
stop pump *pump_index*	pump_index: (main | wash | br1) - selects the pump. Multiple selection is possible.	stop selected pump(s)
valve *valve_index new_state*	valve_index: (0 - 31) - selects the valve to be controlled	controls the 32 solenoid valves
	new_state: (0 or 1) - 0= turn valve OFF, 1= turn valve ON	
pulsevalve *valve_index time*	valve_index: (0 - 31) - selects the valve to be controlled	switches the selected valve ON only for specified time
	time: (1 - 30000) - pulse time in milliseconds	
sample2capillary *sample_idx capilary_idx*	sample_idx: (1 - 8) - index of the sample to be pumped	pump selected sample to selected capillary
	capilary_idx: (1 - 8) - index of the capillary the sample will be pumped into	
empty path *path_idx volume speed*	path_idx: (0 - 19) - index of the fluidic path to be emptied	empty the selected path with defined volume and speed
	volume: (0 - 32000) - pumped volume in µl to empty the fluidic path	
	speed: (1 - 5000) - pumping speed in µl/min	
wash path *path_idx volume speed*	path_idx: (0 - 19) - index of the fluidic path to be emptied	wash the selected path with defined volume and speed
	volume: (0 - 32000) - pumped volume in µl to empty the fluidic path	
	speed: (1 - 5000) - pumping speed in µl/min	
measure sample		measure the fluorescence of current sample
empty all		empty all fluidic paths in the system
wash all		wash all fluidic paths in the system
set rtc *hours minutes seconds*	hours: (0 - 23) - the hour-part of the new time to be set	set new RTC time
	minutes: (0 - 23) - the minutes-part of the new time to be set	
	seconds: (0 - 23) - the seconds-part of the new time to be set	
debug motors		enter to the stepper motors debug menu
end clean		start the end-cleaning procedure
stop cleaning		immediately stop all cleaning services
callbootloader		start the bootloader
callcisservice		enter photometric/fluorimetric sensor measurement service

ELISA menu commands		
exit		exit ELISA menu and return back to the main menu
sequence start *starting_step*	starting_step: (1 - 250) - selects the starting point of the sequence	start execution of the ELISA sequence
sequence stop		immediately stop executing the ELISA sequence
sequence pause		pause the ELISA sequence execution, but finish the current step
sequence continue		resumes the execution of the ELISA sequence
current incubation time *new_time*	new_time: (0 - 30000) - new incubation time in seconds	change the recently running incubation time
substrate time *new_time*	new_time: (0 - 3000) - new incubation time in seconds	change the substrate incubation time
global incubation time *new_time*	new_time: (0 - 30000) - new incubation time in seconds	change the all incubation periods in the sequence except the current time and the substrate time
measurements *repeats*	repeats: (1 - 10) - number of measurement repeats for each channel	set the measurement repeating for each channel, the interval between successive measurements is defined by substrate time command
dosing speed *speed*	speed: (1 - 1000) - new pumping speed to be set	set the pumping speed when pumping liquid to the capillary
sample mode *mode*	mode: (br1 | prepared) - select one of two sampling modes	configure the sampling mode: br1 - take sample directly from the culturing device, prepared - take the sample from the sample container
shortcut *start end*	start: (1 - 255) - the shortcut starting point (as the protocol step) - will be not executed end: (1 - 255) - the shortcut end point (as the protocol step) - will be executed	allows to skip certain steps in the ELISA sequence
manifold speed *speed*	speed: (1 - 1000) - new pumping speed to be set	set the pumping speed when pumping liquid to the manifolds
wash manifold speed *speed*	speed: (1 - 1000) - new pumping speed to be set	set the pumping speed when pumping liquid to the manifolds during washing operations
wash dosing speed *speed*	speed: (1 - 1000) - new pumping speed to be set	set the pumping speed when pumping liquid to the capillary during the washing operations
wash manifold volume *volume*	volume: (1 - 10000) - new washing volume to be set	set the volume for washing the manifold fluidic paths
wash dosing volume *volume*	volume: (1 - 10000) - new washing volume to be set	set the volume for washing the capillary fluidic paths
wash repeat *repeats*	repeats: (1 - 10) - number of washing steps	set the number of washing repeats between the ELISA protocol steps
Bioreactor related commands		
br control *new_state*	new_state: (0 | 1) - turn ON (1) or OFF (0) the perfusion	control the culturing unit bioreactor perfusion
br change medium *volume speed*	volume: (-10000 - 10000) - pumped volume in µl, negative number means opposite direction speed: (1 - 5000) - pumping speed in µl/min	change the medium inside the bioreactor
br sample *volume speed*	volume: (0 - 10000) - pumped volume in µl speed: (1 - 5000) - pumping speed in µl/min	sample the bioreactor medium of defined volume with defined speed
br mix	volume: (0 - 30000) - pumped volume of air in µl speed: (1 - 5000) - pumping speed in µl/min	mix the sampled bioreactor medium using stream of air
br prepare sample		start the bioreactor sample preparation - comprise sampling and mixing operations
br set speed *speed*	speed: (1 - 500) - perfusion speed in µl/min	set new bioreactor perfusion speed

br stop		reset all ongoing bioreactor operations
Cleaner related commands		
cleaner start *starting_step*	starting_step: (1 - 250) - selects the starting point of the sequence	start the cleaning of the fluidics from defined step
cleaner stop		immediatelly stop the clening procedure
cleaner pause		pause the cleaning sequence execution, but finish the current step
cleaner continue		resumes the execution of the cleaning sequence
cleaner shortcut *start end*	start: (1 - 255) - the shortcut start point (as the protocol step) - will be not executed	allows to skip certain steps in the cleaning sequence
	end: (1 - 255) - the shortcut end point (as the protocol step) - will be executed	

3.5 Intermediate summary

As can be seen, the development of the proof of the concept analytical device for the automated flow-through human albumin sandwich ELISA was relatively complex. The performance in the terms of sensitivity and assay time will can be potentially improved to the standard MTP ELISA procedure. This was evaluated during the testing phase and it is described in the later chapters. As with every prototype device some optimization was necessary to reach the required performance level. Figure 3.39 and Figure 3.40 show the finished ELISA analytical unit and the control unit respectively.

Figure 3.39 The automated flow-through ELISA module prototype

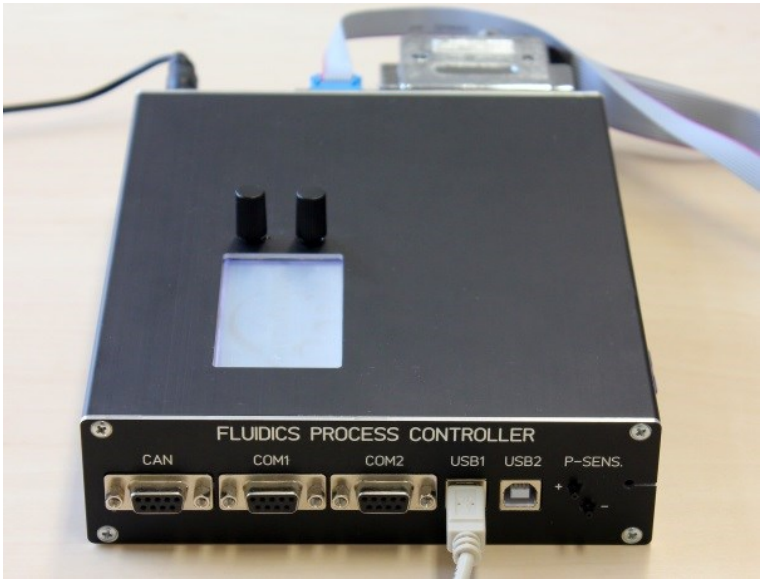

Figure 3.40 The control unit prototype

4 Designing the prototype of automated 3D cell culture device

4.1 Introduction

The use of 3D cell cultures, especially when perfused, are more closely related to *in vivo* conditions, making them potentially a more relevant model than 2D cultures [2]. Recently, two new tools, micro bioreactors and MatriGrid® porous polycarbonate (PC) scaffolds, were developed at Ilmenau University of Technology for 3D culturing of cell [4]. The culture device described here serves as an extension of these two tools to create an incubator environment compatible with a platform having integrated active perfusion and automated medium change. This chapter first briefly describes the MatriGrid® and micro bioreactor devices, following by the design and functionality description of the automated culture unit.

4.2 Required functionality definition of the culture unit

The intended purpose of the culture unit within the automated system is to automate the 3D cell culturing process and provide the fluidic interface for transferring medium samples to the analytical module for further analysis. The basic functionality of the culture unit comprises:

- Compatibility with the existing micro bioreactor and MatriGrid® devices
- Active perfusion of the cell culture
- Automated medium change
- Sampling of the culture medium for the purpose of analysis with the optional possibility of dilution

Aside from the main functionality, some additional properties are also required: The culture unit should be compatible with the incubator environment, it should allow easy handling and maintenance. The culture unit size should be therefore compact. The prototype device may be constructed from commercially available fluidic components to verify the design concept and required functionality.

4.3 The culture platform: Micro Bioreactor and MatriGrid®

The porous polycarbonate scaffolds termed MatriGrid® (Figure 4.1, left side) were previously developed at Ilmenau University of Technology for 3D cell culturing [4]. The scaffold contains up to 187 microcavities in which the cells are cultured. In contrast to 2D cultures, cells grow 3-dimensionally due to the limited space inside these microcavities. The scaffold consists of a rectangular 50 µm thick biocompatible polycarbonate piece with a microstructured seeding area of 5 x 5 mm^2. Porous polycarbonate foils are structured to achieve pore sizes that are necessary for the nutrient supply in active perfusion of 3D cultured cells during bioreactor culture.

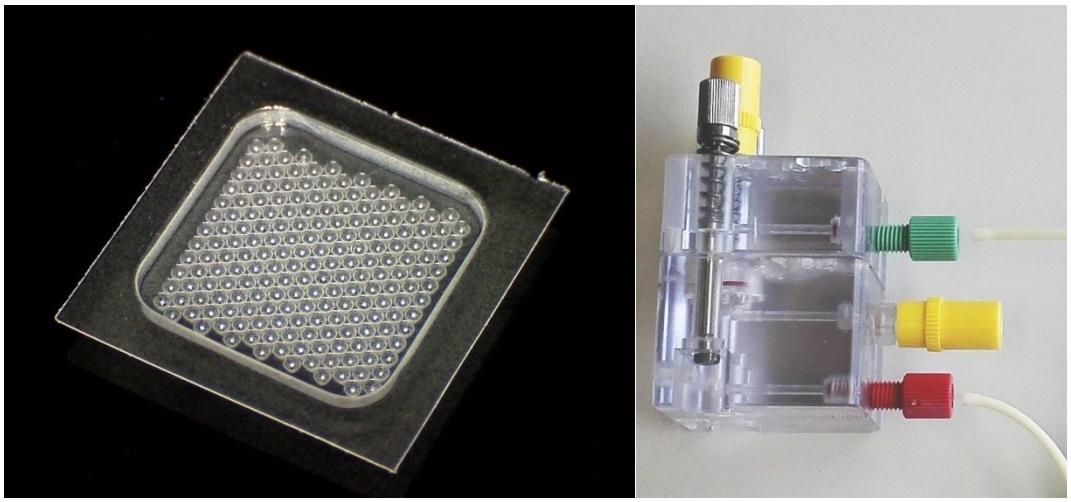

Figure 4.1 The MatriGrid® scaffold (left) and the micro bioreactor (right)

The externally perfused micro bioreactor (Figure 4.1, right side) used for 3D organotypic cell culture has a total fluid volume of 1350 µl and outer dimensions of 41 x 43 x 40 mm (W x D x H) [4]. The housing consists of heat-resistant biocompatible PC components and is therefore autoclavable. Within the bioreactor, two fluid chambers are located above and below the inserted 3D cell carrier MatriGrid® which are connected to an in- and out-flow channel to facilitate medium exchange and sample extraction. The inlet and outlet of the micro bioreactor are connected to medium containing tubes via 1/4-28 UNF flangeless tube connectors (Upchurch Scientific, IDEX Health & Science LLC, USA). De-aeration of the fluid cycle is via an infusion port (B. Braun Melsungen AG, Germany).

4.4 Fluidics design

Based on the requirements defined before the prototype version of the fluidics for the culture unit has been designed (Figure 4.2). It contains five active fluidic components: one peristaltic pump (type 61131.000, Boxer GmbH), one two-way solenoid valve (075P2NC12-23B, Bio-Chem Fluidics Inc.) and three three-way solenoid valves (075P3MP12-23B, Bio-Chem Fluidics Inc.). The use of the components of the same type as in the case of the analytical module is advantageous, because this allows using the control unit also for driving the culture unit components. The control unit has sufficient hardware resources to control one analytical unit and two culture units simultaneously. Two kinds of tubing were used for the culture unit: The C-Flex® with internal diameter of 0.58 mm (type 10025-23B, Bio-Chem Fluidics Inc.) and the PharMed® BPT tubing with internal diameter of 0.51 mm (type SC0339, Cole-Parmer GmbH). Additionally, three 15 ml Eppendorf tubes serve as the reservoirs for the fresh medium, sample and waste containers.

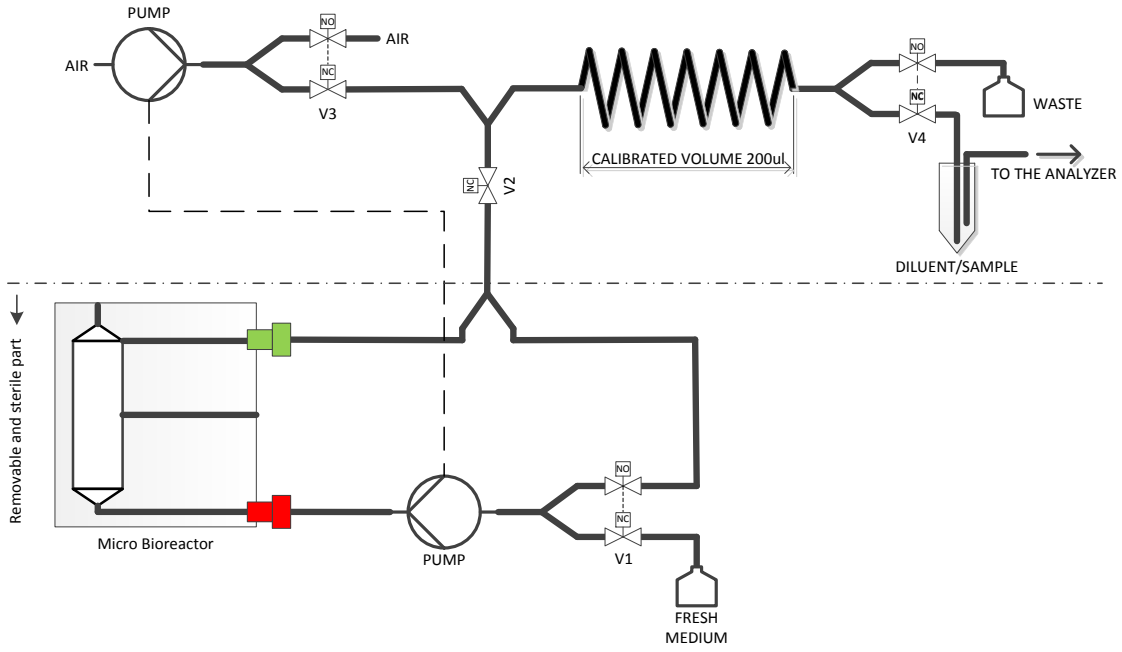

Figure 4.2 Fluidic diagram of the culture unit

The fluidic network is divided into two parts. The circulation loop including the bioreactor and the fresh medium reservoir must work under sterile conditions. Therefore, this part was designed as removable (Figure 4.2, lower part). The cell culture supported on the MatriGrid® can be inserted into the bioreactor and the whole circulation loop can

be filled with culture medium under the clean bench. Once completed, the circulation loop can be mounted on the culture unit outside the clean bench. The PharMed® material was selected for circulation loop tubing, because in contrast to the C-Flex® material it can withstand the autoclaving cycle. The second part of the fluidic network (Figure 4.2, upper part) is designed to handle sample or the waste medium from the bioreactor and the sterility or autoclaving operation is not required. The C-Flex tubing is therefore used for this second fluidic part.

The culture unit functionality will be described in more detail in the following paragraphs separately for each operating mode.

4.4.1 Culture unit during the active perfusion of the cell culture

Most of the time the culture unit perfuses the cell culture located in the micro bioreactor. The peristaltic pump maintains the circulation. The medium flow path is shown on the Figure 4.3. The valve V2 stays closed and V1 is also powered off, which means the V1-NO part remains open and the V1-NC part remains closed. This way the cell culture medium circulates in the loop and the atmospheric oxygen diffuses through the tubing walls and facilitates the medium oxygenation. The entire culture unit may be placed into incubator with controlled temperature and atmosphere. For that reason, the culture unit does not contain any electronics, which could cause problems with the heat management and moreover the electronics would need to be protected from the humid incubator atmosphere as well. The solenoid valve and peristaltic pump drivers are located inside the control unit. The perfusion speed should be selected sufficiently low that the cells are not loaded with excessive shear stress. At the other hand too low perfusion can limit the oxygen supply to the cells. The typical perfusion speed is in the low tens of microliters per minute. The culture unit allows setting the perfusion speed in the range from 1 µl/min to 500 µl/min. The direction of the perfusion can be also changed if needed. The second channel of the peristaltic pump is not used in this mode. The circulation loop dead volume is about 250 µl including the pump. The total internal volume including the micro bioreactor is 1600 µl. For a perfusion speed of 25 µl/min the time for one complete medium cycle will be 64 minutes.

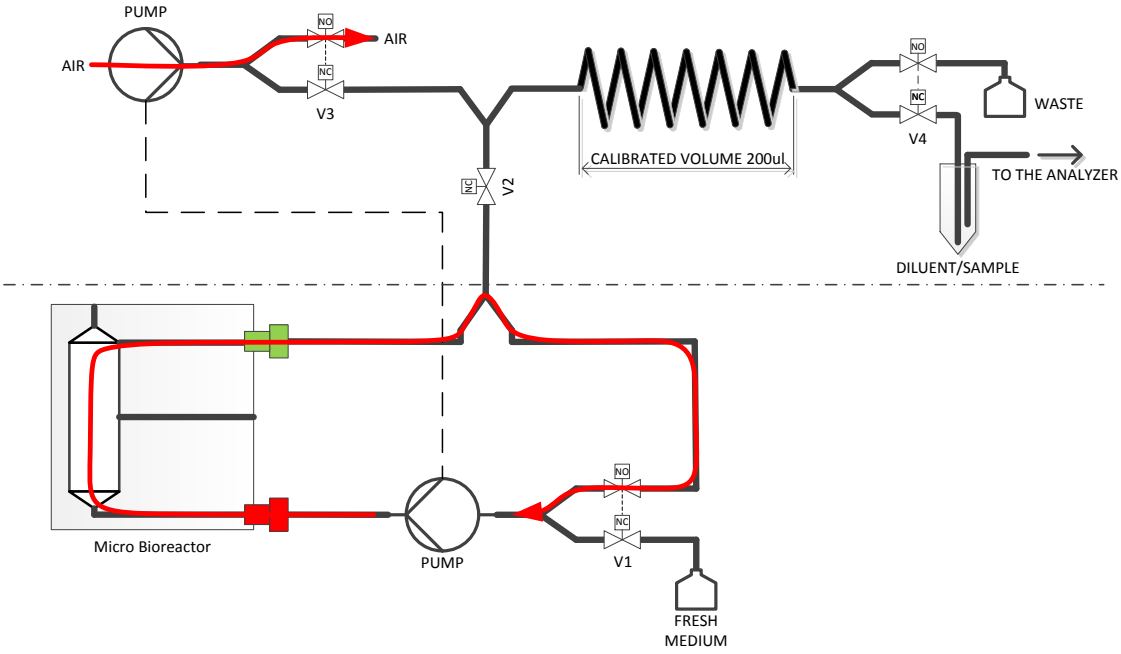

Figure 4.3 Culture unit active perfusion flow path

4.4.2 Automated medium change

The regular refreshment of the cell culture medium is essential in the cell culturing experiments. It is always connected with the risk of microbial contamination, so it must be carried out under aseptic conditions. The cell culture unit was designed to automate the medium change process and minimize the risk of contamination. This is achieved by keeping the fluidic system closed during the medium change operation. The whole procedure consists of two phases. During the first phase the fresh medium is pumped into the bioreactor while the old medium is pumped out of the bioreactor to the waste container. The second phase empties the fluidic paths. The respective flow paths are shown on the Figure 4.4 and Figure 4.5. During the Phase I the solenoid valves V1 and V2 are powered on, while the valves V3 and V3 stay powered off. The circulation loop is opened and the pump delivers now the fresh medium to the bottom end of the bioreactor.

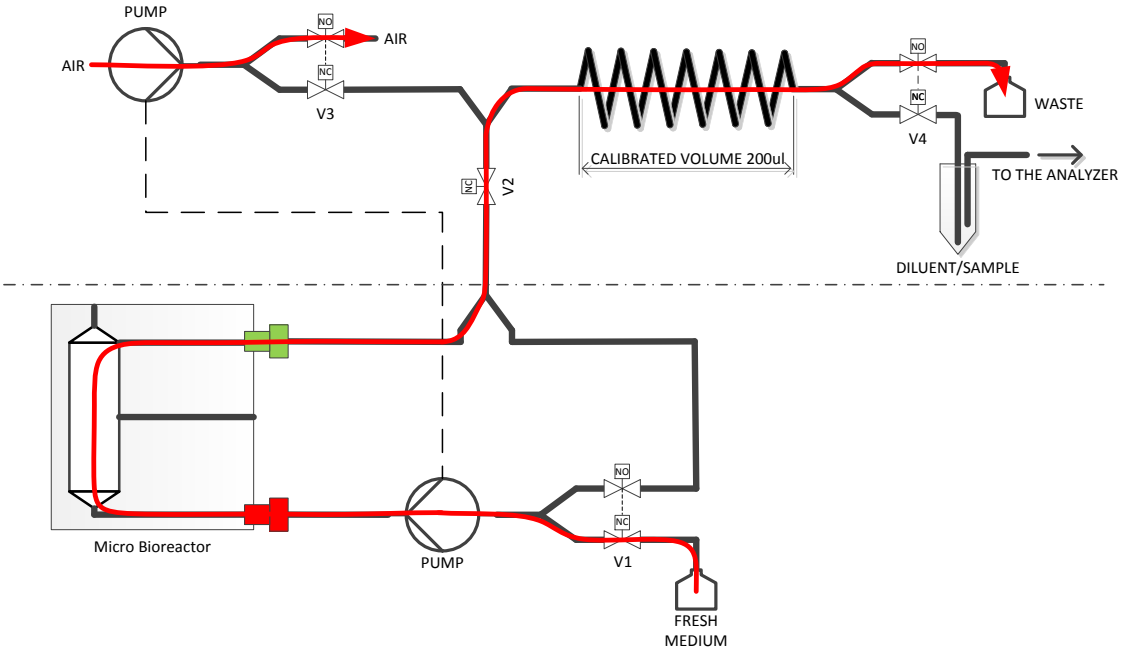

Figure 4.4 The culture unit flow path during the medium change or sampling, phase I

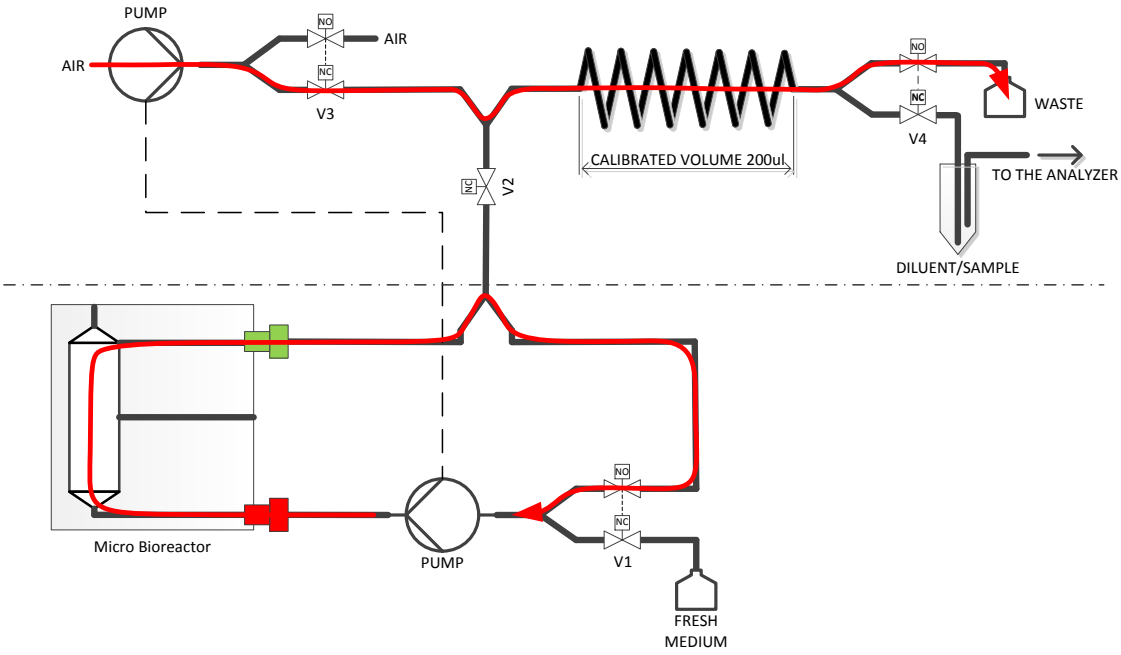

Figure 4.5 The culture unit flow path during the medium change phase II

The old medium is continuously displaced from the top part of the bioreactor through the valve V2 and the calibrated volume fluidic part (which plays no role in this mode of operation) to the waste container. By keeping the flow inside the bioreactor laminar (which is always the case for the relevant perfusion speed range) the mixing between old and new medium is limited to the diffusion. The user has the freedom to select the medium

change volume and speed. Partial or full medium exchange can be achieved by varying the exchange volume. The exchange speed is usually the same like the perfusion speed, but it can be also increased for speeding up the medium change process if the cultured cells can handle such perfusion rate increase. At the end of phase I the valves V1 and V2 switch off again, which will restore the circulation loop and the cell culture continues to be perfused with the refreshed medium. The fluidic paths behind the valve V2 are filled with the old medium, which needs to be removed. This is the purpose of the phase II in which the valve V3 switches on and the old medium in the tubing behind the valve V2 will be displaced by the air pumped by the second channel of the peristaltic pump. At the end of phase II, the valve V3 will be switched off. The cell culture perfusion is not affected during the phase II. The volume of the fresh medium container (15 ml) together with the medium exchange volume sets the limit how many times the medium exchange can be performed without refilling it and thus opening the aseptic part. Another limitation is the stability of the medium at the incubator temperature.

4.4.3 Automated sampling of the cell culture medium

The basic feature of the integrated culture and analytic system is the possibility of automated online medium sampling and subsequent analysis. The culture medium can be supplemented with vehicle control or test chemical as needed. The sampling procedure is similar to the medium change and consists of two phases as well. During the first phase the solenoid valves V1 and V2 are powered on. The medium being sampled flows out from the top side of the bioreactor through the valve V2 and the calibrated fluidic part to the waste container. The fresh medium flows through the valve V1 to the bottom side of the bioreactor. The situation is shown on the Figure 4.4. However, the volume displaced during the phase I is chosen so that it will fill the fluidic paths until the point of valve V4. The required volume will be slightly more than the calibrated value if 200 µl. The difference accounts for the dead volume of the tubing connecting the calibrated part with the valves V2 and V4. The medium stored in the calibrated part is displaced into the sample container during the phase II (Figure 4.6). This way the volume delivered to the sample container will be always known and constant. This approach was preferred over the simple metering by peristaltic pump. Although the stepper motor can rotate for exact angle, the pumped volume depends also on the exact position of the tubing inside the peristaltic head or the degree of tubing wearing. During the sampling phase II the valves

V3 and V4 are powered on, while the valves V1 and V2 remain powered off. The circulation loop is closed and the cell culture remains to be perfused.

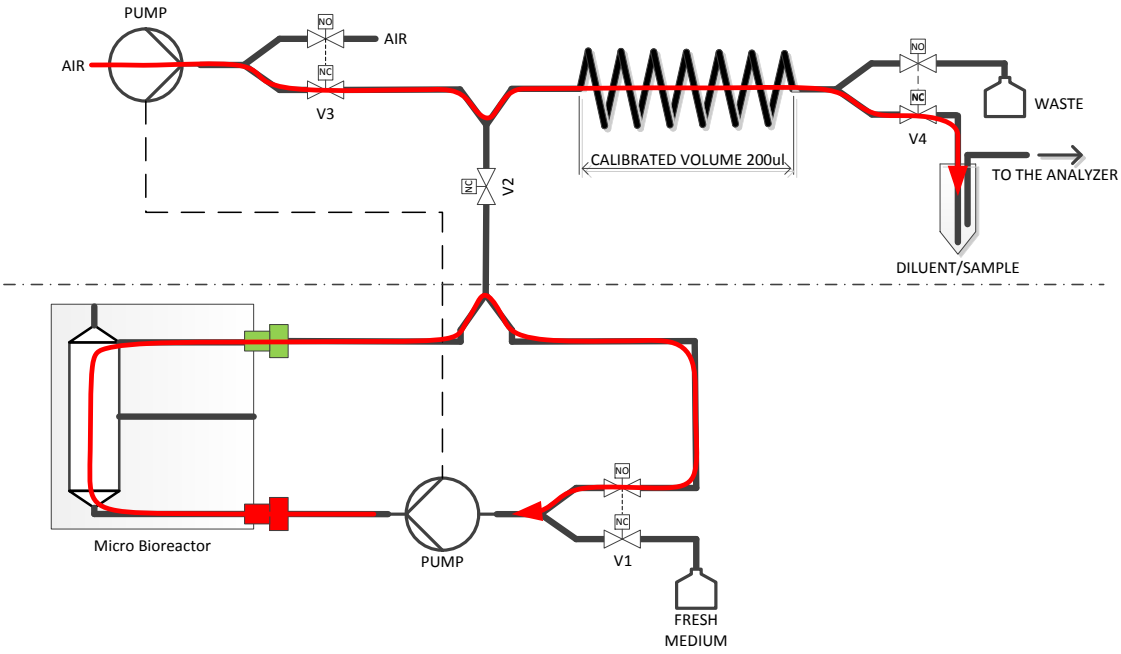

Figure 4.6 The culture unit flow path during the medium sampling phase II

The optional sample dilution can be reached by filling the required amount of diluent into the sampling container prior to sampling. The amount of sample is known so the amount of diluent can be calculated for required dilution factor. Once the sample was displaced into the diluent there is possibility to mix the resulting solution by the stream of air. In this case the pumped volume during the phase II is increased several times, so after the sample was flushed into the diluent the air continues to be further pumped to the solution. The sample container tubing must end at the bottom of the container. At the end of phase II, the sample ready for the analysis is present in the sample container. The additional tubing provides the fluidic connection between the culture unit sample container and the analytical module. Alternatively, the sample can be cryopreserved and analyzed later.

4.5 Intermediate summary

The culture unit described in this chapter together with the analytical module and the control unit presents a new tool for cell culturing with high degree of automation and system integration. The possibility of automated medium change while keeping the

fluidic system closed minimizes risk of cell culture contamination. The prototype of the culture unit shows Figure 4.7. The evaluation of this culturing system will be described in the following chapters.

Figure 4.7 The culture unit prototype

5 Evaluation of the prototype system

5.1 Introduction

This chapter describes initial testing of the culturing and analytical system after the hardware and software development has been finished as described in the previous two chapters. First the basic functionality of the fluidic components was verified. Next the flow rates were optimized for the culturing part and the analytical part. The measurement of standard curves for ELISA protocols was performed to confirm the assay validity after its adaptation to the flow-through topology. During this testing phase some problems were discovered, which required some design changes and protocol optimizations. Those changes are discussed in detail in the following chapter.

5.2 Fluidics evaluation

The initial fluidic testing of the analyzer unit was performed with water as the working fluid. The complete flow-through ELISA sequence was executed and the proper function of active fluidic components (peristaltic pumps and solenoid valves) was visually inspected. No leaks were detected. Furthermore, the pumping speed was optimized. It is desirable to use the highest possible pumping speed to shorten the assay time. The upper limit of the pumping speed is 2000 µl/min (software limitation). However, pumping speeds above 500 µl/min caused significant increases in the liquid pressure, which together with the tubing elasticity caused inconsistency in the pumped volume. Experimental testing showed that it is necessary to keep the pumping speed bellow 500 µl/min in order to maintain the pumped volume accuracy. The pumping speed needed to be further reduced for pumping liquids to the capillaries to prevent the desorption of antibodies and to not affect the assay accuracy. Pumping volumes were determined according to dead volume of the respective fluidic paths and increased by 15 – 25% to compensate for priming phase and peristaltic tubing wearing. The satisfactory pumping parameters for various assay fluidic operations are listed in the Table 5.1.

Similarly, the fluidic system of the culture unit was inspected for proper functionality. Since its fluidic system is much simpler compared to the analytical unit and the pumping

speed used for perfusing cell culture are typically in the range of tens of µl/min, no additional optimization of the fluidic network was necessary during the initial testing phase. One minor problem however, was occasionally observed. The MatriGrid® mounting in the Bioreactor proved to be critical. If the MatriGrid® was nor perfectly aligned with its support, the O-rings sealing of the bioreactor was leaky and the loop circulation in this case failed.

Table 5.1 Optimized fluidic parameters for flow-through ELISA assay

	Fluidic operation	Pumped volume [µl]	Pumping speed [µl/min]
1	Filling manifolds with a reagent	300	350
2	Filling capillary with a reagent	50	100
3	Filling manifolds with washing buffer	500	350
4	Filling capillary with washing buffer	50	100
5	Filling manifolds with a sample	300	350
6	Filling capillary with a sample	50	100
7	Removing reagent or sample from manifolds	350	350
8	Removing sample from sample manifold	50	100
9	Removing reagent from reagent manifold	50	100
10	Washing after sample	500	350
11	Washing reagent manifold	50	350

5.3 Readout system evaluation

Functionality of the fluorimeter as the readout subsystem was evaluated first by measuring the noise level in the dark. The transfer curve - the dependence of measured fluorescence on the resorufin concentration was measured and the limit of detection was determined. The fluorimeter was also tested in cooperation with the sample changer to evaluate the “autofocusing” algorithm for proper sample alignment.

The fluorescence measurement sequence begins with the ADC sampling in the dark, i.e. with the excitation laser turned off. This ADC reading includes the transconductance amplifier offset and it is used as a baseline for the final fluorescence calculation. Those dark ADC readings can be also used for the fluorimeter electrical noise evaluation. The dark ADC readings obtained during the 48 florescence measurements of the resorufin standards were used to estimate the noise level. Out of the 48 readings, the ADC generated the output number 32828 - 21 times and the number 32829 27 times. No other codes were

generated. Obviously, the peak-to-peak noise is 1 LSB and the design goal in the terms of noise performance was met.

Because the transconductance amplifier transfer function has a negative slope, the actual fluorescence reading is calculated by subtracting the measured value from the dark (baseline) value. This way the fluorescence reading is proportional to the fluorophore concentration and blank solution provide the reading of 0. The sensitivity of the fluorimeter was evaluated by measuring a series of concentration standards of resorufin sodium salt (R3257-5G, Sigma Aldrich). The measured dependence is shown on the Figure 5.1.

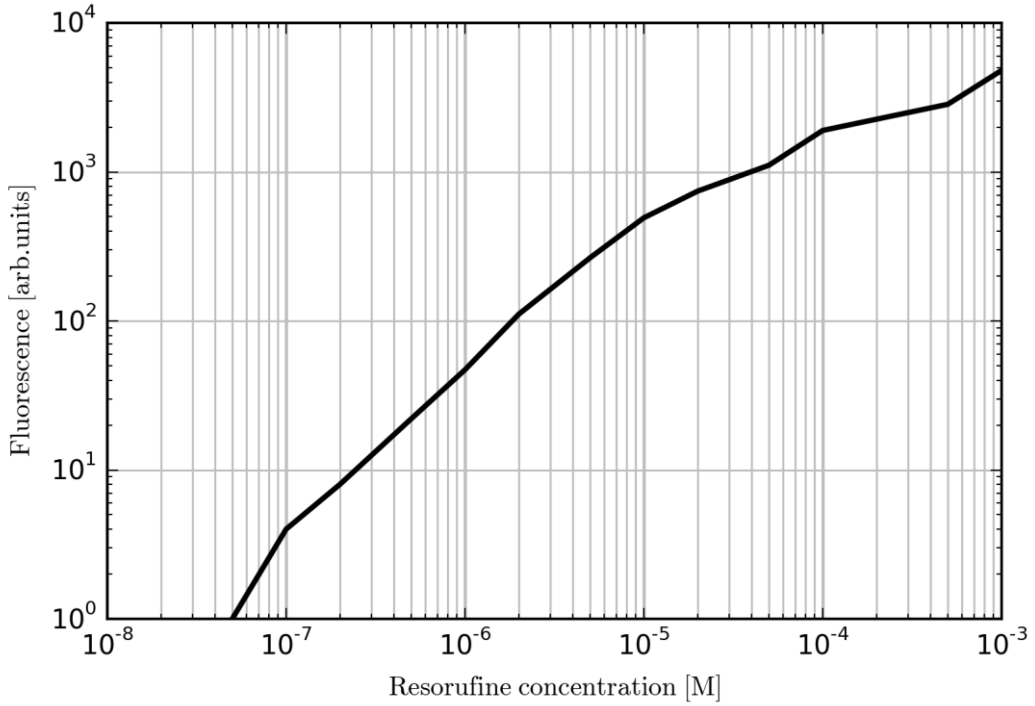

Figure 5.1 Dependence of the fluorimeter reading on the resorufin concentration. Blank, 10nM and 20nM solution of resorufin was not detected (reading of 0). Lower limit of detection is 50nM of resorufin.

It can be seen that blank solutions resulted in consistent zero readings. This proves we had the appropriate optical filter selection, because no excitation light caused false readings. The resorufin concentration of 10 nM and 20 nM was too low and was not detected by the fluorimeter. The lower limit of detection was found to be 50 nM of

resorufin. This limit is more than adequate for adapted assay, as will be shown later by measuring the standard curve.

The fluorimeter was designed to work together with the rotary sample changer. The rotary sample changer is formed by a double disk and contains eight capillaries equally spaced around its circumference. The implemented algorithm eliminates problems with the sample alignment with the focal line of the fluorimeter. During the actual measurement the capillary is positioned at some angular distance before the focal line. The sample changer then rotates through the focal line while continuously measuring the fluorescence. The extent of rotation is adjustable, but the value of 100 microsteps was found to be satisfactory. The sample changer stepper motor is configured to make 6400 microsteps per one revolution, so 100 microsteps corresponds to the angle of 5.625°. The Figure 5.2 shows the fluorescence dependence on the angular distance (number of microsteps) for 10µM resorufin solution. It can be seen that the area near the maximum is relatively flat, implying that the fluorescence is not very sensitive to the angular position.

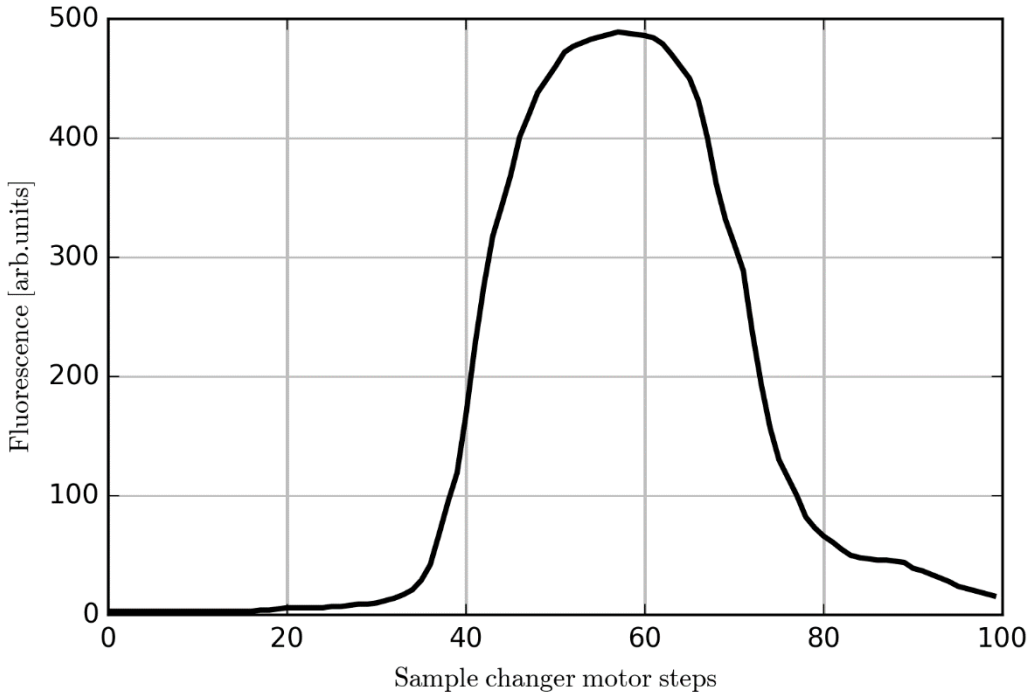

Figure 5.2 Dependence of the fluorescence on the capillary angular position during the sample holder rotation. The x-axis span shown (100 microsteps) corresponds to angular distance 5.625°. The capillary was filled with 10µM resorufin solution.

5.4 Albumin assay evaluation

To fully evaluate the analyzer functionality, complete human albumin assay was run on the analyzer module as described in the **Chapter 7**. A series of human albumin solutions with known concentration was used to measure the standard curve. The actual concentrations were selected according to quantitation kit manufacturer recommendations (Bethyl, E80-129): 0, 6.25, 12.5, 25, 50, 100 and 200 ng/ml of albumin. According to used QuantaRed™ substrate manufacturer instructions (15159, Thermo Fisher Scientific), the enzymatic color reaction needs to be stopped using the stop solution, following the fluorescence measurement in the MTP reader. In contrast to that, in a flow-through protocol there is no need to use the stop solution, because the fluorimeter creates an integral part of the analyzer and the fluorescence of each capillary is measured after exactly the same incubation time. This way, no manipulation with the capillaries is required and all related errors are thus eliminated. Moreover, this approach allows fluorescence measurement in multiple time points which allows internal quality control and the measurement of reaction kinetics, if required. For this reason, all fluorescence data measured by the analyzer module was taken for at least three time points. Table 5.2 shows the fluorescence data for the standard curve test. The measurement at three substrate incubation times: 370s, 740s and 1110s.

Table 5.2 Standard curve test – measured fluorescence

Albumin conc. [ng/ml]	Fluorescence [arb. units]		
	370s	740s	1110s
0	172	281	401
6.25	363	642	849
12.5	492	881	1318
25	1013	1863	2593
50	1529	2649	3890
100	2923	4846	6926
200	3463	5722	8154

The shape of dose-dependent curves of the sandwich immunological assay has sigmoidal shape [38] and the four-parameter logistic model is considered the most suitable for nonlinear regression of such assays [39]. The four-parameter model is based on the following equation:

$$y = d + \frac{a - d}{1 + \left(\frac{x}{c}\right)^b} \tag{5.1}$$

The parameters *a*, *b*, *c* and *d* have following meaning:

a – corresponds to response (*y*) at zero analyte concentration (*x*)
b – represents the slope of the sigmoid curve
c – represents the sigmoid curve inflexion point
d – represents expected response (*y*) for infinitely high analyte concentration (*x*)

The equation x is typically solved numerically using various iterative algorithms (e.g. Gauss–Newton algorithm), to find model parameters for a given concentrations and measured fluorescence [38]. Initial parameter setting is needed as starting point, and following procedure was used for this purpose:

a – use the minimal value of the response *y*
b – use the slope defined by minimal and maximal response *y*
c – use the response value *y* which is closest to the middle point between minimum and maximum
d – use the maximal value of the response *y*

The regression algorithm uses residual sum of squares (RSS) as the assessment criteria for the quality of curve fit. The RSS is calculated according the following equation:

$$RSS = \sum_{i=1}^{l} w_i [Y_i - (Y_c)_i]^2 \tag{5.2}$$

Where Y is the observed response and Y_c is calculated response. The w_i is weighting factor for *i*-th data point. The use of weighting factor greatly improves the accuracy of curve fit, because the error tends to be proportional to the signal (Y) magnitude [40, 41].
The curve fitting in the frame of this work was done using an automated MS Excel sheet [42], which implements weighted logistic model and Solver add-in tool to calculate model parameters. The estimated model parameters for the standard curve measurement contained in Table 5.2 are shown in the Table 5.3, and corresponding curves are shown in Figure 5.3.

Table 5.3 Estimated model parameters for standard curve for three different incubation times

	370s	740s	1110s
a	4789	7824.23	10726.3
b	-1.26769	-1.21539	-1.25709
c	89.07887	84.08654	76.5626
d	176.9873	286.172	404.1411
R^2	0.9959	0.9965	0.9978

Once the model parameters are known; any sample analyte concentration X can be calculated from the corresponding fluorescence Y using the following equation:

$$X = c\left(\frac{a-d}{Y-d} - 1\right)^{\frac{1}{b}} \tag{5.3}$$

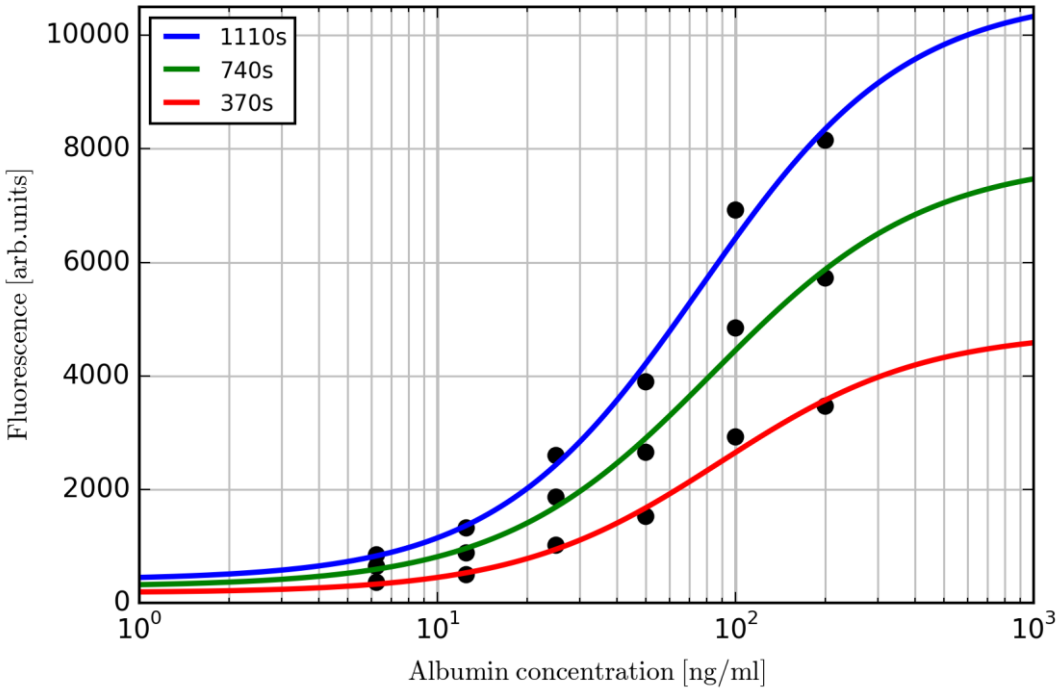

Figure 5.3 Standard curve fitting of data shown in Table 5.2. Quality of fit: R^2=0.9959 for 370s incubation time; R^2=0.9965 for 740s incubation time; R^2=0.9978 for 1110s incubation time.

The measured standard curve shown on Figure 5.3 confirm that the analyzer unit is functional and the adaptation of standard MTP based ELISA protocol was successfully modified to a flow-through version. However, the fluorescence data is not ideal. The recorded fluorescence for blank standard is approximately 10x higher than expected. This

high background signal decreases overall dynamic range of the assay and indicates problems with contamination of fluidic paths or insufficient cleaning procedure. To find the concrete source of the problem, additional testing was necessary, which is described in the following chapter.

5.5 Intermediate summary

The initial hardware and software testing of the whole system described in this chapter successfully verified basic functionality of the whole culturing and analytical system and proves its suitability for intended purpose – that is an automated cell culturing and automated cell culture analysis utilizing the sandwich ELISA assay. During this testing phase, minor problems were identified. By solving those problems, the accuracy and dynamic range of the immunoassay could be increased on the side of analytical module, as well as the manipulation reliability on the side of culturing unit.

6 Optimizing the performance of analytical module

6.1 Introduction

This chapter describes the analyzer unit modifications to solve some assay accuracy problems as described in the previous chapter. The analyzer performance after those modifications made the measurement of standard curves in low concentration ranges possible, allowing the measurement of values lower than recommended by the substrate kit manufacturer. The accuracy of the assay was verified by comparing with standard microtiter plate assay and the results are presented.

6.2 Analyzer unit cleaning procedure

Cleaning of all the fluidic pathways before the start of new assays is of critical importance. A proper cleaning procedure removes all chemically active residuals which remain after the previous assay, such as the adsorbed proteins on the tubing and manifold walls. The extent of removal must be sufficient to not influence or distort the results of the following assay. The assay interference caused by insufficient cleaning can be observed as increased background fluorescence or the incoherent standard curve.

The initial cleaning procedure involved the emptying of all fluidic paths and washing it with 70% ethanol, followed by washing with deionized (DI) water. Although it is known that 70% ethanol denatures proteins and it is an efficient disinfectant, this cleaning procedure did not provide a satisfactory result. The background fluorescence was significantly higher after each assay, which indicated insufficient desorption and / or denaturation of proteins. The cleaning procedure was therefore extended, and as the first step a desorption solution of the following composition was used: 70% ethanol, 2% citric acid and 0.5% sodium dodecyl sulfate (SDS). In the second step, washing with 70% ethanol was used, followed by final washing with DI water. All three components of the desorption solution cause protein denaturation by various mechanisms. The ethanol disrupts the hydrophobic interactions of the protein core, the citric acid at concentration of 100 mM lowers pH to approximately 2, which causes negative charge neutralization of the protein amino acid residues. SDS is a well-known detergent and causes protein

unfolding by binding of the SDS hydrophobic part to the protein amino acids [43]. Moreover, disrupting hydrophobic interaction between the protein and fluidic channel surface helps to desorb and remove the proteins. The use of above described three stage cleaning procedure proved to be satisfactory and allows running multiple assays without replacing the C-Flex™ tubing. The only part required to be replaced for each assay run are the capillaries which are considered to be critical component, because they serve as the assay's solid phase. The cleaning procedure is automated and it is part of the control unit firmware.

6.2.1 Extension of the cleaning procedure

During later tests using the optimized FEP manifolds it was found that occasionally, those manifolds become clogged by precipitated proteins. The cleaning procedure was therefore extended for additional washing by 1M sodium hydroxide solution. It is known that proteins, including BSA, can be desorbed and solubilized by NaOH at high pH values [44, 45]. All used fluidic component materials are chemically compatible with 1M NaOH solution: PharMed™, C-Flex™, PVC, FEP and PEEK. The sodium hydroxide washing was selected as the first step in the cleaning sequence to dissolve and remove most of the proteins, which prevents formation of the protein precipitates in the following steps. The citric acid in the second cleaning step helps to neutralize rests of the alkaline solution trapped in the fluidic network. At the end of the last cleaning step (DI water), a pH measurement confirmed the neutral reaction of the effluent. The final version of the cleaning procedure is listed in the Table 6.1.

Table 6.1 The final cleaning sequence of the analyzer fluidic network

Cleaning sequence step	Description
1	Empty the fluidic network
2	Wash the network with 1M NaOH solution
3	Empty the fluidic network
4	Wash the network with desorption solution
5	Empty the fluidic network
6	Wash the network with 70% Ethanol
7	Empty the fluidic network
8	Wash the network with DI water
9	Empty the fluidic network

6.3 The cause of decreased assay performance

Looking at the fluorescence data listed in the Table 5.2, two problems can be identified: the relatively high background fluorescence of the blank standard, and additionally slightly decreased accuracy which can be seen on the Figure 5.3, where the measured points do not coincide closely with the standard curve (i.e. it was not possible to make curve fittings with a smaller error). This may be caused by several reasons, such as improper washing, instability of the substrate, contamination of the substrate by secondary antibody or various assay protocol problems. To diagnose the origin of this inaccuracy, a series of tests were conducted and evaluated.

The stability of the substrate was tested for stability as the first step. The substrate mix consists of the non-fluorescent ADHP dye, the enhancer and stabilized hydrogen peroxide solution. This mix should be used within 30 minutes according the manufacturer instructions. The presence of traces of HRP enzyme or possibly other interfering compounds may result in resorufin dye development. The flow-through ELISA sequence was used for this test as described in the **Chapter 3.2.4** with one modification: it was started from the step 13 (Introducing the substrate to the capillaries). Also, new capillaries were used for this test and the whole fluidic system was washed using previously described three-step cleaning procedure. Recorded fluorescence is shown in the Table 6.2. Ideally, fluorescence values of zero or close to zero should be recorded for all 7 channels. It can be seen that the fluorescence after 5 minutes is zero for all channels except the channels 3 and 7, where minimal fluorescence was recorded.

Table 6.2 Substrate stability test – measured fluorescence

Channel	Fluorescence [arb. units]		
	5 min	10 min	15 min
1	0	10	15
2	0	9	15
3	5	13	18
4	0	9	14
5	0	3	6
6	0	12	16
7	4	9	10

The fluorescence in all channels slowly increased with time and after 15 minutes the fluorescence remained below 18 units. Those values are more than 20 times lower than the values for channel 0 (blank) listed in the Table 5.2. It may be therefore assumed that the substrate is sufficiently stable and does not contribute to problems with assay accuracy.

Logically, the next test should involve the conjugated secondary antibody. This test was aimed to evaluate substrate contamination by the conjugated antibody. Ideally, the conjugated antibody is pumped to all capillaries, followed by a washing step using the washing buffer and finally the substrate solution will be pumped to the capillaries. If the fluidic system will be completely washed of the conjugated antibody, the fluorescence reading should stay close to the levels listed in Table 6.2. The whole fluidic system must be previously blocked by BSA blocking buffer, to prevent non-specific binding of conjugate antibody to the surface of capillaries and fluidic pathways. Again, an incomplete flow-through ELISA sequence was used for this test, starting from the step 4 (pumping the BSA blocking buffer to the capillaries). The blocking incubation time was extended to 1 hour to allow for sufficient surface blocking. The steps 7, 8 and 9 were skipped as no standards were using in this test and the sequence continued by the step 10 (pumping the conjugate antibody to the system). The fluorescence levels measured in the last step (14) are listed in the Table 6.3.

Table 6.3 Conjugated antibody contamination test – fluorescence data

Channel	Fluorescence [arb. units]		
	5 min	10 min	15 min
1	467	928	1365
2	426	708	1085
3	432	834	1125
4	400	770	1055
5	265	490	666
6	349	685	949
7	325	558	798

The result of the test shows high fluorescence values, implying substrate contamination by the enzyme conjugated antibody. However, it is not clear whether this contamination originates from cross-contamination of fluidic pathways or there might be some conjugated antibody adsorbed on the capillary walls caused by possible insufficient

blocking. To investigate this possibility, the test was repeated with one important modification: at the end of step 13, when the conjugate antibody was pumped to the capillaries and washed afterwards, all capillaries were replaced for new ones and the sequence then continued with the step 14 (pumping the substrate). This way, it was guaranteed that no conjugated antibody was present in new capillaries and eventual substrate color reaction must be inevitably caused by conjugated antibody remained in the fluidic system. The fluorescence data as the result of this experiment is presented in the Table 6.4.

Table 6.4 Conjugated antibody contamination test with replaced capillaries – fluorescence data

Channel	Fluorescence [arb. units]		
	5 min	10 min	15 min
1	84	158	234
2	155	259	351
3	225	441	651
4	164	258	390
5	394	717	945
6	494	955	1291
7	25	39	47

Comparing the fluorescence data with the previous test it can be concluded that the contamination was reduced but not significantly. Moreover, the fluorescence between the channels differs significantly! This fact implies various degree of contamination for various channels. This cannot be attributed to improper washing directly, because the washing cycle in the step 12 is repeated 3 times with exactly same pumping volumes and timing for all capillaries. It is obvious that, despite intense washing cycle, traces of conjugate antibody solution remain in the fluidic system. Logically, the most probable place in the fluidic system where the washing could be problematic are the places with highest topologic complexity, more specific – the manifolds. The analyzer unit used during those tests the commercial 9-port manifolds with the "star" topology (P-191, IDEX Health & Science, Figure 6.1 - left).

It is difficult to prevent the contamination of adjacent channels in the star joint. A small portion of solution always diffuses to all channels connected to the center point. Also, the washing procedure sequenced as switching of one arm only at same time, will not result in perfect cleaning for the same reason.

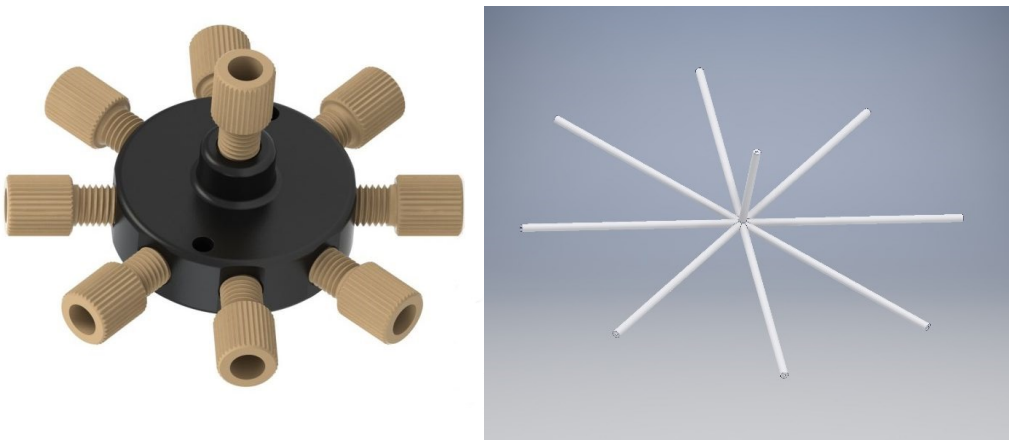

Figure 6.1 The 9-port manifold with "star" fluidic topology (left), and its internal fluidic channels (right)

This idea led to change of the manifold topology, which would allow more efficient reagent separation and cleaning. The topology of nine port manifold was therefore changed from the star configuration to series of 7 "tee" joints. This way, no joint has more than three branches: one inlet, one outlet and one side arm. Also, the problem with diffusion will be limited to one side arm, and the sequencing algorithm can easily correct this effect. The Figure 6.2 shows redesigned 9-port manifold.

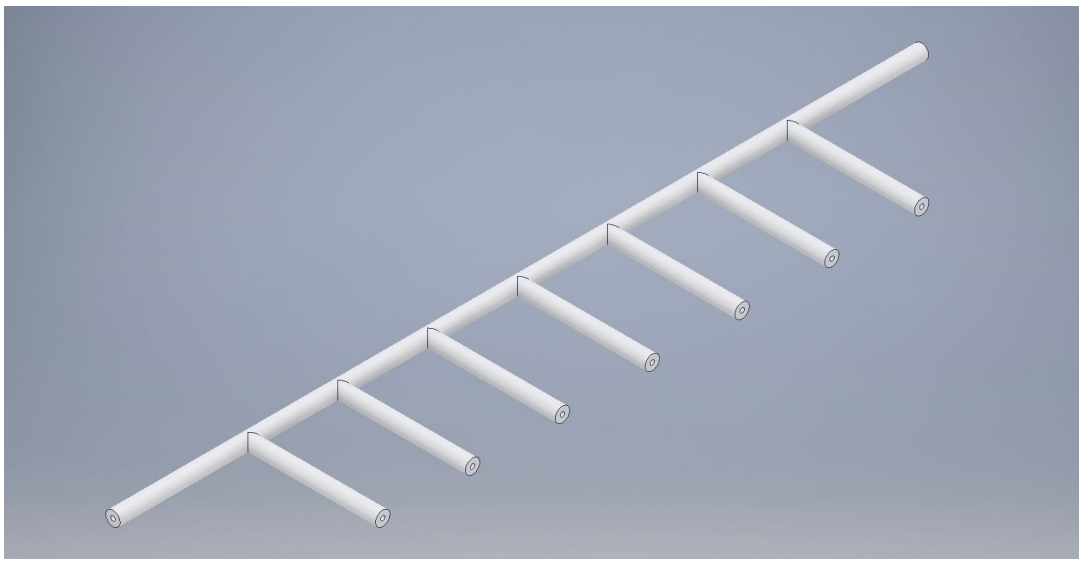

Figure 6.2 The fluidic topology of 9-port manifold composed from series of "T" joints

Three out of four manifolds were replaced with the new version (see Figure 3.4): the sample manifold, the reagent manifold and the capillary manifold. Only the waste manifold was kept unchanged, because it is located in the waste stream and is therefore

irrelevant for the assay accuracy. New manifolds were built by hot-air welding of pieces of FEP tubing (VICI JR-T-6802, 1/16" x 0.5 mm). The capillary manifold is located at the top of the circular sample changer and it is therefore required to have a circular shape. A dedicated manifold holder was therefore manufactured to support the FEP tubing (see Figure 6.3).

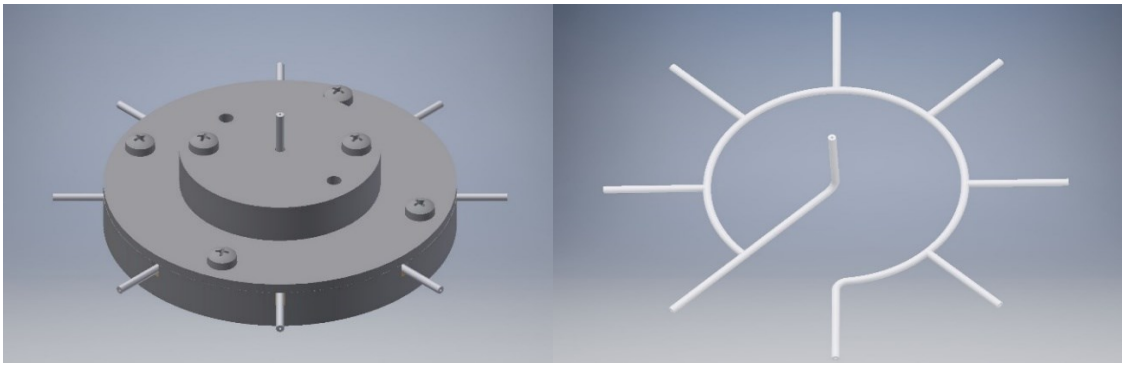

Figure 6.3 The circular holder of the capillary manifold (left), and corresponding FEP tubing manifold (right)

After the manifolds replacement, the analyzer was tested with full sequence to evaluate possible improvements in the background fluorescence and cross-channel contamination. The result of the are listed in the Table 6.5.

Table 6.5 Full sequence test with redesigned manifolds – fluorescence data

Albumin conc. [ng/ml]	Fluorescence [arb. units]		
	7 min	14 min	21 min
0	61	104	150
6.25	172	319	474
12.5	405	773	1126
25	828	1573	2211
50	2000	3689	5135
100	4922	8370	9903
200	6148	9903	9903

The fluorescence data of the assay test after the replacement of the manifolds show significant improvement in the background fluorescence for blank standards. Additionally, the florescence values for monotonically increasing albumin concentrations in standards are also monotonically increasing in uniform steps, which is presumably caused by reduced cross-channel contamination. The fluorescence values of 9903 units

for the high albumin concentrations and longer times (14 min and 21 min) represents the maximum readout value under the fluorimeter amplifier saturation. This value is not constant for every assay and may slightly vary, depending on the offset calibration, which is performed at the beginning of each fluorimeter operation.

6.4 Standard curve measurement in low concentration range

According to Human Albumin Quantitation Set (Bethyl, E80-129) manufacturer instructions, recommended standard concentrations cover the albumin concentration range from 6.25 ng/ml to 400 ng/ml. Given the improved analyzer performance with optimized manifolds and improved cleaning procedure, a trial test was conducted to investigate the analyzer performance with the albumin concentrations below 6.25 ng/ml. Result of this test represents the standard curve in low albumin concentration range and corresponding fluorescence data are listed in the Table 6.6, the 4-parameter logistic model fit parameters are listed in the Table 6.7, and the standard curve is plotted on the Figure 6.4.

Table 6.6 Fluorescence data of standard curve in low albumin concentration range.

Albumin conc. [ng/ml]	Fluorescence [arb. units]		
	7 min	14 min	21 min
0	40	60	60
0.78	43	64	84
1.5	51	85	121
3.13	90	164	234
6.25	172	295	425
12.5	414	788	1141
25	906	1677	2523

Table 6.7 Estimated model parameters for standard curve in low concentration range

	7 min	14 min	21 min
a	2360.463	4427.98	19753.53
b	-1.62841	-1.61524	-1.39027
c	34.35895	34.56781	100.74
d	38.7509	57.90703	60.86779
R^2	0.995	0.9959	0.9973

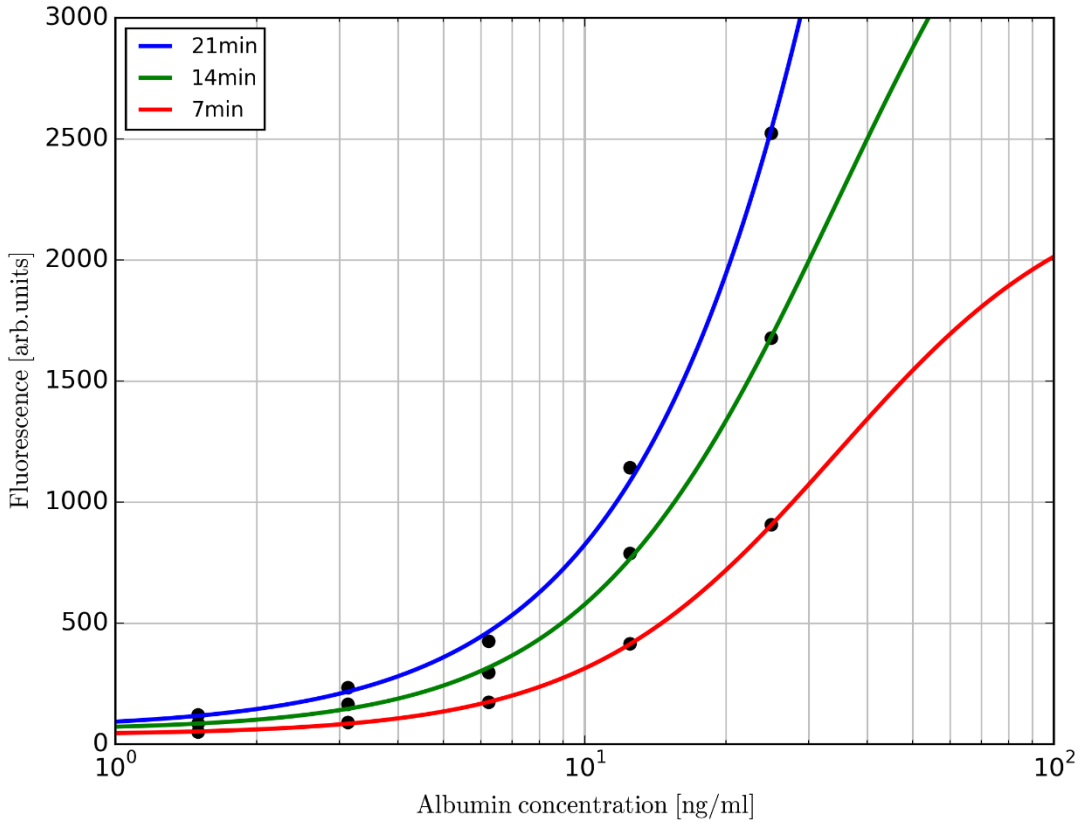

Figure 6.4 Measured standard curve in the low albumin concentration range.

The background fluorescence value was 40 units after 7 minutes, which is completely acceptable. Table 6.6 also shows that even the albumin concentration as low as 0.78 ng/ml can be clearly distinguished from the background and it is above the detection limit. For such low albumin concentration, longer substrate incubation times provide steeper curves, as can be seen on Figure 6.4. The measured points coincide more closely with the standard curve, compared to situation on Figure 5.3, despite the albumin concentration levels are 8x lower.

6.5 Accuracy verification of the flow-through ELISA

The final verification step of the analyzer functional assessment is the comparison of an albumin standard measurement by the analyzer unit utilizing the flow-through protocol, with the measurements of the same standard by standard laboratory ELISA protocols. Two albumin solutions with concentration of 9 ng/ml and 18 ng/ml were prepared and

used for testing by both protocols. Both solutions were measured as triplicates, allowing the evaluation of measurement error. Both protocols used same human albumin quantitation set (Bethyl, E80-129), but different substrates. The QuantaRed™ Enhanced Chemifluorescent substrate (15159, Thermo Fisher Scientific) was used for flow-through ELISA, while the TMB substrate was used for the MTP ELISA (Immunochemistry Technologies, #6275). The Spectramax® M5 reader was used to measure samples optical density when using the MTP ELISA. The substrate incubation time for flow-through ELISA was 7 minutes. The comparison of the measured albumin levels is shown on Figure 6.5 and the statistical assessment of the measurement error is listed in Table 6.8.

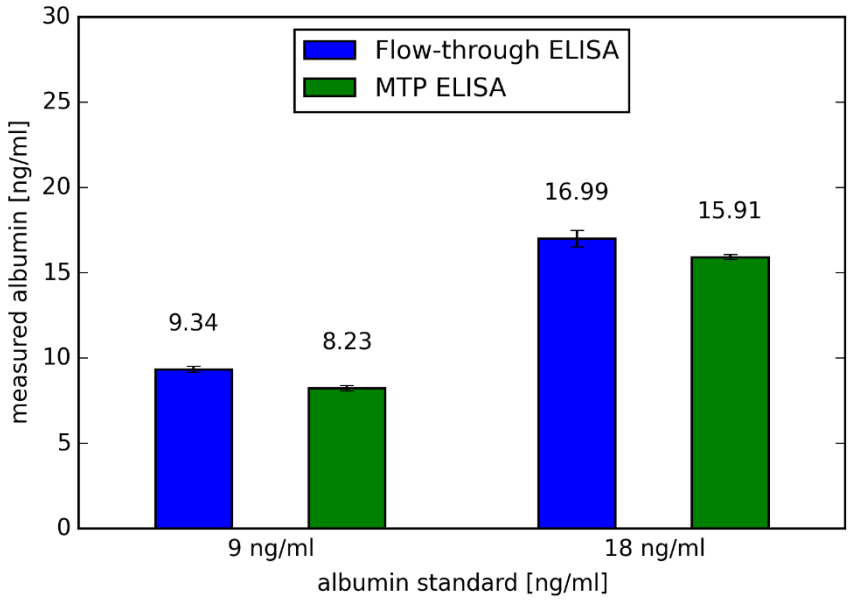

Figure 6.5 Comparison of albumin measurement with two different ELISA protocols. The error bars represent the standard error of mean (SEM)

According to current regulatory guidelines from the U.S. Food and Drug Administration (FDA) and the European Medicines Agency (EMA), a validated assay to quantify the antibody should demonstrate an accuracy with ±20% of the known antibody concentration and precision of less than 20% coefficient of variance [40]. The results in the Table 6.8 shows that the accuracy of the analyzer unit implementing the optimized flow-through ELISA protocol is significantly better than the required 20%. The MTP ELISA protocols provided slightly lower results then the flow-through protocol (11.88% lower result for 9 ng/ml standard and 6.35% lower result for 18 ng/ml standard). There

may be various reasons for that, such as variation of surrounding temperature, or the use of different substrate and different readout method (colorimetric vs. fluorimetric). Diagnosing the source of this error is beyond the scope of this work and it was not carried out.

Table 6.8 Statistical evaluation of measurement accuracy for both ELISA protocols.

Flow - through ELISA					
Albumin standard [ng/ml]	Measured values [ng/ml]	Mean value [ng/ml]	Standard deviation [ng/ml]	Measurement error [%]	Coefficient of variation [%]
9	9.60	9.34	0.29	3.73	3.07
	9.03				
	9.38				
18	16.11	16.99	0.82	-5.59	4.85
	17.13				
	17.74				
MTP ELISA					
Albumin standard [ng/ml]	Measured values [ng/ml]	Mean value [ng/ml]	Standard deviation [ng/ml]	Measurement error [%]	Coefficient of variation [%]
9	7.93	8.23	0.26	-8.60	3.18
	8.42				
	8.34				
18	15.81	15.91	0.26	-11.59	1.65
	15.72				
	16.21				

6.6 Intermediate summary

The issues identified and described in the **Chapter 5** served as a starting point for the analyzer unit improvement. Two key modifications – the redesign of the manifolds and improvement of the cleaning procedure leaded to desired level of assay accuracy, as was confirmed at the end of this chapter. The ability of standard curve measurement at concentration levels 8x below the recommended level was demonstrated. Additionally, the assay accuracy was compared to standard MTP ELISA protocol and it was confirmed that it fulfills recent international standards for validation of antibody quantitation assays. At this point, the analyzer unit is considered fully functional and ready to be used in connection with cell culturing units for on-line albumin level measurement in cell culture medium.

7 Evaluation of the prototype system

7.1 Introduction

The applicability of the automated compact device in online-kinetics measurement of albumin secretion was verified as a proof-of-concept using 3D HepaRG cultures perfused with acetaminophen (APAP) over a period of 96 h. This chapter demonstrates how this integrated *in vitro* system can be used for drug toxicity tests and show the potential for adaptation of online-monitoring to measure other secreted proteins, such as hormones and signaling molecules from 3D mono- and co-cultures. The APAP toxicity is discussed first, followed by the experimental part in which the APAP induced EC_{50} values were determined for HepaRG cell culture in various formats. To provide additional evidence of APAP metabolism in cell culture, the APAP consumption was measured by HPLC methods. Albumin measurements in the perfused 3D HepaRG culture with and without APAP supplemented medium using the integrated culturing and analytical system are described and the results are compared with conventional MTP ELISA.

7.2 APAP toxicity in 2D and 3D hepatocyte cultures

APAP is a well-known representative method for dose-related intrinsic liver toxicity [46, 47], making it an ideal test compound to use in these proof-of-concept studies. While predominantly phase II reactions account for the major metabolites of APAP, namely the APAP-sulfonate and APAP-glucuronide conjugates, toxic doses of APAP result in reactive metabolite formation (N-acetyl-p-benzoquinoneimine (NAPQI), Figure 7.1) via phase I enzymes, namely CYP3A4, CYP1A2 and CYP2E1 [48-50]. At lower concentrations, NAPQI is detoxified by reduced glutathione (GSH) but once GSH is depleted, NAPQI covalently binds to cellular proteins, e.g. from mitochondria [50-52]. The resulting mitochondrial dysfunction leads to a number of forms of toxicity which initiate pathways ultimately leading to acute liver failure (ALF) [53]. These pathways include the impairment of hepatocyte mitochondrial respiration, ATP depletion and formation of reactive oxygen species ([54-56] (such as NO and superoxides resulting from mitochondrial permeability transition (MPT) [57]).

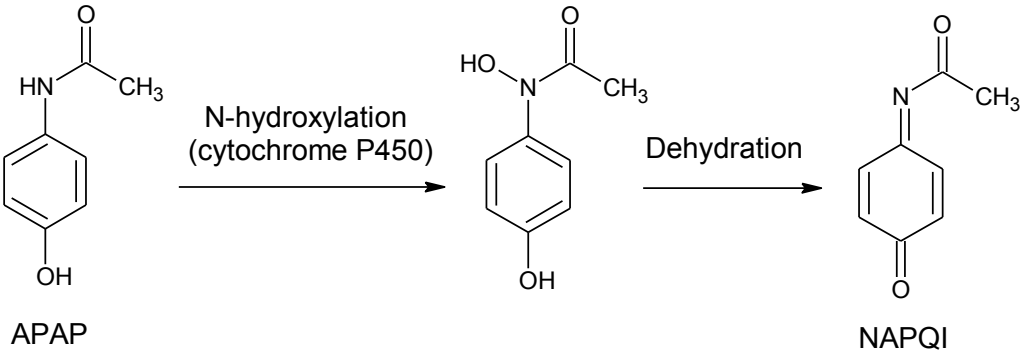

Figure 7.1 Metabolic conversion of acetaminophen (APAP) to toxic N-acetyl-p-benzoquinoneimine (NAPQI)

For individual experiments, HepaRG cells (BIOPREDIC International, Saint Grégoire) were seeded at a density of 50.000 cells per well either in collagen pre-coated (5 µg/cm^2) 24-well plates (monolayer) or collagen pre-coated scaffolds (3D organotypic cell culture) in 24-well plates. Cells were seeded in the scaffolds in a small volume (25 µl) to ensure selective growth in the microcavities. Subsequently, cells were allowed to adhere for 2 h before adding the remaining culture medium into the wells. After seeding, cells were cultured for 2 weeks in the maintenance medium in a cell incubator at 37°C, 95% relative humidity and 5% CO_2. Thereafter, cells were either cultured in the maintenance medium for another 2 weeks or shifted to a differentiation medium on day 14 (supplemented with 1% DMSO). For perfused 3D cell cultures, the MatriGrid® scaffolds were inserted in the supporting bioreactor and mounted on the culturing unit of the automated system. The culture medium was renewed every 2 days in all experiments.

APAP toxicity was measured by the determination of metabolic activity and albumin secretion. Metabolic activity of HepaRG cells was analyzed using the commercially available Alamar Blue® kit (BIO-RAD, BUF012A). Albumin levels in the culture supernatants were analyzed using the Albumin-ELISA Quantitation kit (Bethyl, E80-129) with TMB substrate (Immunochemistry Technologies, #6275). After 4 weeks of differentiation, HepaRG cells grown either in monolayer or scaffolds were incubated with increasing concentrations of APAP (0, 1, 5, 10, 15, 20, 40, 80 mM) in Williams medium E (WME) + 0.1% FBS in wells or in perfused micro-bioreactors for 24 h. Cells grown in monolayer culture (2D) or in scaffolds (3D) were treated with trypsin to return cells to suspension and the total cell number was determined. After centrifugation for 5 min at 515g cells were incubated with resazurin for 2 h at 37°C in the incubator. The

fluorescence of the metabolite, resorufin, was measured at 560 nm excitation and 590 nm emission with a Spectramax® M5 microplate reader. Albumin levels were measured in the culture supernatants according to kit manufacturer instructions. The absorbance of the TMB oxidation product was measured by Spectramax® M5 microplate reader at 450 nm. Metabolic activity and albumin level values were normalized to the total cell number (per million cells), and values of APAP-treated samples were normalized against the control values (i.e. without APAP), which was set to 100 %.

To assess whether the culture format affects APAP-induced toxicity and the effect of APAP on secretion of hepatic albumin, DMSO differentiated HepaRG cells grown under 2D and static (i.e., not perfused) 3D conditions were treated with increasing concentrations of APAP either statically (2D, 3D) or under perfusion (bioreactor "3D BR"). After 24 h of incubation, the concentration dependent toxicity of APAP was measured using two different readout parameters: resazurin metabolism and albumin secretion. As illustrated in Figure 7.2, 3D static cultures of HepaRG cells were more sensitive to APAP than 2D cultures according to resazurin metabolism (EC_{50} 3D: 21.0 mM, EC_{50} 2D: 27.1 mM). This could be due to the more highly differentiated state of the cells under 3D conditions, especially with respect to the presence of the bioactivating CYPs. Continuous perfusion of 3D cultures in a bioreactor ("3D BR") significantly enhanced the sensitivity of HepaRG cells to APAP, with an EC_{50} value of 9.7 mM, which is due to the increased accessibility of the hepatocytes for the drug.

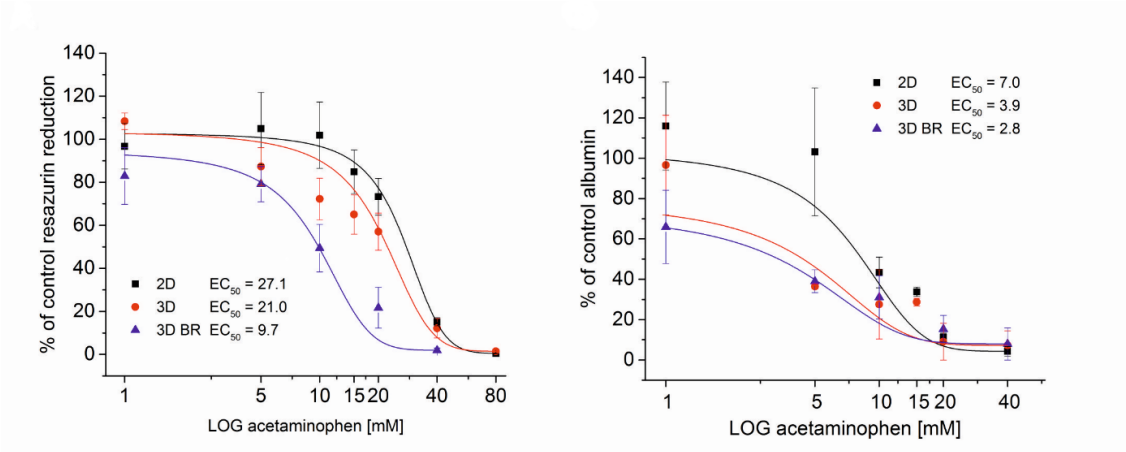

Figure 7.2 Effect of APAP on resazurin metabolism (left) and albumin secretion (right) in HepaRG cells cultured under different conditions. The fluorescence of resorufin, the product of the resazurin assay, was measured with a SpectraMax M5 microplate reader. Each experiment was replicated 3 times (n = 3 per concentration, mean ± SEM).

In comparison to the resazurin assay, inhibition of albumin secretion by APAP occurred at lower concentrations of APAP. The lowest concentration of APAP that inhibited albumin synthesis was detected in perfused 3D cultures (EC_{50} 3D perfused: 2.8 mM). The EC_{50} value for statically cultured 3D cultures was 3.9 mM, while 2D cultures were the least sensitive to albumin inhibition by APAP (EC_{50} 2D: 7.0 mM). The results are consistent with those of others and show that in addition to the impairment of mitochondrial function, the secretion of albumin is also affected by APAP [15, 58-60].

To investigate whether the higher sensitivity of 3D HepaRG cultures to APAP are due to an increased metabolism of APAP (to NAPQI), we measured APAP consumption during the incubations. Differentiated 2D and 3D HepaRG cells were incubated with 20 mM APAP in Williams medium E with 0.1% FBS for 1 h in wells or under perfusion in the micro bioreactor. After incubation, culture supernatant was immediately frozen at -80°C and the total cell number was determined. Before analyzing the samples by HPLC, the supernatant was processed by solid phase extraction using Sephadex® G-50, according to the manufacturer's instructions. A volume of 0.5 µl sample was injected onto the HPLC system for analysis. The depletion of APAP was calculated by comparing the amount of APAP in each sample using the area under the chromatogram peak with the peak area of 20 mM APAP and normalized to the total number of cells.

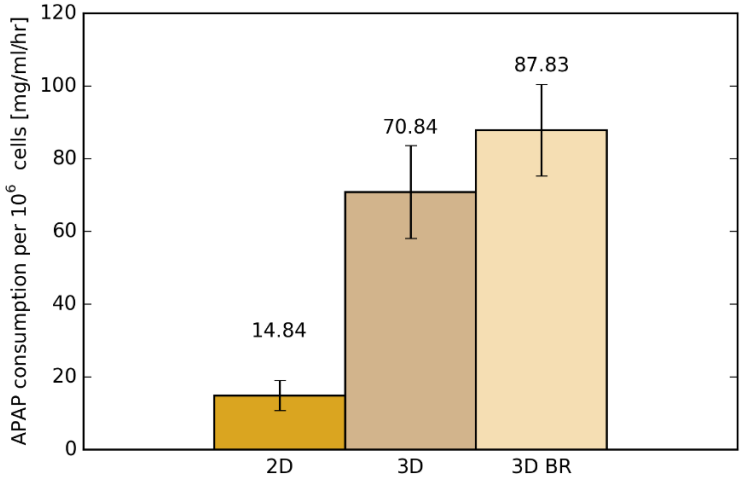

Figure 7.3 Consumption of APAP by HepaRG cells cultured in different formats (2D, 3D and 3D BR). The consumption of APAP in the medium was calculated by HPLC analysis before and after culture with the cells. Each experiment was replicated at least 3 times (mean ± SEM).

In comparison to 2D cultures, APAP consumption was significantly increased in static and perfused 3D cultures (Figure 7.3; 3D vs 2D: 4.8-fold; 3D BR vs 2D: 5.9-fold, respectively). These data correlate well with the observed higher toxicity of APAP in static and perfused 3D HepaRG cultures. Based on these results, it can be concluded that 3D cell cultures show enhanced metabolic activity compared to 2D cell cultures, which is in keeping with other reports that 3D cell culture provides an *in vivo*-like realistic extracellular microenvironment that modulates differentiation and cellular functionality [61]. The extracellular matrix conditions in our MatriGrid® helps to maintain CYP2E1 and CYP3A4 activities, which in turn enhance the metabolism of APAP.

7.3 Online flow ELISA with APAP

The influence of APAP on albumin secretion was measured using online flow ELISA and also by conventional MTP, and the results were compared. After 4 weeks of static culturing, differentiated HepaRG cells in scaffolds were inserted into the micro bioreactor either filled with Williams Medium E with 0.1% FCS, 5 µg/ml insulin, 5 x 10^{-5} M hydrocortisone hemisuccinate, 2 mM glutamine, 100 U/ml penicillin and 100 µg/ml streptomycin (vehicle) or the same medium supplemented with APAP in a final concentration of 5 mM. Micro bioreactors were mounted on the culturing unit and continuously perfused with aforementioned media using an integrated peristaltic pump using the flowrate of 25 µl/min over 96 h. To measure the initial albumin levels, cells were continuously perfused with control cell culture medium for the first 24 h. After the first automatic medium change and sampling run was completed, the cell culture was supplemented with 5 mM APAP for another 72 h. Automatic medium change and sampling was carried out every 24 h.

Albumin secretion increased in vehicle-treated HepaRG cells over 72 h (by more than 150% of the initial level, Figure 7.4). By contrast, albumin secretion in APAP-treated HepaRG cells decreased to approx. 50% of the initial value within the first 24 h and was further decreased to 24% after 72 h. The 50% decrease of albumin levels in the first 24 h correlated well with the APAP EC_{50} value of 2.8 mM after 24 h incubation with perfused HepaRG cells and measured using MTP ELISA (Figure 7.2, right).

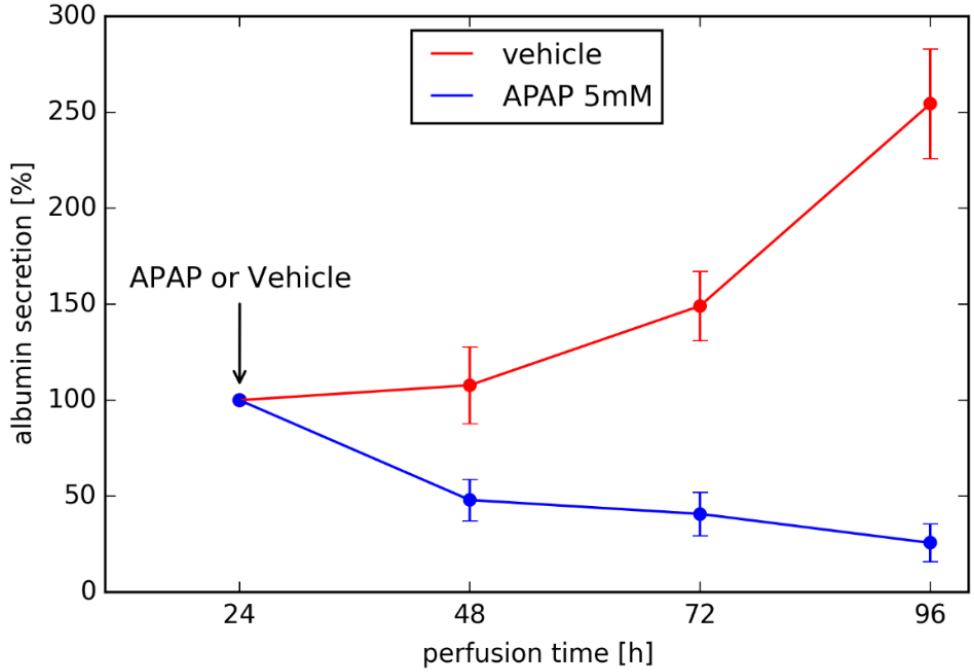

Figure 7.4 Albumin secretion measurement by automated culturing and analysis system with and without administration of 5 mM APAP over the period of 96 h. Values are from at least 3 experiments (mean ± SEM).

The albumin concentrations of the online samples were also measured using traditional MTP ELISA and confirmed the accuracy of the adapted flow-through ELISA (i.e. the values were not significantly different, as can be seen on Figure 7.5).

7.4 Intermediate summary

According to both measurements of toxicity, 3D cell cultures were more sensitive to APAP than 2D cultures, which is in line with the findings of others [2, 62]. The use of 3D cell cultures, especially when perfused, are more closely related to *in vivo* conditions, making them potentially a more relevant model than 2D cultures [2]. The functionality of the automated culturing and analysis device with polycarbonate-scaffold cultured HepaRG organoids was demonstrated. Their excellent hepatofunctional properties can be used with advantage in spheroid culture toxicity assays. The use of this robust 3D cell culturing tool provides advantages of automated medium change, minimal contamination risk, and additional labor-saving benefit especially in long-term experiments.

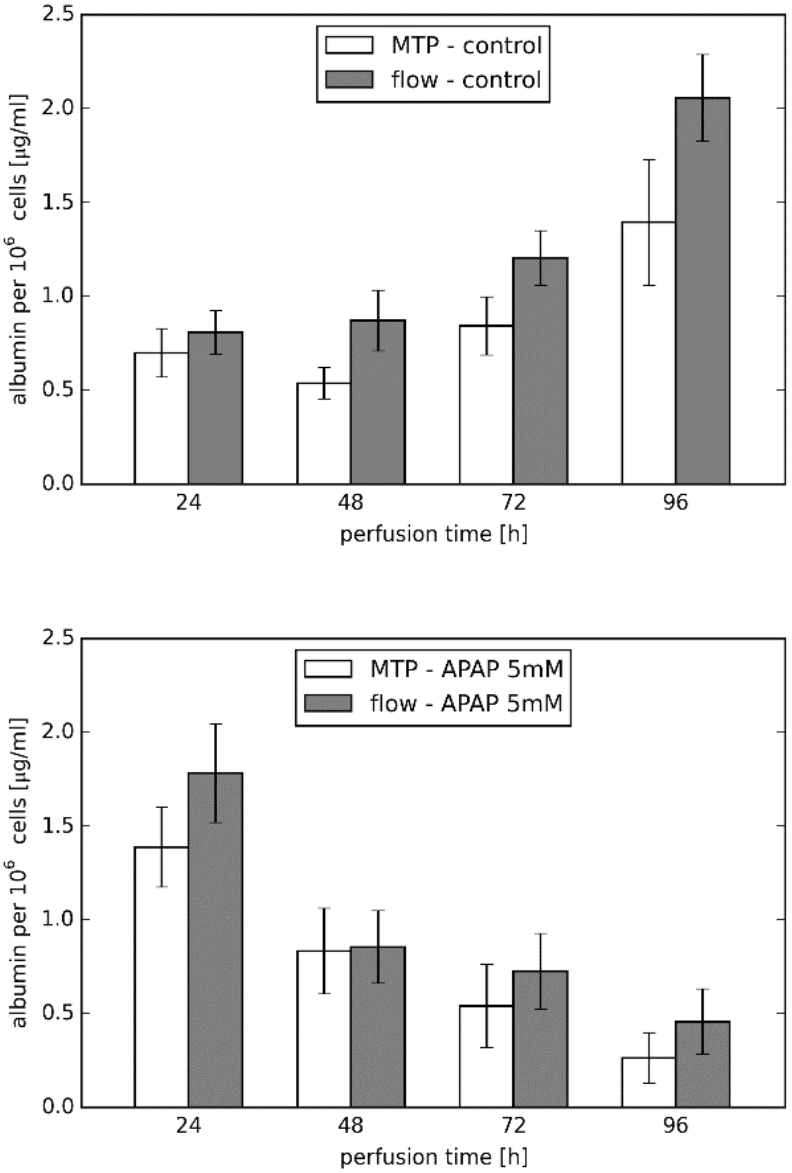

Figure 7.5 Validation of measured albumin levels by automated system with conventional MTP ELISA. Albumin secretion increased in vehicle-treated HepaRG cells (top) and decreased in APAP-treated HepaRG cells (bottom). Values are from at least 3 experiments (mean ± SEM).

8 Culturing and analytic system extensions

8.1 Introduction

The automated system as described in previous chapters was built as a prototype device to verify the automation possibilities of 3D cell culturing with on-demand automated analysis based on ELISA. This functionality was achieved after several design improvements (**Chapter 6**) and demonstrated by the APAP toxicity evaluation in 3D hepatocyte culture (**Chapter 7**). However, the development of this system should not stop at this point. This chapter addresses the most important parts of the automated system, which would further improve the assay accuracy, long term culturing reliability or the analysis throughput. Some of those ideas has been realized, but majority serve as the basis for further development of this automated system.

8.2 Parallelization of the culturing units

During the experiments described in the previous chapter, it was realized that the possibility of conducting the automated culturing in more than one bioreactor would accelerate the experimental work. In a typical toxicity study, one needs to perform cell culturing in the presence of the active compound as well as in the control culture. In both cases, the culturing should be replicated at least three times to provide statistical relevance. Because the control unit of the automated system does not support more than two culturing units, a way for efficient controlling of six or more culturing unit was needed.

A concept of "the smart driver" was chosen, where each culturing unit would be controlled by a separate driver module. The driver module should be able to directly control the peristaltic pump and solenoid valves of corresponding culturing unit. It should be equipped with suitable microcontroller, programmed to manage the complete functionality of the culturing unit in an autonomous way. This covers the culture perfusion, automated medium change and also medium sampling. This driver should be controlled by a host system via high level commands and it should provide culturing unit status information on demand. Six or more culturing units with corresponding drivers

should be controlled by one common controller module, which would also serve as the user interface (Figure 8.1).

This parallel culturing system was realized as described in the two following subchapters and it was found to be extremely useful for conducting long term cell culturing in multiple bioreactors. The automated medium change significantly reduces the risk of culture contamination and additionally provides labor saving benefit. This second aspect becomes significant in long term experiments requiring operation of six or more bioreactors.

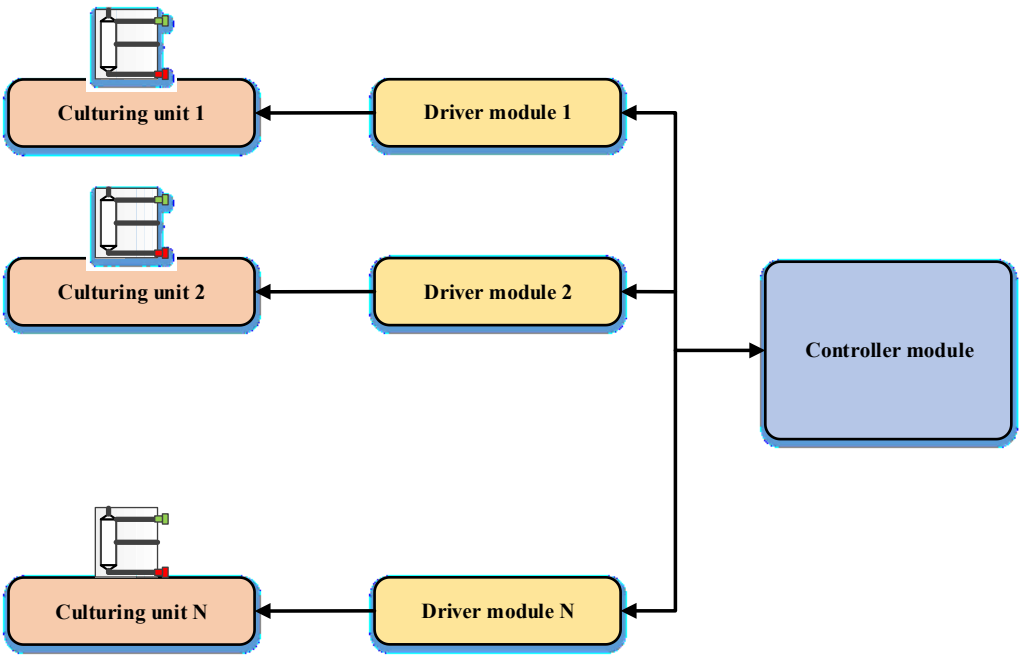

Figure 8.1 Block diagram of parallel operation of multiple culturing units.

8.2.1 Driver design for culturing unit

In contrast to the automated culturing and analysis system control unit, where the main objective was universality, the driver was designed with focus on simplicity and covers the culturing unit control requirements with little additional functionality. The driver is based on the 8-bit ATmega32 microcontroller with more than adequate CPU speed. It provides the following functionality:

- One stepper motor driver with microstepping support
- Eight solenoid valve drivers

- Two galvanically isolated digital inputs
- Two analog inputs
- Asynchronous serial interface (UART)
- I^2C serial interface

The communication between driver units and the controller is accomplished via I^2C serial interface. It is two-wire bi-directional serial interface with addressing. All drivers are connected to this interface in parallel and each driver must have assigned a unique address. The asynchronous serial interface serves for debugging purposes. The driver PCB was designed as two layers board with dimensions of 100 x 65 mm. The driver board is placed in aluminum housing and it is physically separated from the culturing unit. The main reason for that is incompatibility of power electronics with humid incubator environment. Figure 8.2 shows the assembled driver board without housing. The driver schematic can be found in the Appendix 6 and the corresponding printed circuit board layout in the Appendix 7.

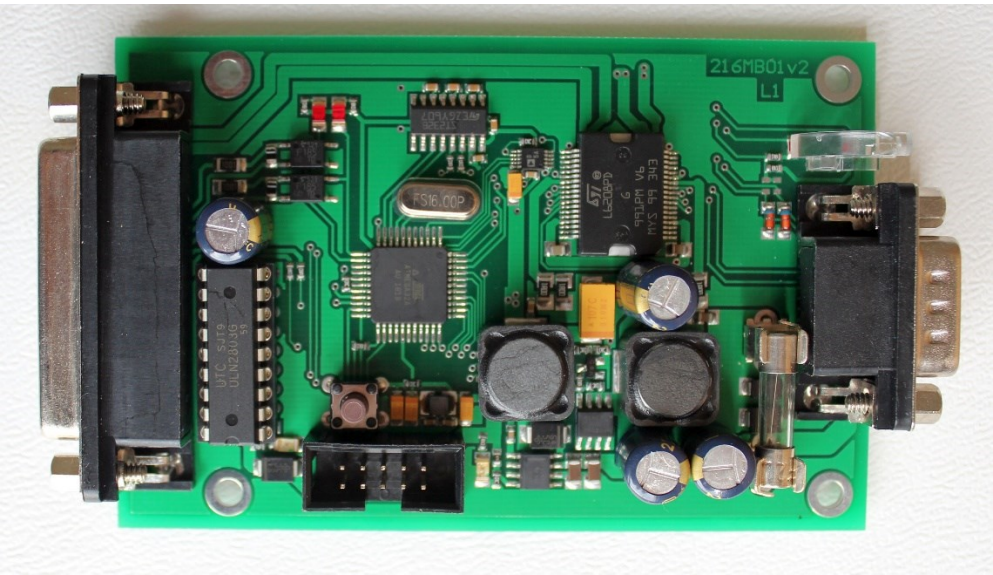

Figure 8.2 The assembled culturing unit driver module.

Similar to the automated system control unit, the driver code comprises two parts – the bootloader and the driver application, and the source code was written in C programming language. The bootloader simplifies the application upgrade procedure, as no special tool is necessary and the upgrade is made via the UART interface. The recent application version is v1.1 and it contains more than 2000 lines of code (excluding the bootloader).

The driver source code is listed in the Appendix 10. The implemented command set for communicating with the control unit is listed in Table 8.1.

Table 8.1 Implemented command set of the culturing unit driver module (I^2C interface)

Command name	Command code (Hex)	Number of parameter bytes	Parameter: (valid range) - parameter description	Command description
COM_TEST	0x30	0		Toggles red LED
COM_BRCONTROL	0x31	1	P1 – new state: (0 or 1): 0= perfusion OFF; 1= perfusion ON	Switches ON or OFF bioreactor perfusion
COM_BRSTOP	0x32	0		Cancels any BR operation (perfusion, medium change or sampling)
COM_BRSPEED	0x33	2	P1 - speed: (1 - 500) - 16-bit value of BR perfusion speed	Sets BR perfusion speed
COM_PREPSAMPLE	0x34	0		Starts medium sampling operation
COM_BRCHANGEMED	0x35	4	P1 - volume: (-10000 - 10000) -16-bit value, pumped volume in µl, negative number means opposite direction P2 - speed: (1 - 500) - 16-bit value, pumping speed in µl/min	Start the medium change operation by pumping selected volume at selected speed
COM_VALVE	0x38	2	P1 – valve index: (1 - 8) - selects the valve to be controlled P2 – new state: (0 or 1) - 0= turn the valve OFF, 1= turn the valve ON	controls the 8 solenoid valves
COM_STARTPUMP	0x39	4	P1 - volume: (-10000 - 10000) -16-bit value, pumped volume in µl, negative number means opposite direction P2 - speed: (0 - 500) - 16-bit value, pumping speed in µl/min; 0 = stop the pump	Start perfusion pump to pump selected volume at selected speed
COM_BRSTATUS	0x41	0		Reads the culture unit status

8.2.2 The common control module for culturing units

The control module provides a convenient way for a user to interact with multiple culturing units. It works in real time, supports simultaneous controlling up to 8 culturing units with corresponding drivers, and allows medium change scheduling in regular intervals independently for each culturing unit. In addition, it allows manual control of all solenoid valves and the pump of any connected culturing unit, as well as the manual (i.e. immediate or non-scheduled) medium change. The control unit periodically monitors the state of all culturing units and displays corresponding information or the medium change progress for all connected modules (see Figure 8.3). The graphic TFT display with resolution of 800 x 480 pixels and associated resistive touchscreen provides a convenient user interface. The internal real time clock (RTC) module allows user to program multiple medium change events on specific dates or hours.

The control unit was built using commercially available electronic modules as it did not require any special or precise functionality. The construction is based on the Arduino® Mega 2560 board and 5-inch TFT display module with resolution of 800x480 pixels and integrated touchscreen. The ITDB02 Arduino MEGA Shield v2.0 is needed to interface the display module to Arduino board. Additional RTC module based on the DS3231 chip is connected directly to Arduino board via the I^2C interface.

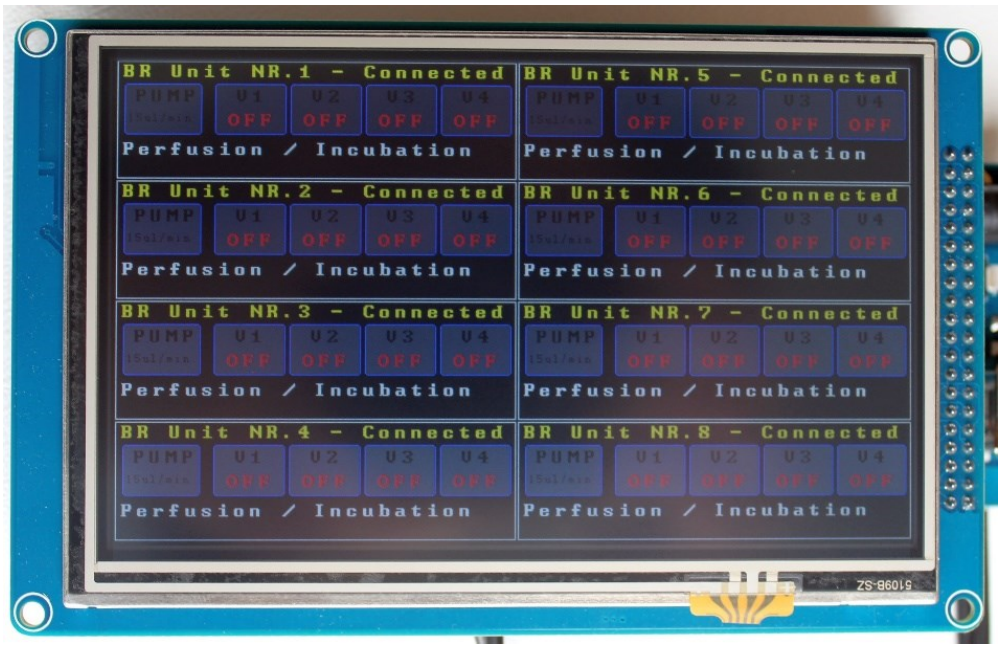

Figure 8.3 The control unit displaying status information from 8 culturing unit, showing the perfusion in progress with perfusion speed 15 µl/min.

The whole assembly is mounted to aluminum housing for convenient handling. The control unit was programmed using the Arduino integrated development environment (IDE). The recent version of the source code is v1.2 and contains approximately 1500 lines of code. This source code is listed in Appendix 11.

8.3 Increasing throughput of the analyzer module

The analyzer module has 7 measurement channels which can be used either for standard curve or the bioreactor sample measurement. The reliable standard curve measurement requires a minimum of 5 points (including the blank), which leaves 2 channels for bioreactor samples. This configuration was sufficient to operate whole automated system with two culturing units connected to it. However, after the extension of the culturing

system to support up to 8 culturing units with corresponding bioreactors, a possibility of measuring all 8 samples in one run would save significand amount of time, and simultaneously increase the accuracy of sample comparison as all of them would be referenced to same standard curve. Additionally, each point of standard curve could be measured in triplicates, which would also improve standard curve precision. That way, a minimum of 23 (8+15) channels would be required to be analyzed in a single run.

The simplest way to increase the analyzer throughput would be to keep the operation principle and construction same but extend the number of capillary channels. This approach is straightforward, requiring the extension of the capillary and waste manifold for more branches, the addition of more capillaries to the sample holder, the addition of one more solenoid valve for every new capillary, and the appropriate adjustment of the control unit software. This way of system extension has its limits mainly for two reasons: firstly - a limited number of capillaries will fit to the rotary sample changer, and secondly - as the number of channel increases, the time needed to sequentially fill all capillaries increases proportionally. This second point would mean significant increase of total assay time. For those reason the practical upper limit for channel number increase without conceptual change of the analyzer architecture is approximately 16 channels.

To increase the number of channels beyond this limit, preferably to 24, 36, or even more, the analyzer working principle must be changed to extensively use parallel operation where possible. The following list contains some key points for the design of such analyzer:

- Filling of capillary with reagent, sample or washing buffer should be done in parallel for all channels
- Readout should be done for all channels in parallel. This implies to use one readout device per channel. For this reason, the readout design should be kept as simple as possible, interfacing of optical sensors to capillaries should preferably use the optical fibers. Additionally, all readout devices will need to be calibrated.
- Use preferably modular design. Each module may contain 2 to 6 channels (capillaries) with associated readout devices and isolation valves. Each module should be provided with sample fluidic input or container. The modules should be connected to common manifold distributing the reagents.

- The capillaries are disposable, therefore the design should allow quick and easy replacement of used capillaries for new ones.

It is obvious that the design of such modular analyzer for 24 channels will be challenging. Nevertheless, the experience gained during the development of the prototype analyzer unit will make this task easier.

8.4 Other future system improvements

8.4.1 Adding sensorics to the culturing unit

Continuous monitoring of the environmental conditions of the cell culture in the bioreactor will certainly increase culturing reliability. A simple flow sensor inserted to the perfusion loop would provide valuable and early information about any failure of the perfusion. High accuracy sensors are not needed for this purpose. A simple mass flow sensor based on the thermal principle would be fully adequate.

Another useful environmental parameter is the monitoring of the oxygen saturation of the cell culture medium. A trial experiments were conducted using the OXY-4 mini device (PreSens, Precision Sensing GmbH), which based on the noninvasive fluorescence quenching measurement principle. Two SP-PSt3-NAU sensor spots were placed inside the bioreactor, one of the sensors was located in the bioreactor reservoir compartment near the perfusion inlet, while the other sensor was located on near the fluidic outlet. That way, the oxygen concentration difference representing cell culture oxygen consumption can be measured. The oxygen levels were measured during the 72h long culturing period of HepaRG cell with automated medium change every 24h (Figure 8.4). The oxygen consumption can serve as indirect indicator of cell culture metabolic activity.

Additionally, the acidity (pH) of the cell culture medium can be monitored either to prove correct culturing conditions or to provide early warning signals to generate medium change requests. The sensing principle can be used either using the pH sensor spots (e.g. SP-LG1-SA, PreSens), or spectrophotometrically if the culture medium is supplemented with pH indicator (e.g. phenol red). Both approaches are non-invasive.

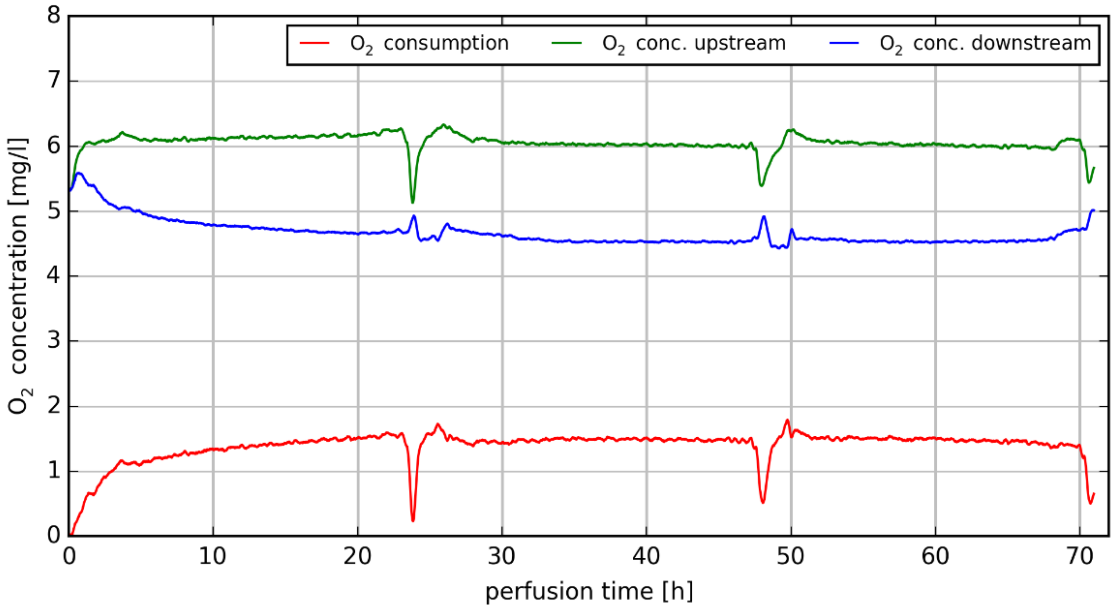

Figure 8.4 The oxygen levels measured in the 3D HepaRG cell culture located in the bioreactor. Automated medium change was performed every 24h and is visible as negative glitch on the consumption curve (red).

8.4.2 Temperature management of the analyzer unit

Keeping the temperature constant is important for consistent measurements. The prototype analyzer unit does not have the capability of thermostating the assay. This is usually not a problem as far as the measurement is conducted in the laboratory with controlled temperature. However, keeping the reagents and capillaries during the assay at constant temperature would increase the measurement accuracy and analyzer robustness.

Another improvement, especially for long term experiments, would be embedding the reagents cooling option to the analyzer. Most of the reagents have limited shelf-life at room temperature, so keeping them in a cooled state (e.g. 4 ºC) would enable multiple runs without the need to replace the reagents.

8.4.3 Analyzer unit – on-site substrate preparation

The shelf-life of ADHP substrates is 30 minutes at room temperature. Because the assay run duration is approximately 3 hours, the substrate must be prepared fresh and placed to the analyzer shortly before it will be consumed (protocol step 13, see **Chapter 3.2.4**). A simple way of mixing 3 substrate components in pre-measured quantities would further

increase the comfort of analyzer operation. The components should be preferably stored in cooled state and brought to the reaction temperature after mixing.

8.4.4 Analyzer unit – further optimization of cleaning protocol

The recent analyzer cleaning procedure which must be conducted after each assay was optimized with respect to cleaning efficiency. However, this cleaning procedure takes approximately three hours to complete, so further optimization with respect to the cleaning duration would shorten the minimal time between two successive assays.

8.4.5 Control unit – implementation of curve fitting algorithms

The curve fitting was done using the external software tools (MATLAB or MS Excel). However, extending the control unit software to include four- or five- parameter logistic regression algorithms should be straightforward. The ARM CORTEX M3 architecture-based microcontroller used in the control unit provides sufficient computational power to allow this option.

8.4.6 Analyzer unit – extending the readout system for absorbance measurement possibility

The current readout configuration allows using fluorescent or luminescent substrates. However, the colorimetric substrates are very common and extending the readout system to support those substrates would increase the range of assays the analyzer unit would support. The conceptual layout of combined fluorometric and absorbance sensor is shown on Figure 8.5. The redesign of the readout system will be required for extending the analyzer to more than 16 channels, and utilization of plastic optical fibers (POF) will make such miniaturized and combined sensor design feasible.

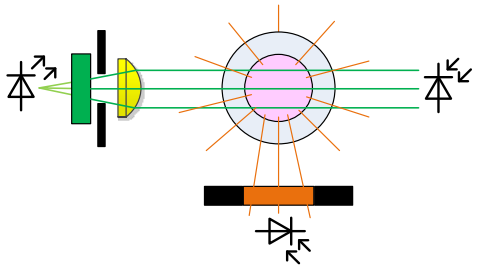

Figure 8.5 The layout of combined fluorescence and absorbance sensor. The excitation LED is on the left side, the light passes through the excitation filter and is focused on the fluidic channel. The emission light passes at 90º angle through the emission filter and is detected by photodiode (bottom side). The absorbance is measured by photodiode aligned with the fluidic channel and emission light source (right side).

8.4.7 Miniaturization of the analyzer unit

One of the design goal of the whole automated system was to make the whole system portable. Miniaturization of the analyzer unit is generally required. The current design of the analyzer prototype represents by no means the limit in down-scaling. It is not based on microfluidic chip technology, which might be seen as obvious step towards miniaturization. While it is true that using the microfluidic chip technology would decrease the dead volume of the fluidic network, most of the space will remain to be occupied by valves. The use of on-chip integrated pneumatic valves does not provide overall space advantage, because switching of individual pressurized lines must be provided by another set of some off-chip solenoid valves. Additionally, handling and connecting of delicate microfluidic chips is more difficult, which may result in handling discomfort for analyzer operator. The key factor to scale down in this case will be the replacement of active fluidic components such as the peristaltic pumps and solenoid valves. The current analyzer design uses two peristaltic pumps, which are larger than required for this application. Similarly, more than 20 solenoid valves with the diameter 19 mm were used. Replacement of those valves with types with smaller footprint will save considerable amount of space. If the decision will be made to use microfluidic chip technology, care must be taken to use compatible materials with respect to low-protein binding properties or the compatibility with cleaning agents. Materials based on PDMS should be avoided because of the problems with the analyte adsorption on the surface and associated cleaning difficulties.

9 Application possibilities of the culturing and analytical system

In this chapter, application possibilities of the 3D-culturing and analytical system are described. The overview is not exhaustive and covers main areas of use. The system is very universal and it can be adapted to many other specific tasks. The applications are described in a general way and references to specific examples are provided if related experimental work was performed.

9.1 Applications of the Cell culturing systems

9.1.1 Drug toxicity tests

The combination of the automated culturing unit and the analytic unit provides many benefits for conducting short- and long-term toxicity assays with 3D cultures. This was demonstrated in acetaminophen toxicity study on 3D HepaRG cell culture [63]. In the study the culturing was performed in actively perfused bioreactor with automated medium exchange every 24 hours. The concentration of hepatocyte metabolite albumin was repeatedly determined by analytical module using the flow-through ELISA. High sensitivity of the assay combined with low protein binding materials for fluidics allowed to measure albumin concentrations as low as one nanogram per milliliter.
Presented tool can be used in variety of other toxicology studies such as screening of anti-cancer therapeutics using the 3D cell cultures or assessing of toxicity of various nanoparticles or toxic substance on 3D cell culture. The ease of parallelization and automation of the drug application to the cell culture makes this system ideal for toxicity assays.

9.1.2 Microenvironments testing

The cellular behavior of organoid in 3D cell culture is influenced by particular geometrical and biochemical boundaries of the growth microenvironment [64]. In the case of scaffold-based 3D cell cultures, the properties of cultured cells are influenced by physical and chemical properties of supporting scaffold. Therefore, another broad field of application of this novel culturing system is to evaluate various scaffold morphologies,

materials and its chemical modifications on cell proliferation, viability or differentiation. Possible applications include cell co-culturing, including organ-on-a-chip operation. Moreover, connecting multiple bioreactors in serial manner enables more complex body-on-a-chip experiments. In a previous work, human neuroblastoma cell lines (BE(2)-C, IMR-32) were grown on MatriGrid® scaffolds in the form of spheroids [65]. Although generally it is known that 3D-cultures are difficult to handle, the use of 3D-culturing units reduces and simplifies the handling operations and improves the consistency of the experiment results.

9.1.3 Influence of the fluidic shear stress in the cell culture

The fluid flow can directly influence cell proliferation [66] by means of mechanical actions such as compression, shear stress or pressure. These are important factors for organ development and function [67]. Advanced cell culture techniques such as previously mentioned organ-on-chips offer the possibility to control some of these factors. Therefore, advanced culturing techniques are suitable for studying biological phenomena that depend on tissue microarchitecture and perfusion [68]. The ability of the culture unit to precisely control of the perfusion flow rate in a time dependent manner allows it to be used for advanced cell culturing applications. Previous work with cooperation with the Jena University Hospital on the placenta explants revealed that the placenta tissue in the explants is very sensitive to fluidic shear stress. In order to increase the cell viability in the explant, a new adapted scaffold named TissGrid® was developed, which incorporated a protective cylinder made of porous film [69]. The explant was inserted inside the cylinder, where it was protected from excessive fluidic shear stress, but thanks to the porous scaffold material, the explant remained supplied with nutrients.

9.1.4 Cell line maintenance

One of the frequent operations in the biological laboratory is the cell line maintenance and passaging. The culturing unit can be used with advantage to maintain sensitive cell lines requiring perfusion with oxygenated medium. Continuous perfusion prevents consumptive oxygen depletion in cultures sensitive to hypoxia such as hepatocytes [70]. Automated medium change provides additional benefits of reducing the risk of contamination and simultaneously reducing the manual labor.

9.2 Applications of the analytical module

The analytical module was optimized for flow-through albumin florescence assay. The choice of Amplex™ Red (ADHP) as the fluorescence substrate was based on its desirable properties, such as chemical and thermal stability, low background and increased dynamic range and fluorescence emission outside the range of compound autofluorescence. Because of these advantaged the substrate, it is used in many commercially available assays. A few examples of fluorescent assays using the ADHP as substrate are listed here:

- Glucose / Glucose oxidase assay (A22189, ThermoFischer)
- Cholesterol assay (A12216, ThermoFischer)
- Catalase assay (A22180, ThermoFischer)
- Hydrogen peroxide / Peroxidase assay (A22188, ThermoFischer)
- Acetylcholine / Acetylcholinesterase assay (A12217, ThermoFischer)
- Galactose / Galactose oxidase assay (A22179, ThermoFischer)
- Glutamic acid / Glutamate oxidase assay (A12221, ThermoFischer)
- Monoamine oxidase assay (A12214, ThermoFischer)
- Neuraminidase assay (A22178, ThermoFischer)
- Phospholipase D assay (A12219, ThermoFischer)
- Phosphate assay (P22061, ThermoFischer)
- Pyrophosphate assay (P22062, ThermoFischer)
- Sphingomyelinase assay (A12220, ThermoFischer)
- Uric acid / Uricase assay (A22181, ThermoFischer)
- Xanthine / Xanthine oxidase assay (A22182, ThermoFischer)

Additionally, LDH assay using ADHP related substrate resazurin (C20302, ThermoFischer) is also available as commercial product. Because all these assays are based on the same fluorescent molecule (resorufin) as the albumin assay which the analytical unit was designed for, no change on the readout part (fluorimeter) would be required if the analytical unit should be adapted for one of these assays. The fluidic part is flexible in design and it would not require extensive modifications to support these assays. Obviously, the software of the control unit would require extensions in order to support any additional assay.

Other fluorophores than resorufin can be supported if the fluorimeter would be appropriately modified. For example, to adapt the fluorimeter for one other common fluorophore – fluorescein, the excitation laser wavelength would need to be changed to 450 nm or 488nm (e.g. L450P1600MM or L488P60, Thorlabs) and the emission filter would also need to be replaced to a suitable bandpass type (e.g. 513-556 nm, #67-017, Edmund Optics). Thus, by replacing two components, the fluorimeter can be adapted to an assay using different fluorophore.

10 Summary

"Genes are effectively one-dimensional. If you write down the sequence of A, C, G and T, that's kind of what you need to know about that gene. But proteins are three-dimensional. They have to be because we are three-dimensional, and we're made of those proteins. Otherwise we'd all sort of be linear, unimaginably weird creatures"

(Francis Collins, led the Human Genome Project, director of the National Institutes of Health in Bethesda, Maryland, United States, 2001)

The observation made by Francis Collins, namely that humans are three-dimensional, and not linear, unimaginably weird creatures, is amusing. Similarly, this thesis was started with an amusing quote from the book *Flatland*, in which a series of characters interact in a purely two-dimensional world, but a three-dimensional world is later discovered. These quotes, which juxtapose 1D and 2D environments with 3D environments, are amusing because the nature of our 3D environment is so self-evidently obvious to us. Here, it would be a poor transition to now simply say, "so stop using 2D cultures, because 3D is…, etc". A more relevant transition would be to say, "although our 3D environment is so self-evidently obvious, is there a justification for reducing the dimensionality?". The answer is certainly yes. There are many benefits of 2D systems (cost, ease of observation, ease of measurement) and these benefits have been discussed in the context of cell culturing in this text. For simpler systems, and simpler questions, the complexity of a 3D environment (and the inclusion of automation) is not always necessary.

However, at the clinical level, humans are three-dimensional, and the issue is the following. Although unautomated cell culturing systems in 2D can be used for some pre-clinical questions, it is without question that cells act in much different manner when surrounded by other cells in 3D. Therefore, **the problem is predictivity**. By introducing a 3D system more representative of the clinical environment with a more complex technology, this may reduce both the monetary cost and the degree of failures of drugs and therapies at the level clinical trials which previously passed 2D screening systems.

Certainly, pharmaceutical companies dedicate astonishing amounts of money to R&D each year for pre-clinical drug trials, with the majority of such trials ending in failure once transferred to the clinical level. The introduction of 3D systems may give a better degree of predictivity in trails at the pre-clinical level. Essentially, as stated in the introduction, ***a culture* should be 3D to increase predictivity, but *the culturing* should be automated to increase through-put and applicability to large-scale pre-clinical testing**.

Indeed, there is currently a genuine renaissance in attempts to include forms of automation, or at least so-called *online* measurement, which allows the testing of certain parameters of the biological system without actually opening or disturbing the system. "New generation" organ-on-a-chip systems are equipped with biosensors or bioimaging that enable the online monitoring of pH and oxygen [71, 72], the cellular metabolic state [73, 74] and the detection of cell-derived analytes in the culture medium by microfluidic enzyme linked immunosorbent assay (ELISA) [13-15]. Thus, cells do not need to be removed from the perfused culture systems to define drug toxicity and cellular health. In particular, quantitative analysis of cell-secreted proteins by **microfluidic ELISA provides a novel method of measuring non-invasively the toxicity of drugs to cells in complex culture systems** [63] where the removal of cells or the opening/exposure of the system could compromise long-term experiments.

In the presented study of this text, **a fully-automated and robust culturing system was developed, which combined 3D cell culturing with automated perfusion, medium change, and sampling, followed by an automated flow-ELISA for detection of cell-derived albumin for the assessment of hepatotoxicity**. The focus was on developing a scaffold-based 3D culture and analysis system which allowed excellent exposure of the cells to the applied drug and minimized adsorption and absorption of small molecules, drugs, and biomolecules by the system. The ELISA analyzer module was designed in such a way that almost *any commercially available ELISA assay kit* can be used with this system and therefore made available to a wide range of users.

The presented study describes the development and operation of an automated 3D culture system with a non-invasive online analysis system and its relevance compared to routinely-performed standard sandwich ELISA protocols. It was demonstrated that 3D cultures of HepaRG cells differ from 2D monolayer cultures in sensitivity to toxic

compounds, making them appropriate for online toxicity studies. Finally, the proper system functionality was verified using the applicability of the device in online-kinetics measurement of albumin secretion as a proof-of-concept using 3D HepaRG cultures perfused with APAP over a period of 96 hours. This study demonstrates how this highly integrated in vitro system can be used for drug toxicity tests and shows the potential for adaptation of the online-detection to include other secreted proteins, such as hormones and signaling molecules from 3D mono- and co-cultures.

References

1. Brenner, S., *Life sentences: Detective Rummage investigates.* Genome Biol, 2002. **3**(9): p. comment1013 1-2.

2. Fey, S.J. and K. Wrzesinski, *Determination of Drug Toxicity Using 3D Spheroids Constructed From an Immortal Human Hepatocyte Cell Line.* Toxicol Sci, 2012. **127**(2): p. 403-11.

3. Jensen, C. and Y. Teng, *Is It Time to Start Transitioning From 2D to 3D Cell Culture?* Front Mol Biosci, 2020. **7**: p. 33.

4. Fernekorn, U., et al., *Microbioreactor design for 3-D cell cultivation to create a pharmacological screening system.* Engineering in Life Sciences, 2011. **11**(2): p. 133-139.

5. Geckil, H., et al., *Engineering hydrogels as extracellular matrix mimics.* Nanomedicine (Lond), 2010. **5**(3): p. 469-84.

6. Berthier, E., E.W. Young, and D. Beebe, *Engineers are from PDMS-land, Biologists are from Polystyrenia.* Lab Chip, 2012. **12**(7): p. 1224-37.

7. Nilsson, M., H. Håkanson, and B. Mattiasson, *Flow-injection ELISA for process monitoring and control.* Analytica Chimica Acta, 1991. **249**(1): p. 163-168.

8. Nilsson, M., et al., *On-line monitoring of product concentration by flow-ELISA in an integrated fermentation and purification process.* Journal of Fermentation and Bioengineering, 1994. **78**(5): p. 356-360.

9. Ramachandran, S., et al., *A Rapid, Multiplexed, High-Throughput Flow-Through Membrane Immunoassay: A Convenient Alternative to ELISA.* Diagnostics, 2013. **3**(2): p. 244-260.

10. Sani, A., C. Cao, and D. Cui, *Toxicity of gold nanoparticles (AuNPs): A review.* Biochem Biophys Rep, 2021. **26**: p. 100991.

11. Sreedevi, C., et al., *Development and evaluation of flow through assay for detection of antibodies against porcine cysticercosis*. Vol. 28. 2011. 160-170.

12. Lebogang, L., et al., *Electrochemical Flow-ELISA for Rapid and Sensitive Determination of Microcystin-LR Using Automated Sequential Injection System.* Sensors (Basel), 2017. **17**(7).

13. Riahi, R., et al., *Automated microfluidic platform of bead-based electrochemical immunosensor integrated with bioreactor for continual monitoring of cell secreted biomarkers.* Scientific Reports, 2016. **6**: p. 24598.

14. Shin, S.R., et al., *Label-Free and Regenerative Electrochemical Microfluidic Biosensors for Continual Monitoring of Cell Secretomes.* Advanced Science, 2017. **4**(5): p. 1600522-n/a.

15. Zhang, Y.S., et al., *Multisensor-integrated organs-on-chips platform for automated and continual in situ monitoring of organoid behaviors.* Proceedings of the National Academy of Sciences, 2017. **114**(12): p. E2293-E2302.

16. Halldorsson, S., et al., *Advantages and challenges of microfluidic cell culture in polydimethylsiloxane devices.* Biosens Bioelectron, 2015. **63**: p. 218-31.

17. LeCluyse, E.L., et al., *Organotypic liver culture models: meeting current challenges in toxicity testing.* Crit Rev Toxicol, 2012. **42**(6): p. 501-48.

18. Shekarchi, I.C., et al., *Evaluation of various plastic microtiter plates with measles, toxoplasma, and gamma globulin antigens in enzyme-linked immunosorbent assays.* J Clin Microbiol, 1984. **19**(2): p. 89-96.

19. *Biopharmaceutical Products*. 2013, Sani-Tech West, Inc.

20. Kay, A., *Chapter 1 - Introduction and Review of Statistics*, in *Operational Amplifier Noise*. 2012, Newnes: Boston. p. 1-11.

21. M., J., *Photodetection and Measurement: Maximizing Performance in Optical System.* 2003, New York: McGraw-Hill.

22. Gallant, M.I., *Transimpedance Noise Calculation*. 2012, JavaScience Consulting.

23. Oppenheim, A.V., *Discrete-time Signal Processing*. 1998: Prentice-Hall.

24. Collura, T.F., *Averaging, Noise, and Statistics*, in *Comprehensive clinical Neurophysilology*, K. Levin, Luders, H., Editor. 1995, Elsevier.

25. STMicroelectronics, *STM32F103xC STM32F103xD STM32F103xE High-density performance line ARM-based 32-bit MCU with 256 to 512KB Flash, USB, CAN, 11 timers, 3 ADCs, 13 communication interfaces*. April 2011.

26. Altium, *Altium Designer*. 2017.

27. CETONI, *neMESYS LOW PRESSURE SYRINGE PUMP*. 2015.

28. Hamamatsu Photonics, *CMOS linear image sensor S8377/S8378 series*. 2002.

29. STMicroelectronics, *L6208 DMOS driver for bipolar stepper motor*. 2014.

30. NXP Semiconductors, *MPX5100 MPXV5100 Series Integrated Silicon Pressure Sensor On-Chip Signal Conditioned, Temperature Compensated and Calibrated*. 2010.

31. Analog Devices, *AD7147 CapTouch Programmable Controller for Single-Electrode Capacitance Sensors*. 2015.

32. Feaser, *OpenBLT GNU GPL Bootloader*. 2017.

33. Birkler, J., *[avr-libc-dev] Re: printf in avr-libc*. 2002.

34. Fischer, M., *YAGARTO*. 2012.

35. Foundation, T.E., *Eclipse Juno*. 2017.

36. Rath, D., *Open On-Chip Debugger*. 2017.

37. Rath, D., *Open On-Chip Debugger*, in *Department of Computer Science*. 2005, University of Applied Sciences Augsburg: Augsburg.

38. Diamandis, E.P. and T.K. Christopoulos, *Immunoassay*. 1996, San Diego: Academic Press.

39. Dudley, R.A., et al., *Guidelines for immunoassay data processing.* Clin Chem, 1985. **31**(8): p. 1264-71.

40. Wild, D., *The immunoassay handbook : theory and applications of ligand binding, ELISA, and related techniques*. 2013, Oxford: Elsevier.

41. Gottschalk, P.G. and J.R. Dunn, *The five-parameter logistic: a characterization and comparison with the four-parameter logistic.* Anal Biochem, 2005. **343**(1): p. 54-65.

42. Swart, A. *Fully automated spreadsheet for 4– and 5– parameter logistics curve fitting of bioassay calibrations.* 2012 [cited 2017 May, 5th]; Available from: http://rheumatologie-neuss.net/index_files/ELISA%20AUTO%20CURVE%20FIT.xlsm.

43. Reynolds, J.A. and C. Tanford, *Binding of Dodecyl Sulfate to Proteins at High Binding Ratios. Possible Implications for the State of Proteins in Biological Membranes.* Proceedings of the National Academy of Sciences of the United States of America, 1970. **66**(3): p. 1002-1007.

44. Takehara, A., H. Urano, and S. Fukuzaki, *Cleaning of Alumina Fouled with Bovine Serum Albumin by the Combined Use of Gaseous Ozone and Alkaline Electrolyzed Water.* Biocontrol Science, 2001. **6**(2): p. 103-106.

45. GE Healthcare Life Sciences, *High-throughput process development for design of cleaning-in-place protocols*. 2010.

46. Roth, R.A. and P.E. Ganey, *Intrinsic versus idiosyncratic drug-induced hepatotoxicity--two villains or one?* J Pharmacol Exp Ther, 2010. **332**(3): p. 692-7.

47. Blieden, M., et al., *A perspective on the epidemiology of acetaminophen exposure and toxicity in the United States.* Expert Rev Clin Pharmacol, 2014. **7**(3): p. 341-8.

48. Prescott, L.F., *Kinetics and metabolism of paracetamol and phenacetin.* British Journal of Clinical Pharmacology, 1980. **10**(Suppl 2): p. 291S-298S.

49. Cheung, C., et al., *The cyp2e1-humanized transgenic mouse: role of cyp2e1 in acetaminophen hepatotoxicity.* Drug Metab Dispos, 2005. **33**(3): p. 449-57.

50. Cohen, S.D., et al., *Selective protein covalent binding and target organ toxicity.* Toxicol Appl Pharmacol, 1997. **143**(1): p. 1-12.

51. Qiu, Y., L.Z. Benet, and A.L. Burlingame, *Identification of the hepatic protein targets of reactive metabolites of acetaminophen in vivo in mice using two-dimensional gel electrophoresis and mass spectrometry.* J Biol Chem, 1998. **273**(28): p. 17940-53.

52. Qiu, Y., L.Z. Benet, and A.L. Burlingame, *Identification of hepatic protein targets of the reactive metabolites of the non-hepatotoxic regioisomer of acetaminophen, 3'-hydroxyacetanilide, in the mouse in vivo using two-dimensional gel electrophoresis and mass spectrometry.* Adv Exp Med Biol, 2001. **500**: p. 663-73.

53. Liu, Z.X. and N. Kaplowitz, *Role of innate immunity in acetaminophen-induced hepatotoxicity.* Expert Opin Drug Metab Toxicol, 2006. **2**(4): p. 493-503.

54. Jaeschke, H., *Glutathione disulfide formation and oxidant stress during acetaminophen-induced hepatotoxicity in mice in vivo: the protective effect of allopurinol.* J Pharmacol Exp Ther, 1990. **255**(3): p. 935-41.

55. Cover, C., et al., *Peroxynitrite-induced mitochondrial and endonuclease-mediated nuclear DNA damage in acetaminophen hepatotoxicity.* J Pharmacol Exp Ther, 2005. **315**(2): p. 879-87.

56. Hanawa, N., et al., *Role of JNK Translocation to Mitochondria Leading to Inhibition of Mitochondria Bioenergetics in Acetaminophen-induced Liver Injury.* The Journal of Biological Chemistry, 2008. **283**(20): p. 13565-13577.

57. Reid, A.B., et al., *Mechanisms of acetaminophen-induced hepatotoxicity: role of oxidative stress and mitochondrial permeability transition in freshly isolated mouse hepatocytes.* J Pharmacol Exp Ther, 2005. **312**(2): p. 509-16.

58. Ullrich, A., et al., *Use of a standardised and validated long-term human hepatocyte culture system for repetitive analyses of drugs: repeated administrations of acetaminophen reduces albumin and urea secretion.* Altex, 2007. **24**(1): p. 35-40.

59. Ullrich, A., et al., *Long term cultures of primary human hepatocytes as an alternative to drug testing in animals.* Altex, 2009. **26**(4): p. 295-302.

60. LeBlanc, A., et al., *Absolute quantitation of NAPQI-modified rat serum albumin by LC-MS/MS: monitoring acetaminophen covalent binding in vivo.* Chem Res Toxicol, 2014. **27**(9): p. 1632-9.

61. Pampaloni, F., E.G. Reynaud, and E.H. Stelzer, *The third dimension bridges the gap between cell culture and live tissue.* Nat Rev Mol Cell Biol, 2007. **8**(10): p. 839-45.

62. Schyschka, L., et al., *Hepatic 3D cultures but not 2D cultures preserve specific transporter activity for acetaminophen-induced hepatotoxicity.* Arch Toxicol, 2013. **87**(8): p. 1581-93.

63. Baca, M., et al., *Automated Analysis of Acetaminophen Toxicity on 3D HepaRG Cell Culture in Microbioreactor.* Bioengineering (Basel), 2022. **9**(5).

64. Habanjar, O., et al., *3D Cell Culture Systems: Tumor Application, Advantages, and Disadvantages.* Int J Mol Sci, 2021. **22**(22).

65. Bingel, C., et al., *Three-dimensional tumor cell growth stimulates autophagic flux and recapitulates chemotherapy resistance.* Cell Death Dis, 2017. **8**(8): p. e3013.

66. Petrik, D., et al., *Epithelial Sodium Channel Regulates Adult Neural Stem Cell Proliferation in a Flow-Dependent Manner.* Cell Stem Cell, 2018. **22**(6): p. 865-878.e8.

67. Bhatia, S.N. and D.E. Ingber, *Microfluidic organs-on-chips.* Nature Biotechnology, 2014. **32**(8): p. 760-772.

68. Ingber, D.E., *'Organ-on-a-chip' technology: On trial.*, in *Chemistry and Industry*. 2011, Society of Chemical Industry: London. p. 18-20.

69. Mai, P., et al., *MatriGrid(®) Based Biological Morphologies: Tools for 3D Cell Culturing.* Bioengineering (Basel), 2022. **9**(5).

70. Place, T.L., F.E. Domann, and A.J. Case, *Limitations of oxygen delivery to cells in culture: An underappreciated problem in basic and translational research.* Free Radical Biology and Medicine, 2017. **113**: p. 311-322.

71. Weltin, A., et al., *Cell culture monitoring for drug screening and cancer research: a transparent, microfluidic, multi-sensor microsystem.* Lab Chip, 2014. **14**(1): p. 138-46.

72. Shaegh, S.A.M., et al., *A microfluidic optical platform for real-time monitoring of pH and oxygen in microfluidic bioreactors and organ-on-chip devices.* Biomicrofluidics, 2016. **10**(4): p. 044111.

73. Bavli, D., et al., *Real-time monitoring of metabolic function in liver-on-chip microdevices tracks the dynamics of mitochondrial dysfunction.* Proceedings of the National Academy of Sciences, 2016. **113**(16): p. E2231-E2240.

74. Yu, F., et al., *On chip two-photon metabolic imaging for drug toxicity testing.* Biomicrofluidics, 2017. **11**(3): p. 034108.

List of Figures

List of Tables

Appendixes

List of Appendixes

Appendix 1 – Control unit schematics

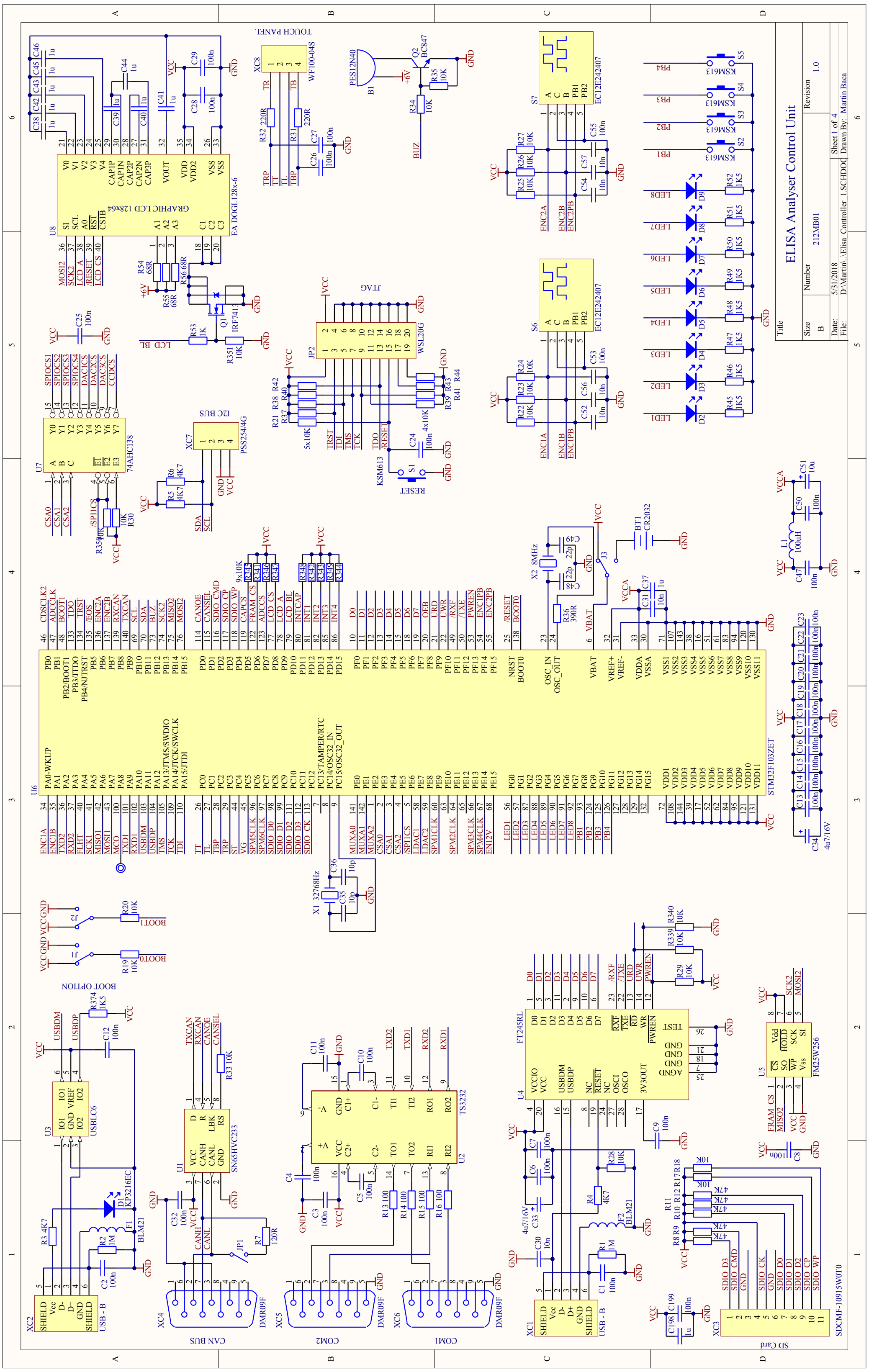

Figure A - 1 Control unit schematic – the microcontroller part

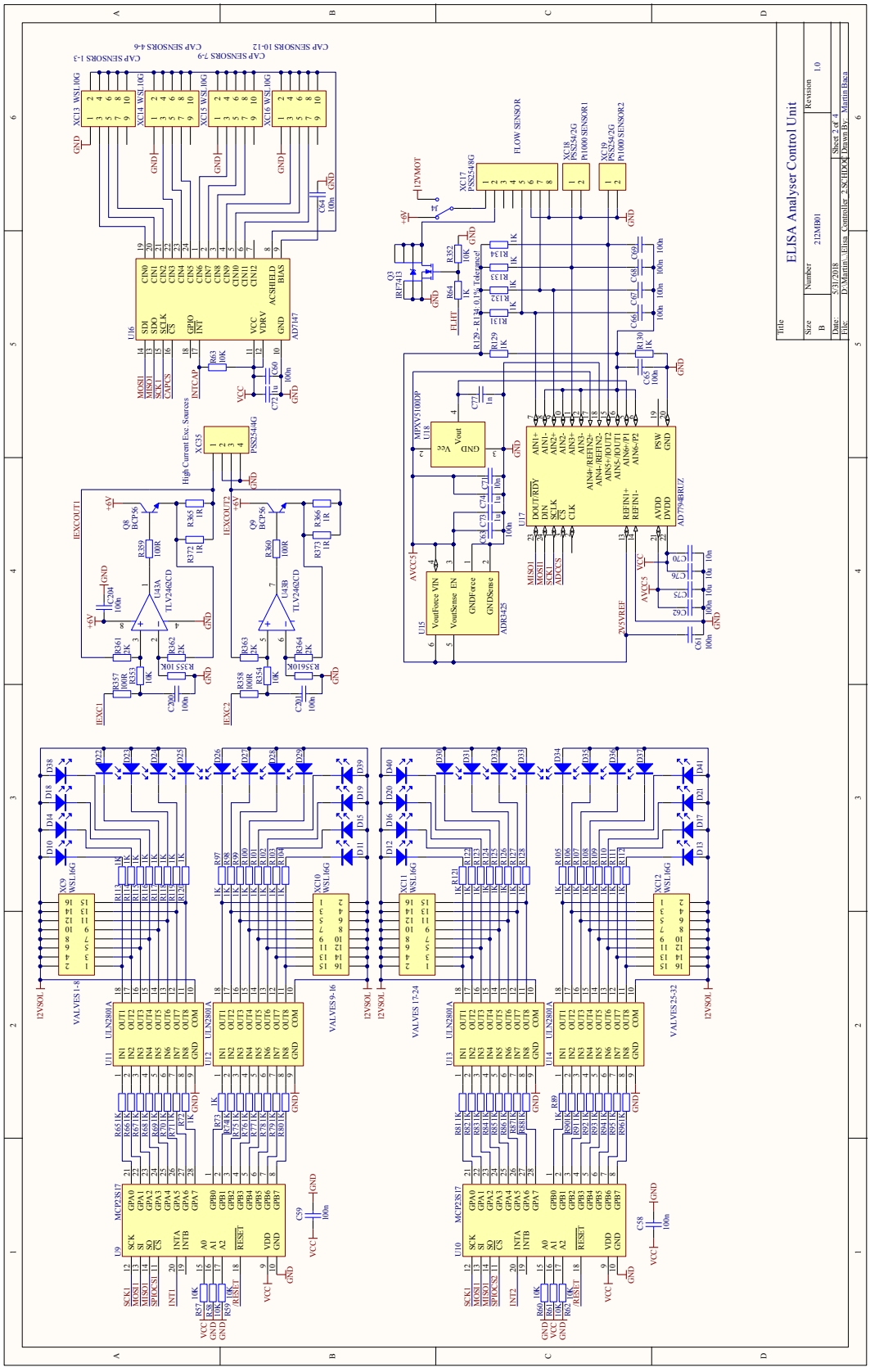

Figure A - 2 Control unit schematic – solenoid valve drivers, constant current sources, capacitive sensors and A/D converter.

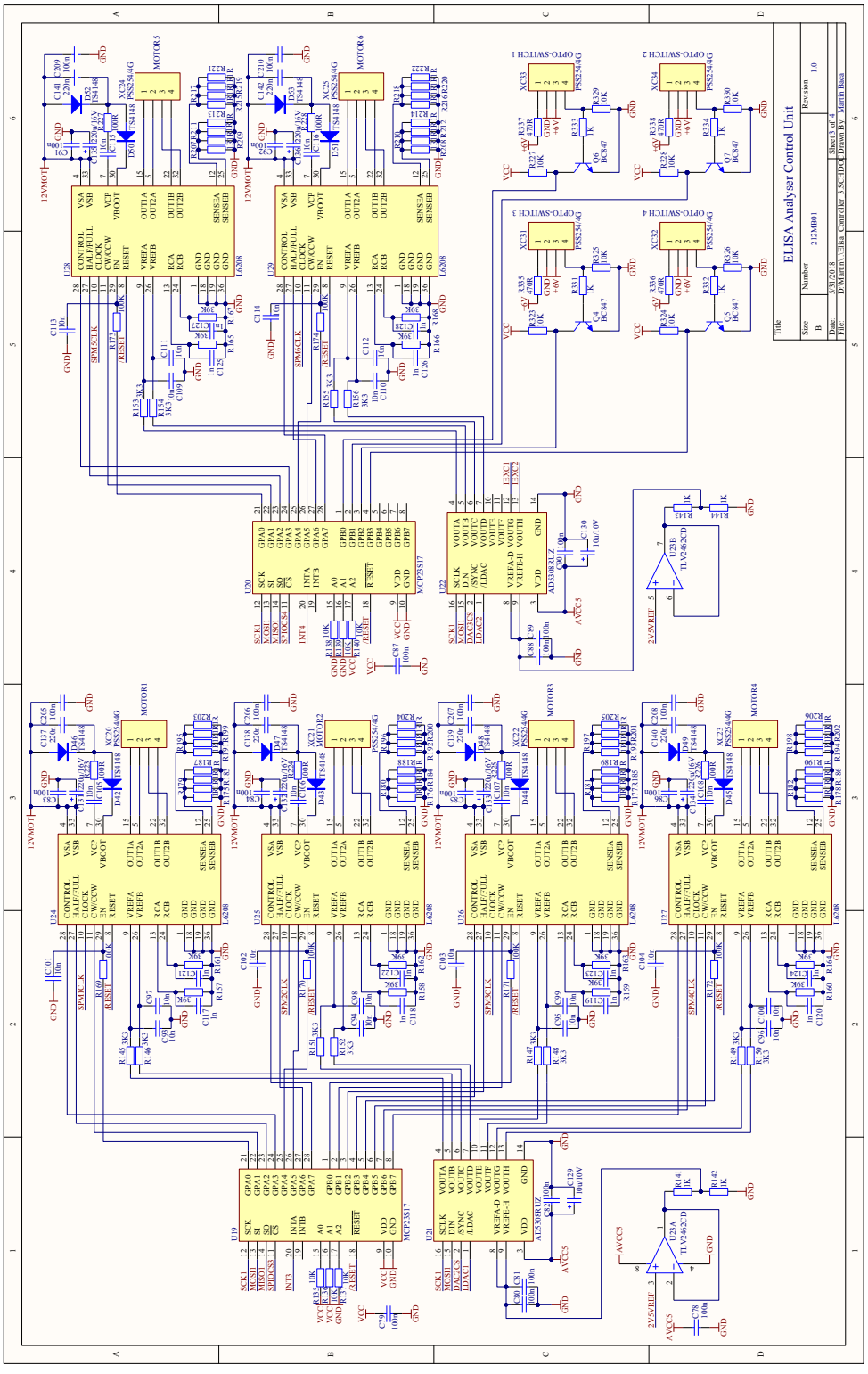

Figure A - 3 Control unit schematic – stepper motor drivers and optical sensors.

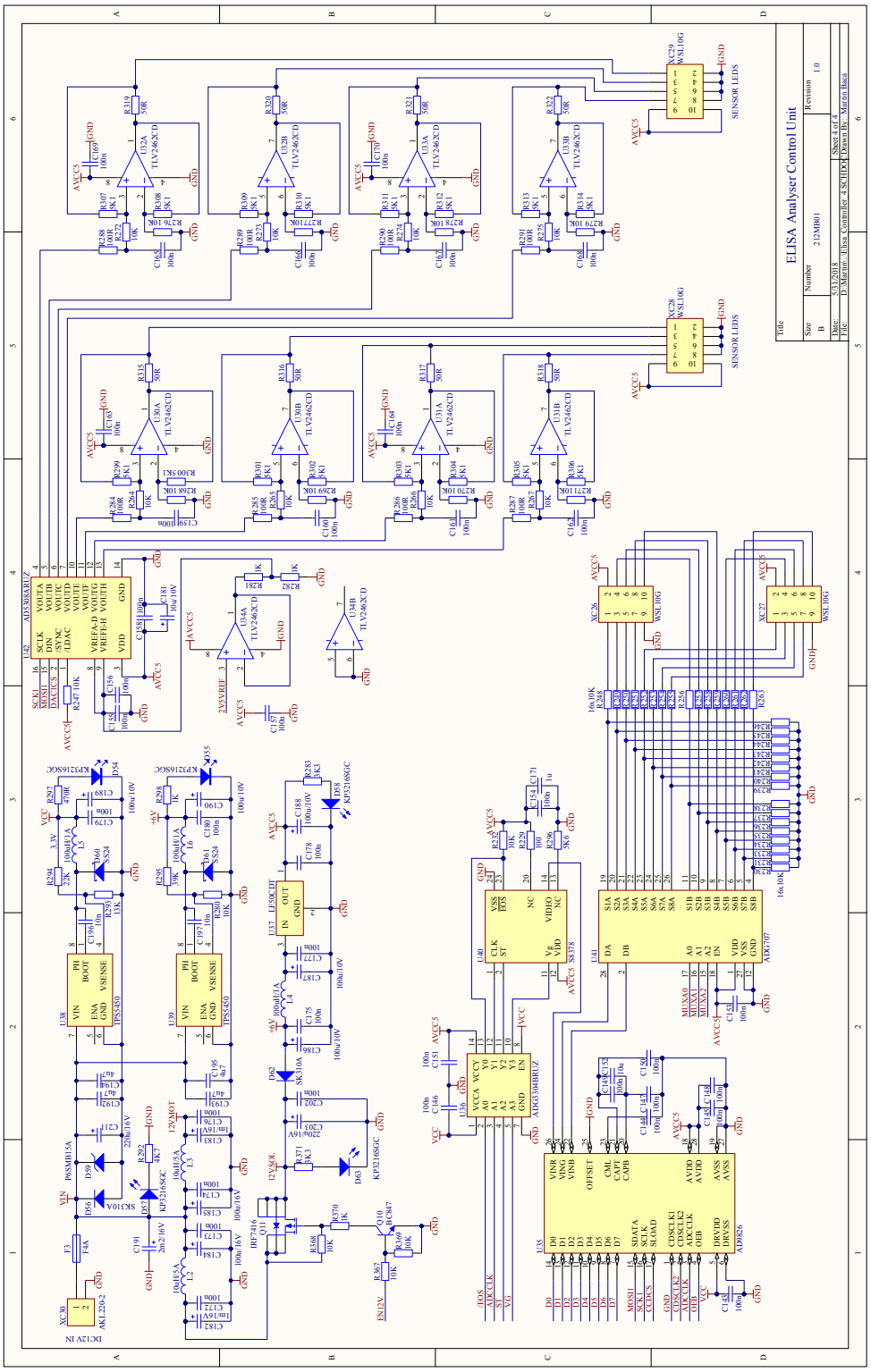

Figure A - 4 Control unit schematic – power supply, spectrometer interface and combined optical sensors interface.

Appendix 2 – Control unit PCB assembly drawing

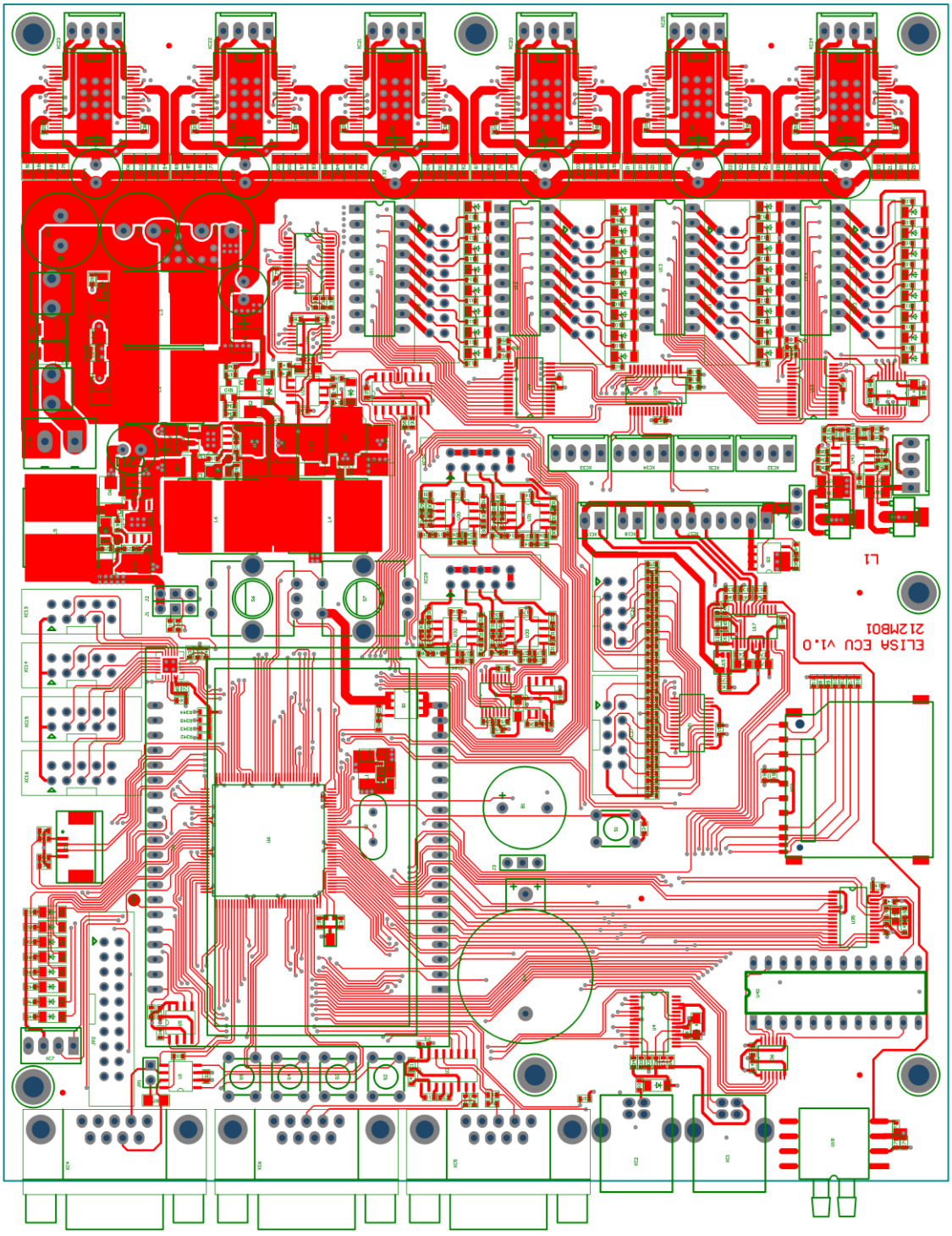

Figure A - 5 Control unit PCB assembly – top side.

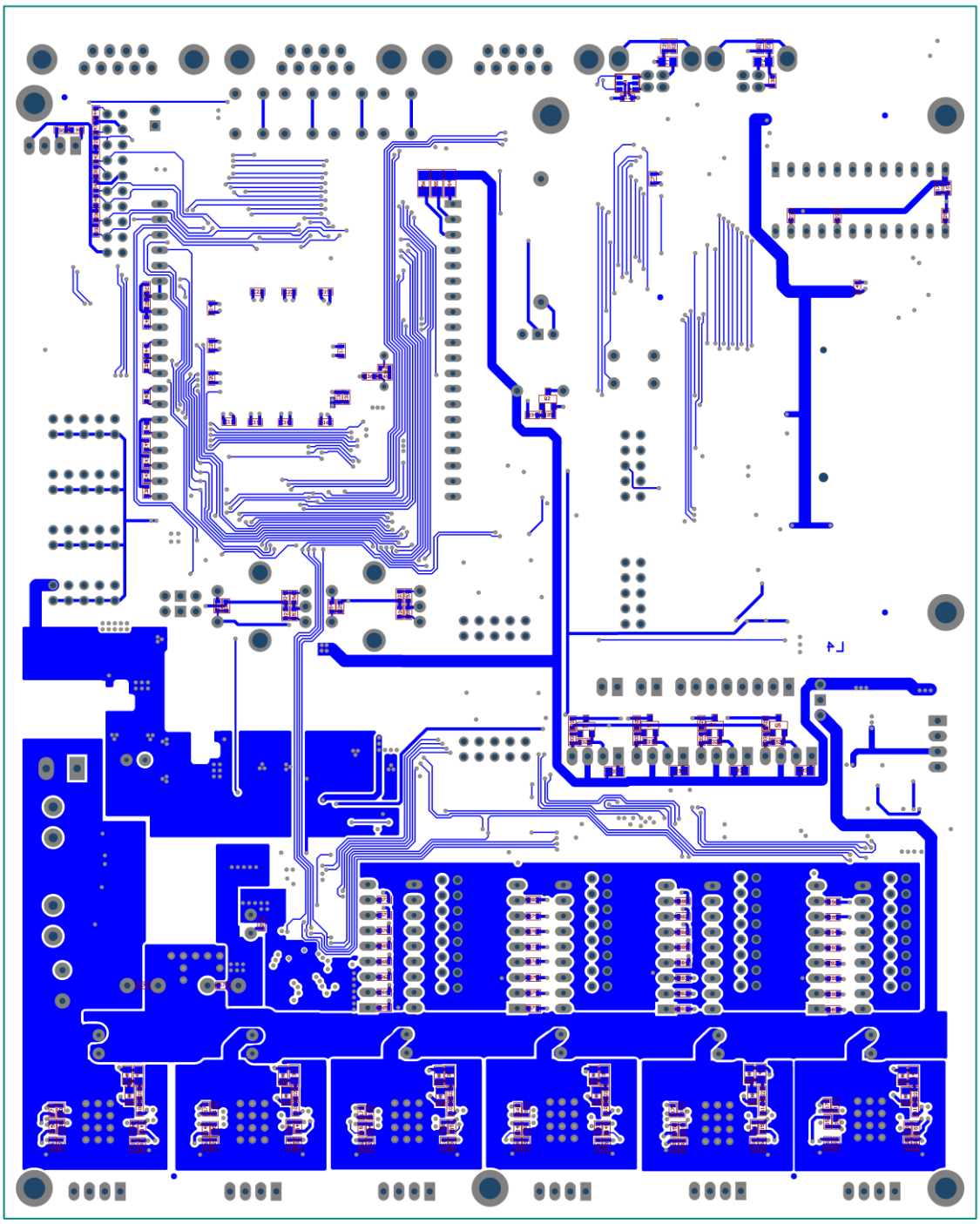

Figure A - 6 Control unit PCB assembly – bottom side.

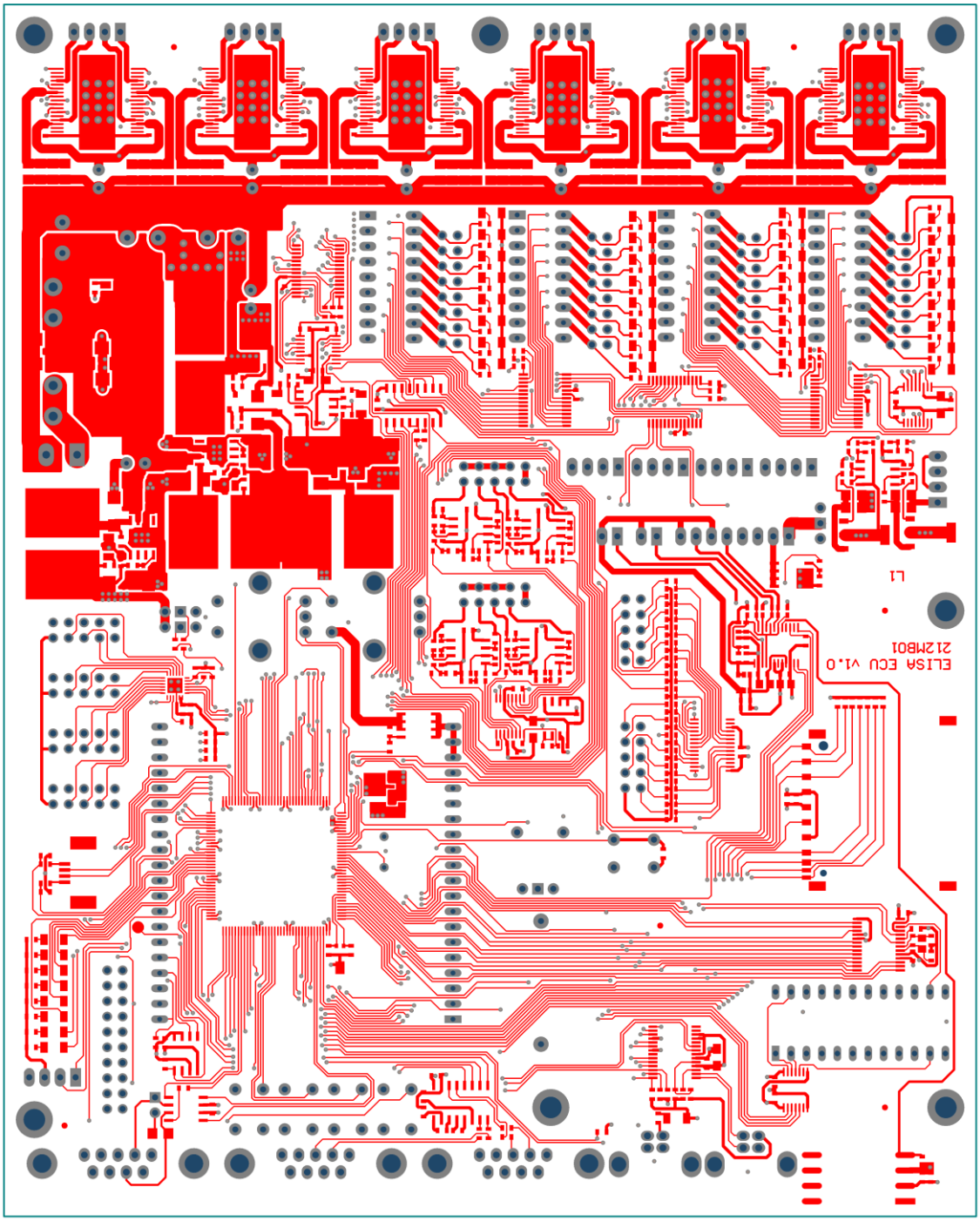

Figure A - 7 Control unit PCB layout – top layer (layer 1).

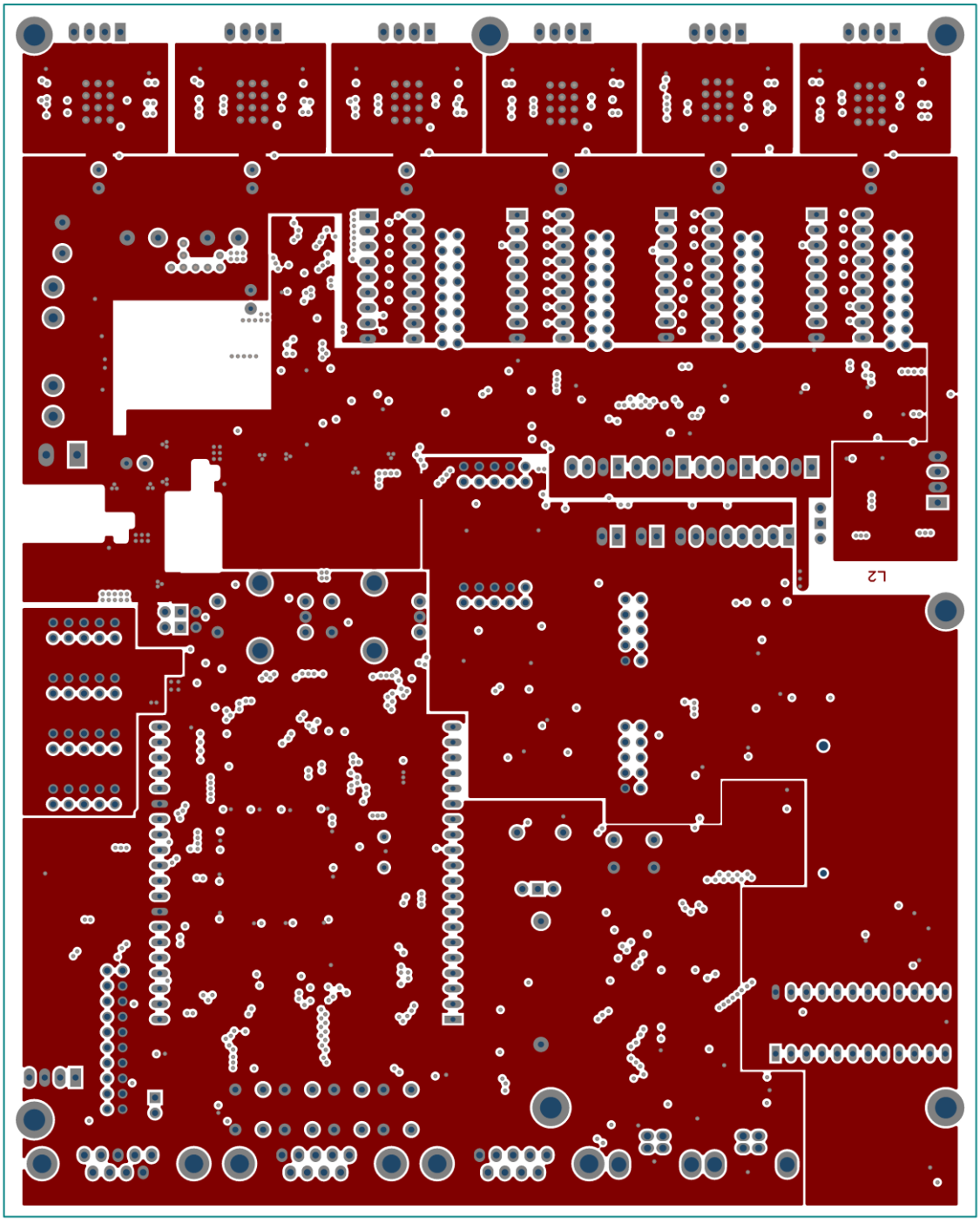

Figure A - 8 Control unit PCB layout - ground layer (layer2).

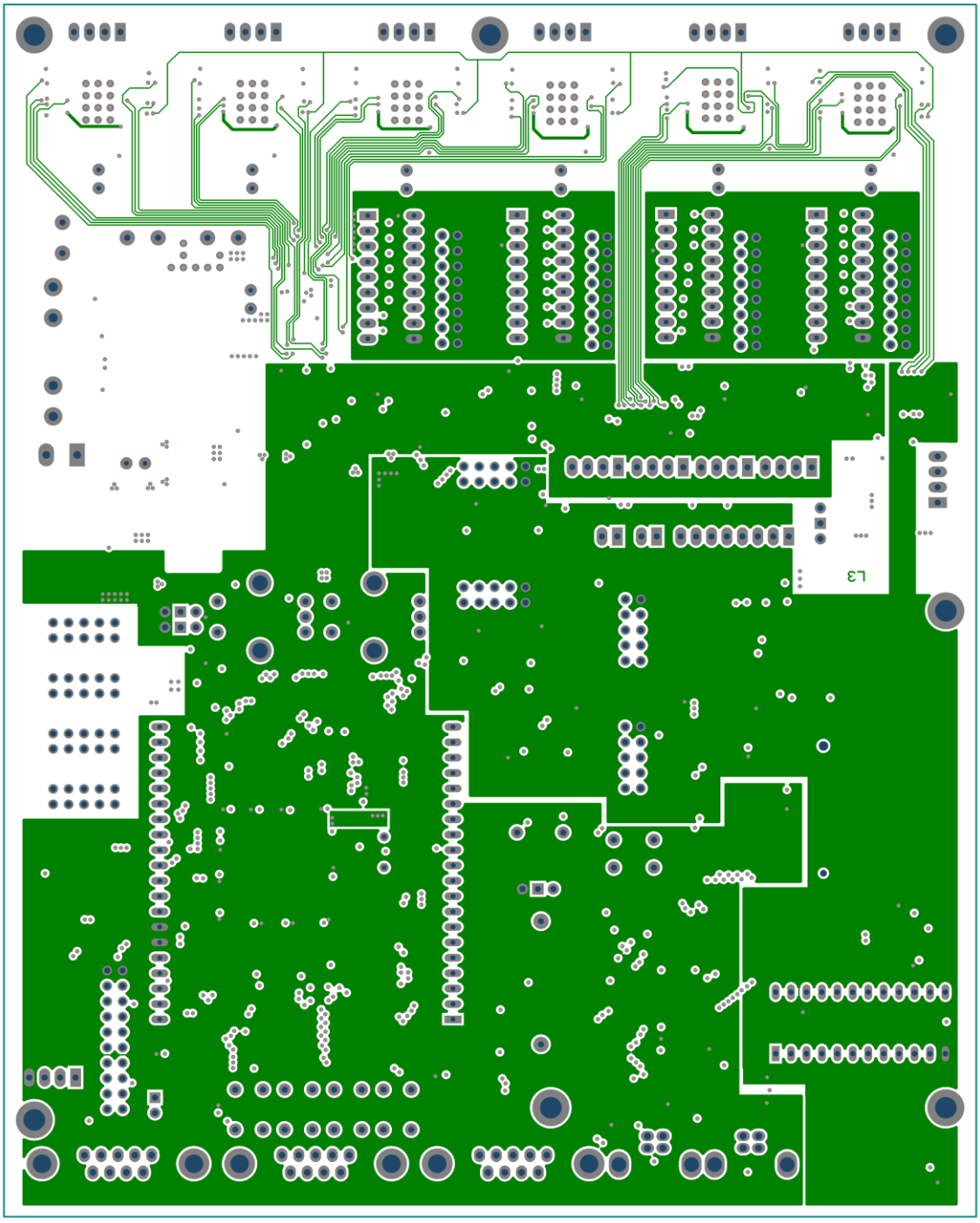

Figure A - 9 Control unit layout – power plane layer (layer 3).

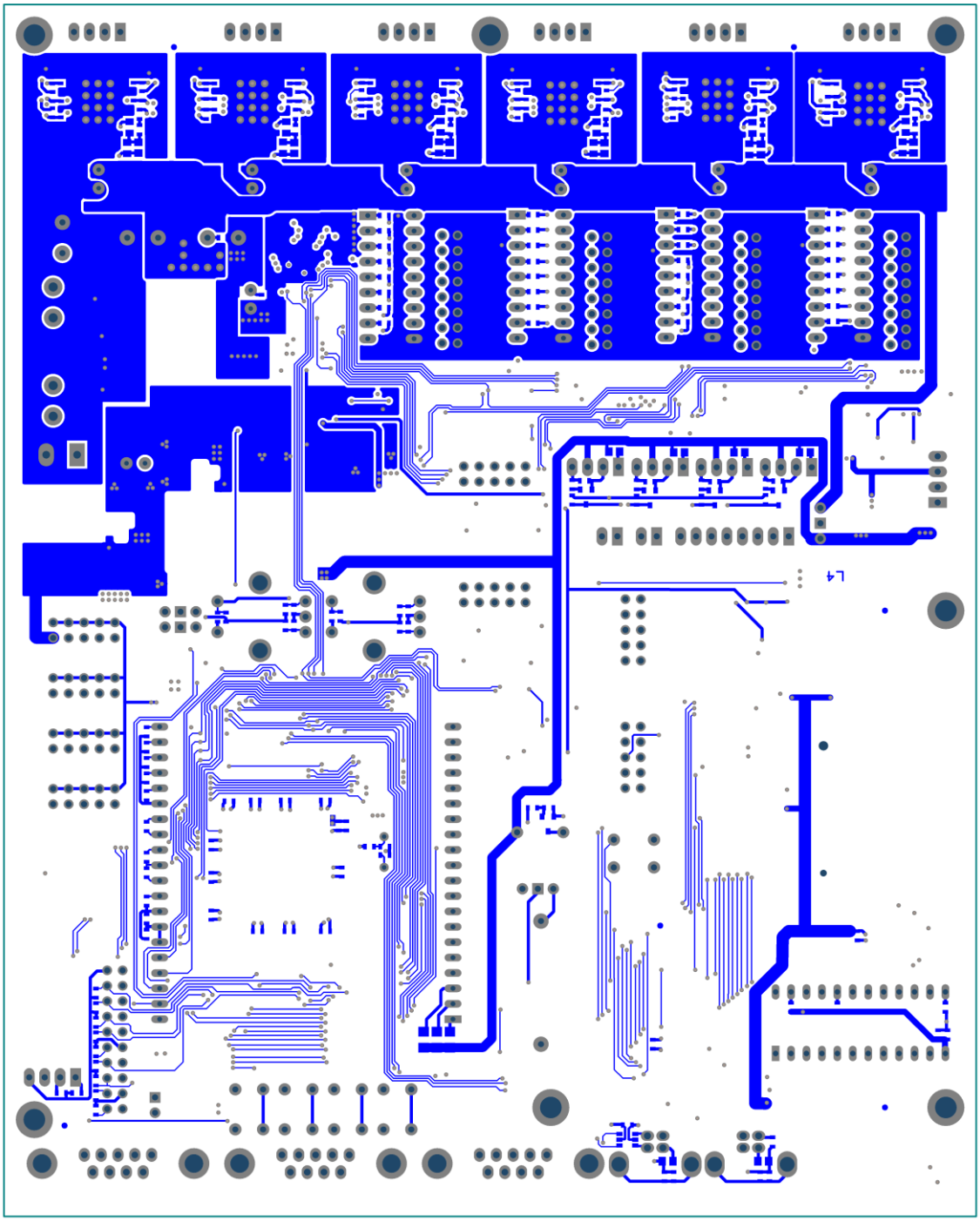

Figure A - 10 Control unit PCB layout – bottom layer (layer 4).

Appendix 3 – Fluorimeter amplifier module schematic

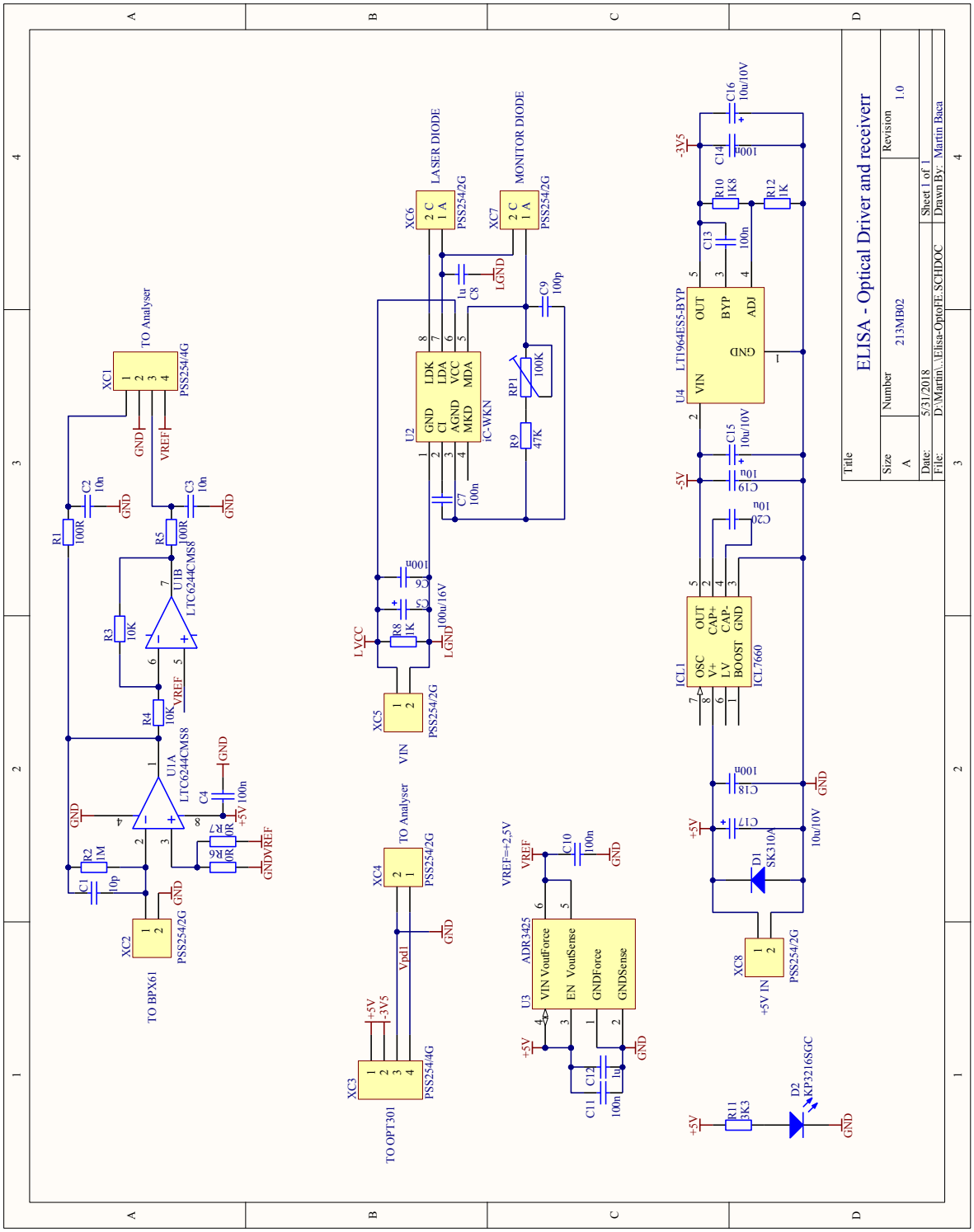

Figure A - 11 Fluorimeter front-end amplifier and laser driver schematic.

Appendix 4 – Fluorimeter amplifier PCB layout

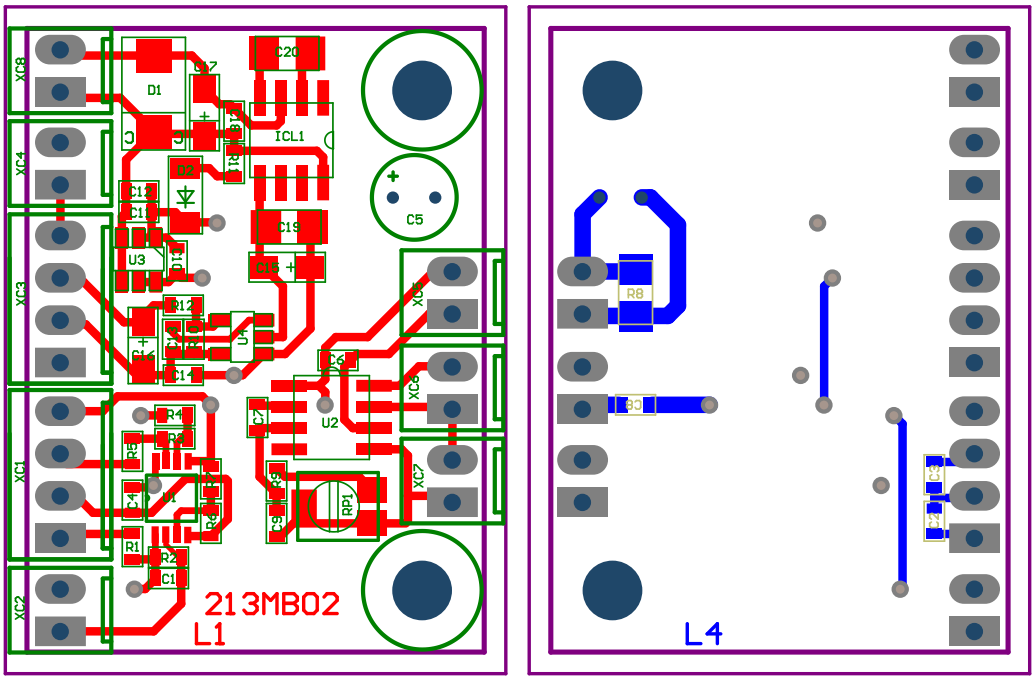

Figure A - 12 Fluorimeter amplifier PCB assembly of top layer (left) and the right layer (right).

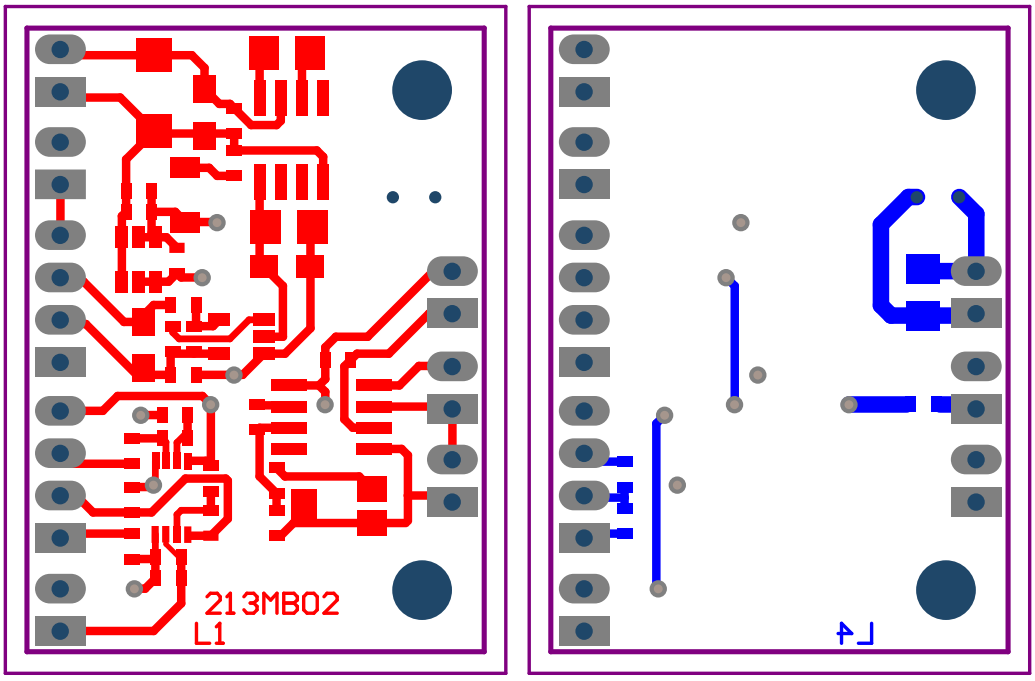

Figure A - 13 Fluorimeter amplifier PCB top layer (left) and bottom layer (right).

Appendix 5 – Culturing module driver schematic

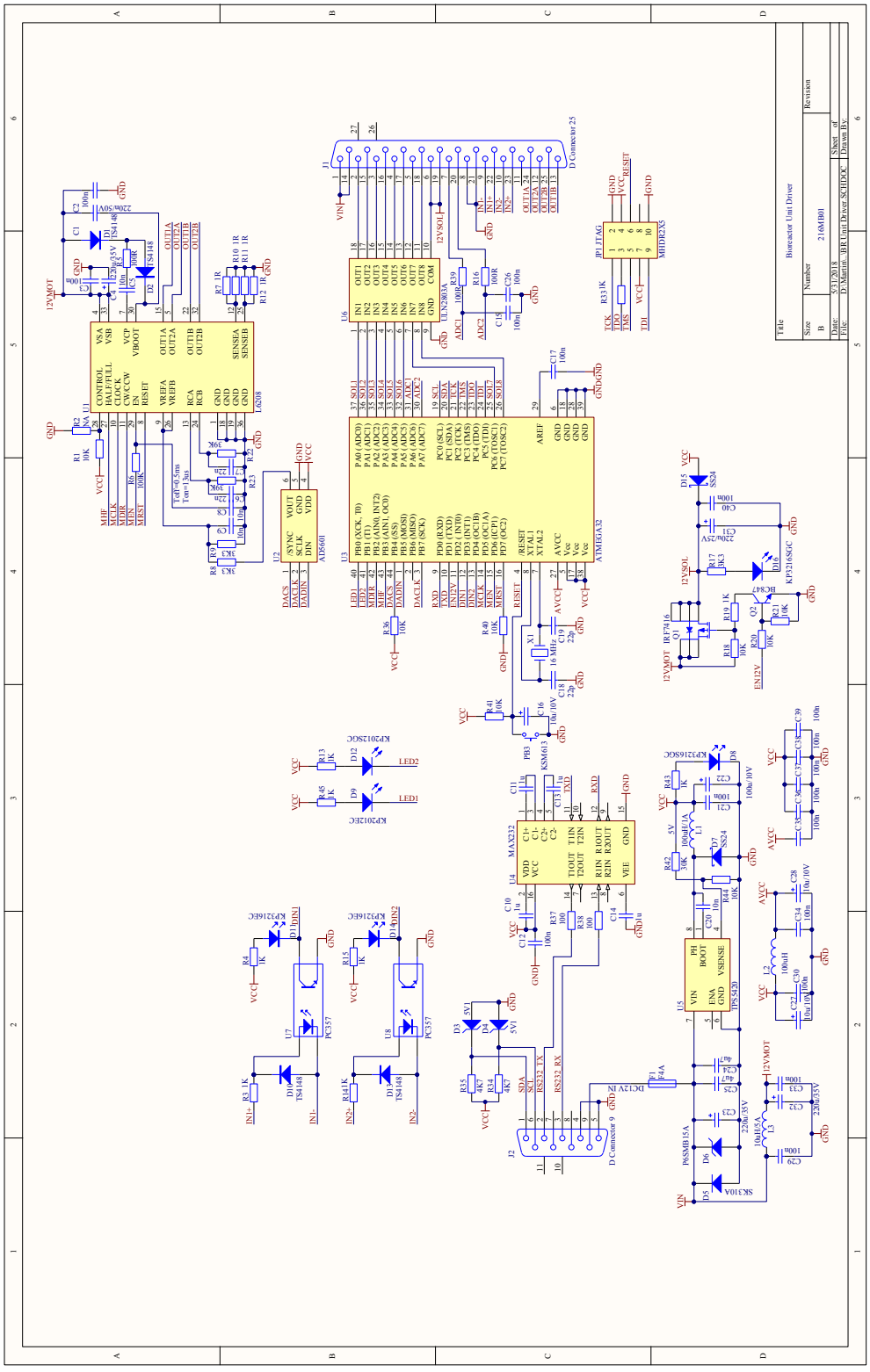

Figure A - 14 Culturing module smart driver schematic.

Appendix 6 – Culturing unit driver PCB assembly plan

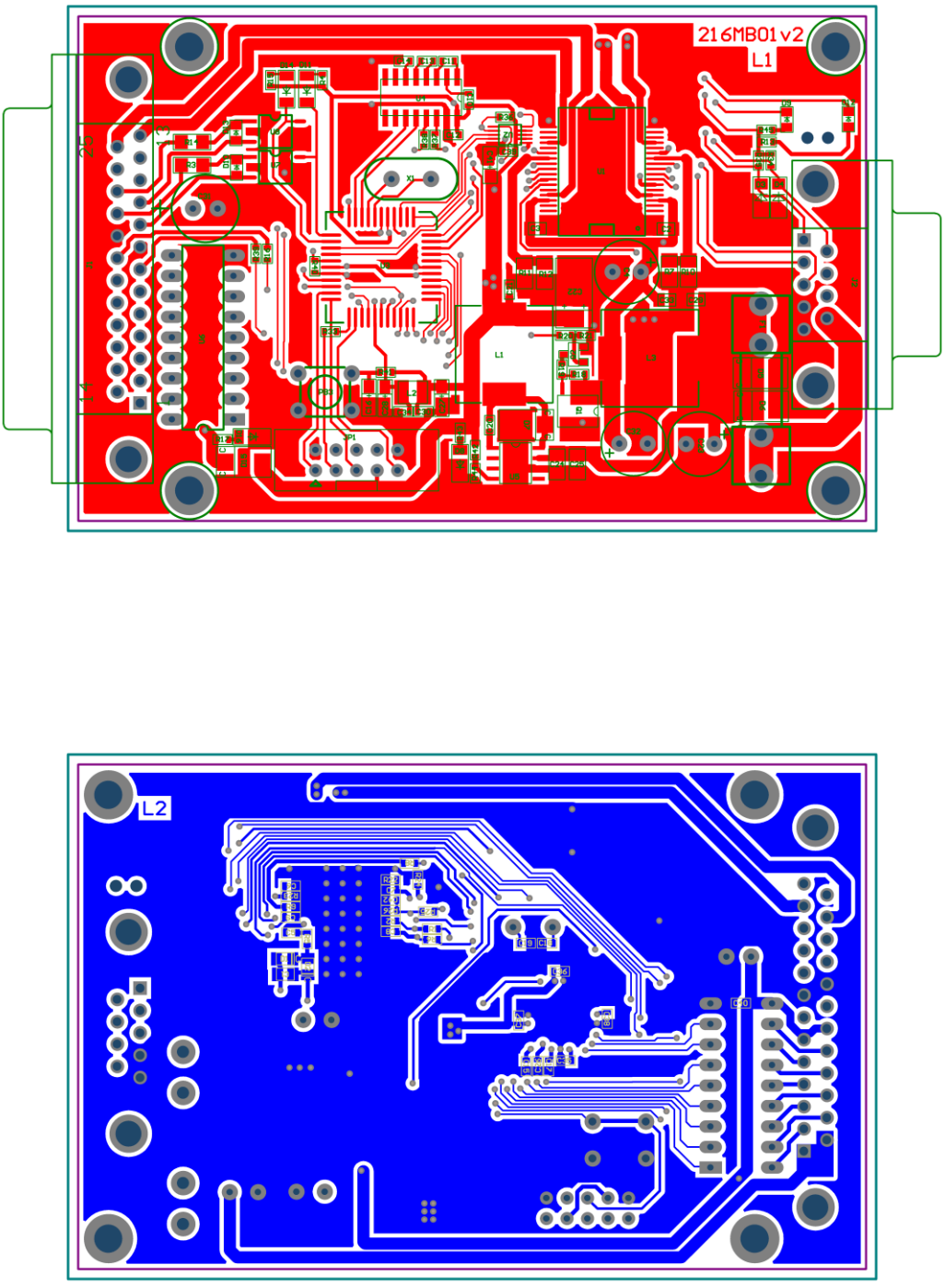

Figure A - 15 Assembly of the driver unit – top side (top) and bottom side (bottom).

Appendix 7 – Culturing unit driver PCB layout

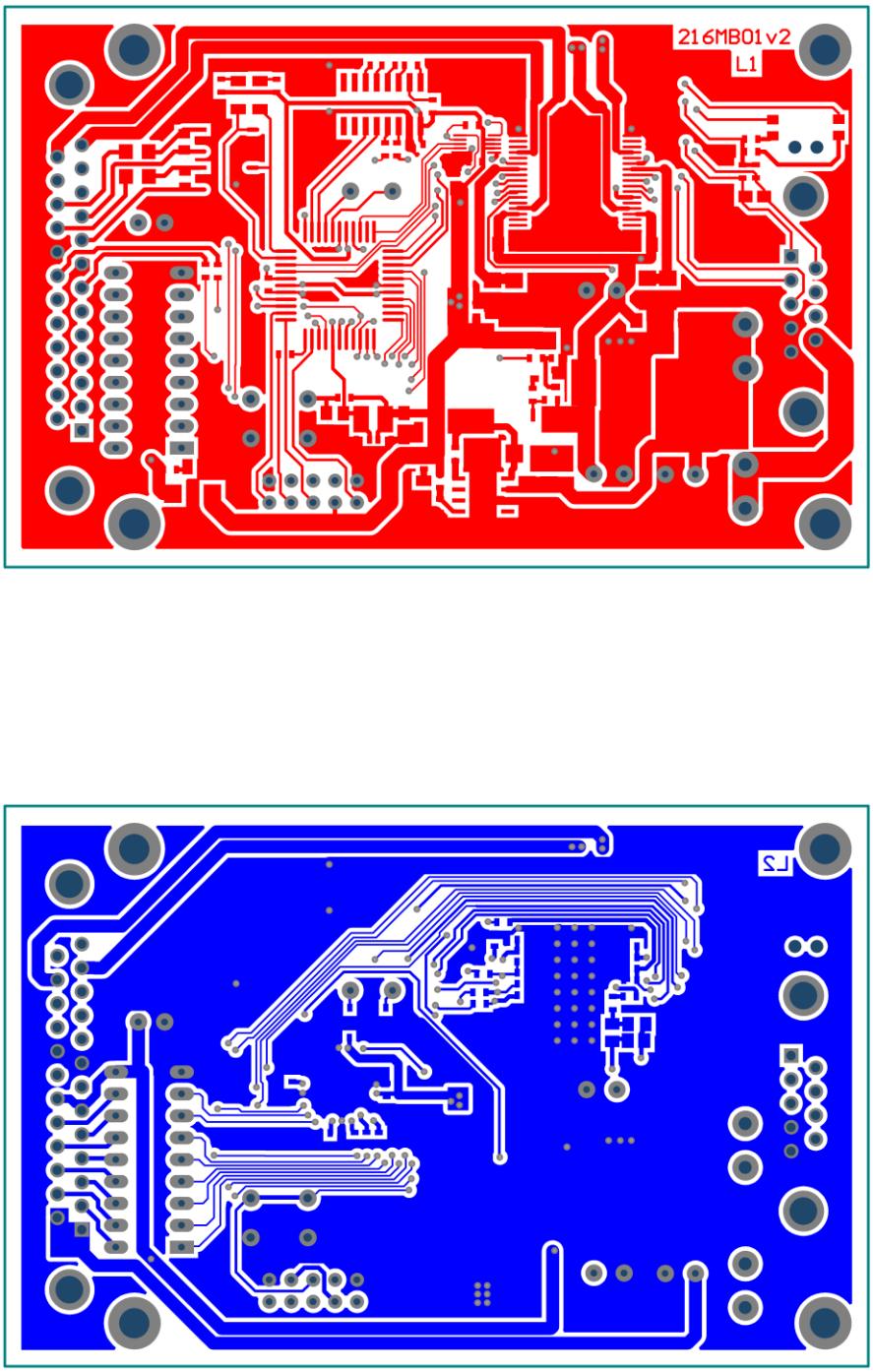

Figure A - 16 Culturing unit smart driver PCB layout – top layer (top) and bottom layer (bottom).

Appendix 8 – Schematics of the OpenOCD debugger hardware

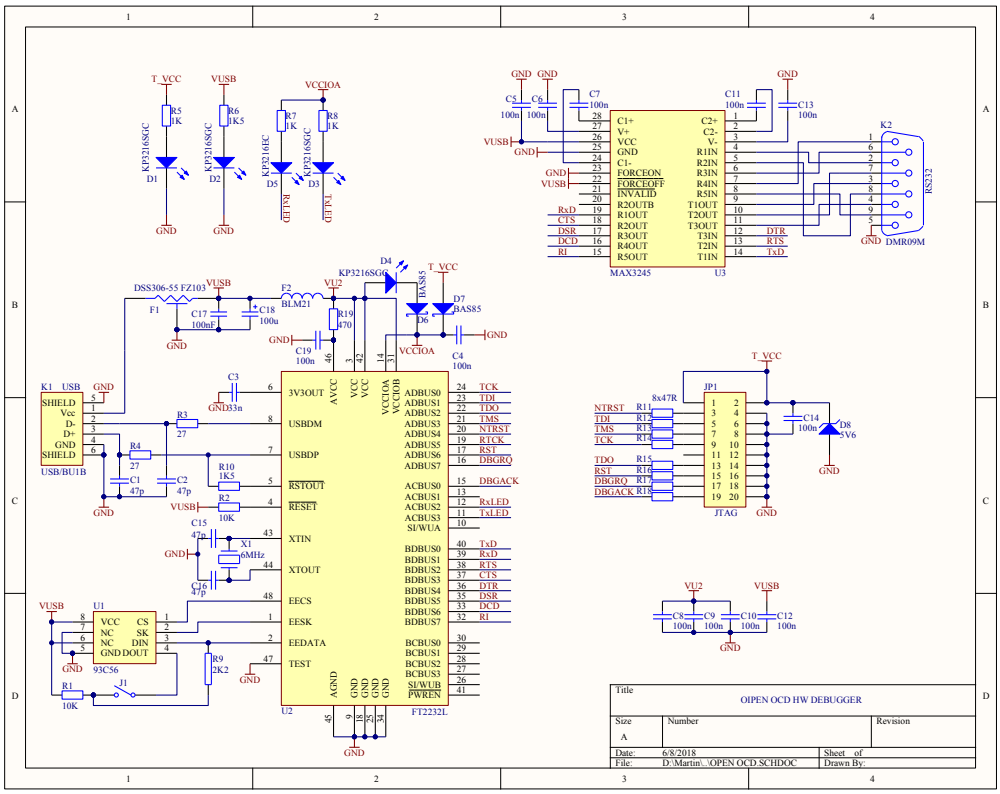

Figure A - 17 The schematic of the OpenOCD debugger

Appendix 9 – The assembly plan and layout of the OpenOCD debugger PCB

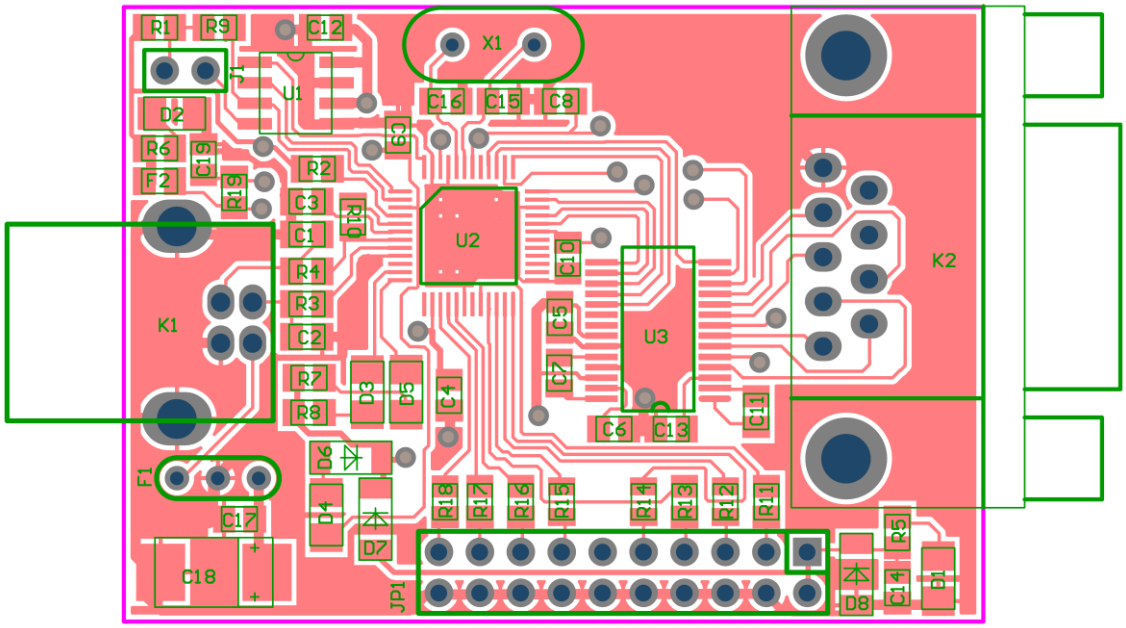

Figure A - 18 The assembly of the OpenOCD debugger – top layer

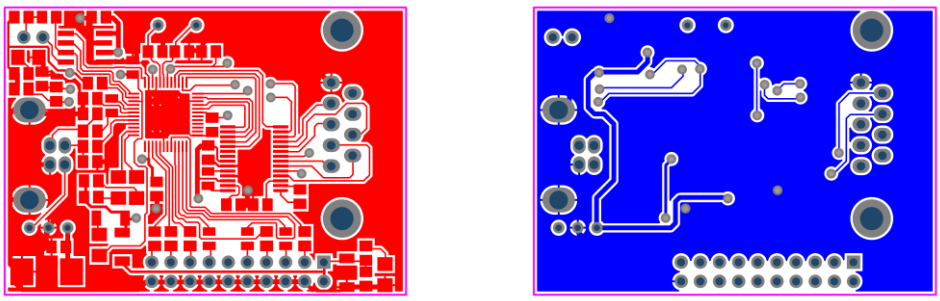

Figure A - 19 The layout of the OpenOCD debugger, top layer (left) and bottom layer (right)

Appendix 10 – Culturing unit driver source code listing

```
/*****************************************************************************\
*               BR Driver -  HW ver1.0                                        *
*-----------------------------------------------------------------------------*
* Description : Stepper motor controller with solenoid valves driver          *
*               controlled over I2C bus                                       *
*-----------------------------------------------------------------------------*
* Author      : Martin Baca                                                   *
* Developed   : 05.04.2016                Last Update : 12.09.2016            *
* Version     : 1.1                                                           *
*-----------------------------------------------------------------------------*
* Compiler    : avrgcc                                                        *
* Source file : brdriver.c                                                    *
*-----------------------------------------------------------------------------*
* Target system : 216MB01 - HW Version 1.0                                    *
*                 ATmega32 @16 MHz, UART:  115200,N,8,1                       *
* Emulator HW   :                                                             *
\*****************************************************************************/



#include <avr/io.h>
#include <avr/sleep.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <avr/eeprom.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <ctype.h>


/*** uncomment one the following lines according to HW version ***/

//#define  PCB216MB01
#define    PCB216MB01v2


/*** uncomment the following line for rotation direction change ***/
#define SM_REVERSE_ROTATION


//*** EEPROM MAP ***
#define EE_BR1_MAKESAMPLE      0x04
#define EE_BR1_PERFUSION       0x05
#define EE_BR_SPEED            0x06
#define EE_I2C_ADDR            0x08
#define EE_NEXT_FREE           0x09

//reserved for bootloader
#define EE_BOOT_FLAG           E2END

//*** END OF EEPROM MAP ***

#define Reset2Boot  2 * 0x3C00 // The reset address of bootloader

//baudrate division constants for Xtal 16 MHz
#define    baud1200        831
#define    baud2400        415
#define    baud4800        207
#define    baud9600        103
#define baud19200           51
#define baud38400           25
#define baud57600           16
#define baud115200           8

// values for TWPS=00 - prescalling by 1 and XTAL=16MHz
#define I2C_400k            12
#define I2C_100k            72
#define I2C_50k            128

#define I2C_RXBUFSIZE       50
#define I2C_TXBUFSIZE       50

#define I2C_START      TWCR=0xA5
#define I2C_STOP       TWCR=0x95
#define I2C_CLRTWINT   TWCR=0xC5
#define I2C_NACK       TWCR=0x85
#define I2C_CLRTWINT_ACK    0xC4
#define I2C_CLRTWINT_NACK   0x84
#define I2C_INT_DIS         0x00
#define I2C_INT_ENA         0x01
#define I2C_TIMEOUT            2    //1LSB = 100ms,
#define COM_PARAM_ERR       0xFD    //-3 - I2C parameter error - error state
#define COM_NOTVALID        0xFE    //-2 - I2C command not valid - error state
#define GEN_ERROR           0xFF    //-1 - I2C general error - error state
#define NO_ERROR            0x00    //0 - I2C no error - normal state

// I2C commands:
// I2C command syntax: I2C_address, Command,
// Parameters (optional - 1 or more bytes), '\n'(end of msg character)
//
#define COM_TEST        0x30    //no parameters, toggles the LED
#define COM_BRCONTROL   0x31    //1 parameter: 0x00 - disable br control,
                                //0x01 - enable br control
#define COM_BRSTOP      0x32    //no parameters
#define COM_BRSPEED     0x33    //2 parameter bytes: SpeedMSB, SpeedLSB,
```

```
                                    //min speed=1, max speede=500 ul/min
#define COM_PREPSAMPLE  0x34    //no parameters
#define COM_BRCHANGEMED 0x35    //4 parameter bytes: VolumeMSB, VolumeLSB,
                                //SpeedMSB, SpeedLSB, volume limits: <-10000;10000> ul,
                                //speed limits: <1;500> ul/min
//#define COM_BRSAMPLE  0x36    //4 parameter bytes: VolumeMSB, VolumeLSB, SpeedMSB,
                                //SpeedLSB, volume limits: <1;10000> ul,
                                //speed limits: <1;500> ul/min
//#define COM_BRMIX     0x37    //4 parameter bytes: VolumeMSB, VolumeLSB,
                                //SpeedMSB, SpeedLSB, volume limits: <1;30000> ul,
                                //speed limits: <1;500> ul/min
#define COM_VALVE       0x38    // 2 parameters: Vavlve_Number: <1;8>, new_state: 0 or 1
#define COM_STARTPUMP   0x39    //4 parameter bytes: VolumeMSB, VolumeLSB, SpeedMSB,
                                //SpeedLSB, volume limits: <1;30000> ul,
                                //speed limits: <1;500> ul/min, speed=0 means STOP
#define COM_BRSTATUS    0x41    // no parameters


#define AIN1_ADC_CH 6
#define AIN2_ADC_CH 7
#define RXBUFSIZE  80


#define LEDR_ON    (PORTB&=~0x01)
#define LEDR_OFF    (PORTB|=0x01)
#define LEDR_TOGGLE (PORTB^=0x01)

#define LEDG_ON    (PORTB&=~0x02)
#define LEDG_OFF    (PORTB|=0x02)
#define LEDG_TOGGLE (PORTB^=0x02)

#define DACS_0     (PORTB&=~0x10)
#define DACS_1      (PORTB|=0x10)

#define DACSB_0    (PORTB&=~0x10)
#define DACSB_1     (PORTB|=0x10)

#define DACSA_0    (PORTB&=~0x02)
#define DACSA_1     (PORTB|=0x02)

#define SM_CLK_0   (PORTD&=~0x20)
#define SM_CLK_1    (PORTD|=0x20)

#define SM_DISABLE (PORTD&=~0x40)
#define SM_ENABLE   (PORTD|=0x40)

#define SM_RESET   (PORTD&=~0x80)
#define SM_UNRESET  (PORTD|=0x80)

#define SM_DIR_CCW (PORTB&=~0x04)
#define SM_DIR_CW   (PORTB|=0x04)

#define SM_STEP_FULL (PORTB&=~0x08)
#define SM_STEP_HALF  (PORTB|=0x08)


#define VALVE1 1
#define VALVE2 2
#define VALVE3 3
#define VALVE4 4
#define VALVE5 5
#define VALVE6 6
#define VALVE7 7
#define VALVE8 8


#define SOL_SET_5V (PORTD&=~0x04)
#define SOL_SET_12V (PORTD|=0x04)

#define OFF        0
#define ON         1

#define DAC_CHAN_A 0
#define DAC_CHAN_B 1

// Speed ramp states
#define STOP       0
#define ACCEL      1
#define DECEL      2
#define RUN        3
#define STOPPED    4

#define NONE       0
#define SPEEDUP    1
#define SLOWDOWN   2
#define SLOWSTOP   3

#define TRUE       1
#define FALSE      0

#define CW         0
#define CCW        1

//#define HALFSTEPS
#define FULLSTEPS
//#define SLOWDECAY
#define FASTDECAY

// Timer/Counter 1 running on 16MHz / 64 = 0.25MHz (4uS). (T1-FREQ 250000)
#define T1_FREQ 250000

//! Number of (full)steps per round on stepper motor in use.
```

```
200  #define FSPR 200
201
202
203
204  // Maths constants. To simplify maths when calculating in speed_cntr_Move().
205  #define ALPHA (2*3.14159/FSPR)                    // 2*pi/spr
206  #define A_T_x1000 ((long)(ALPHA*T1_FREQ*1000))    // (ALPHA / T1_FREQ)*1000
207  #define A_T_x100 ((long)(ALPHA*T1_FREQ*100))      // (ALPHA / T1_FREQ)*100
208  #define T1_FREQ_148 ((int)((T1_FREQ*0.676)/100)) // divided by 100 and scaled by 0.676
209  #define A_SQ (long)(ALPHA*2*10000000000)          // ALPHA*2*10000000000
210  #define A_x20000 (int)(ALPHA*20000)               // ALPHA*20000
211
212
213  #define T0_TICKS_PS    250
214  #define TIME_100MS      25
215  #define TIME_200MS      50
216  #define TIME_500MS     125
217  #define TIME_1S        250
218
219  #define DISP_BLINK_1S  125 //period 1s
220
221  #define SOLENOID_12V_TIME  1000    //250   //turn-on time with 12V powering, 1LSB=1ms
222
223  #define CMD_OK             0
224  #define CMD_ERR_CMD        1
225  #define CMD_ERR_PARAM      2
226  #define CMD_ERR_PARAMCNT   3
227
228  #define STEP_RUNNING       1
229  #define STEP_FINISHED      0
230
231  #define BR1_V1                      VALVE1
232  #define BR1_V2                      VALVE4
233  #define BR1_V3                      VALVE3
234  #define BR1_V4                      VALVE2
235  #define PUMP1_VOL_PER_REVOLUTION   15120   //12600 //pump volume per revolution in nl
236
237  #define CALIBRATED_DEAD_VOLUME     120     //the dead volume of the callibrated tubing
238
239  #define BR1_PERFUS_VOL             30000
240  #define BR1_PERFUS_SPEED           br1_pump_speed
241  #define BR1_MIX_VOL                500
242  #define BR1_MIX_SPEED              br1_pump_speed
243  #define BR1_SAMP_VOL               400
244  #define BR1_SAMP_SPEED             br1_pump_speed
245
246
247
248  typedef struct {
249    volatile uint8_t run_state : 3; //! What part of the speed ramp we are in.
250    uint8_t dir : 1;              //! Direction stepper motor should move.
251    uint16_t step_delay;          //! Peroid of next timer delay. At start this
252                                  //  value set the accelration rate.
253    uint32_t decel_start;             //! What step_pos to start decelaration
254    int32_t decel_val;            //! Sets deceleration rate.
255    int16_t min_delay;            //! Minimum time delay (max speed)
256    int32_t accel_count;          //! Counter used when accelerateing/decelerateing
257                                  //  to calculate step_delay.
258
259    uint32_t max_s_lim;           //! Number of steps before we hit max speed.
260    uint32_t accel_lim;           //! Number of steps before we must start deceleration
261                                  //  (if accel does not hit max speed).
262
263    uint16_t new_step_delay;      // Holds next delay period.
264    int16_t last_accel_delay;         // Remember the last step delay used when accelrating.
265    uint32_t step_count;          // Counting steps when moving.
266    uint16_t rest;                // Keep track of remainder from new_step-delay
267                                  // calculation to incrase accurancy
268
269    uint16_t max_iphase;          // maximum phase current
270    uint16_t standby_iphase;      // standby phase current
271    uint16_t i_phasea;            // actual current for phase a
272    uint16_t i_phaseb;            // actual current for phase b
273    int8_t mstep_counter;         // microsteps counter
274    uint8_t driver_state;         // L6208 internal state counter
275    int8_t microstep_inc;         // increment size for microstepping
276    uint8_t microsteps;           // allowed values: 2,4,8,16,32 - microstepping
277                                  // enabled and resolution
278    uint8_t driver_control;       // bits 0-3: mirror of L6208 control pins (bit3 - CONTROL,
279                                  // bit2 - HALF/FULL, bit1 - CW/CCW, bit0 - EN)
280    int32_t abs_position;         //absolute position in microsteps
281  } speedRampData;
282
283
284  typedef struct {
285     uint8_t percent_done;        //
286     int16_t time2end;            //time to finish current task in seconds
287  } brtask_statistics;
288
289
290  //bit0 - br1_perfusion
291  //bit1 - br1_make sample
292  //bit2 - br1_changing_medium
293  //bit7 - pump running(1) or stopped(0)
294  unsigned char br_status_flags=0;
295
296
297  volatile unsigned int adc_result;
298  volatile unsigned char adc_flag;
299
300
301  volatile unsigned int tim_frac;
```

```
volatile unsigned char tim_sec;
volatile unsigned char tim_min;
volatile unsigned int tim_hrs;
volatile unsigned char tim_ena=0;

volatile uint16_t solenoid_pwr_timer;//1ms interval

unsigned char baudrate;


char rxbuf[RXBUFSIZE];
volatile unsigned char rx_ptr;
volatile unsigned char rx_overflow=0;
//unsigned char read_ptr;
volatile unsigned char new_msg=0;

// i2c variables
unsigned char i2c_rxbuf[I2C_RXBUFSIZE];    //i2c RX buffer
unsigned char i2c_txbuf[I2C_TXBUFSIZE];    //i2c TX buffer
volatile unsigned int i2c_rxptr_top=1;
volatile unsigned int i2c_rxptr_bot=0;
volatile unsigned char i2c_txptr_top=1;
volatile unsigned char i2c_txptr_bot=0;
volatile unsigned char i2c_rxbuf_err=0;    //buffer owerflow flag
volatile unsigned char i2c_txbuf_err=0;    //buffer owerflow flag
volatile unsigned char i2c_buserror=0;     //i2c communication error flag
volatile unsigned char i2c_newmsg=0;

volatile unsigned char last_command=0;
volatile unsigned char last_txbyte=1;
volatile unsigned char i2c_status=0;
volatile unsigned char error=0;

unsigned char disp_blink_period=0; //8ms per LSB, defined display on and off time,
                                   // range 8ms - 2s, 0=blink disabled

unsigned char cmd_valid=0;

signed int acceleration; // Accelration to use.
signed int deceleration; // Deceleration to use.
signed int steps;        // Number of steps to move.
signed int speed;        // current Speed to use.

signed int last_speed;   // current Speed to use.

uint8_t br1_perfusion=0;
uint8_t br1_make_sample=0;
uint8_t br1_changing_medium=0;
int16_t medium_change_vol=5;
uint16_t medium_change_speed=25;
uint16_t br1_pump_speed=25;

volatile uint16_t delay_timer_br;      //10ms interval

speedRampData srd_sm1;

#define MICROSTEPS_TABSIZE 32

const unsigned char microstep_tab[2][MICROSTEPS_TABSIZE]={
   {0,5,10,15,20,24,29,34,38,43,47,51,56,60,63,67,71,74,
    77,80,83,86,88,90,92,94,96,97,98,99,100,100},
    {100,100,99,99,98,97,96,94,92,90,88,86,83,80,77,74,71,
    67,63,60,56,51,47,43,38,34,29,24,20,15,10,5}};

uint8_t progress[8]={0,0,0,0, 0,0,0,0}c;


void speed_cntr_Init_Timer1(void);
unsigned int min(unsigned int x, unsigned int y);

void speed_cntr_Move(int32_t step, uint16_t accel, uint16_t decel, uint16_t speed);
void sm_set_iphase(uint16_t  iphasea, uint16_t  iphaseb);
void sm_set_istandby(speedRampData *srd_motorx, int16_t istandby);
void sm_motor_init(speedRampData *srd_motorx, uint8_t microsteps,
                   uint16_t max_iphase, uint16_t standby_iphase);
void sm_motor_deinit(speedRampData *srd_motorx);

void i2c_send_byte(void);
void i2c_recv_byte(void);
void parse_i2c_command(void);
void stop_pump(void);
uint8_t br1_init(void);
void br_status(void);
void control_solenoid(uint8_t solenoid_id, unsigned char state);
void start_pump(int16_t volume, uint16_t speed);



const char cmd_valve[]         PROGMEM="valve";
const char cmd_startpump[]     PROGMEM="start pump";
const char cmd_stoppump[]      PROGMEM="stop pump";
const char cmd_pumpstatus[]    PROGMEM="ps";
const char cmd_stats[]         PROGMEM="stats";
const char cmd_callboot[]      PROGMEM="CALL BOOT";
const char cmd_seti2caddr[]    PROGMEM="set i2c addr";

//bioreactor commands
const char cmdb_control[]      PROGMEM="br control";
const char cmdb_changemed[]    PROGMEM="br change medium";
const char cmdb_sample[]       PROGMEM="br sample";
const char cmdb_mix[]          PROGMEM="br mix";
const char cmdb_prepsample[]   PROGMEM="br prepare sample";
```

```
404  const char cmdb_setspeed[]     PROGMEM="br set speed";
405  const char cmdb_stop[]         PROGMEM="br stop";
406
407  const char resp_ok[]           PROGMEM="\r\nOK.\r\n";
408  const char resp_err_cmd[]      PROGMEM="\r\nCOMMAND SYNTAX ERROR!\r\n";
409  const char resp_err_param[]    PROGMEM="\r\nPARAMETER SYNTAX ERROR!\r\n";
410  const char resp_err_paramcnt[] PROGMEM="\r\nPARAMETER COUNT ERROR!\r\n";
411
412
413  uint64_t uint64_mul32 (uint64_t a, uint32_t b)
414  {uint64_t r = 0;
415
416   while (b) {
417     if ((uint8_t)b & 1)
418         r += a;
419     a <<= 1;
420     b >>= 1;
421      }
422      return r;
423  }
424
425
426  uint64_t uint64_div32 (uint64_t a, uint32_t b)
427  {uint64_t r = 0;
428   uint32_t h = 0;
429   uint8_t c = 64, h2;
430
431   /* This looks much smoother in assembler (carry)... */
432      while (c--) {
433     h2 = (h & 0x80000000) ? 1 : 0;
434     h <<= 1;
435     if (a & 0x8000000000000000ULL)
436         h |= 1;
437     a = (a<<1);
438     r = (r<<1);
439     if (h2 || h >= b) {
440         h -= b;
441         r |= 1;
442         }
443      }
444   return r;
445  }
446
447  static unsigned long sm_sqrt(unsigned long x)
448  {
449    register unsigned long xr;  // result register
450    register unsigned long q2;  // scan-bit register
451    register unsigned char f;   // flag (one bit)
452
453    xr = 0;                     // clear result
454    q2 = 0x40000000L;           // higest possible result bit
455    do
456    {
457      if((xr + q2) <= x)
458      {
459        x -= xr + q2;
460        f = 1;                  // set flag
461      }
462      else{
463        f = 0;                  // clear flag
464      }
465      xr >>= 1;
466      if(f){
467        xr += q2;               // test flag
468      }
469    } while(q2 >>= 2);          // shift twice
470    if(xr < x){
471      return xr +1;             // add for rounding
472    }
473    else{
474      return xr;
475    }
476  }
477
478  unsigned int min(unsigned int x, unsigned int y)
479  {
480    if(x < y){
481      return x;
482    }
483    else{
484      return y;
485    }
486  }
487
488
489
490
491  void delay(unsigned int ticks)            //oneskorenie asi 100ms
492  {volatile unsigned char j,k;
493   volatile unsigned int i;
494
495   for(i=0;i<ticks;i++)
496      for(j=0;j<255;j++)
497          k++;
498  }
499
500  void longdelay(char a)
501  {
502   while(a){a--;
503             delay(1000);
504             };
505  }
```

```
506
507
508  // -----------------------------------------------------------------------
509  unsigned int SetDelay (unsigned int t)
510  {
511    return((tim_frac + t + 1)%1000);
512  }
513
514  // -----------------------------------------------------------------------
515  char CheckDelay(unsigned int t)
516  {
517   if(((signed int)t - (signed int)tim_frac)>0) return(0);
518   return(1);
519  }
520
521  // -----------------------------------------------------------------------
522  void Delay_ms(unsigned int w)
523  {
524   unsigned int akt;
525   akt = SetDelay(w);
526   while (!CheckDelay(akt));
527  }
528
529
530  void ioinit(void)              // init portov, watchdog
531  {// define inputs & outputs
532    DDRA = 0x3F;
533    DDRB = 0xBF;
534    DDRC = 0xC0;
535    DDRD = 0xE4;
536    PORTA = 0x00;
537    PORTB = 0x13;
538    PORTC = 0x00;
539    PORTD = 0x18;
540  }
541
542
543
544
545  void uartinit(unsigned char baud_rate)
546  {
547   UBRRL = baud_rate % 256;
548   UBRRH = baud_rate / 256;
549   UCSRA = 0x00;
550   UCSRB = 0x98;     //RXEN=1, TXEN=1, RXIE=1
551   UCSRC = 0x86;     // 8 bit, 1 stop bit, no parity, asynchro
552   rx_ptr=0;
553  // read_ptr=0;
554   rx_overflow=0;
555   new_msg=0;
556  }
557
558  void spi_init(void)
559  {
560    SPCR=0x00;       //SPI Disable
561    SPCR=0x58;       //Enable, Master, MSB first, SPI Mode 2, Cpuclk/4
562  }
563
564
565  void i2c_init(unsigned char i2c_baud)
566  {
567
568    i2c_rxptr_top=1;
569    i2c_rxptr_bot=0;
570    i2c_txptr_top=1;
571    i2c_txptr_bot=0;
572    i2c_rxbuf_err=0;
573    i2c_txbuf_err=0;
574    i2c_buserror=0;
575    TWBR=i2c_baud;
576    TWSR=0x00;               //set prescaller to 1
577    TWAR=(eeprom_read_byte((uint8_t *)EE_I2C_ADDR)<<1);
578    TWCR=0xC5;               //enable TWI, enable interrupt, enable ACK
579  }
580
581
582  void ad5601_write(unsigned char data, unsigned char channel)
583  {unsigned int tmp;
584   tmp=(unsigned int)data<<6;
585
586    if(channel==DAC_CHAN_A) DACSA_0; else DACSB_0;
587    SPDR=tmp>>8;
588    while(!(SPSR&0x80));
589
590    SPDR=tmp&0xFF;
591    while(!(SPSR&0x80));
592    if(channel==DAC_CHAN_A) DACSA_1; else DACSB_1;
593  }
594
595  void ad5302_write(unsigned char data, unsigned char channel)
596  {unsigned int tmp;
597
598   tmp=(unsigned int)data<<4;
599   if(channel==DAC_CHAN_B) tmp|=0x8000;
600
601    DACS_0;
602    SPDR=tmp>>8;
603    while(!(SPSR&0x80));
604
605    SPDR=tmp&0xFF;
606    while(!(SPSR&0x80));
607    DACS_1;
```

```
}

int uartsend (char a, FILE *dummy)
{
  while(!(UCSRA & 0x20)); // wait for UDRE=1
  UDR = a;
  return 0;
}

void uart_SendByte(unsigned char data)
{
  uartsend (data,0);
}



void T0_start(void)
{
    TCNT0 = 0x00;        // set sampling frequency
   OCR0 = 250;
   TIFR  |= 0x02;      // timer1 overflow flag clear
   TCCR0 = 0x0B;       // Timer mode CTC, /64 prescalling
   TIMSK |= 0x02;      // timer1 overflow interrupt enable

   tim_frac=0;
   tim_sec=0;
   tim_min=0;
   tim_hrs=0;
   tim_ena=0;
}



void adc_start(void)
{
 ADMUX=0x40;   //CH0, right adjust result, Vref=AVCC
 ADCSRA=0x9E;  //ADC enable, ADC start, prescaller = 64

 set_sleep_mode(SLEEP_MODE_ADC);
 adc_flag=0;
}


void set_adc_channel(unsigned char channel)
{
 channel&=0x07;
 ADMUX&=0xF8;
 ADMUX|=channel;
}

 unsigned int get_adc_sample(unsigned char channel)
 {
  set_adc_channel(channel);
  ADCSRA|=0x40;
  while(!adc_flag);
  //delay(1);
  adc_flag=0;

  ADCSRA|=0x40;
  while(!adc_flag);
  adc_flag=0;

  return(adc_result);
 }



SIGNAL(SIG_OUTPUT_COMPARE0)  //TIMER0 OCR0, serviced every 1ms
{
 if(solenoid_pwr_timer) solenoid_pwr_timer--;
 if(solenoid_pwr_timer==1) SOL_SET_5V; //set solenoid power to +5V

 if(tim_ena) tim_frac++;

 if(!(tim_frac%10)){
                   if(delay_timer_br) delay_timer_br--;
                   }

 if(tim_frac>=1000) {
                   tim_frac=0;
                   tim_sec++;
                   if(tim_sec>59) {tim_sec=0;
                                   tim_min++;
                                   if(tim_min>59) {tim_min=0;
                                                   tim_hrs++;
                                                   };
                                   };
                   };

}


SIGNAL(SIG_ADC)  //ADC ISR
{static unsigned char i,j;

  j=ADCL;
  i=ADCH;
  adc_result=i<<8;
  adc_result+=j;
  adc_flag=1;
}
```

```
710
711  SIGNAL(SIG_UART_RECV)  //UART receive ISR
712  {
713     rxbuf[rx_ptr]=UDR;
714     uart_SendByte(rxbuf[rx_ptr]);   //echo character
715     if(((rxbuf[rx_ptr]=='\r') || (rxbuf[rx_ptr]=='\n')) && rx_ptr) new_msg=1;
716     if(rx_ptr<(RXBUFSIZE-1)) rx_ptr++; else rx_overflow=1;
717  }
718
719
720  ISR(TWI_vect)  //I2C ISR
721  {
722   //i2c_spy[spyptr]=TWSR;
723   //if(spyptr<29) spyptr++;
724
725   /*
726     //sendhex(TWSR);
727    // MASTER TRANSMIT MODE
728     if((TWSR&0xF8)==0x08){   //START SENT
729                           i2c_send_byte();
730                           if(TWCR&0x80) I2C_CLRTWINT;
731                           return;
732                           }
733     if((TWSR&0xF8)==0x10){   //REPEATED START SENT
734                           i2c_send_byte();
735                           if(TWCR&0x80) I2C_CLRTWINT;
736                           return;
737                           }
738     if((TWSR&0xF8)==0x18){   //SLA+W SENT, ACK recv.
739                           i2c_send_byte();
740                           if(TWCR&0x80) I2C_CLRTWINT;
741                           i2c_busy=0;
742                           return;
743                           }
744     if((TWSR&0xF8)==0x28){   //DATA SENT, ACK recv.
745                           i2c_send_byte();
746                           if(TWCR&0x80) I2C_CLRTWINT;
747                           return;
748                           }
749     if((TWSR&0xF8)==0x20){   //SLA+W SENT, NACK recv.
750                           I2C_STOP;
751                           i2c_busy=1;
752                           return;
753                           }
754     if((TWSR&0xF8)==0x30){   //DATA SENT, NACK recv.
755                           I2C_STOP;
756                           return;
757                           }
758     if((TWSR&0xF8)==0x38){   //ARBITRATION LOST
759                           I2C_STOP;
760                           return;
761                           }
762
763   // MASTER RECIEVER MODE
764     if((TWSR&0xF8)==0x40){   //SLA+R SENT, ACK recv.
765                           //if(TWCR&0x80) I2C_CLRTWINT;
766                           if((TWCR&0x80) && ack_gen) TWCR=I2C_CLRTWINT_ACK|i2c_int_ctrl;
767                           if((TWCR&0x80) && (!ack_gen)) TWCR=I2C_CLRTWINT_NACK|i2c_int_ctrl;
768                           return;
769                           }
770
771     if((TWSR&0xF8)==0x48){   //SLA+R SENT, NACK recv.
772                           I2C_STOP;
773
774                           return;
775                           }
776     if((TWSR&0xF8)==0x50){   //DATA RECIEVED, ACK sent.
777                           i2c_recv_byte();
778                           if((TWCR&0x80) && ack_gen) TWCR=I2C_CLRTWINT_ACK|i2c_int_ctrl;
779                           if((TWCR&0x80) && (!ack_gen)) TWCR=I2C_CLRTWINT_NACK|i2c_int_ctrl;
780                           return;
781                           }
782
783     if((TWSR&0xF8)==0x58){   //DATA RECIEVED, NACK sent.
784                           i2c_recv_byte();
785                           I2C_STOP;
786                           i2c_active=0;
787                           return;
788                           }
789      */
790   // SLAVE RECEIVE MODE
791     if((TWSR&0xF8)==0xA0){   //STOP OR REP. START RECV.
792                          // LEDR_OFF;
793                           if(TWCR&0x80) I2C_CLRTWINT;
794                           return;
795                           }
796     if((TWSR&0xF8)==0x60){   //SLA+W RECV., ACK returned
797                          // LEDR_ON;
798                           if(TWCR&0x80) I2C_CLRTWINT;
799                           return;
800                           }
801     if((TWSR&0xF8)==0x68){   //ARBITRATION LOST, SLA+W RECV., ACK returned
802                           if(TWCR&0x80) I2C_CLRTWINT;
803                           return;
804                           }
805    if((TWSR&0xF8)==0x70){    //GENERAL CALL RECV., ACK returned
806                           if(TWCR&0x80) I2C_CLRTWINT;
807                           return;
808                           }
809    if((TWSR&0xF8)==0x78){    //ARBITRATION LOST,GENERAL CALL RECV., ACK returned
810                           if(TWCR&0x80) I2C_CLRTWINT;
811                           return;
```

```
812                          }
813   if((TWSR&0xF8)==0x80){    //SLA+W RECV.,DATA recv., ACK returned
814                         i2c_recv_byte();
815
816                         //delay(1);
817                          if(TWCR&0x80) I2C_CLRTWINT;
818                          return;
819                          }
820   if((TWSR&0xF8)==0x88){    //SLA+W RECV.,DATA recv., NACK returned
821                         i2c_recv_byte();
822                          if(TWCR&0x80) I2C_CLRTWINT;
823                          return;
824                          }
825   if((TWSR&0xF8)==0x90){    //GENERAL CALL RECV.,DATA recv., ACK returned
826                         i2c_recv_byte();
827                          if(TWCR&0x80) I2C_CLRTWINT;
828                          return;
829                          }
830   if((TWSR&0xF8)==0x98){    //GENERAL CALL RECV. RECV.,DATA recv., NACK returned
831                         i2c_recv_byte();
832                          if(TWCR&0x80) I2C_CLRTWINT;
833                          return;
834                          }
835
836   // SLAVE TRANSMITTER MODE
837    if((TWSR&0xF8)==0xA8){   //SLA+R RECV., ACK returned
838                         //LEDR_ON;
839                          //parse_i2c_command();
840                          i2c_send_byte();
841                          if(TWCR&0x80) {
842                                        if(last_txbyte) TWCR=0x85;
843                                                        else I2C_CLRTWINT;
844                                        };
845                          return;
846                          }
847    if((TWSR&0xF8)==0xB0){   //ARBITRATION LOST, SLA+R RECV., ACK returned
848                          if(TWCR&0x80) I2C_CLRTWINT;
849                          return;
850                          }
851    if((TWSR&0xF8)==0xB8){   //DATA BYTE TRANSMITTED, ACK received
852                         i2c_send_byte();
853                          if(TWCR&0x80) {
854                                        if(last_txbyte) TWCR=0x85;
855                                                        else I2C_CLRTWINT;
856                                        };
857                          return;
858                          }
859   if((TWSR&0xF8)==0xC0){    //DATA BYTE TRANSMITTED, NOT ACK received
860                          if(TWCR&0x80) I2C_CLRTWINT;
861                          return;
862                          }
863   if((TWSR&0xF8)==0xC8){    //LAST DATA BYTE TRANSMITTED, NOT ACK received
864                          if(TWCR&0x80) I2C_CLRTWINT;
865                          return;
866                          }
867
868
869   // I2C BUS ERROR
870    if((TWSR&0xF8)==0x00){   //I2C BUS ERROR
871                          if(i2c_buserror!=0xFF) i2c_buserror++;
872                          if(TWCR&0x80) I2C_CLRTWINT;
873                          return;
874                          }
875
876   //undefined state
877   if(TWCR&0x80){
878                  I2C_CLRTWINT;
879                 };
880  }
881
882  void i2c_putchar(unsigned char byte)
883  {unsigned char i;
884
885   i=i2c_txptr_top;
886   i2c_txbuf[i2c_txptr_top]=byte;
887   i2c_txptr_top++;
888   if(i2c_txptr_top==I2C_TXBUFSIZE) i2c_txptr_top=0;
889   if(i2c_txptr_top==i2c_txptr_bot) {i2c_txbuf_err=1;
890                                     i2c_txptr_top=i;
891                                     };
892  }
893
894  void i2c_send_byte(void)
895  {unsigned char i;
896
897   i=i2c_txptr_bot+1;
898   if(i>=I2C_TXBUFSIZE) i=0;
899   if(i==i2c_txptr_top) {
900                         return;};
901   i2c_txptr_bot=i;
902   TWDR=i2c_txbuf[i2c_txptr_bot];
903
904   last_txbyte=0;
905   i=i2c_txptr_bot+1;
906   if(i>=I2C_TXBUFSIZE) i=0;
907   if(i==i2c_txptr_top){ last_txbyte=1;
908                        i2c_txptr_top=1;
909                        i2c_txptr_bot=0;
910                        };
911  }
912
913  void i2c_recv_byte(void)
```

```
914  {static unsigned int i;
915
916   i=TWDR;
917
918  // j=i2c_rxptr_top;
919   i2c_rxbuf[i2c_rxptr_top]=i;
920   i2c_rxptr_top++;
921   if(i2c_rxptr_top>=I2C_RXBUFSIZE) i2c_rxptr_top=0;
922   if(i2c_rxptr_top==i2c_rxptr_bot) {i2c_rxbuf_err=1;
923                                    };
924   if(i=='\r' || i=='\n') i2c_newmsg=1;
925  }
926
927  void i2c_txflush(void)
928  {
929   i2c_txptr_top=1;
930   i2c_txptr_bot=0;
931   i2c_txbuf_err=0;
932  }
933
934  unsigned char i2c_getchar(void)
935  {unsigned int tmp;
936
937   if(i2c_rxbuf_err) return(0);
938   tmp=i2c_rxptr_bot;
939   i2c_rxptr_bot++;
940   if(i2c_rxptr_bot>=I2C_RXBUFSIZE) i2c_rxptr_bot=0;
941   if(i2c_rxptr_bot==i2c_rxptr_top) {i2c_rxptr_bot=tmp; return(0);};
942   return(i2c_rxbuf[i2c_rxptr_bot]);
943  }
944
945  void parse_i2c_command(void)
946  {unsigned char i=0;
947   unsigned int tmp;
948   signed int param1=0,param2=0;
949
950     i2c_status=COM_NOTVALID;
951     i2c_newmsg=0;
952
953     i=i2c_getchar();
954     if(i==COM_TEST){last_command=COM_TEST;
955                  i2c_status=NO_ERROR;
956                  LEDR_TOGGLE;
957                 }
958     else if(i==COM_BRCONTROL){
959                  i2c_status=NO_ERROR;
960                  i=i2c_getchar();
961                  if(i==0 || i==1){
962                     br1_perfusion=i;
963                     eeprom_write_byte((uint8_t *)EE_BR1_PERFUSION,br1_perfusion);
964                     if(i==0) stop_pump();
965                     }
966                 }
967     else if(i==COM_BRSTOP){
968                  i2c_status=NO_ERROR;
969                  br1_changing_medium=0;
970                   br1_make_sample=0;
971                   eeprom_write_byte((uint8_t *)EE_BR1_MAKESAMPLE,br1_make_sample);
972                   br1_init();
973                   progress[0]=0;
974                   progress[1]=0;
975                   progress[2]=0;
976                   progress[3]=0;
977                   progress[4]=0;
978                   stop_pump();
979                 }
980     else if(i==COM_BRSPEED){
981                  i2c_status=NO_ERROR;
982                  tmp=i2c_getchar();
983                  tmp=tmp*256;
984                  tmp+=i2c_getchar();
985                  if(tmp<1 || tmp>500) i2c_status=COM_PARAM_ERR;
986                     else{
987                         br1_pump_speed=tmp;
988                         eeprom_write_word((uint16_t *)EE_BR_SPEED,br1_pump_speed);
989                         }
990                 }
991     else if(i==COM_PREPSAMPLE){
992                  i2c_status=NO_ERROR;
993                  br1_make_sample=1;
994                  eeprom_write_byte((uint8_t *)EE_BR1_MAKESAMPLE,br1_make_sample);
995                 }
996     else if(i==COM_BRCHANGEMED){
997                  i2c_status=NO_ERROR;
998                  tmp=i2c_getchar();
999                  tmp=tmp*256;
1000                 tmp+=i2c_getchar();
1001                 param1=(int16_t)tmp;   //1st parameter - pumped volume
1002                 if(param1<-10000 || param1>10000) i2c_status=COM_PARAM_ERR;
1003                 tmp=i2c_getchar();
1004                 tmp=tmp*256;
1005                 tmp+=i2c_getchar();
1006                 param2=(int16_t)tmp;   //2nd parameter - pump speed
1007                 if(param2<1 || param2>500) i2c_status=COM_PARAM_ERR;
1008                 if(i2c_status==NO_ERROR){
1009                                          medium_change_vol=param1;
1010                                          medium_change_speed=param2;
1011                                          br1_changing_medium=1;
1012                                        }
1013                 }
1014     else if(i==COM_VALVE){
1015                  i2c_status=NO_ERROR;
```

```
1016                tmp=i2c_getchar();
1017                param1=tmp;        //1st parameter - valve number
1018                if(param1<1 || param1>8) i2c_status=COM_PARAM_ERR;
1019                tmp=i2c_getchar();
1020                param2=tmp;        //2nd parameter - new state: 0-OFF, 1-ON
1021                if(param2<0 || param2>1) i2c_status=COM_PARAM_ERR;
1022                if(i2c_status==NO_ERROR){
1023                                        control_solenoid(param1, param2);
1024                                    }
1025              }
1026   else if(i==COM_BRSTATUS){
1027                            i2c_status=NO_ERROR;
1028                            br_status();
1029                           }
1030   else if(i==COM_STARTPUMP){
1031                            i2c_status=NO_ERROR;
1032                            tmp=i2c_getchar();
1033                            tmp=tmp*256;
1034                            tmp+=i2c_getchar();
1035                            param1=(int16_t)tmp;   //1st parameter - pumped volume
1036                            if(param1<-10000 || param1>10000) i2c_status=COM_PARAM_ERR;
1037                            tmp=i2c_getchar();
1038                            tmp=tmp*256;
1039                            tmp+=i2c_getchar();
1040                            param2=(int16_t)tmp;   //2nd parameter - pump speed
1041                            if(param2<0 || param2>500) i2c_status=COM_PARAM_ERR;
1042                            if(i2c_status==NO_ERROR){
1043                                                    if(param2==0) stop_pump();
1044                                                         else{
1045                                                             start_pump(param1,param2);
1046                                                         }
1047
1048                                                }
1049
1050                            }
1051
1052
1053  i2c_rxptr_top=1;
1054  i2c_rxptr_bot=0;
1055 }
1056
1057 void sm_start_timer(void)
1058 {//set clock - division factor 64
1059   TCCR1B |= ((0<<CS12)|(1<<CS11)|(1<<CS10));
1060   // Timer/Counter 1 Output Compare A Match Interrupt enable.
1061   TIMSK |= (1<<OCIE1A);
1062 }
1063
1064 void sm_stop_timer(void)
1065 {// stop the clock
1066   TCCR1B &= ~0x07;
1067   // Timer/Counter 1 Output Compare A Match Interrupt enable.
1068   TIMSK |= (1<<OCIE1A);
1069 }
1070
1071 void sm_driver_reset(void)
1072 {
1073  SM_DISABLE;
1074
1075  SM_RESET;
1076  delay(10);
1077  SM_UNRESET;
1078 }
1079
1080 //Sets L6208 control pins according to controldata bits 0-3
1081 //(CONTROL, HALF/FULL, CW/CCW, EN)
1082 void sm_set_driver_control(uint8_t controldata)
1083 {
1084  if(controldata&0x01) SM_ENABLE; else SM_DISABLE;
1085  if(controldata&0x02) SM_DIR_CW; else SM_DIR_CCW;
1086  if(controldata&0x04) SM_STEP_HALF; else SM_STEP_FULL;
1087 }
1088
1089 //sets the max. phase current for stepper motors
1090 //iphasea, iphaseb - current limit in mA, range: 0 - 3000 mA
1091 //1V of DAC voltage corresponds to 2000mA phase current
1092 void sm_set_iphase(uint16_t  iphasea, uint16_t  iphaseb)
1093 {uint32_t voltcode;
1094
1095  if(iphasea>3000) iphasea=3000;
1096  if(iphaseb>3000) iphaseb=3000;
1097 // if(iphasea==0 && iphaseb!=0) iphasea=iphaseb;
1098
1099  voltcode=(uint32_t)iphasea*((uint16_t)(1.024*1000)); // scaled up by factor 100
1100  voltcode=voltcode/20000;
1101  if(voltcode>=256) voltcode=255;
1102 #ifdef PCB216MB01
1103  ad5601_write((uint8_t)voltcode, DAC_CHAN_A ); //write to DAC - value for Phase A
1104 #else
1105  ad5302_write((uint8_t)voltcode, DAC_CHAN_A ); //write to DAC - value for Phase A
1106 #endif
1107
1108  voltcode=(uint32_t)iphaseb*((uint16_t)(1.024*1000)); // scaled up by factor 100
1109  voltcode=voltcode/20000;
1110  if(voltcode>=256) voltcode=255;
1111 #ifdef PCB216MB01
1112  ad5601_write((uint8_t)voltcode, DAC_CHAN_B);  //write to DAC - value for Phase B
1113 #else
1114  ad5302_write((uint8_t)voltcode, DAC_CHAN_B);  //write to DAC - value for Phase B
1115 #endif
1116
1117 }
```

```
1118
1119 void sm_set_istandby(speedRampData *srd_motorx, int16_t istandby)
1120 {
1121  srd_motorx->standby_iphase = istandby;
1122 }
1123
1124
1125
1126 // *  \param imax_phase  Max phase current, in mA (range: 50 - 3000mA).
1127 void sm_motor_init(speedRampData *srd_motorx, uint8_t microsteps,
1128                    uint16_t max_iphase, uint16_t standby_iphase)
1129 {
1130
1131  if(microsteps && microsteps!=2 && microsteps!=4 &&
1132     microsteps!=8 && microsteps!=16 && microsteps!=32) microsteps=2;
1133
1134  srd_motorx->run_state = STOPPED;
1135  srd_motorx->step_count = 0;
1136  srd_motorx->mstep_counter = 0;
1137  srd_motorx->rest = 0;
1138  srd_motorx->abs_position = 0;
1139
1140  //srd_motorx->driver_state=0;
1141  srd_motorx->microstep_inc=0;
1142  if(microsteps)srd_motorx->microstep_inc=MICROSTEPS_TABSIZE/microsteps;
1143  srd_motorx->mstep_counter = srd_motorx->microstep_inc;    //initial state is
1144                                          // the first state after zero
1145  srd_motorx->microsteps=microsteps;
1146  srd_motorx->max_iphase = max_iphase;
1147  srd_motorx->standby_iphase = standby_iphase;
1148  srd_motorx->i_phaseb = (standby_iphase/100)*(microstep_tab[0][srd_motorx->microstep_inc]);
1149  srd_motorx->i_phasea = (standby_iphase/100)*(microstep_tab[1][srd_motorx->microstep_inc]);
1150
1151  srd_motorx->driver_control=0x05;  //enable on, Halfsteps
1152 #ifdef SLOWDECAY
1153  srd_motorx->driver_control|=0x08; //additionaly set decay mode
1154 #endif
1155
1156
1157  sm_set_driver_control(srd_motorx->driver_control);
1158
1159  //initial state - phase a current set to 0, phase b current set to standby
1160  sm_set_iphase(srd_motorx->i_phasea, srd_motorx->i_phaseb);
1161
1162 }
1163
1164 // *  disables motor driver and timer channel
1165 void sm_motor_deinit(speedRampData *srd_motorx)
1166 {
1167
1168  // Timer/Counter 1 Output Compare A Match Interrupt disable.
1169   TIMSK &= ~(1<<OCIE1A);
1170
1171  srd_motorx->run_state = STOP;
1172  srd_motorx->driver_control&=(~0x01);  //enable OFF
1173
1174  sm_set_driver_control(srd_motorx->driver_control);
1175 }
1176
1177
1178 void sm_driver_gostandby(speedRampData *srd_motorx)
1179 {int8_t next_xidx,next_yidx;
1180  int16_t i_phasea,i_phaseb;
1181
1182  next_yidx=(0x66>>srd_motorx->driver_state)&0x01;
1183  next_xidx=srd_motorx->mstep_counter;
1184
1185  i_phaseb=(srd_motorx->standby_iphase/100)*(microstep_tab[next_yidx][next_xidx]);
1186  i_phasea=(srd_motorx->standby_iphase/100)*(microstep_tab[next_yidx^0x01][next_xidx]);
1187  sm_set_iphase(i_phasea, i_phaseb);
1188 }
1189
1190
1191
1192 void speed_cntr_Move(int32_t step, uint16_t accel, uint16_t decel, uint16_t speed)
1193 {speedRampData *srd_motorx;
1194  int8_t next_xidx,next_yidx;
1195  volatile uint64_t tmp;
1196
1197  if(step==0) return;
1198  srd_motorx=&srd_sm1;
1199
1200  //wait until previous movement will finish
1201  while(srd_motorx->run_state != STOPPED);
1202
1203  srd_motorx->step_count = 0;
1204
1205 #ifdef SM_REVERSE_ROTATION
1206    step = -step;
1207 #endif
1208
1209   // Set direction from sign on step value.
1210   if(step < 0){
1211     srd_motorx->dir = CCW;
1212     step = -step;
1213     srd_motorx->driver_control&=(~0x02);
1214     if(srd_motorx->microstep_inc>0) srd_motorx->microstep_inc = -srd_motorx->microstep_inc;
1215   }
1216   else{
1217     srd_motorx->dir = CW;
1218     srd_motorx->driver_control|=0x02;          //change direction
1219     if(srd_motorx->microstep_inc<0) srd_motorx->microstep_inc = -srd_motorx->microstep_inc;
```

```
1220   }
1221
1222    sm_set_driver_control(srd_motorx->driver_control);
1223
1224    // calculate phase currents for the next step
1225    next_yidx=srd_motorx->driver_state;
1226    if(next_yidx==0 || next_yidx==7 || next_yidx==3 || next_yidx==4) next_yidx=0;
1227        else next_yidx=1;
1228    next_xidx=srd_motorx->mstep_counter+srd_motorx->microstep_inc;
1229    if(next_xidx>=MICROSTEPS_TABSIZE){ next_xidx=0;
1230                                       next_yidx=(~next_yidx)&0x01;
1231                                       }
1232    if(next_xidx<0){ next_xidx+=MICROSTEPS_TABSIZE;
1233                     next_yidx^=0x01;
1234                     }
1235    srd_motorx->i_phaseb=
1236            (srd_motorx->max_iphase/100)*(microstep_tab[next_yidx][next_xidx]);
1237    srd_motorx->i_phasea=
1238            (srd_motorx->max_iphase/100)*(microstep_tab[(~next_yidx)&0x01][next_xidx]);
1239
1240
1241    // If moving only 1 step.
1242      if(step == 1){
1243        // Move one step...
1244        srd_motorx->accel_count = -1;
1245        // ...in DECEL state.
1246        srd_motorx->run_state = DECEL;
1247        // Just a short delay so main() can act on 'running'.
1248        srd_motorx->step_delay = 1000;
1249       // OCR1A=100;
1250      }
1251
1252    // Only move if number of steps to move is not zero.
1253    else if(step != 0){
1254      // Refer to documentation for detailed information about these calculations.
1255
1256      // Set max speed limit, by calc min_delay to use in timer.
1257      // min_delay = (alpha / tt)/ w
1258      srd_motorx->min_delay = A_T_x1000 / (speed*srd_motorx->microsteps);
1259
1260      // Set accelration by calc the first (c0) step delay .
1261      // step_delay = 1/tt * sqrt(2*alpha/accel)
1262      // step_delay =
1263     // ( tfreq*0.676/100 )*100 * sqrt( (2*alpha*10000000000) / (accel*100) )/10000
1264      srd_motorx->step_delay =
1265              (T1_FREQ_148 * sm_sqrt(A_SQ / (accel*srd_motorx->microsteps)))/100;
1266
1267      // Find out after how many steps does the speed hit the max speed limit.
1268      // max_s_lim = speed^2 / (2*alpha*accel)
1269      srd_motorx->max_s_lim = (int32_t)speed*speed/
1270              (int32_t)(((int32_t)A_x20000*accel*100)/(100*srd_motorx->microsteps));
1271
1272      // If we hit max speed limit before 0,5 step it will round to 0.
1273      // But in practice we need to move atleast 1 step to get any speed at all.
1274      if(srd_motorx->max_s_lim == 0){
1275         srd_motorx->max_s_lim = 1;
1276      }
1277
1278      // Find out after how many steps we must start deceleration.
1279      // n1 = (n1+n2)decel / (accel + decel)
1280      tmp=uint64_mul32(step,decel);
1281      tmp=uint64_div32(tmp, (accel+decel));
1282      srd_motorx->accel_lim = (uint32_t)(tmp);
1283      // We must accelrate at least 1 step before we can start deceleration.
1284      if(srd_motorx->accel_lim == 0){
1285         srd_motorx->accel_lim = 1;
1286      }
1287
1288      // Use the limit we hit first to calc decel.
1289      if(srd_motorx->accel_lim <= srd_motorx->max_s_lim){
1290        srd_motorx->decel_val = srd_motorx->accel_lim - step;
1291      }
1292      else{
1293        srd_motorx->decel_val = -((int32_t)srd_motorx->max_s_lim*accel)/decel;
1294      }
1295      // We must decelrate at least 1 step to stop.
1296      if(srd_motorx->decel_val == 0){
1297        srd_motorx->decel_val = -1;
1298      }
1299
1300      // Find step to start decleration.
1301      srd_motorx->decel_start = step + srd_motorx->decel_val;
1302
1303      // If the maximum speed is so low that we dont need to go via accelration state.
1304      if(srd_motorx->step_delay <= srd_motorx->min_delay){
1305        srd_motorx->step_delay = srd_motorx->min_delay;
1306        srd_motorx->run_state = RUN;
1307      }
1308      else{
1309        srd_motorx->run_state = ACCEL;
1310      }
1311
1312      // Reset counter.
1313      srd_motorx->accel_count = 0;
1314    }
1315
1316    OCR1A=100;
1317    sm_start_timer();
1318 }
1319
1320 void speed_cntr_Init_Timer1(void)
1321 {
```

```
1322   // Tells what part of speed ramp we are in.
1323   srd_sm1.run_state = STOPPED;
1324   // Timer/Counter 1 in mode 4 CTC (Not running).
1325   TCCR1B = (1<<WGM12);
1326   // Timer/Counter 1 Output Compare A Match Interrupt enable.
1327   TIMSK |= (1<<OCIE1A);
1328 }
1329
1330
1331 void Motor_Init(void)
1332 {
1333   // Init of IO pins
1334   sm_driver_reset();
1335
1336   sm_motor_init(&srd_sm1, 32, 750, 100);
1337   srd_sm1.driver_state=0;
1338   SM_CLK_0;
1339
1340   // Init of Timer/Counter1
1341   speed_cntr_Init_Timer1();
1342 }
1343
1344
1345 void sm1_driver_Step(void)
1346 {int8_t next_xidx,next_yidx;
1347  uint8_t step_flag=0;
1348
1349  sm_set_iphase(srd_sm1.i_phasea, srd_sm1.i_phaseb);
1350
1351  if(srd_sm1.microstep_inc>0) srd_sm1.abs_position++; else srd_sm1.abs_position--;
1352  srd_sm1.mstep_counter+=srd_sm1.microstep_inc;
1353  if(srd_sm1.mstep_counter>=MICROSTEPS_TABSIZE){srd_sm1.mstep_counter=0;
1354                                                step_flag=1;
1355                                               }
1356        else if(srd_sm1.mstep_counter<=0) { if(srd_sm1.mstep_counter<0)
1357                                              srd_sm1.mstep_counter+=MICROSTEPS_TABSIZE;
1358                                            step_flag=1;
1359                                           }
1360                    else if(srd_sm1.driver_state&0x01){
1361                                                       step_flag=1;
1362                                                      }
1363
1364  if(step_flag==1 ) { step_flag=0;
1365                      SM_CLK_1;      //set SPM1CLK to 1
1366                      if(srd_sm1.dir==CW) srd_sm1.driver_state++; else srd_sm1.driver_state--;
1367                      srd_sm1.driver_state&=0x07;
1368                      SM_CLK_0;      //set SPM1CLK to 0
1369                      }
1370
1371  next_yidx=(0x66>>srd_sm1.driver_state)&0x01;
1372  next_xidx=srd_sm1.mstep_counter+srd_sm1.microstep_inc;
1373  if(next_xidx>=MICROSTEPS_TABSIZE){ next_xidx=0;
1374                                     next_yidx^=0x01;
1375                                     }
1376  if(next_xidx<0){ next_xidx+=MICROSTEPS_TABSIZE;
1377                   next_yidx^=0x01;
1378                 }
1379
1380  srd_sm1.i_phaseb=(srd_sm1.max_iphase/100)*(microstep_tab[next_yidx][next_xidx]);
1381  srd_sm1.i_phasea=(srd_sm1.max_iphase/100)*(microstep_tab[next_yidx^0x01][next_xidx]);
1382 }
1383
1384
1385 //void sm1_update(void)
1386 SIGNAL(SIG_OUTPUT_COMPARE1A)
1387 {
1388   OCR1A = srd_sm1.step_delay;
1389 //LEDR_ON;
1390      switch(srd_sm1.run_state) {
1391        case STOP:
1392          srd_sm1.step_count = 0;
1393          srd_sm1.rest = 0;
1394          sm_stop_timer();  // Stop Timer/Counter 1.
1395          sm_driver_gostandby(&srd_sm1);
1396          srd_sm1.run_state = STOPPED;
1397          break;
1398
1399        case ACCEL:
1400          sm1_driver_Step();
1401          srd_sm1.step_count++;
1402          srd_sm1.accel_count++;
1403          srd_sm1.new_step_delay = srd_sm1.step_delay - (((2 * (int32_t)srd_sm1.step_delay)
1404                                                 + srd_sm1.rest)/(4 * srd_sm1.accel_count + 1));
1405          srd_sm1.rest = ((2 * (int32_t)srd_sm1.step_delay)+
1406                                                   srd_sm1.rest)%(4 * srd_sm1.accel_count + 1);
1407          // Check if we should start decelration.
1408          if(srd_sm1.step_count >= srd_sm1.decel_start) {
1409            srd_sm1.accel_count = srd_sm1.decel_val;
1410            srd_sm1.run_state = DECEL;
1411          }
1412          // Check if we hitted max speed.
1413          else if(srd_sm1.new_step_delay <= srd_sm1.min_delay) {
1414              srd_sm1.last_accel_delay = srd_sm1.new_step_delay;
1415              srd_sm1.new_step_delay = srd_sm1.min_delay;
1416              srd_sm1.rest = 0;
1417            srd_sm1.run_state = RUN;
1418          }
1419          break;
1420
1421        case RUN:
1422          sm1_driver_Step();
1423          srd_sm1.step_count++;
```

```
1424          srd_sm1.new_step_delay = srd_sm1.min_delay;
1425          // Check if we should start decelration.
1426          if(srd_sm1.step_count >= srd_sm1.decel_start) {
1427            srd_sm1.accel_count = srd_sm1.decel_val;
1428            // Start decelration with same delay as accel ended with.
1429            srd_sm1.new_step_delay = srd_sm1.last_accel_delay;
1430            srd_sm1.run_state = DECEL;
1431          }
1432          break;
1433
1434        case DECEL:
1435          sm1_driver_Step();
1436          srd_sm1.step_count++;
1437          srd_sm1.accel_count++;
1438          srd_sm1.new_step_delay = srd_sm1.step_delay -
1439            (((2*(int32_t)srd_sm1.step_delay)+srd_sm1.rest)/(4*srd_sm1.accel_count+1));
1440          srd_sm1.rest=((2*(int32_t)srd_sm1.step_delay)+
1441                                    srd_sm1.rest)%(4*srd_sm1.accel_count+1);
1442          // Check if we at last step
1443          if(srd_sm1.accel_count >= 0){
1444            srd_sm1.run_state = STOP;
1445          }
1446          break;
1447      }
1448      srd_sm1.step_delay = srd_sm1.new_step_delay;
1449 //   LEDR_OFF;
1450 }
1451
1452
1453
1454
1455 void control_solenoid(uint8_t solenoid_id, unsigned char state)
1456 {unsigned char data;
1457  volatile uint8_t *dataport;
1458  //char msgstring[32];
1459  const char *tmpstr;
1460
1461
1462  switch(solenoid_id){
1463     case 1:
1464             data=0x01;
1465             dataport=&PORTA;
1466             break;
1467    case 2:
1468             data=0x02;
1469             dataport=&PORTA;
1470             break;
1471    case 3:
1472             data=0x04;
1473             dataport=&PORTA;
1474             break;
1475    case 4:
1476             data=0x08;
1477             dataport=&PORTA;
1478             break;
1479    case 5:
1480             data=0x10;
1481             dataport=&PORTA;
1482             break;
1483    case 6:
1484             data=0x20;
1485             dataport=&PORTA;
1486             break;
1487    case 7:
1488             data=0x40;
1489             dataport=&PORTC;
1490             break;
1491    case 8:
1492             data=0x80;
1493             dataport=&PORTC;
1494             break;
1495    default:
1496             data=0;
1497             dataport=&PORTA;
1498             break;
1499    };
1500
1501  if(state==OFF){*dataport&=(~data);
1502                 tmpstr="OFF";
1503                 };
1504  if(state==ON) {*dataport|= data;
1505                 solenoid_pwr_timer=SOLENOID_12V_TIME;
1506                 SOL_SET_12V;
1507                 tmpstr="ON";
1508                };
1509
1510 // sprintf(msgstring,"\n\rSWITCHING SOLENOID %d %s ",solenoid_id,tmpstr);
1511 // uart_SendString(msgstring);
1512  printf_P(PSTR("\n\rSWITCHING SOLENOID %d %s"),solenoid_id,tmpstr);
1513 }
1514
1515
1516 //parameters:
1517 //volume - volume to be pumped in ul units,
1518 //negative value means backward pumping, range 1-16000 ul
1519 //speed - pump speed in ul/min, allowed range: 1 - 5000 ul/min
1520 void start_pump(int16_t volume, uint16_t speed)
1521 {int64_t steps,tmp;
1522  uint16_t accel=628;
1523  int8_t sign=1;
1524  int32_t stepsdir,tmp2;
1525  speedRampData *srd_motorx;
```

```
1526 // char tmpstr[64];
1527  //const char *pstr=str_unknown;
1528
1529  last_speed=speed;
1530  srd_motorx=&srd_sm1;
1531
1532  if(!speed || !volume) return;
1533  if(speed>5000) speed=5000;
1534
1535  if(volume<0){sign=-sign;
1536              volume=-volume;
1537             };
1538
1539  //calculate number of steps for required volume
1540  steps= uint64_mul32(volume,((uint32_t)32*FSPR*1000));
1541  steps= uint64_div32(steps, (uint32_t)PUMP1_VOL_PER_REVOLUTION);
1542  stepsdir=(int32_t)steps;
1543  stepsdir*=sign;
1544
1545  //calculate speed in radians per second *1000
1546  tmp=uint64_mul32(speed,((uint32_t)1000*6283));
1547  tmp+=(uint32_t)PUMP1_VOL_PER_REVOLUTION*60/2; //decrease rounding error
1548  tmp=uint64_div32(tmp,((uint32_t)PUMP1_VOL_PER_REVOLUTION*60));
1549
1550  //adjust acceleration and deceleration
1551  //accel=(speed/1000)*628;
1552  tmp2=speed*628;
1553  accel=(uint16_t)(tmp2/1000);
1554  if (accel<628) accel=628;
1555
1556  srd_motorx->max_iphase = 750;
1557  if(speed>=100) srd_motorx->max_iphase = 1000;
1558  if(speed>=1000) srd_motorx->max_iphase = 1500;
1559
1560  speed_cntr_Move(stepsdir, accel, accel, (uint16_t)tmp);
1561
1562  //print information
1563  printf_P(PSTR("\n\rStarting BR pump. Pumping %dul @ %dul/min."),volume*sign,speed);
1564 }
1565
1566 void stop_pump(void)
1567 {speedRampData *srd_motorx;
1568
1569 srd_motorx=&srd_sm1;
1570
1571 if(srd_motorx->run_state == DECEL) return; //return if already decelerating
1572 while(srd_motorx->run_state == ACCEL); //wait until acceleration is over
1573 if(srd_motorx->run_state == RUN){
1574                                  srd_motorx->step_count = srd_motorx->decel_start;
1575                                 }
1576  printf_P(PSTR("\n\rBR Pump has been stopped."));
1577 }
1578
1579 //checks if pump is running
1580 uint8_t get_pump_status(void)
1581 {speedRampData *srd_motorx=&srd_sm1;
1582
1583 if(srd_motorx->run_state == STOPPED) return (OFF);
1584    else
1585        return(ON);
1586 }
1587
1588 //checks the pumping progress. returns 0 if the pump is stopped, otherwise 1
1589 uint8_t get_pump_progress(brtask_statistics *stats)
1590 {speedRampData *srd_motorx=&srd_sm1;
1591  uint32_t tmp;
1592 // uint32_t time2end;
1593  //uint8_t percent_done;
1594  uint8_t rtn_val;
1595
1596  if(srd_motorx->run_state == STOPPED){rtn_val=0;
1597                                       stats->percent_done=100;
1598                                       stats->time2end=0;
1599                                     //  printf_P(PSTR("\n\rBR pump is stopped.\n\r"));
1600                                       }
1601    else{
1602         rtn_val=1;
1603         tmp=100UL*srd_motorx->step_count;
1604         stats->percent_done=(uint8_t)(tmp/srd_motorx->decel_start);
1605
1606         if(srd_motorx->decel_start > srd_motorx->step_count)
1607                     tmp=srd_motorx->decel_start-srd_motorx->step_count; else tmp=0;
1608         stats->time2end=(tmp*srd_motorx->min_delay)/250000UL;
1609
1610
1611        // printf_P(PSTR("\n\rBR pump is running.\n\r"));
1612        // printf_P(PSTR("\n\rBR pump status: \n\rProgress %d %% \n\rTime to end: %d sec"),
1613        //                                    stats->percent_done,(int16_t)stats->time2end);
1614         }
1615
1616  return(rtn_val);
1617 }
1618
1619
1620
1621 void get_medchange_stats(brtask_statistics *stats, int16_t volume, uint16_t speed)
1622 {brtask_statistics stat1;
1623  uint32_t total_time2end;
1624
1625  total_time2end=(60UL*(abs(volume)+abs(CALIBRATED_DEAD_VOLUME)))/speed;
1626  //printf_P(PSTR("\n\rTotal time to end: %u\n\r"),total_time2end);
1627
```

```
1628   if(progress[1]<5){stats->percent_done=0;
1629                     stats->time2end=total_time2end;
1630                    }
1631   else if(progress[1]<6){
1632          stats->time2end=(60ul*abs(CALIBRATED_DEAD_VOLUME))/speed;
1633          get_pump_progress(&stat1);
1634          stats->time2end += stat1.time2end;
1635          stats->percent_done=100-
1636                (uint8_t)((100UL*stats->time2end+total_time2end/2)/total_time2end);
1637          }
1638   else if(progress[1]<8){
1639          stats->time2end = (60ul*abs(CALIBRATED_DEAD_VOLUME))/speed;
1640          stats->percent_done=100-
1641                (uint8_t)((100UL*stats->time2end+total_time2end/2)/total_time2end);
1642          }
1643   else{
1644          get_pump_progress(&stat1);
1645          stats->time2end = stat1.time2end;
1646          stats->percent_done=100-
1647                (uint8_t)((100UL*stats->time2end+total_time2end/2)/total_time2end);
1648          }
1649   if(stats->percent_done>100) stats->percent_done=0;
1650 }
1651
1652 void get_mix_stats(brtask_statistics *stats, int16_t volume, uint16_t speed)
1653 {brtask_statistics stat1;
1654  uint32_t total_time2end;
1655
1656  total_time2end=(60ul*(abs(volume)))/speed;
1657  //printf_P(PSTR("\n\rTotal time to end: %u\n\r"),total_time2end);
1658
1659   if(progress[3]<5){stats->percent_done=0;
1660                     stats->time2end=total_time2end;
1661                    }
1662   else{
1663      get_pump_progress(&stat1);
1664      stats->time2end = stat1.time2end;
1665      stats->percent_done=100-
1666            (uint8_t)((100UL*stats->time2end+total_time2end/2)/total_time2end);
1667      }
1668   if(stats->percent_done>100) stats->percent_done=0;
1669 }
1670
1671 void get_prepsamp_stats(brtask_statistics *stats)
1672 {brtask_statistics stat1;
1673  uint32_t total_time2end;
1674
1675  total_time2end=60ul*(abs(BR1_MIX_VOL))/BR1_MIX_SPEED;
1676  total_time2end+=60ul*(abs(BR1_SAMP_VOL))/BR1_SAMP_SPEED;
1677  //printf_P(PSTR("\n\rTotal time to end: %u\n\r"),total_time2end);
1678
1679   if(progress[4]==0){
1680         get_mix_stats(stats, BR1_MIX_VOL,BR1_MIX_SPEED);
1681         get_medchange_stats(&stat1, BR1_SAMP_VOL,BR1_SAMP_SPEED);
1682
1683         stats->time2end+=stat1.time2end;
1684         stats->percent_done=100-
1685            (uint8_t)((100UL*stats->time2end+total_time2end/2)/total_time2end);
1686        }
1687   else{
1688          get_mix_stats(stats, BR1_MIX_VOL,BR1_MIX_SPEED);
1689          stats->percent_done=100-
1690                (uint8_t)((100UL*stats->time2end+total_time2end/2)/total_time2end);
1691        }
1692   if(stats->percent_done>100) stats->percent_done=0;
1693 }
1694
1695
1696 void br_status(void)
1697 {uint8_t i;
1698  brtask_statistics stat1;
1699
1700 /*
1701   br_status_flags
1702   br1_pump_speed
1703   valves
1704   pump: percent done
1705   pump: time to end
1706
1707   task: percent done
1708   task: time to end
1709
1710   perfusion speed
1711   medium change speed
1712   medium change volume
1713 */
1714  i2c_txflush();
1715
1716  i=0;
1717  if(br1_perfusion) i|=0x01;
1718  if(br1_make_sample) i|=0x02;
1719  if(br1_changing_medium) i|=0x04;
1720  if(get_pump_status()==ON) i|=0x80;
1721  i2c_putchar(i);
1722  /*
1723  if(br1_changing_medium){
1724                          i2c_putchar(medium_change_speed/256);
1725                          i2c_putchar(medium_change_speed%256);
1726                          }
1727         else{
1728             i2c_putchar(br1_pump_speed/256);
1729             i2c_putchar(br1_pump_speed%256);
```

```
1730             }
1731  */
1732  i2c_putchar(last_speed/256);
1733  i2c_putchar(last_speed%256);
1734
1735  i=PINA&0x3F;
1736  i|=PINC&0xC0;
1737  i2c_putchar(i);
1738
1739  get_pump_progress(&stat1);
1740  i2c_putchar(stat1.percent_done);
1741  i2c_putchar(stat1.time2end/256);
1742  i2c_putchar(stat1.time2end%256);
1743
1744  if(br1_make_sample) get_prepsamp_stats(&stat1);
1745  else if(br1_changing_medium)
1746        get_medchange_stats(&stat1,medium_change_vol,medium_change_speed);
1747
1748  i2c_putchar(stat1.percent_done);
1749  i2c_putchar(stat1.time2end/256);
1750  i2c_putchar(stat1.time2end%256);
1751
1752  i2c_putchar(br1_pump_speed/256);
1753  i2c_putchar(br1_pump_speed%256);
1754
1755  i2c_putchar('\n');
1756
1757 }
1758
1759 //compare string with message in RX buffer
1760 unsigned char buf_strcmp(PGM_P stringp,const char *strbuf)
1761 { unsigned char  i,len;
1762   char tmp[20];
1763
1764     len=strlen_P(stringp); // get length of string to compare
1765     if(len>19) len=19;
1766     for(i=0;i<len;i++){tmp[i]= *strbuf++;
1767                        };
1768    tmp[i]=0x00;
1769
1770    i=strcasecmp_P(tmp,stringp);
1771    if(i) return(1);
1772    return (0);
1773 }
1774 /*
1775 void send_status(void)
1776 {
1777
1778  printf_P(PSTR("\r\nCURRENT STATE: "));
1779  if(srd_sm1.run_state==STOP) printf_P(PSTR("STOP \r\n"));
1780  if(srd_sm1.run_state==RUN) printf_P(PSTR("RUNNING \r\n"));
1781  if(srd_sm1.run_state==ACCEL) printf_P(PSTR("ACCELERATING \r\n"));
1782  if(srd_sm1.run_state==DECEL) printf_P(PSTR("DECELERATING \r\n"));
1783
1784 }
1785 */
1786 char* find_number(char *position)
1787 {
1788   while(!isdigit((int)position) && (*position!='+') && (*position!='-')){
1789                                       position++;
1790                                       if(position>=rxbuf+RXBUFSIZE) return(NULL);
1791                                       }
1792   return(position);
1793 }
1794
1795 char* find_next_number(char *position)
1796 {
1797   while(isdigit((int)position)){ position++;
1798                                 if(position>=rxbuf+RXBUFSIZE) return(NULL);
1799                                 }
1800   position=find_number(position);
1801   return(position);
1802 }
1803
1804
1805
1806
1807
1808 //********************** BIOREACTOR ROUTINES **************
1809
1810
1811 uint8_t br1_init(void)
1812 {
1813  control_solenoid(BR1_V1, OFF);
1814  control_solenoid(BR1_V2, OFF);
1815  control_solenoid(BR1_V3, OFF);
1816  control_solenoid(BR1_V4, OFF);
1817
1818  return(STEP_FINISHED);
1819 }
1820
1821
1822 uint8_t br1_change_medium(int16_t volume, uint16_t speed)
1823 {
1824  //first stop the pump
1825  if(progress[1]==0){
1826                 progress[1]=1;
1827                 stop_pump();
1828                 return(STEP_RUNNING);};
1829  //then switch all valves to correct position
1830  if(progress[1]==1){if(get_pump_status()!=OFF) return(STEP_RUNNING);
1831                 progress[1]=2;
```

```
1832                control_solenoid(BR1_V1, ON);
1833                delay_timer_br=20;
1834                return(STEP_RUNNING);};
1835  if(progress[1]==2){if(delay_timer_br) return(STEP_RUNNING);
1836                progress[1]=3;
1837                control_solenoid(BR1_V2, ON);
1838                delay_timer_br=20;
1839                return(STEP_RUNNING);};
1840  if(progress[1]==3){if(delay_timer_br) return(STEP_RUNNING);
1841                progress[1]=4;
1842                control_solenoid(BR1_V3, OFF);
1843                control_solenoid(BR1_V4, OFF);
1844                delay_timer_br=20;
1845                return(STEP_RUNNING);};
1846  //start the pump at medium change speed and volume
1847  if(progress[1]==4){if(delay_timer_br) return(STEP_RUNNING);
1848                    progress[1]=5;
1849                    start_pump(volume,speed);
1850                    return(STEP_RUNNING);};
1851  if(progress[1]==5){if(get_pump_status()==ON) return(STEP_RUNNING);
1852                progress[1]=6;
1853                control_solenoid(BR1_V1, OFF);
1854                control_solenoid(BR1_V2, OFF);
1855                delay_timer_br=20;
1856                return(STEP_RUNNING);};
1857
1858  if(progress[1]==6){if(delay_timer_br) return(STEP_RUNNING);
1859                progress[1]=7;
1860                control_solenoid(BR1_V3, ON);
1861                delay_timer_br=20;
1862                return(STEP_RUNNING);};
1863
1864  //start the pump at medium change speed to flush the medium from calibrated tubing
1865  if(progress[1]==7){if(delay_timer_br) return(STEP_RUNNING);
1866                    progress[1]=8;
1867                    start_pump(CALIBRATED_DEAD_VOLUME,speed);
1868                    return(STEP_RUNNING);};
1869  if(progress[1]==8){if(get_pump_status()==ON) return(STEP_RUNNING);
1870                progress[1]=9;
1871                control_solenoid(BR1_V3, OFF);
1872                delay_timer_br=20;
1873                return(STEP_RUNNING);};
1874  //END
1875  if(progress[1]==9){if(delay_timer_br) return(STEP_RUNNING);
1876                progress[1]=0;
1877                return(STEP_FINISHED);
1878                };
1879
1880  //this is never reached
1881  return(STEP_RUNNING);
1882 }
1883
1884 uint8_t br1_sample(int16_t volume, uint16_t speed)
1885 {
1886  //first stop the pump
1887  if(progress[2]==0){
1888                progress[2]=1;
1889                stop_pump();
1890                return(STEP_RUNNING);};
1891  //then switch all valves to correct position
1892  if(progress[2]==1){if(get_pump_status()!=OFF) return(STEP_RUNNING);
1893                progress[2]=2;
1894                control_solenoid(BR1_V1, ON);
1895                delay_timer_br=20;
1896                return(STEP_RUNNING);};
1897  if(progress[2]==2){if(delay_timer_br) return(STEP_RUNNING);
1898                progress[2]=3;
1899                control_solenoid(BR1_V2, ON);
1900                delay_timer_br=20;
1901                return(STEP_RUNNING);};
1902  if(progress[2]==3){if(delay_timer_br) return(STEP_RUNNING);
1903                progress[2]=4;
1904                control_solenoid(BR1_V4, ON);
1905                delay_timer_br=20;
1906                return(STEP_RUNNING);};
1907  if(progress[2]==4){if(delay_timer_br) return(STEP_RUNNING);
1908                progress[2]=5;
1909                control_solenoid(BR1_V3, OFF);
1910                delay_timer_br=20;
1911                return(STEP_RUNNING);};
1912  //then run the pump at sampling speed and volume
1913  if(progress[2]==5){if(delay_timer_br) return(STEP_RUNNING);
1914                progress[2]=6;
1915                    start_pump(volume,speed);
1916                    return(STEP_RUNNING);};
1917  //then switch off all valves
1918  if(progress[2]==6){if(get_pump_status()==ON) return(STEP_RUNNING);
1919                progress[2]=7;
1920                control_solenoid(BR1_V1, OFF);
1921                control_solenoid(BR1_V2, OFF);
1922                control_solenoid(BR1_V4, OFF);
1923                delay_timer_br=20;
1924                return(STEP_RUNNING);};
1925  //END
1926  if(progress[2]==7){if(delay_timer_br) return(STEP_RUNNING);
1927                progress[2]=0;
1928                return(STEP_FINISHED);
1929                };
1930
1931  //this is never reached
1932  return(STEP_RUNNING);
1933 }
```

```
1934
1935 uint8_t br1_mix_sample(int16_t volume, uint16_t speed)
1936 {
1937  //first stop the pump
1938  if(progress[3]==0){
1939                 progress[3]=1;
1940                 stop_pump();
1941                 return(STEP_RUNNING);};
1942  //then switch all valves to correct position
1943  if(progress[3]==1){if(get_pump_status()!=OFF) return(STEP_RUNNING);
1944                 progress[3]=2;
1945                 control_solenoid(BR1_V3, ON);
1946                 delay_timer_br=20;
1947                 return(STEP_RUNNING);};
1948  if(progress[3]==2){if(delay_timer_br) return(STEP_RUNNING);
1949                 progress[3]=3;
1950                 control_solenoid(BR1_V4, ON);
1951                 delay_timer_br=20;
1952                 return(STEP_RUNNING);};
1953  if(progress[3]==3){if(delay_timer_br) return(STEP_RUNNING);
1954                 progress[3]=4;
1955                 control_solenoid(BR1_V1, OFF);
1956                 control_solenoid(BR1_V2, OFF);
1957                 delay_timer_br=20;
1958                 return(STEP_RUNNING);};
1959  //take the time point and wait mix_time
1960  if(progress[3]==4){if(delay_timer_br) return(STEP_RUNNING);
1961                 progress[3]=5;
1962                 if(get_pump_status()==OFF)  start_pump(volume,speed);
1963                     return(STEP_RUNNING);};
1964  //then switch off all valves
1965  if(progress[3]==5){if(get_pump_status()==ON) return(STEP_RUNNING);
1966                 progress[3]=6;
1967                 control_solenoid(BR1_V3, OFF);
1968                 control_solenoid(BR1_V4, OFF);
1969                 delay_timer_br=20;
1970                 return(STEP_RUNNING);};
1971  //END
1972  if(progress[3]==6){if(delay_timer_br) return(STEP_RUNNING);
1973                 progress[3]=0;
1974                 return(STEP_FINISHED);
1975                 };
1976
1977  //this is never reached
1978  return(STEP_RUNNING);
1979 }
1980
1981 uint8_t br1_prepare_sample(void)
1982 {uint8_t result;
1983
1984  //first fill the calibrated tubing length with the old medium
1985  if(progress[4]==0){result=br1_change_medium(BR1_SAMP_VOL,BR1_SAMP_SPEED);
1986                 if(result==STEP_RUNNING) return(STEP_RUNNING);
1987                 progress[4]=1;
1988                 return(STEP_RUNNING);};
1989  //then flush that volume into the dilution container and mix with air
1990  if(progress[4]==1){result=br1_mix_sample(BR1_MIX_VOL,BR1_MIX_SPEED);
1991                 if(result==STEP_RUNNING) return(STEP_RUNNING);
1992                 progress[4]=0;
1993                 return(STEP_FINISHED);};
1994
1995  //this is never reached
1996  return(STEP_RUNNING);
1997 }
1998
1999 void bioreactor_sequencer(void)
2000 {uint8_t status;
2001
2002 if((!br1_make_sample) && (!br1_changing_medium)){
2003                   if(br1_perfusion){
2004                                   if(get_pump_status()==OFF)
2005                                       start_pump(BR1_PERFUS_VOL, BR1_PERFUS_SPEED);
2006                                  }
2007                       else{
2008                          //if(get_pump_status()==ON) stop_pump();
2009                          }
2010                 };
2011
2012  if(br1_make_sample){if(br1_make_sample==1){br1_make_sample++;
2013                   eeprom_write_byte((uint8_t *)EE_BR1_MAKESAMPLE,br1_make_sample);
2014                   printf_P(PSTR("\n\rpreparing sample"));}
2015                   status=br1_prepare_sample();
2016                   if(status==STEP_FINISHED) {
2017                       br1_make_sample=0;
2018                       eeprom_write_byte((uint8_t *)EE_BR1_MAKESAMPLE,br1_make_sample);
2019                       printf_P(PSTR("\n\rsample ready"));
2020                       }
2021                   }
2022
2023  if(br1_changing_medium && (!br1_make_sample)){
2024     if(br1_changing_medium==1){br1_changing_medium++;
2025                  printf_P(PSTR("\n\rstart of the medium change"));}
2026                  status=br1_change_medium(medium_change_vol,medium_change_speed);
2027                  if(status==STEP_FINISHED) {
2028                      br1_changing_medium=0;
2029                      printf_P(PSTR("\n\rmedium change has been finished"));
2030                      }
2031                  }
2032
2033 }
2034
2035
```

```
2036
2037
2038
2039 void parse_msg(void)
2040 {uint8_t cmderr=CMD_ERR_CMD;
2041  char *msgptr;
2042  int16_t cmdparameters[8];
2043  brtask_statistics mstats;
2044
2045  msgptr=rxbuf;
2046
2047
2048
2049  //switch on/off the solenoid valve
2050  if(!buf_strcmp(cmd_valve,msgptr)){cmderr=CMD_OK;
2051       msgptr+=strlen(cmd_valve);
2052       //read the 1st parameter - the valve number
2053       cmdparameters[0]=(uint16_t)strtol(msgptr, &msgptr, 0);
2054       if(cmdparameters[0]<1 || cmdparameters[0]>8) cmderr=CMD_ERR_PARAM;
2055       if(*msgptr != ' ') cmderr=CMD_ERR_PARAMCNT;
2056       //read the 2nd parameter - on(1) of off(0)
2057       cmdparameters[1]=(uint16_t)strtol(msgptr, &msgptr, 0);
2058       if(cmdparameters[1]!=0 && cmdparameters[1]!=1) cmderr=CMD_ERR_PARAM;
2059
2060       if(cmderr==CMD_OK) control_solenoid(cmdparameters[0], cmdparameters[1]);
2061     }
2062
2063  //start pump
2064  else if(!buf_strcmp(cmd_startpump,msgptr)){cmderr=CMD_OK;
2065        msgptr+=strlen(cmd_startpump);
2066
2067       //read the 1st numeric parameter - pumping volume
2068       cmdparameters[1]=(int16_t)strtol(msgptr, &msgptr, 0);
2069       if(!cmdparameters[1] || cmdparameters[1]<-20000 || cmdparameters[1]>20000)
2070              cmderr=CMD_ERR_PARAM;
2071       if(*msgptr != ' ') cmderr=CMD_ERR_PARAMCNT;
2072
2073       //read the 2nd parameter - pumping speed
2074       cmdparameters[2]=(int16_t)strtol(msgptr, &msgptr, 0);
2075       if(cmdparameters[2]<1 || cmdparameters[2]>5000) cmderr=CMD_ERR_PARAM;
2076         //printf("\n\rParameters: %d %d %d",cmdparameters[0],
2077         //cmdparameters[1], cmdparameters[2]);
2078       if(cmderr==CMD_OK){ start_pump(cmdparameters[1], cmdparameters[2]);
2079        //speed_cntr_Move(cmdparameters[1],acceleration,deceleration,cmdparameters[2]);
2080                          }
2081       }
2082  //stop pump
2083   else if(!buf_strcmp(cmd_stoppump,msgptr)){cmderr=CMD_OK;
2084                                             //msgptr+=strlen(cmd_startpump);
2085                                             stop_pump();
2086                                            }
2087  //get pump status
2088   else if(!buf_strcmp(cmd_pumpstatus,msgptr)){cmderr=CMD_OK;
2089
2090                                             get_pump_progress(&mstats);
2091                                            }
2092   else if(!buf_strcmp(cmd_stats,msgptr)){cmderr=CMD_OK;
2093
2094        get_medchange_stats(&mstats,medium_change_vol,medium_change_speed);
2095        //get_prepsamp_stats(&mstats);
2096        printf_P(PSTR("\n\rM.Ch. done: %d %%, "),mstats.percent_done);
2097        printf_P(PSTR("Time to end: %d sec\n\r"),mstats.time2end);
2098                                             }
2099
2100   else if(!buf_strcmp(cmd_callboot,msgptr)){cmderr=CMD_OK;
2101                                        printf_P(PSTR("\n\rStarting Bootloader\n\r"));
2102                                        eeprom_write_byte((uint8_t *)E2END,0xFF);
2103                                       // Delay_ms(100);
2104                                        (*((void(*)(void))(Reset2Boot)))();
2105                                       }
2106
2107   else if(!buf_strcmp(cmd_seti2caddr,msgptr)){cmderr=CMD_OK;
2108        msgptr+=strlen(cmd_seti2caddr);
2109
2110       //read the 1st numeric parameter - the I2C address, allowed range: 0x01 - 0x7F
2111       cmdparameters[0]=(int16_t)strtol(msgptr, &msgptr, 0);
2112       if(cmdparameters[0]<1 || cmdparameters[0]>127) cmderr=CMD_ERR_PARAM;
2113
2114       if(cmderr==CMD_OK){
2115              eeprom_write_byte((uint8_t *)EE_I2C_ADDR,(uint8_t)cmdparameters[0]);
2116              TWAR=(uint8_t)cmdparameters[0]<<1;
2117             }
2118       }
2119
2120 //BIOREACTOR COMMANDS
2121    //turn on/off the perfusion of the bioreactor
2122    else if(!buf_strcmp(cmdb_control,msgptr)){cmderr=CMD_OK;
2123          msgptr+=strlen(cmdb_control);
2124          //read the 1st parameter - the on off witch> 0=off, 1=on
2125          cmdparameters[0]=(uint16_t)strtol(msgptr, &msgptr, 0);
2126          if(cmdparameters[0]<0 || cmdparameters[0]>1) cmderr=CMD_ERR_PARAM;
2127
2128          if(cmderr==CMD_OK){
2129                      br1_perfusion=cmdparameters[0];
2130                      eeprom_write_byte((uint8_t *)EE_BR1_PERFUSION,br1_perfusion);
2131                      if(cmdparameters[0]==0) stop_pump();
2132                     }
2133        }
2134    //change the cultivation medium
2135    else if(!buf_strcmp(cmdb_changemed,msgptr)){cmderr=CMD_OK;
2136          msgptr+=strlen(cmdb_changemed);
2137          //read the 1st parameter - the medium volume to change
```

```
2138          cmdparameters[0]=(uint16_t)strtol(msgptr, &msgptr, 0);
2139          if(cmdparameters[0]<-10000 || cmdparameters[0]>10000) cmderr=CMD_ERR_PARAM;
2140          if(*msgptr != ' ') cmderr=CMD_ERR_PARAMCNT;
2141           //read the 2nd parameter - the pump speed
2142           cmdparameters[1]=(uint16_t)strtol(msgptr, &msgptr, 0);
2143           if(cmdparameters[1]<1 || cmdparameters[1]>500) cmderr=CMD_ERR_PARAM;
2144
2145          if(cmderr==CMD_OK){
2146                               medium_change_vol=cmdparameters[0];
2147                               medium_change_speed=cmdparameters[1];
2148                               br1_changing_medium=1;
2149                              }
2150          }
2151    //sample the cultivation medium
2152    else if(!buf_strcmp(cmdb_sample,msgptr)){cmderr=CMD_OK;
2153          msgptr+=strlen(cmdb_sample);
2154          //read the 1st parameter - the medium volume to change
2155          cmdparameters[0]=(uint16_t)strtol(msgptr, &msgptr, 0);
2156          if(cmdparameters[0]<0 || cmdparameters[0]>10000) cmderr=CMD_ERR_PARAM;
2157          if(*msgptr != ' ') cmderr=CMD_ERR_PARAMCNT;
2158           //read the 2nd parameter - the pump speed
2159           cmdparameters[1]=(uint16_t)strtol(msgptr, &msgptr, 0);
2160           if(cmdparameters[1]<1 || cmdparameters[1]>500) cmderr=CMD_ERR_PARAM;
2161
2162          if(cmderr==CMD_OK){
2163              while(br1_sample(cmdparameters[0],cmdparameters[1]) != STEP_FINISHED);
2164             }
2165        }
2166    //mix the sample during dilution
2167    else if(!buf_strcmp(cmdb_mix,msgptr)){cmderr=CMD_OK;
2168          msgptr+=strlen(cmdb_mix);
2169          //read the 1st parameter - the air volume to pump
2170          cmdparameters[0]=(uint16_t)strtol(msgptr, &msgptr, 0);
2171          if(cmdparameters[0]<1 || cmdparameters[0]>30000) cmderr=CMD_ERR_PARAM;
2172          if(*msgptr != ' ') cmderr=CMD_ERR_PARAMCNT;
2173           //read the 2nd parameter - the pump speed
2174           cmdparameters[1]=(uint16_t)strtol(msgptr, &msgptr, 0);
2175           if(cmdparameters[1]<1 || cmdparameters[1]>500) cmderr=CMD_ERR_PARAM;
2176
2177          if(cmderr==CMD_OK){
2178              while(br1_mix_sample(cmdparameters[0],cmdparameters[1]) != STEP_FINISHED);
2179             }
2180        }
2181    //mix the sample during dilution
2182    else if(!buf_strcmp(cmdb_prepsample,msgptr)){cmderr=CMD_OK;
2183
2184          if(cmderr==CMD_OK){
2185                      br1_make_sample=1;
2186                      eeprom_write_byte((uint8_t *)EE_BR1_MAKESAMPLE,br1_make_sample);
2187                     }
2188        }
2189    //change washing manifold pump speed
2190    else if(!buf_strcmp(cmdb_setspeed,msgptr)){cmderr=CMD_OK;
2191         msgptr+=strlen(cmdb_setspeed);
2192         //read the 1st parameter - the bioreactor perfusion speed
2193         cmdparameters[0]=(uint16_t)strtol(msgptr, &msgptr, 0);
2194         if(cmdparameters[0]<1 || cmdparameters[0]>500) cmderr=CMD_ERR_PARAM;
2195
2196         if(cmderr==CMD_OK){
2197                             br1_pump_speed=cmdparameters[0];
2198                             eeprom_write_word((uint16_t *)EE_BR_SPEED,br1_pump_speed);
2199                             }
2200
2201         }
2202    //mix the sample during dilution
2203    else if(!buf_strcmp(cmdb_stop,msgptr)){cmderr=CMD_OK;
2204
2205          if(cmderr==CMD_OK){
2206                  br1_changing_medium=0;
2207                  br1_make_sample=0;
2208                  eeprom_write_byte((uint8_t *)EE_BR1_MAKESAMPLE,br1_make_sample);
2209                  progress[0]=0;
2210                  progress[1]=0;
2211                  progress[2]=0;
2212                  progress[3]=0;
2213                  progress[4]=0;
2214                  br1_init();
2215                  stop_pump();
2216                 }
2217    }
2218
2219
2220  if(cmderr==CMD_OK) printf_P(resp_ok);
2221  else if(cmderr==CMD_ERR_CMD) printf_P(resp_err_cmd);
2222  else if(cmderr==CMD_ERR_PARAM) printf_P(resp_err_param);
2223  else if(cmderr==CMD_ERR_PARAMCNT) printf_P(resp_err_paramcnt);
2224
2225  new_msg=0;
2226  rx_ptr=0;
2227
2228 }
2229
2230
2231
2232
2233 void load_backup(void)
2234 {
2235  br1_make_sample=eeprom_read_byte((uint8_t *)EE_BR1_MAKESAMPLE);
2236  br1_perfusion=eeprom_read_byte((uint8_t *)EE_BR1_PERFUSION);
2237  br1_pump_speed=eeprom_read_word((uint16_t *)EE_BR_SPEED);
2238
2239  if(br1_make_sample==0xFF){br1_make_sample=0;
```

```
2240                          eeprom_write_byte((uint8_t *)EE_BR1_MAKESAMPLE,
2241                          br1_make_sample);
2242                          };
2243  if(br1_perfusion==0xFF){br1_perfusion=0;
2244                         eeprom_write_byte((uint8_t *)EE_BR1_PERFUSION,
2245                         br1_perfusion);
2246                         };
2247  if(br1_pump_speed==0xFFFF){br1_pump_speed=25;
2248                         eeprom_write_word((uint16_t *)EE_BR_SPEED,
2249                         br1_pump_speed);
2250                         };
2251 }
2252
2253
2254 int main( void )
2255 {//unsigned char i;
2256  //int tmp;
2257
2258
2259  steps = 200;
2260  acceleration = 2000;
2261  deceleration = 2000;
2262  speed=10;
2263
2264  baudrate=baud115200;
2265
2266  GICR=0x01;        //move interrupt vectors to appl section
2267  GICR=0x00;
2268
2269  ioinit();
2270  uartinit(baudrate);
2271  fdevopen(uartsend,0);
2272  printf_P(PSTR("\n\rBIOREACTOR DRIVER START\n\r"));
2273  T0_start();
2274  adc_start();
2275  spi_init();
2276  i2c_init(I2C_100k);
2277
2278  sei();
2279  tim_ena=1;
2280  Motor_Init();
2281
2282  LEDR_ON;
2283  delay(250);
2284  LEDR_OFF;
2285  LEDG_ON;
2286  delay(250);
2287  LEDG_OFF;
2288
2289
2290
2291   SOL_SET_5V;
2292   br1_init();
2293   load_backup();
2294  /*
2295   i2c_putchar('1');
2296   i2c_putchar('2');
2297   i2c_putchar('3');
2298   i2c_putchar('\n');
2299   */
2300   while(1) {
2301
2302           if(new_msg) parse_msg();
2303           if(i2c_newmsg) parse_i2c_command();
2304
2305           bioreactor_sequencer();
2306
2307           if(i2c_buserror){TWCR=0;
2308                           //i2c_buserror=0;
2309                           i2c_init(I2C_100k);
2310                           };
2311
2312           }
2313
2314  return 0;
2315 }
```

Appendix 11 – Listing of the source code for control unit for smart drivers

```
/*********************************************************************\
*               Controller for BR drivers -  HW ver1.0                *
*--------------------------------------------------------------------*
* Description : Controller for up to 8 BR drivers,                    *
*               controlled over I2C bus                               *
*--------------------------------------------------------------------*
* Author      : Martin Baca                                           *
* Developed   : 07.06.2016               Last Update : 29.12.2017     *
* Version     : 1.2                                                   *
*--------------------------------------------------------------------*
* Compiler    : arduino                                               *
* Source file : BR_Controller_RTC_Diag.ino                            *
*--------------------------------------------------------------------*
* Target system : Arduino Mega 2560 board, Rev.3                      *
*                 ITDB50 - 5" TFT Display 800x480,                    *
*                 DS3231 RTC module                                   *
* Target CPU    : ATmega2560 @16 MHz, UART:  115200,N,8,1             *
* Emulator HW   :                                                     *
\*********************************************************************/


#include <UTFT.h>
#include <Wire.h>
#include <UTouch.h>
#include <UTFT_Buttons_ITDB.h>
#include <DS3231.h>
#include <EEPROM.h>
#include "BR_Lib.h"

//EEPROM MAP
#define EE_I2CADR1  0x01
#define EE_I2CADR2  0x02
#define EE_I2CADR3  0x03
#define EE_I2CADR4  0x04
#define EE_I2CADR5  0x05
#define EE_I2CADR6  0x06
#define EE_I2CADR7  0x07
#define EE_I2CADR8  0x08

#define EE_MEDCHG_VOL1  0x10
#define EE_MEDCHG_VOL2  0x12
#define EE_MEDCHG_VOL3  0x14
#define EE_MEDCHG_VOL4  0x16
#define EE_MEDCHG_VOL5  0x18
#define EE_MEDCHG_VOL6  0x1A
#define EE_MEDCHG_VOL7  0x1C
#define EE_MEDCHG_VOL8  0x1E

#define EE_MEDCHG_SPEED1  0x20
#define EE_MEDCHG_SPEED2  0x22
#define EE_MEDCHG_SPEED3  0x24
#define EE_MEDCHG_SPEED4  0x26
#define EE_MEDCHG_SPEED5  0x28
#define EE_MEDCHG_SPEED6  0x2A
#define EE_MEDCHG_SPEED7  0x2C
#define EE_MEDCHG_SPEED8  0x2E

#define EE_TINT_HRS1  0x30
#define EE_TINT_HRS2  0x31
#define EE_TINT_HRS3  0x32
#define EE_TINT_HRS4  0x33
#define EE_TINT_HRS5  0x34
#define EE_TINT_HRS6  0x35
#define EE_TINT_HRS7  0x36
#define EE_TINT_HRS8  0x37

#define EE_TINT_MIN1  0x40
#define EE_TINT_MIN2  0x41
#define EE_TINT_MIN3  0x42
#define EE_TINT_MIN4  0x43
#define EE_TINT_MIN5  0x44
#define EE_TINT_MIN6  0x45
#define EE_TINT_MIN7  0x46
#define EE_TINT_MIN8  0x47

#define EE_TREP_TOT1  0x50
#define EE_TREP_TOT2  0x51
#define EE_TREP_TOT3  0x52
#define EE_TREP_TOT4  0x53
#define EE_TREP_TOT5  0x54
#define EE_TREP_TOT6  0x55
#define EE_TREP_TOT7  0x56
#define EE_TREP_TOT8  0x57

#define EE_TREP_LEFT1  0x60
#define EE_TREP_LEFT2  0x61
#define EE_TREP_LEFT3  0x62
#define EE_TREP_LEFT4  0x63
#define EE_TREP_LEFT5  0x64
#define EE_TREP_LEFT6  0x65
#define EE_TREP_LEFT7  0x66
#define EE_TREP_LEFT8  0x67

```

```
94   #define EE_TENA1  0x70
95   #define EE_TENA2  0x71
96   #define EE_TENA3  0x72
97   #define EE_TENA4  0x73
98   #define EE_TENA5  0x74
99   #define EE_TENA6  0x75
100  #define EE_TENA7  0x76
101  #define EE_TENA8  0x77
102
103  #define EE_TFIRST_START1  0x80
104  #define EE_TFIRST_START2  0x88
105  #define EE_TFIRST_START3  0x90
106  #define EE_TFIRST_START4  0x98
107  #define EE_TFIRST_START5  0xA0
108  #define EE_TFIRST_START6  0xA8
109  #define EE_TFIRST_START7  0xB0
110  #define EE_TFIRST_START8  0xB8
111
112  #define EE_TNEXT_START1  0xC0
113  #define EE_TNEXT_START2  0xC8
114  #define EE_TNEXT_START3  0xD0
115  #define EE_TNEXT_START4  0xD8
116  #define EE_TNEXT_START5  0xE0
117  #define EE_TNEXT_START6  0xE8
118  #define EE_TNEXT_START7  0xF0
119  #define EE_TNEXT_START8  0xF8
120
121
122
123  void timer_init(void);
124  void update_schedule_status(uint8_t channel);
125  void recalculate_schedule(uint8_t channel);
126  void print_global_diag(void);
127
128  extern uint8_t SmallFont[];
129  extern uint8_t BigFont[];
130  extern uint8_t Dingbats1_XL[];
131
132  // Remember to change the model parameter to suit your display module!
133  UTFT          myGLCD(ITDB50,38,39,40,41);
134  UTouch        myTouch(6,5,4,3,2);
135
136  // Finally we set up UTFT_Buttons :)
137  UTFT_Buttons  myButtons(&myGLCD, &myTouch);
138
139  // Init the DS3231 using the hardware interface
140  DS3231  rtc(SDA, SCL);
141
142  // Init a Time-data structure
143  Time  t;
144
145  int inByte = 0;         // incoming serial byte
146  byte x = 0;
147  unsigned char i2c_txbuf[16];
148
149  int but0, but1, but2, but3, but4, but5, but6;
150  int but7, but8, but9, butDEL, butOK, butBACK;
151  int but_perstart, but_perstop, but_medstart;
152  int but_sampstart, but_reset, but_timer, but_sched;
153  int but_v1, but_v2, but_v3, but_v4, but_pump;
154
155  uint8_t i2c_adr_tab[8];
156  uint16_t medchg_vol_tab[8];
157  uint16_t medchg_speed_tab[8];
158
159  char* dw_tab[]={"Mon","Tue","Wed","Thu","Fri","Sat","Sun"};
160
161  #define MAX_REPEATES  10
162  #define MAX_INTERVAL_HOURS  99
163  #define INTERCHANNEL_DELAY  3
164
165  uint8_t timer_intervals_hour[8];
166  uint8_t timer_intervals_minutes[8];
167  uint8_t timer_repeates_total[8];
168  uint8_t timer_repeates_left[8];
169  uint8_t timer_enables[8];
170  Time timer_next_start[8];
171  Time timer_first_start[8];
172
173  char diag_msg[128];
174  uint8_t br_connections[8]={0,0,0,0,0,0,0,0};
175
176  void save_start_time_eeprom(uint8_t channel, Time* src)
177  {
178   EEPROM.write(EE_TFIRST_START1+8*(channel-1)+0, src->sec);
179   EEPROM.write(EE_TFIRST_START1+8*(channel-1)+1, src->min);
180   EEPROM.write(EE_TFIRST_START1+8*(channel-1)+2, src->hour);
181   EEPROM.write(EE_TFIRST_START1+8*(channel-1)+3, src->date);
182   EEPROM.write(EE_TFIRST_START1+8*(channel-1)+4, src->mon);
183   EEPROM.write(EE_TFIRST_START1+8*(channel-1)+5, (src->year)/256);
184   EEPROM.write(EE_TFIRST_START1+8*(channel-1)+6, (src->year)%256);
185   EEPROM.write(EE_TFIRST_START1+8*(channel-1)+7, src->dow );
186  }
187
188
189  void load_start_time_eeprom(uint8_t channel, Time* dest)
190  {uint8_t error = 0;
191
192   dest->sec= EEPROM.read(EE_TFIRST_START1+8*(channel-1)+0);
193   if(dest->sec > 59)  {dest->sec=0; error++;}
194   dest->min= EEPROM.read(EE_TFIRST_START1+8*(channel-1)+1);
195   if(dest->min > 59) {dest->min = 0; error++;}
```

```
 dest->hour= EEPROM.read(EE_TFIRST_START1+8*(channel-1)+2);
 if(dest->hour > 23)  {dest->hour = 12; error++;}
 dest->date= EEPROM.read(EE_TFIRST_START1+8*(channel-1)+3);
 if(dest->date > 31)  {dest->date = 1; error++;}
 dest->mon= EEPROM.read(EE_TFIRST_START1+8*(channel-1)+4);
 if(dest->mon > 12)  {dest->mon = 1; error++;}
 dest->year= 256*EEPROM.read(EE_TFIRST_START1+8*(channel-1)+5);
 dest->year += EEPROM.read(EE_TFIRST_START1+8*(channel-1)+6);
 if(dest->year > 9999)  {dest->year = 2000; error++;}
 dest->dow = EEPROM.read(EE_TFIRST_START1+8*(channel-1)+7);
 if(dest->dow > 7)  {dest->dow = 1; error++;}

 if(error) save_start_time_eeprom(channel, dest);
}



void timer_init(void)
{char i;

   for(i=0;i<8;i++){
    timer_intervals_hour[i]=EEPROM.read(EE_TINT_HRS1+i);
    if( timer_intervals_hour[i]>MAX_INTERVAL_HOURS)  timer_intervals_hour[i]=MAX_INTERVAL_HOURS;
    timer_intervals_minutes[i]=EEPROM.read(EE_TINT_MIN1+i);
    if(timer_intervals_minutes[i]>59) timer_intervals_minutes[i]=59;
    timer_repeates_total[i]=EEPROM.read(EE_TREP_TOT1+i);
    if(timer_repeates_total[i]>MAX_REPEATES) timer_repeates_total[i]=MAX_REPEATES;
    timer_repeates_left[i]=EEPROM.read(EE_TREP_LEFT1+i);
    if(timer_repeates_left[i]>MAX_REPEATES) timer_repeates_left[i]=MAX_REPEATES;
    timer_enables[i]=EEPROM.read(EE_TENA1+i);
    if(timer_enables[i]>1) timer_enables[i]=1;
    load_start_time_eeprom(i+1, &timer_first_start[i]);
    calc_next_start(i+1);
   }
}


void setup()
{char i;
 uint8_t tmp;
 uint16_t tmp16;

  Wire.begin(); // join i2c bus (address optional for master)
  Serial.begin(115200);

  // Setup the LCD
  myGLCD.InitLCD();
  myGLCD.setFont(SmallFont);

  // Clear the screen and draw the frame
  myGLCD.clrScr();

  myTouch.InitTouch(LANDSCAPE);
  myTouch.setPrecision(PREC_MEDIUM);
  myTouch.calibrateRead();    //used to properly initialize XPT2046 - and enable the IRQ

  myButtons.setTextFont(BigFont);
  myButtons.setSymbolFont(Dingbats1_XL);

  // Initialize the rtc object
  rtc.begin();

  timer_init();

  for(i=0;i<8;i++){
                  tmp=EEPROM.read(EE_I2CADR1+i);
                  if(!tmp || tmp>127){tmp=127; EEPROM.write(EE_I2CADR1+i,tmp);}
                  i2c_adr_tab[i]=tmp;

                  tmp16=EEPROM.read(EE_MEDCHG_VOL1+2*i);
                  tmp16=tmp16<<8;
                  tmp16+=EEPROM.read(EE_MEDCHG_VOL1+2*i+1);
                  if(!tmp16 || tmp16>9999){tmp16=9999;
                                           EEPROM.write(EE_MEDCHG_VOL1+2*i,tmp16/256);
                                           EEPROM.write(EE_MEDCHG_VOL1+2*i+1,tmp16&0xFF);
                                           };
                  medchg_vol_tab[i]=tmp16;

                  tmp16=EEPROM.read(EE_MEDCHG_SPEED1+2*i);
                  tmp16=tmp16<<8;
                  tmp16+=EEPROM.read(EE_MEDCHG_SPEED1+2*i+1);
                  if(!tmp16 || tmp16>500){tmp16=50;
                                           EEPROM.write(EE_MEDCHG_SPEED1+2*i,tmp16/256);
                                           EEPROM.write(EE_MEDCHG_SPEED1+2*i+1,tmp16&0xFF);
                                           };
                  medchg_speed_tab[i]=tmp16;
                  };

  diag_out("System Power ON.");
  for(i=1;i<9;i++) draw_status(i,0);
  print_global_diag();
}

uint8_t is_leap_year(uint16_t year)
{
 return ((year & 3) == 0) && ((year % 400 == 0) || (year % 100 != 0));
}

//returns 1 if thistime os on future
// returns 0 otherwise
int8_t is_time_future(uint8_t channel, Time* thistime)
{Time curtime;
```

```
298
299    thistime->sec = ((channel-1)*INTERCHANNEL_DELAY)%60;
300
301    curtime = rtc.getTime();
302    if(thistime->year > curtime.year) return(1);
303    if(thistime->year < curtime.year) return(0);
304
305    if(thistime->mon > curtime.mon) return(1);
306    if(thistime->mon < curtime.mon) return(0);
307
308    if(thistime->date > curtime.date) return(1);
309    if(thistime->date < curtime.date) return(0);
310
311    if(thistime->hour > curtime.hour) return(1);
312    if(thistime->hour < curtime.hour) return(0);
313
314    if(thistime->min > curtime.min) return(1);
315    if(thistime->min < curtime.min) return(0);
316
317    if(thistime->sec > curtime.sec) return(1);
318    if(thistime->sec < curtime.sec) return(0);
319
320    return(0);
321  }
322
323  void calc_incr_start(uint8_t channel, Time* begining, Time* result)
324  {uint16_t tmp_hours;
325   uint16_t tmp_minutes;
326   uint8_t day_limit;
327   Time tmptime;
328
329   tmp_hours = timer_intervals_hour[channel-1];
330   tmp_minutes = timer_intervals_minutes[channel-1];
331
332   tmptime.sec=0;
333   tmptime.min=begining->min;
334   tmptime.hour=begining->hour;
335   tmptime.date=begining->date;
336   tmptime.mon=begining->mon;
337   tmptime.year=begining->year;
338
339   tmptime.min+=tmp_minutes;
340   if(tmptime.min>59){tmptime.min-=60; tmptime.hour++;};
341   tmptime.hour+=tmp_hours % 24;
342   if(tmptime.hour>23){tmptime.hour-=24; tmptime.date++;};
343   tmptime.date+=(tmp_hours / 24);
344
345   day_limit=31;
346   if(tmptime.mon==1 || tmptime.mon==3 || tmptime.mon==5 || tmptime.mon==7 ||
347      tmptime.mon==8 || tmptime.mon==10 || tmptime.mon==12) day_limit--;
348   if(tmptime.mon==2){ day_limit=28;
349                       if(is_leap_year(tmptime.year)) day_limit++;
350                     };
351   if(tmptime.date>day_limit){tmptime.date-=day_limit; tmptime.mon++;};
352   if(tmptime.mon>12){tmptime.mon=1; tmptime.year++;};
353
354   result->sec=tmptime.sec;
355   result->min=tmptime.min;
356   result->hour=tmptime.hour;
357   result->date=tmptime.date;
358   result->mon=tmptime.mon;
359   result->year=tmptime.year;
360  }
361
362  uint8_t calc_next_start(uint8_t channel)
363  {//uint16_t tmp_hours;
364   //uint16_t tmp_minutes;
365   uint8_t day_limit,i;
366   Time tmptime;
367
368
369   //tmp_hours = (uint16_t)timer_repeates_left[channel-1] * timer_intervals_hour[channel-1];
370   //tmp_minutes = (uint16_t)timer_repeates_left[channel-1] * timer_intervals_minutes[channel-1];
371
372   tmptime.sec=0;
373   tmptime.min=timer_first_start[channel-1].min;
374   tmptime.hour=timer_first_start[channel-1].hour;
375   tmptime.date=timer_first_start[channel-1].date;
376   tmptime.mon=timer_first_start[channel-1].mon;
377   tmptime.year=timer_first_start[channel-1].year;
378
379   i=timer_repeates_total[channel-1];
380   while(i){
381            if(is_time_future(channel, &tmptime)) break;
382            calc_incr_start(channel, &tmptime, &tmptime);
383            i--;
384    }
385
386   timer_repeates_left[channel-1]=i;
387   timer_next_start[channel-1].sec=tmptime.sec;
388   timer_next_start[channel-1].min=tmptime.min;
389   timer_next_start[channel-1].hour=tmptime.hour;
390   timer_next_start[channel-1].date=tmptime.date;
391   timer_next_start[channel-1].mon=tmptime.mon;
392   timer_next_start[channel-1].year=tmptime.year;
393   return(1);
394  }
395
396
397  void br_reset(uint8_t channel)
398  {
399    sprintf(diag_msg,"Reseting BR unit %d.",channel);
```

```
  diag_out(diag_msg);

  i2c_txbuf[0]=2;
  i2c_txbuf[1]=0x32;
  i2c_txbuf[2]='\n';
  send_i2c_msg(i2c_adr_tab[channel-1],i2c_txbuf);
  }


void control_perfusion(uint8_t channel, uint8_t new_state)
{ if (new_state) sprintf(diag_msg,"Perfusion START for BR unit %d.",channel);
     else
        sprintf(diag_msg,"Perfusion STOP for BR unit %d.",channel);
  diag_out(diag_msg);

  i2c_txbuf[0]=3;
  i2c_txbuf[1]=0x31;
  i2c_txbuf[2]=0x00;
  if(new_state) i2c_txbuf[2]++;
  i2c_txbuf[3]='\n';
  send_i2c_msg(i2c_adr_tab[channel-1],i2c_txbuf);
}

void control_valve(uint8_t channel,uint8_t valve, uint8_t new_state)
{ if (new_state) sprintf(diag_msg,"Switch Valve%d ON on BR unit %d.",valve,channel);
     else
        sprintf(diag_msg,"Switch Valve%d OFF on BR unit %d.",valve,channel);
  diag_out(diag_msg);

  i2c_txbuf[0]=4;
  i2c_txbuf[1]=0x38;
  i2c_txbuf[2]=valve;
  i2c_txbuf[3]=0;
  if(new_state) i2c_txbuf[3]++;
  i2c_txbuf[4]='\n';
  send_i2c_msg(i2c_adr_tab[channel-1],i2c_txbuf);
}

void set_perfusion_speed(uint8_t channel,uint16_t pspeed)
{ sprintf(diag_msg,"Setting prerfusion speed to %d ul/min for BR unit %d.",pspeed,channel);
  diag_out(diag_msg);

  i2c_txbuf[0]=4;
  i2c_txbuf[1]=0x33;
  i2c_txbuf[2]=(uint8_t)(pspeed/256);
  i2c_txbuf[3]=(uint8_t)(pspeed&0xFF);
  i2c_txbuf[4]='\n';
  send_i2c_msg(i2c_adr_tab[channel-1],i2c_txbuf);
}

void start_pump(uint8_t channel, int16_t volume, int16_t pspeed)
{ sprintf(diag_msg,"Starting pump of BR unit %d. Volume: %d, Speed: %d",channel, volume, pspeed);
  diag_out(diag_msg)

  i2c_txbuf[0]=6;
  i2c_txbuf[1]=0x39;
  i2c_txbuf[2]=(uint8_t)(volume/256);
  i2c_txbuf[3]=(uint8_t)(volume&0xFF);
  i2c_txbuf[4]=(uint8_t)(pspeed/256);
  i2c_txbuf[5]=(uint8_t)(pspeed&0xFF);
  i2c_txbuf[6]='\n';
  send_i2c_msg(i2c_adr_tab[channel-1],i2c_txbuf);
}

void medium_change(uint8_t channel, int16_t volume, int16_t pspeed)
{ sprintf(diag_msg,"Starting medium change on BR unit %d. Volume: %d, Speed: %d",channel, volume, pspeed);
  diag_out(diag_msg);

  i2c_txbuf[0]=6;
  i2c_txbuf[1]=0x35;
  i2c_txbuf[2]=(uint8_t)(volume/256);
  i2c_txbuf[3]=(uint8_t)(volume&0xFF);
  i2c_txbuf[4]=(uint8_t)(pspeed/256);
  i2c_txbuf[5]=(uint8_t)(pspeed&0xFF);
  i2c_txbuf[6]='\n';
  send_i2c_msg(i2c_adr_tab[channel-1],i2c_txbuf);
}


uint8_t get_selected_ch(void)
{uint8_t   result;
 int   touch_x;
 int   touch_y;

  myTouch.read();
  touch_x = myTouch.getX();
  touch_y = myTouch.getY();

  result=touch_y/STAT_SIZEY+1;
  if(touch_x >= STAT_SIZEX) result+=4;

  return(result);
}

#define BUTSIZEX 100
#define BUTSIZEY 50

void draw_keyboard(void)
{
  but1 = myButtons.addButton(5+0*(BUTSIZEX+10), 479-15-2*BUTSIZEY, BUTSIZEX,  BUTSIZEY, "1");
  but2 = myButtons.addButton(5+1*(BUTSIZEX+10), 479-15-2*BUTSIZEY, BUTSIZEX,  BUTSIZEY, "2");
  but3 = myButtons.addButton(5+2*(BUTSIZEX+10), 479-15-2*BUTSIZEY, BUTSIZEX,  BUTSIZEY, "3");
```

```
  but4 = myButtons.addButton(5+3*(BUTSIZEX+10), 479-15-2*BUTSIZEY, BUTSIZEX,  BUTSIZEY, "4");
  but5 = myButtons.addButton(5+4*(BUTSIZEX+10), 479-15-2*BUTSIZEY, BUTSIZEX,  BUTSIZEY, "5");
  but6 = myButtons.addButton(5+0*(BUTSIZEX+10), 479-5-BUTSIZEY, BUTSIZEX,  BUTSIZEY, "6");
  but7 = myButtons.addButton(5+1*(BUTSIZEX+10), 479-5-BUTSIZEY, BUTSIZEX,  BUTSIZEY, "7");
  but8 = myButtons.addButton(5+2*(BUTSIZEX+10), 479-5-BUTSIZEY, BUTSIZEX,  BUTSIZEY, "8");
  but9 = myButtons.addButton(5+3*(BUTSIZEX+10), 479-5-BUTSIZEY, BUTSIZEX,  BUTSIZEY, "9");
  but0 = myButtons.addButton(5+4*(BUTSIZEX+10), 479-5-BUTSIZEY, BUTSIZEX,  BUTSIZEY, "0");

  butDEL = myButtons.addButton(5+5*(BUTSIZEX+10), 479-5-BUTSIZEY, BUTSIZEX,  BUTSIZEY, "DEL");
  butOK = myButtons.addButton(5+6*(BUTSIZEX+10), 479-5-BUTSIZEY, BUTSIZEX,  BUTSIZEY, "OK");
  butBACK = myButtons.addButton(5+5*(BUTSIZEX+10), 479-15-2*BUTSIZEY,  BUTSIZEX*2+10,  BUTSIZEY, "BACK");

  myButtons.drawButtons();
}

int16_t read_keyboard(int16_t xpos, int16_t ypos, uint8_t len)
{char num[8];
 uint8_t return_home,numptr,i;
 int  pressed_button, result;

 if(len>5) len=5;
 if(!len) return(0);

 myGLCD.setFont(BigFont);
 myGLCD.setBackColor(VGA_WHITE);
 myGLCD.setColor(VGA_MAROON);

 return_home=0;
 numptr=0;
 while(!return_home){
          if(myTouch.dataAvailable()==true){
                    pressed_button = myButtons.checkButtons();
                    if (pressed_button==but0) {num[numptr++]='0';}
                    else if(pressed_button==but1){num[numptr++]='1';}
                    else if(pressed_button==but2){num[numptr++]='2';}
                    else if(pressed_button==but3){num[numptr++]='3';}
                    else if(pressed_button==but4){num[numptr++]='4';}
                    else if(pressed_button==but5){num[numptr++]='5';}
                    else if(pressed_button==but6){num[numptr++]='6';}
                    else if(pressed_button==but7){num[numptr++]='7';}
                    else if(pressed_button==but8){num[numptr++]='8';}
                    else if(pressed_button==but9){num[numptr++]='9';}
                    else if(pressed_button==butDEL){
                                                    if(numptr) numptr--;
                                                    }
                    else if(pressed_button==butOK){
                                                   return_home=1;
                                                   }
                    if(numptr>len) numptr=len;

                    for(i=numptr;i<len;i++) num[i]=' ';
                    num[len]=0;
                    myGLCD.print(num,xpos,ypos);
                    }
              }

 sscanf(num,"%d",&result);
 while (myTouch.dataAvailable() == true);
 return(result);
}

void check_num_fields(uint8_t channel)
{int result;
 char tmp[8];

  myTouch.read();
  int   touch_x = myTouch.getX();
  int   touch_y = myTouch.getY();
  if(touch_x>190 && touch_x<260 && touch_y>40 && touch_y<80){ //Perfusion speed
                            myGLCD.setColor(VGA_WHITE);
                            myGLCD.fillRect(200, 50,250,69);
                            result=read_keyboard(202, 52,3);
                            if(result<1) result=1;
                            if(result>500) result=500;
                            myGLCD.setColor(VGA_GRAY);
                            myGLCD.fillRect(200, 50,250,69);
                            myGLCD.setBackColor(VGA_GRAY);
                            myGLCD.setColor(VGA_RED);
                            sprintf(tmp,"%d",result);
                            myGLCD.print(tmp,202,52);
                            set_perfusion_speed(channel,result);
                            }
  if(touch_x>0 && touch_x<83 && touch_y>210 && touch_y<260){  //Medium Change Volume
                            myGLCD.setColor(VGA_WHITE);
                            myGLCD.fillRect(5, 230,5+68,249);
                            result=read_keyboard(7, 232,4);
                            if(result<1) result=1;
                            if(result>9999) result=9999;
                            myGLCD.setColor(VGA_GRAY);
                            myGLCD.fillRect(5, 230,5+68,249);
                            myGLCD.setBackColor(VGA_GRAY);
                            myGLCD.setColor(VGA_RED);
                            sprintf(tmp,"%d",result);
                            myGLCD.print(tmp,7, 232);
                            medchg_vol_tab[channel-1]=result;
                            EEPROM.write(EE_MEDCHG_VOL1+2*(channel-1),result/256);
                            EEPROM.write(EE_MEDCHG_VOL1+2*(channel-1)+1,result&0xFF);
                            }
  if(touch_x>190 && touch_x<262 && touch_y>210 && touch_y<260){  //Medium Change Speeed
                            myGLCD.setColor(VGA_WHITE);
                            myGLCD.fillRect(200, 230,200+52,249);
                            result=read_keyboard(202, 232,3);
```

```
                              if(result<1) result=1;
                              if(result>500) result=500;
                              myGLCD.setColor(VGA_GRAY);
                              myGLCD.fillRect(200, 230,200+52,249);
                              myGLCD.setBackColor(VGA_GRAY);
                              myGLCD.setColor(VGA_RED);
                              sprintf(tmp,"%d",result);
                              myGLCD.print(tmp,202, 232);
                              medchg_speed_tab[channel-1]=result;
                              EEPROM.write(EE_MEDCHG_SPEED1+2*(channel-1),result/256);
                              EEPROM.write(EE_MEDCHG_SPEED1+2*(channel-1)+1,result&0xFF);
                              }
    if(touch_x>539 && touch_x<610 && touch_y>200 && touch_y<240){  //I2C Address
                              myGLCD.setColor(VGA_WHITE);
                              myGLCD.fillRect(405+144, 210,405+144+52,229);
                              result=read_keyboard(551,212,3);
                              if(result<1) result=1;
                              if(result>127) result=127;
                              myGLCD.setColor(VGA_GRAY);
                              myGLCD.fillRect(405+144, 210,405+144+52,229);
                              myGLCD.setBackColor(VGA_GRAY);
                              myGLCD.setColor(VGA_RED);
                              sprintf(tmp,"%d",result);
                              myGLCD.print(tmp,551, 212);
                              i2c_adr_tab[channel-1]=result;
                              EEPROM.write(EE_I2CADR1+channel-1,result);
                              }

}

uint8_t wd(uint16_t year, uint8_t month, uint8_t day)
{ uint32_t JND;
  uint16_t a,m,y;

  a = (14-month)/12;
  m = month + (12*a) - 3;
  y = year + 4800 - a;

  JND = day;
  JND += (((153 * m) + 2) / 5) ;
  JND += (365ul * y);
  JND += (y / 4);
  JND -= (y / 100);
  JND += (y / 400);
  JND -= 32045;

  return((JND % 7)+1);
}

char *get_name_weekday(uint8_t day)
{
 if (day<1) day=1;
 if (day>7) day=7;
 day--;
 return(dw_tab[day]);
}

void check_timer_fields(uint8_t channel)
{int result;
 char tmp[8];
 Time tmp_time;

  myTouch.read();
  int   touch_x = myTouch.getX();
  int   touch_y = myTouch.getY();
  if(touch_x>235 && touch_x<280 && touch_y>48 && touch_y<87){ //Hours
                              myGLCD.setColor(VGA_WHITE);
                              myGLCD.fillRect(240, 58,272,77);
                              result=read_keyboard(240, 60,2);
                              if(result<0) result=0;
                              if(result>23) result=23;
                              myGLCD.setColor(VGA_GRAY);
                              myGLCD.fillRect(240, 58,272,77);
                              myGLCD.setBackColor(VGA_GRAY);
                              myGLCD.setColor(VGA_RED);
                              sprintf(tmp,"%02d",result);
                              myGLCD.print(tmp,240,60);
                              tmp_time=rtc.getTime();
                              rtc.setTime(result, tmp_time.min, tmp_time.sec);
                              }
  if(touch_x>283 && touch_x<328 && touch_y>48 && touch_y<87){  //Minutes
                              myGLCD.setColor(VGA_WHITE);
                              myGLCD.fillRect(240+48, 58,240+48+32,77);
                              result=read_keyboard(240+48, 60,2);
                              if(result<0) result=0;
                              if(result>59) result=59;
                              myGLCD.setColor(VGA_GRAY);
                              myGLCD.fillRect(240+48, 58,240+48+32,77);
                              myGLCD.setBackColor(VGA_GRAY);
                              myGLCD.setColor(VGA_RED);
                              sprintf(tmp,"%02d",result);
                              myGLCD.print(tmp,240+48, 60);
                              tmp_time=rtc.getTime();
                              rtc.setTime(tmp_time.hour, result, tmp_time.sec);
                              }
  if(touch_x>331 && touch_x<376 && touch_y>48 && touch_y<87){  //Seconds
                              myGLCD.setColor(VGA_WHITE);
                              myGLCD.fillRect(240+96, 58,240+96+32,77);
                              result=read_keyboard(240+96, 60,2);
                              if(result<0) result=0;
                              if(result>59) result=59;
                              myGLCD.setColor(VGA_GRAY);
```

```
706                             myGLCD.fillRect(240+96, 58,240+96+32,77);
707                             myGLCD.setBackColor(VGA_GRAY);
708                             myGLCD.setColor(VGA_RED);
709                             sprintf(tmp,"%02d",result);
710                             myGLCD.print(tmp,240+96, 60);
711                             tmp_time=rtc.getTime();
712                             rtc.setTime(tmp_time.hour, tmp_time.min, result);
713                             }
714     if(touch_x>456 && touch_x<504 && touch_y>48 && touch_y<87){  //Day
715                             myGLCD.setColor(VGA_WHITE);
716                             myGLCD.fillRect(464, 58,464+32,77);
717                             result=read_keyboard(464, 60,2);
718                             if(result<1) result=1;
719                             if(result>31) result=31;
720                             myGLCD.setColor(VGA_GRAY);
721                             myGLCD.fillRect(464, 58,464+32,77);
722                             myGLCD.setBackColor(VGA_GRAY);
723                             myGLCD.setColor(VGA_RED);
724                             sprintf(tmp,"%02d",result);
725                             myGLCD.print(tmp,464, 60);
726                             tmp_time=rtc.getTime();
727                             rtc.setDate(result, tmp_time.mon, tmp_time.year);
728                             rtc.setDOW(wd(tmp_time.year, tmp_time.mon, result));
729                             }
730     if(touch_x>504 && touch_x<552 && touch_y>48 && touch_y<87){  //Month
731                             myGLCD.setColor(VGA_WHITE);
732                             myGLCD.fillRect(464+48, 58,464+48+32,77);
733                             result=read_keyboard(464+48, 60,2);
734                             if(result<1) result=1;
735                             if(result>12) result=12;
736                             myGLCD.setColor(VGA_GRAY);
737                             myGLCD.fillRect(464+48, 58,464+48+32,77);
738                             myGLCD.setBackColor(VGA_GRAY);
739                             myGLCD.setColor(VGA_RED);
740                             sprintf(tmp,"%02d",result);
741                             myGLCD.print(tmp,464+48, 60);
742                             tmp_time=rtc.getTime();
743                             rtc.setDate(tmp_time.date, result, tmp_time.year);
744                             rtc.setDOW(wd(tmp_time.year, result, tmp_time.date));
745                             }
746     if(touch_x>552 && touch_x<632 && touch_y>48 && touch_y<87){  //Year
747                             myGLCD.setColor(VGA_WHITE);
748                             myGLCD.fillRect(560, 58, 624, 77);
749                             result=read_keyboard(560, 60, 4);
750                             if(result<2000) result=2000;
751                             if(result>9999) result=9999;
752                             myGLCD.setColor(VGA_GRAY);
753                             myGLCD.fillRect(560, 58, 624, 77);
754                             myGLCD.setBackColor(VGA_GRAY);
755                             myGLCD.setColor(VGA_RED);
756                             sprintf(tmp,"%04d",result);
757                             myGLCD.print(tmp, 560, 60);
758                             tmp_time=rtc.getTime();
759                             rtc.setDate(tmp_time.date, tmp_time.mon, result);
760                             rtc.setDOW(wd(result, tmp_time.mon, tmp_time.date));
761                             }
762     if(touch_x>235 && touch_x<280 && touch_y>88 && touch_y<127){ //Hours of the First Start
763                             myGLCD.setColor(VGA_WHITE);
764                             myGLCD.fillRect(240, 98,272,117);
765                             result=read_keyboard(240, 100,2);
766                             if(result<0) result=0;
767                             if(result>23) result=23;
768                             myGLCD.setColor(VGA_GRAY);
769                             myGLCD.fillRect(240, 98,272,117);
770                             myGLCD.setBackColor(VGA_GRAY);
771                             myGLCD.setColor(VGA_RED);
772                             sprintf(tmp,"%02d",result);
773                             myGLCD.print(tmp,240,100);
774                             timer_first_start[channel-1].hour=result;
775                             recalculate_schedule(channel);
776                             update_schedule_status(channel);
777                             save_start_time_eeprom(channel, &timer_first_start[channel-1]);
778                             }
779     if(touch_x>283 && touch_x<328 && touch_y>88 && touch_y<127){  //Minutes of the First start
780                             myGLCD.setColor(VGA_WHITE);
781                             myGLCD.fillRect(240+48, 98,240+48+32,117);
782                             result=read_keyboard(240+48, 100,2);
783                             if(result<0) result=0;
784                             if(result>59) result=59;
785                             myGLCD.setColor(VGA_GRAY);
786                             myGLCD.fillRect(240+48, 98,240+48+32,117);
787                             myGLCD.setBackColor(VGA_GRAY);
788                             myGLCD.setColor(VGA_RED);
789                             sprintf(tmp,"%02d",result);
790                             myGLCD.print(tmp,240+48, 100);
791                             timer_first_start[channel-1].min=result;
792                             recalculate_schedule(channel);
793                             update_schedule_status(channel);
794                             save_start_time_eeprom(channel, &timer_first_start[channel-1]);
795                             }
796     if(touch_x>456 && touch_x<504 && touch_y>88 && touch_y<127){  //Day of the First Start
797                             myGLCD.setColor(VGA_WHITE);
798                             myGLCD.fillRect(464, 98,464+32,117);
799                             result=read_keyboard(464, 100,2);
800                             if(result<1) result=1;
801                             if(result>31) result=31;
802                             myGLCD.setColor(VGA_GRAY);
803                             myGLCD.fillRect(464, 98,464+32,117);
804                             myGLCD.setBackColor(VGA_GRAY);
805                             myGLCD.setColor(VGA_RED);
806                             sprintf(tmp,"%02d",result);
807                             myGLCD.print(tmp,464, 100);
```

```
                            timer_first_start[channel-1].date=result;
                            myGLCD.setBackColor(VGA_BLACK);
                            myGLCD.setColor(VGA_WHITE);
                            myGLCD.print(get_name_weekday(wd(timer_first_start[channel-1].year,
                           timer_first_start[channel-1].mon, timer_first_start[channel-1].date)),240+160,100);
                            recalculate_schedule(channel);
                            update_schedule_status(channel);
                            save_start_time_eeprom(channel, &timer_first_start[channel-1]);
                            }
    if(touch_x>504 && touch_x<552 && touch_y>88 && touch_y<127){  //Month of the First Start
                            myGLCD.setColor(VGA_WHITE);
                            myGLCD.fillRect(464+48, 98,464+48+32,117);
                            result=read_keyboard(464+48, 100,2);
                            if(result<1) result=1;
                            if(result>12) result=12;
                            myGLCD.setColor(VGA_GRAY);
                            myGLCD.fillRect(464+48, 98,464+48+32,117);
                            myGLCD.setBackColor(VGA_GRAY);
                            myGLCD.setColor(VGA_RED);
                            sprintf(tmp,"%02d",result);
                            myGLCD.print(tmp,464+48, 100);
                            timer_first_start[channel-1].mon=result;
                            myGLCD.setBackColor(VGA_BLACK);
                            myGLCD.setColor(VGA_WHITE);
                            myGLCD.print(get_name_weekday(wd(timer_first_start[channel-1].year,
                           timer_first_start[channel-1].mon, timer_first_start[channel-1].date)),240+160,100);
                            recalculate_schedule(channel);
                            update_schedule_status(channel);
                            save_start_time_eeprom(channel, &timer_first_start[channel-1]);
                            }
    if(touch_x>552 && touch_x<632 && touch_y>88 && touch_y<127){  //Year of the First Start
                            myGLCD.setColor(VGA_WHITE);
                            myGLCD.fillRect(560, 98, 624, 117);
                            result=read_keyboard(560, 100, 4);
                            if(result<2000) result=2000;
                            if(result>9999) result=9999;
                            myGLCD.setColor(VGA_GRAY);
                            myGLCD.fillRect(560, 98, 624, 117);
                            myGLCD.setBackColor(VGA_GRAY);
                            myGLCD.setColor(VGA_RED);
                            sprintf(tmp,"%04d",result);
                            myGLCD.print(tmp, 560, 100);
                            timer_first_start[channel-1].year=result;
                            myGLCD.setBackColor(VGA_BLACK);
                            myGLCD.setColor(VGA_WHITE);
                            myGLCD.print(get_name_weekday(wd(timer_first_start[channel-1].year,
                           timer_first_start[channel-1].mon, timer_first_start[channel-1].date)),240+160,100);
                            recalculate_schedule(channel);
                            update_schedule_status(channel);
                            save_start_time_eeprom(channel, &timer_first_start[channel-1]);
                            }
     if(touch_x>235 && touch_x<280 && touch_y>128 && touch_y<167){ //Hours of Interval of change
                            myGLCD.setColor(VGA_WHITE);
                            myGLCD.fillRect(240, 138,272,157);
                            result=read_keyboard(240, 140,2);
                            if(result<0) result=0;
                            if(result>MAX_INTERVAL_HOURS) result=MAX_INTERVAL_HOURS;
                            myGLCD.setColor(VGA_GRAY);
                            myGLCD.fillRect(240, 138,272,157);
                            myGLCD.setBackColor(VGA_GRAY);
                            myGLCD.setColor(VGA_RED);
                            sprintf(tmp,"%02d",result);
                            myGLCD.print(tmp,240,140);
                            timer_intervals_hour[channel-1]=result;
                            recalculate_schedule(channel);
                            update_schedule_status(channel);
                            EEPROM.write(EE_TINT_HRS1+channel-1,timer_intervals_hour[channel-1]);
                            }
    if(touch_x>283 && touch_x<328 && touch_y>128 && touch_y<167){  //Minutes of interval of change
                            myGLCD.setColor(VGA_WHITE);
                            myGLCD.fillRect(240+48, 138,240+48+32,157);
                            result=read_keyboard(240+48, 140,2);
                            if(result<0) result=0;
                            if(result>59) result=59;
                            myGLCD.setColor(VGA_GRAY);
                            myGLCD.fillRect(240+48, 138,240+48+32,157);
                            myGLCD.setBackColor(VGA_GRAY);
                            myGLCD.setColor(VGA_RED);
                            sprintf(tmp,"%02d",result);
                            myGLCD.print(tmp,240+48, 140);
                            timer_intervals_minutes[channel-1]=result;
                            recalculate_schedule(channel);
                            update_schedule_status(channel);
                            EEPROM.write(EE_TINT_MIN1+channel-1,timer_intervals_minutes[channel-1]);
                            }
    if(touch_x>692 && touch_x<740 && touch_y>128 && touch_y<167){  //Number of changes
                            myGLCD.setColor(VGA_WHITE);
                            myGLCD.fillRect(700, 138,700+32,157);
                            result=read_keyboard(700, 140,2);
                            if(result<0) result=0;
                            if(result>59) result=59;
                            myGLCD.setColor(VGA_GRAY);
                            myGLCD.fillRect(700, 138,700+32,157);
                            myGLCD.setBackColor(VGA_GRAY);
                            myGLCD.setColor(VGA_RED);
                            sprintf(tmp,"%02d",result);
                            myGLCD.print(tmp,700, 140);
                            timer_repeates_total[channel-1]=result;
                            recalculate_schedule(channel);
                            update_schedule_status(channel);
                            EEPROM.write(EE_TREP_TOT1+channel-1,timer_repeates_total[channel-1]);
                            }
```

```
910
911
912  }
913
914  void draw_brcontrol_controls(uint8_t channel)
915  {char msg[24];
916
917     myGLCD.setColor(VGA_BLACK);
918     myGLCD.fillRect(0, 0,799,479);
919
920     myGLCD.setFont(BigFont);
921     myGLCD.setBackColor(VGA_BLACK);
922     myGLCD.setColor(VGA_YELLOW);
923     sprintf(msg,"BR Unit NR.%d CONTROL",channel);
924
925     myGLCD.print(msg,CENTER,0);
926     while (myTouch.dataAvailable() == true);
927
928     myGLCD.setColor(VGA_WHITE);
929     myGLCD.drawRect(0, 20,400-2,180-2);
930     myGLCD.print("PERFUSION",127,20);
931     myGLCD.print("Perf.Speed:",5,50);
932     myGLCD.print("ul/min",255,50);
933     myGLCD.setColor(VGA_GRAY);
934     myGLCD.fillRect(200, 50,250,69);
935
936     but_perstart = myButtons.addButton(25, 150-BUTSIZEY, 150,  BUTSIZEY, "START");
937     but_perstop = myButtons.addButton(225, 150-BUTSIZEY, 150,  BUTSIZEY, "STOP");
938
939     myGLCD.setColor(VGA_WHITE);
940     myGLCD.setBackColor(VGA_BLACK);
941     myGLCD.drawRect(0, 180,400-2,360-2);
942     myGLCD.print("MEDIUM CHANGE",95,180);
943     myGLCD.print("Volume:",5,210);
944     myGLCD.print("ul",77,230);
945     myGLCD.print("Speed:",200,210);
946     myGLCD.print("ul/min",257,230);
947     myGLCD.setColor(VGA_GRAY);
948     myGLCD.fillRect(5, 230,5+68,249);
949     myGLCD.fillRect(200, 230,200+52,249);
950     myGLCD.setBackColor(VGA_GRAY);
951     myGLCD.setColor(VGA_RED);
952     sprintf(msg,"%d",medchg_vol_tab[channel-1]);
953     myGLCD.print(msg,7,232);
954     sprintf(msg,"%d",medchg_speed_tab[channel-1]);
955     myGLCD.print(msg,202,232);
956
957     but_medstart = myButtons.addButton(25, 325-BUTSIZEY, 150,  BUTSIZEY, "START");
958     but_timer = myButtons.addButton(225, 325-BUTSIZEY, 150,  BUTSIZEY, "TIMER");
959     myGLCD.setBackColor(VGA_BLACK);
960     myGLCD.setColor(VGA_WHITE);
961     myGLCD.print("Next: ",10,338);
962     if(timer_repeates_left[channel-1] && timer_enables[channel-1]){
963                     sprintf(msg,"%02d",timer_next_start[channel-1].hour);
964                     myGLCD.print(msg,106,338);
965                     sprintf(msg,"%02d",timer_next_start[channel-1].min);
966                     myGLCD.print(msg,106+48,338);
967                     sprintf(msg,"%02d",timer_next_start[channel-1].date);
968                     myGLCD.print(msg,106+108,338);
969                     sprintf(msg,"%02d",timer_next_start[channel-1].mon);
970                     myGLCD.print(msg,106+108+48,338);
971                     sprintf(msg,"%04d",timer_next_start[channel-1].year);
972                     myGLCD.print(msg,106+108+96,338);
973                     myGLCD.print(":",106+32,338);
974                     myGLCD.print(".",106+108+32,338);
975                     myGLCD.print(".",106+108+80,338);
976           }
977           else
978           {
979             myGLCD.print(" ---              ",106,338);
980             }
981
982     myGLCD.setColor(VGA_WHITE);
983     myGLCD.setBackColor(VGA_BLACK);
984     myGLCD.drawRect(400, 20,799,180-2);
985     myGLCD.print("PREPARE SAMPLE",487,20);
986
987     but_sampstart = myButtons.addButton(400+125, 150-BUTSIZEY, 150,  BUTSIZEY, "START");
988
989     myGLCD.setColor(VGA_WHITE);
990     myGLCD.setBackColor(VGA_BLACK);
991     myGLCD.drawRect(400, 180,799,360-2);
992     myGLCD.print("MANUAL CONTROL",487,180);
993
994     myGLCD.print("I2C ADDR:",405,210);
995     myGLCD.setColor(VGA_GRAY);
996     myGLCD.fillRect(405+144, 210,405+144+52,229);
997     myGLCD.setBackColor(VGA_GRAY);
998     myGLCD.setColor(VGA_RED);
999     sprintf(msg,"%d",i2c_adr_tab[channel-1]);
1000    myGLCD.print(msg,405+144+2,210);
1001
1002    but_v1 = myButtons.addButton(400+10, 350-2*BUTSIZEY-10, 75,  BUTSIZEY, "");
1003    but_v2 = myButtons.addButton(400+10+75+10, 350-2*BUTSIZEY-10, 75,  BUTSIZEY, "");
1004    but_v3 = myButtons.addButton(400+10, 350-BUTSIZEY, 75,  BUTSIZEY, "");
1005    but_v4 = myButtons.addButton(400+10+75+10, 350-BUTSIZEY, 75,  BUTSIZEY, "");
1006    but_pump = myButtons.addButton(800-150-10, 350-BUTSIZEY, 150,  BUTSIZEY, "");
1007    but_reset = myButtons.addButton(800-150-10, 350-2*BUTSIZEY-10, 150,  BUTSIZEY, "!RESET!");
1008
1009    draw_keyboard();
1010
1011    myGLCD.setBackColor(VGA_BLUE);
```

```
1012    myGLCD.setColor(VGA_WHITE);
1013    myGLCD.print("V1",431,249);
1014    myGLCD.print("V2",431+85,249);
1015    myGLCD.print("V3",431,249+60);
1016    myGLCD.print("V4",431+85,249+60);
1017    myGLCD.print("PUMP",720-32,249+60);
1018 }
1019
1020 void recalculate_schedule(uint8_t channel)
1021 {
1022   if(timer_repeates_total[channel-1]==0){
1023                                         timer_repeates_left[channel-1]=0;
1024                                         return;
1025                                        }
1026  calc_next_start(channel);
1027 }
1028
1029 void update_schedule_status(uint8_t channel)
1030 {char msg[30];
1031
1032   myGLCD.setBackColor(VGA_BLACK);
1033    myGLCD.setColor(VGA_WHITE);
1034
1035    sprintf(msg,"%02d",timer_repeates_left[channel-1]);
1036    myGLCD.print(msg,470,300);
1037
1038   if(timer_repeates_left[channel-1] && timer_enables[channel-1]){
1039                    sprintf(msg,"%02d",timer_next_start[channel-1].hour);
1040                    myGLCD.print(msg,380,330);
1041                    sprintf(msg,"%02d",timer_next_start[channel-1].min);
1042                    myGLCD.print(msg,380+48,330);
1043                    sprintf(msg,"%02d",timer_first_start[channel-1].sec);
1044                    myGLCD.print(msg,380+96,330);
1045                    sprintf(msg,"%02d",timer_next_start[channel-1].date);
1046                    myGLCD.print(msg,604,330);
1047                    sprintf(msg,"%02d",timer_next_start[channel-1].mon);
1048                    myGLCD.print(msg,604+48,330);
1049                    sprintf(msg,"%04d",timer_next_start[channel-1].year);
1050                    myGLCD.print(msg,604+96,330);
1051                    myGLCD.print(get_name_weekday(wd(timer_next_start[channel-1].year,
1052                     timer_next_start[channel-1].mon, timer_next_start[channel-1].date)),540,330);
1053                    myGLCD.print(":",412,330);
1054                    myGLCD.print(":",412+48,330);
1055                    myGLCD.print(".",588+48,330);
1056                    myGLCD.print(".",588+96,330);
1057                    myGLCD.print(",",588,330);
1058           }
1059           else
1060           {
1061             myGLCD.print("---                     ",380,330);
1062             }
1063
1064 }
1065
1066 void medium_change_schedule(uint8_t channel)
1067 {char msg[30];
1068  uint8_t i,tmp,return_home;
1069  int  pressed_button;
1070  unsigned long nextUpdate=0;
1071  Time curr_time;
1072
1073  while (myTouch.dataAvailable() == true);
1074
1075    draw_keyboard();
1076
1077    myGLCD.setColor(VGA_BLACK);
1078    myGLCD.fillRect(0, 20,799,360);
1079    myGLCD.setColor(VGA_WHITE);
1080    myGLCD.setBackColor(VGA_BLACK);
1081    myGLCD.drawRect(0, 20,799,360-2);
1082    myGLCD.print("Medium Change scheduler settings:",150,20);
1083    myGLCD.print("Current time: ",20,60);
1084    myGLCD.setBackColor(VGA_BLACK);
1085    myGLCD.setColor(VGA_WHITE);
1086    myGLCD.print(":",272,60);
1087    myGLCD.print(":",272+48,60);
1088    myGLCD.print(".",496,60);
1089    myGLCD.print(".",496+48,60);
1090    myGLCD.setColor(VGA_GRAY);
1091    myGLCD.fillRect(240,58,272,77);
1092    myGLCD.fillRect(288,58,320,77);
1093    myGLCD.fillRect(336,58,368,77);
1094    myGLCD.fillRect(464,58,496,77);
1095    myGLCD.fillRect(512,58,544,77);
1096    myGLCD.fillRect(560,58,624,77);
1097
1098    myGLCD.setColor(VGA_WHITE);
1099    myGLCD.print("First start: ",20,100);
1100    myGLCD.setBackColor(VGA_BLACK);
1101    myGLCD.setColor(VGA_WHITE);
1102    myGLCD.print(":",272,100);
1103    myGLCD.print(":",272+48,100);
1104    myGLCD.print(".",496,100);
1105    myGLCD.print(".",496+48,100);
1106    myGLCD.print(",",448,100);
1107    myGLCD.setColor(VGA_GRAY);
1108    myGLCD.fillRect(240,98,272,117);
1109    myGLCD.fillRect(288,98,320,117);
1110   // myGLCD.fillRect(336,98,368,117);
1111    myGLCD.fillRect(464,98,496,117);
1112    myGLCD.fillRect(512,98,544,117);
1113    myGLCD.fillRect(560,98,624,117);
```

```
1114
1115     myGLCD.setBackColor(VGA_GRAY);
1116     myGLCD.setColor(VGA_RED);
1117     sprintf(msg,"%02d",timer_first_start[channel-1].hour);
1118     myGLCD.print(msg,240,100);
1119     sprintf(msg,"%02d",timer_first_start[channel-1].min);
1120     myGLCD.print(msg,240+48,100);
1121     sprintf(msg,"%02d",timer_first_start[channel-1].date);
1122     myGLCD.print(msg,240+224,100);
1123     sprintf(msg,"%02d",timer_first_start[channel-1].mon);
1124     myGLCD.print(msg,240+224+48,100);
1125     sprintf(msg,"%04d",timer_first_start[channel-1].year);
1126     myGLCD.print(msg,240+224+96,100);
1127     myGLCD.setBackColor(VGA_BLACK);
1128     myGLCD.setColor(VGA_WHITE);
1129     //sprintf(msg,"%s,",rtc.getDOWStr(FORMAT_SHORT));
1130     myGLCD.print(get_name_weekday(wd(timer_first_start[channel-1].year,
1131      timer_first_start[channel-1].mon, timer_first_start[channel-1].date)),240+160,100);
1132     sprintf(msg,"%02d",timer_first_start[channel-1].sec);
1133     myGLCD.print(msg,240+96,100);
1134
1135    myGLCD.setColor(VGA_WHITE);
1136    myGLCD.print("Change Period: ",20,140);
1137    myGLCD.print("Number of Changes: ",400,140);
1138    myGLCD.setBackColor(VGA_BLACK);
1139    myGLCD.setColor(VGA_WHITE);
1140    myGLCD.print(":",272,140);
1141    myGLCD.setColor(VGA_GRAY);
1142    myGLCD.fillRect(240,138,272,157);
1143    myGLCD.fillRect(288,138,320,157);
1144    myGLCD.fillRect(700,138,700+32,157);
1145    myGLCD.setBackColor(VGA_GRAY);
1146    myGLCD.setColor(VGA_RED);
1147    sprintf(msg,"%02d",timer_intervals_hour[channel-1]);
1148    myGLCD.print(msg,240,140);
1149    sprintf(msg,"%02d",timer_intervals_minutes[channel-1]);
1150    myGLCD.print(msg,240+48,140);
1151    sprintf(msg,"%02d",timer_repeates_total[channel-1]);
1152    myGLCD.print(msg,700,140);
1153
1154    myGLCD.setBackColor(VGA_BLACK);
1155    myGLCD.setColor(VGA_WHITE);
1156
1157    myGLCD.print("Scheduling is switched ",20,220);
1158    but_sched = myButtons.addButton(400, 220-BUTSIZEY/2+8, 75,  BUTSIZEY, "");
1159    myButtons.drawButtons();
1160
1161    myGLCD.print("Remaining scheduled starts: ",20,300);
1162    myGLCD.print("Next scheduled start: ",20,330);
1163    myGLCD.print(":",380+32,330);
1164    myGLCD.print(":",380+80,330);
1165    myGLCD.print(",",588,330);
1166    myGLCD.print(".",588+48,330);
1167    myGLCD.print(".",588+96,330);
1168
1169    update_schedule_status(channel);
1170
1171
1172  return_home=0;
1173  while(!return_home){
1174
1175           if (millis() >= nextUpdate){
1176                                    nextUpdate = millis() + 250; // set up the next timeout period
1177
1178                                    // Get data from the DS3231
1179                                    curr_time = rtc.getTime();
1180
1181                                    myGLCD.setBackColor(VGA_GRAY);
1182                                    myGLCD.setColor(VGA_RED);
1183                                    sprintf(msg,"%02d",curr_time.hour);
1184                                    myGLCD.print(msg,240,60);
1185                                    sprintf(msg,"%02d",curr_time.min);
1186                                    myGLCD.print(msg,240+48,60);
1187                                    sprintf(msg,"%02d",curr_time.sec);
1188                                    myGLCD.print(msg,240+96,60);
1189                                    sprintf(msg,"%02d",curr_time.date);
1190                                    myGLCD.print(msg,240+224,60);
1191                                    sprintf(msg,"%02d",curr_time.mon);
1192                                    myGLCD.print(msg,240+224+48,60);
1193                                    sprintf(msg,"%04d",curr_time.year);
1194                                    myGLCD.print(msg,240+224+96,60);
1195                                    myGLCD.setBackColor(VGA_BLACK);
1196                                    myGLCD.setColor(VGA_WHITE);
1197                                    sprintf(msg,"%s,",rtc.getDOWStr(FORMAT_SHORT));
1198                                    myGLCD.print(msg,240+160,60);
1199
1200                                    myGLCD.setBackColor(VGA_BLUE);
1201                                    myGLCD.setColor(VGA_WHITE);
1202                                    if(timer_enables[channel-1]==0) myGLCD.print("OFF",415,220);
1203                                           else myGLCD.print(" ON ",415-8,220);
1204                                    recalculate_schedule(channel);
1205                                    update_schedule_status(channel);
1206                                    }
1207
1208           if(myTouch.dataAvailable()==true){
1209                     check_timer_fields(channel);
1210                     pressed_button = myButtons.checkButtons();
1211                     if (pressed_button==butBACK){
1212                                              return_home=1;
1213                                              }
1214                     if (pressed_button==but_sched){nextUpdate = millis();
1215                          if(timer_enables[channel-1]) timer_enables[channel-1]=0;
```

```
1216                                    else timer_enables[channel-1]=1;
1217                            if(EEPROM.read(EE_TENA1+channel-1)!= timer_enables[channel-1])
1218                                    EEPROM.write(EE_TENA1+channel-1,timer_enables[channel-1]);
1219                           }
1220
1221
1222                     }
1223              }
1224
1225  while (myTouch.dataAvailable() == true);
1226
1227  myButtons.deleteAllButtons();
1228  draw_brcontrol_controls(channel);
1229 }
1230
1231 void draw_brcontrol(uint8_t channel)
1232 {char msg[24];
1233  uint8_t i,tmp,return_home;
1234  int  pressed_button;
1235  static uint8_t valve_states[8]={0,0,0,0,0,0,0,0};
1236  br_status tmpstatus;
1237  char err;
1238  unsigned long nextUpdate=0;
1239
1240  err=get_br_status(channel, &tmpstatus);
1241    if(!err){tmp=1;
1242             for(i=0;i<8;i++){valve_states[i]=0;
1243                              if(tmpstatus.valves&tmp) valve_states[i]++;
1244                              tmp=tmp<<1;
1245                             }
1246     }
1247
1248  draw_brcontrol_controls(channel);
1249
1250  return_home=0;
1251  while(!return_home){
1252
1253          if (millis() >= nextUpdate){
1254            nextUpdate = millis() + 250; // set up the next timeout period
1255            err=get_br_status(channel, &tmpstatus);
1256            if(!err){tmp=1;
1257             for(i=0;i<8;i++){valve_states[i]=0;
1258                      if(tmpstatus.valves&tmp) valve_states[i]++;
1259                      tmp=tmp<<1;
1260                     }
1261               myGLCD.setBackColor(VGA_BLUE);
1262               myGLCD.setColor(VGA_WHITE);
1263               if(!valve_states[BR_V1-1]) myGLCD.print("OFF",431-8,249+16);
1264                      else myGLCD.print(" ON ",431-16,249+16);
1265               if(!valve_states[BR_V2-1]) myGLCD.print("OFF",431+85-8,249+16);
1266                      else myGLCD.print(" ON ",431+85-16,249+16);
1267               if(!valve_states[BR_V3-1]) myGLCD.print("OFF",431-8,249+60+16);
1268                      else myGLCD.print(" ON ",431-16,249+60+16);
1269               if(!valve_states[BR_V4-1]) myGLCD.print("OFF",431+85-8,249+60+16);
1270                      else myGLCD.print(" ON ",431+85-16,249+60+16);
1271               if((tmpstatus.flags&0x80)==0) myGLCD.print("OFF",720-24,249+60+16);
1272                      else myGLCD.print(" ON ",720-32,249+60+16);
1273               myGLCD.setBackColor(VGA_GRAY);
1274               myGLCD.setColor(VGA_RED);
1275               sprintf(msg,"%d",tmpstatus.perfusion_speed);
1276               myGLCD.print(msg,202,52);
1277              }
1278            }
1279
1280          if(myTouch.dataAvailable()==true){
1281               check_num_fields(channel);
1282               pressed_button = myButtons.checkButtons();
1283               if (pressed_button==butBACK){
1284                                           return_home=1;
1285                                           }
1286               else if(pressed_button==but_perstart){
1287                                                    control_perfusion(channel,1);
1288                                                   }
1289               else if(pressed_button==but_perstop){
1290                                                    control_perfusion(channel,0);
1291                                                   }
1292               else if(pressed_button==but_reset){
1293                                                    br_reset(channel);
1294                                                   }
1295               else if(pressed_button==but_v1){
1296                       if(!valve_states[BR_V1-1]){valve_states[BR_V1-1]=1;
1297                                                  control_valve(channel,BR_V1,1);
1298                                  }
1299                                  else {valve_states[BR_V1-1]=0;
1300                                        control_valve(channel,BR_V1,0);
1301                                        }
1302                       }
1303               else if(pressed_button==but_v2){
1304                       if(!valve_states[BR_V2-1]){valve_states[BR_V2-1]=1;
1305                                                  control_valve(channel,BR_V2,1);
1306                                  }
1307                                  else {valve_states[BR_V2-1]=0;
1308                                        control_valve(channel,BR_V2,0);
1309                                        }
1310                       }
1311               else if(pressed_button==but_v3){
1312                       if(!valve_states[BR_V3-1]){valve_states[BR_V3-1]=1;
1313                                                  control_valve(channel,BR_V3,1);
1314                                  }
1315                                  else {valve_states[BR_V3-1]=0;
1316                                        control_valve(channel,BR_V3,0);
1317                                        }
```

```
1318                    }
1319                 else if(pressed_button==but_v4){
1320                        if(!valve_states[BR_V4-1]){valve_states[BR_V4-1]=1;
1321                                                   control_valve(channel,BR_V4,1);
1322                                    }
1323                                    else {valve_states[BR_V4-1]=0;
1324                                          control_valve(channel,BR_V4,0);
1325                                          }
1326                        }
1327                else  if(pressed_button==but_pump && (!(tmpstatus.flags&0x7F))){
1328                    //only in the stanby mode
1329                        if((tmpstatus.flags&0x80)==0){
1330                             start_pump(channel,medchg_vol_tab[channel-1],medchg_speed_tab[channel-1]);
1331                            }
1332                            else {
1333                                  start_pump(channel,1,0);  //stop the pump
1334                                  }
1335                        }
1336                else  if(pressed_button==but_medstart){
1337                        if((tmpstatus.flags&0x06)==0){
1338                        //only if not sampling and not changing the medium
1339                             medium_change(channel,medchg_vol_tab[channel-1],medchg_speed_tab[channel-1]);
1340                              }
1341
1342                        }
1343                else  if(pressed_button==but_timer){
1344                        if(/*(tmpstatus.flags&0x06)==0*/1){
1345                        //only if not sampling and not changing the medium
1346                                               myButtons.deleteAllButtons();
1347                                               medium_change_schedule(channel);
1348                                                }
1349
1350                        }
1351
1352                }
1353               }
1354
1355  while (myTouch.dataAvailable() == true);
1356
1357  myButtons.deleteAllButtons();
1358  for(i=1;i<9;i++) draw_status(i,0);
1359 }
1360
1361 void medium_change_scheduler(void)
1362 {uint8_t i;
1363
1364   for(i=0; i<8; i++){
1365         if(timer_enables[i]){
1366               if(timer_repeates_left[i] && (is_time_future(i+1, &timer_next_start[i])==0)){
1367                                     timer_repeates_left[i]--;
1368                                     medium_change(i+1,medchg_vol_tab[i],medchg_speed_tab[i]);
1369                                     recalculate_schedule(i+1);
1370                                     print_global_diag();
1371                                     }
1372             }
1373       }
1374 }
1375
1376 void print_global_diag(void)
1377 {uint8_t i;
1378   Serial.print("\r\n");
1379   sprintf(diag_msg,"BR unit Connections: %d %d %d %d %d %d %d %d",br_connections[0], \
1380            br_connections[1],br_connections[2],br_connections[3], br_connections[4], \
1381            br_connections[5],br_connections[6],br_connections[7]);
1382   diag_out(diag_msg);
1383   sprintf(diag_msg,"M.E. Timer enabled: %d %d %d %d %d %d %d %d",timer_enables[0], \
1384            timer_enables[1],timer_enables[2],timer_enables[3], timer_enables[4], \
1385            timer_enables[5],timer_enables[6],timer_enables[7]);
1386   diag_out(diag_msg);
1387   sprintf(diag_msg,"M.E. Intervals (hrs): %d %d %d %d %d %d %d %d",timer_intervals_hour[0], \
1388            timer_intervals_hour[1],timer_intervals_hour[2],timer_intervals_hour[3], \
1389            timer_intervals_hour[4],timer_intervals_hour[5],timer_intervals_hour[6], \
1390            timer_intervals_hour[7]);
1391   diag_out(diag_msg);
1392   sprintf(diag_msg,"M.E. Intervals (min): %d %d %d %d %d %d %d %d",timer_intervals_minutes[0], \
1393            timer_intervals_minutes[1],timer_intervals_minutes[2], \
1394             timer_intervals_minutes[3],timer_intervals_minutes[4],timer_intervals_minutes[5], \
1395            timer_intervals_minutes[6],timer_intervals_minutes[7]);
1396   diag_out(diag_msg);
1397   sprintf(diag_msg,"M.E. Repats total: %d %d %d %d %d %d %d %d", timer_repeates_total[0], \
1398            timer_repeates_total[1],timer_repeates_total[2],timer_repeates_total[3],\
1399            timer_repeates_total[4],timer_repeates_total[5],timer_repeates_total[6], \
1400            timer_repeates_total[7]);
1401   diag_out(diag_msg);
1402   sprintf(diag_msg,"M.E. Repats remaining: %d %d %d %d %d %d %d %d",timer_repeates_left[0], \
1403            timer_repeates_left[1],timer_repeates_left[2],timer_repeates_left[3], \
1404             timer_repeates_left[4],timer_repeates_left[5],timer_repeates_left[6], \
1405            timer_repeates_left[7]);
1406   diag_out(diag_msg);
1407
1408   for(i=0;i<8;i++){
1409     sprintf(diag_msg,"BR%d - 1st M.E. start: %d.%d.%d %02d:%02d, NEXT M.E. start: %d.%d.%d %02d:%02d.",i+1, \
1410                      timer_first_start[i].date, timer_first_start[i].mon,timer_first_start[i].year, \
1411                     timer_first_start[i].hour,timer_first_start[i].min, timer_next_start[i].date, \
1412                     timer_next_start[i].mon,timer_next_start[i].year,timer_next_start[i].hour, \
1413                     timer_next_start[i].min);
1414     diag_out(diag_msg);
1415     }
1416   Serial.print("\r\n");
1417 }
1418
1419 void diag_out(char* dg_msg)
```

```
1420 {Time curr_time;
1421  char msg[28];
1422
1423   // Get data from the DS3231
1424   curr_time = rtc.getTime();
1425
1426   sprintf(msg,"\r\n%s,%d.%d. %02d:%02d:%02d> ",rtc.getDOWStr(FORMAT_SHORT),curr_time.date, \
1427           curr_time.mon,curr_time.hour, curr_time.min, curr_time.sec);
1428   Serial.print(msg);
1429   Serial.print(dg_msg);
1430 }
1431
1432 void loop()
1433 {int  pressed_button;
1434  static boolean default_colors = true;
1435  static char upd_ch=1;
1436  char i;
1437  static unsigned long nextUpdate=0;
1438  unsigned long timeout=200;
1439  unsigned int glob_diag_timeout=18000; //in "timeout" units (200ms)
1440  static unsigned int glob_diag_timer=0;
1441
1442  if (millis() >= nextUpdate){
1443                              nextUpdate = millis() + timeout; // set up the next timeout period
1444                              draw_status(upd_ch,1);
1445                              upd_ch++;
1446                              if(upd_ch>8) upd_ch=1;
1447                              medium_change_scheduler();
1448                              glob_diag_timer++;
1449                              if(glob_diag_timer>=glob_diag_timeout) {glob_diag_timer=0;
1450                                                                     print_global_diag();
1451                                                                     }
1452                              }
1453
1454   if (myTouch.dataAvailable() == true)
1455   {
1456     i=get_selected_ch();
1457 //    Serial.print((int)i);
1458 //    Serial.print("\n\r");
1459     draw_brcontrol(i);
1460     }
1461
1462
1463 }
1464
```

```
/*************************************************************************\
*                Controller for BR drivers -  HW ver1.0                   *
*-------------------------------------------------------------------------*
* Description : supporting library for BR driver controller               *
*                                                                         *
*-------------------------------------------------------------------------*
* Author      : Martin Baca                                               *
* Developed   : 07.06.2016                Last Update : 29.12.2017        *
* Version     : 1.2                                                       *
*-------------------------------------------------------------------------*
* Compiler    : arduino                                                   *
* Source file : BR_lib.h                                                  *
*-------------------------------------------------------------------------*
* Target system : Arduino Mega 2560 board, Rev.3                          *
*                 ITDB50 - 5" TFT Display 800x480,                        *
*                 DS3231 RTC module                                       *
* Target CPU    : ATmega2560 @16 MHz, UART:  115200,N,8,1                 *
* Emulator HW   :                                                         *
\*************************************************************************/

#ifndef _BR_Lib_h
#define _BR_Lib_h

//#include <Arduino.h>

#define STAT_SIZEX 400
#define STAT_SIZEY 120

#define BR_V1        1
#define BR_V2        4
#define BR_V3        3
#define BR_V4        2

#define BR_V1_MASK        (1<<(BR_V1-1))
#define BR_V2_MASK        (1<<(BR_V2-1))
#define BR_V3_MASK        (1<<(BR_V3-1))
#define BR_V4_MASK        (1<<(BR_V4-1))

typedef struct {
  uint8_t flags;          //
  int16_t pump_speed;     //
  uint8_t valves;         //
  uint8_t pump_percent;   //
  uint16_t pump_time2end; //
  uint8_t total_percent;  //
  uint16_t total_time2end;//
  uint16_t perfusion_speed;//
} br_status;

extern uint8_t br_connections[];

void draw_status(uint8_t channel, uint8_t update_mode);
uint8_t get_br_status(uint8_t channel, br_status *brstatus);
void draw_valve_state(uint16_t posx, uint16_t posy, uint16_t sizex, uint16_t sizey,
                      uint8_t valvenr, uint8_t state, uint8_t update_mode);
void send_i2c_msg(uint8_t channel, unsigned char *data);

#endif

```

```
1   *------------------------------------------------------------------------*
2   * Description : supporting library for BR driver controller              *
3   *                                                                        *
4   *------------------------------------------------------------------------*
5   * Author      : Martin Baca                                              *
6   * Developed   : 07.06.2016                Last Update : 29.12.2017       *
7   * Version     : 1.2                                                      *
8   *------------------------------------------------------------------------*
9   * Compiler    : arduino                                                  *
10  * Source file : BR_lib.h                                                 *
11  *------------------------------------------------------------------------*
12  * Target system : Arduino Mega 2560 board, Rev.3                         *
13  *                 ITDB50 - 5" TFT Display 800x480,                       *
14  *                 DS3231 RTC module                                      *
15  * Target CPU    : ATmega2560 @16 MHz, UART:  115200,N,8,1                *
16  * Emulator HW   :                                                        *
17  \************************************************************************/
18
19  #include <Wire.h>
20  #include <UTFT.h>
21  #include "BR_Lib.h"
22
23  extern uint8_t SmallFont[];
24  extern uint8_t BigFont[];
25  extern uint8_t Dingbats1_XL[];
26  extern uint8_t i2c_adr_tab[];
27
28  // Remember to change the model parameter to suit your display module!
29  extern UTFT myGLCD;
30
31
32  void send_i2c_msg(uint8_t channel,unsigned char *data)
33  {unsigned char len;
34
35   len=*data++;
36
37   Wire.beginTransmission(channel); // transmit to device
38
39   while(len--){ Wire.write(*data++);              // sends one byte
40                }
41   Wire.endTransmission();    // stop transmitting
42
43  }
44
45
46  void draw_progress_bar(uint16_t posx, uint16_t posy, uint16_t sizex, uint16_t sizey,
47                         uint8_t percent, uint8_t update_mode)
48  {char pstring[5];
49   uint16_t text_xpos;
50
51    if(percent>100) percent=100;
52   if(!update_mode){
53    myGLCD.setColor(VGA_BLACK);
54    myGLCD.fillRect(posx, posy, posx+sizex, posy+sizey);
55    myGLCD.setColor(VGA_BLUE);
56    myGLCD.drawRect(posx+1, posy+1, posx+sizex-1, posy+sizey-1);
57   }
58    myGLCD.setColor(VGA_BLUE);
59    myGLCD.fillRect(posx+1, posy+1, posx+(((long)sizex*percent)/100)-1, posy+sizey-1);
60
61    myGLCD.setFont(SmallFont);
62    if(percent<51)myGLCD.setBackColor(VGA_BLACK); else myGLCD.setBackColor(VGA_BLUE);
63    myGLCD.setColor(VGA_YELLOW);
64
65   text_xpos=posx+sizex/2-16;
66   if(percent<10) text_xpos+=8;
67     else if(percent<100) text_xpos+=4;
68
69    sprintf(pstring,"%d%%",percent);
70    myGLCD.print(pstring, text_xpos, posy+sizey/2-6);
71  }
72
73  void draw_valve_state(uint16_t posx, uint16_t posy, uint16_t sizex, uint16_t sizey, uint8_t valvenr,
74                        uint8_t state, uint8_t update_mode)
75  {char tmp[8];
76   int16_t tmpcolor,tmpx;
77
78   if(valvenr<1 || valvenr>8) return;
79
80  if(!update_mode){
81                myGLCD.setFont(BigFont);
82                myGLCD.setColor(VGA_GRAY);
83                myGLCD.fillRoundRect(posx, posy, posx+sizex, posy+sizey);
84                myGLCD.setColor(VGA_BLUE);
85                myGLCD.drawRoundRect(posx, posy, posx+sizex, posy+sizey);
86               }
87    myGLCD.setFont(BigFont);
88    myGLCD.setBackColor(VGA_GRAY);
89
90   if(!update_mode){
91                  sprintf(tmp,"V%d",valvenr);
92                  myGLCD.setColor(VGA_BLACK);
93                  myGLCD.print(tmp, posx+sizex/2-16, posy+4);
94                 }
95
96    tmpcolor=VGA_RED;
97    if(state) tmpcolor=VGA_LIME;
98    myGLCD.setColor(tmpcolor);
99
100   sprintf(tmp,"OFF");
101   tmpx= posx+sizex/2-24;
102   if(state) {sprintf(tmp," ON ");
```

```
103                tmpx-=8;
104                }
105   myGLCD.print(tmp, tmpx, posy+4+16+8);
106 }
107
108
109 uint8_t get_br_status(uint8_t channel, br_status *brstatus)
110 {uint8_t tmp[16],i;
111  uint32_t timer=200;
112
113  if(channel<1 || channel>8){Serial.print("incorrect channel\n\r"); return(-1);}
114
115  tmp[0]=2;
116  tmp[1]=0x41;
117  tmp[2]='\n';
118  send_i2c_msg(i2c_adr_tab[channel-1],tmp);
119  delay(80);
120 //  Serial.print("Requesting....");
121  Wire.requestFrom(i2c_adr_tab[channel-1], (uint8_t) 13);    // request 13 bytes from slave device
122 //  Serial.print("Done!\r\n");
123  i=0;
124  while (Wire.available()) { // slave may send less than requested
125                             tmp[i++] = Wire.read(); // receive a byte as character
126                             if(i>13) break;
127                             }
128  if(i<11) {//Serial.print("not enough data\n\r");
129            return(-1);
130            }
131  if(tmp[12]!='\n') {//Serial.print("incorrect data\n\r");
132            return(-1);
133            }
134  /*
135  Serial.print(channel);
136  Serial.print("> ");
137  for(i=0;i<13;i++){
138                    if (tmp[i] < 16) {Serial.print("0");}
139                    Serial.print(tmp[i],HEX);
140                    Serial.print(' ');
141                   }
142  Serial.print("\n\r");
143  */
144  brstatus->flags=tmp[0];
145  brstatus->pump_speed=(tmp[1]<<8)+tmp[2];
146  brstatus->valves=tmp[3];
147  brstatus->pump_percent=tmp[4];
148  brstatus->pump_time2end=(tmp[5]<<8)+tmp[6];
149  brstatus->total_percent=tmp[7];
150  brstatus->total_time2end=(tmp[8]<<8)+tmp[9];
151  brstatus->perfusion_speed=(tmp[10]<<8)+tmp[11];
152  return(0);
153 }
154
155
156 void draw_status(uint8_t channel, uint8_t update_mode)
157 {char msg[26];
158  uint16_t posx,posy;
159  uint16_t sizex=STAT_SIZEX;
160  uint16_t sizey=STAT_SIZEY;
161  br_status tmpstatus;
162  char err,tmp;
163
164
165   if(channel<1 || channel>8) return;
166
167   posx=0;
168   if(channel>4) posx+=sizex;
169   posy=((channel-1)%4)*sizey;
170
171
172   err=get_br_status(channel, &tmpstatus);
173  if(!err) tmp=1; else tmp=0;
174  if(br_connections[channel-1]!=tmp){br_connections[channel-1]=tmp; update_mode=0;}
175
176  if(!update_mode){
177               myGLCD.setColor(VGA_BLACK);
178               myGLCD.fillRect(posx, posy,posx+sizex-1,posy+sizey-1);
179               myGLCD.setColor(VGA_WHITE);
180               myGLCD.drawRect(posx+1, posy+1,posx+sizex-2,posy+sizey-2);
181
182               myGLCD.setFont(BigFont);
183               myGLCD.setBackColor(VGA_BLACK);
184               myGLCD.setColor(VGA_YELLOW);
185               sprintf(msg,"BR Unit NR.%d - ",channel);
186               if(!err) sprintf(msg+15,"Connected");
187                      else sprintf(msg+15,"Offline");
188
189               myGLCD.print(msg,posx+4,posy+3);
190              }
191   if(err) return;
192
193   myGLCD.setFont(BigFont);
194   myGLCD.setBackColor(VGA_BLACK);
195   myGLCD.setColor(VGA_WHITE);
196   if(!tmpstatus.flags) sprintf(msg,"Standby                ");
197    else if(tmpstatus.flags&0x04) sprintf(msg,"Changing Medium       ");
198    else if(tmpstatus.flags&0x02) sprintf(msg,"Preparing Sample      ");
199    else if(tmpstatus.flags&0x01) sprintf(msg,"Perfusion / Incubation");
200    else if(tmpstatus.flags&0x80) sprintf(msg,"Manual Pump Control   ");
201   myGLCD.print(msg, posx+4,posy+sizey-1-16-4-18 );
202
203   if(tmpstatus.flags&0x06 || tmpstatus.flags==0x80){
204               draw_progress_bar(posx+4,posy+sizey-1-16-4,sizex-104-16,16,tmpstatus.total_percent, update_mode);
```

```
205               myGLCD.setFont(SmallFont);
206               myGLCD.setBackColor(VGA_BLACK);
207               myGLCD.setColor(VGA_YELLOW);
208               sprintf(msg,"End in:%dmin",tmpstatus.total_time2end/60);
209               myGLCD.print(msg, posx+sizex-1-4-104, posy+sizey-1-4-12);
210               }
211               else{
212                 myGLCD.setColor(VGA_BLACK);
213                 myGLCD.fillRoundRect(posx+4,posy+sizey-1-16-4,posx+sizex-4,posy+sizey-1-16-4+16);
214                 }
215
216 if(!update_mode){
217               myGLCD.setFont(BigFont);
218               myGLCD.setColor(VGA_GRAY);
219               myGLCD.fillRoundRect(posx+8, posy+24, posx+4+80, posy+24+50);
220               myGLCD.setColor(VGA_BLUE);
221               myGLCD.drawRoundRect(posx+8, posy+24, posx+4+80, posy+24+50);
222               myGLCD.setBackColor(VGA_GRAY);
223               myGLCD.setColor(VGA_BLACK);
224               myGLCD.print("PUMP", posx+8+8,posy+24+4);
225             }
226   myGLCD.setBackColor(VGA_GRAY);
227   myGLCD.setColor(VGA_BLACK);
228   if(tmpstatus.flags&0x80) sprintf(msg,"%dul/min",tmpstatus.pump_speed);
229         else sprintf(msg," STOPPED");
230   myGLCD.setFont(SmallFont);
231   myGLCD.print(msg, posx+8+4, posy+24+4+12+12);
232
233
234   draw_valve_state(posx+8+80+10, posy+24, 70, 50, 1, tmpstatus.valves & BR_V1_MASK, update_mode);
235   draw_valve_state(posx+8+80+10+75, posy+24, 70, 50, 2, tmpstatus.valves & BR_V2_MASK, update_mode);
236   draw_valve_state(posx+8+80+10+2*75, posy+24, 70, 50, 3, tmpstatus.valves & BR_V3_MASK, update_mode);
237   draw_valve_state(posx+8+80+10+3*75, posy+24, 70, 50, 4, tmpstatus.valves & BR_V4_MASK, update_mode);
238   }
```

Acknowledgment

I would like to thank to my advisor Prof. Dr. Andreas Schober for providing me a great opportunity to join the nano-biosystem technology research group. I am grateful for open and friendly research environment and for allowing me to make my contribution to this multidisciplinary research field.

Particular gratitude is expressed to Dr. Dana Brauer. Without her help and engagement in overcoming many biology challenges this work would not succeed.

Additionally, I would like to thank to Dr. Sukhdeep Singh for advising me in the field of chemistry and biochemistry and great suggestions during publishing of this work.

This work could not have been possible without the contribution from other members of the nano-biosystem technology group. I am very grateful to Jörg Hampl and Frank Weise for technical support with the bioreactors and fruitful discussions to solve technical challenges. Further, I would like to thank to Dr. Alexander Groß for performing the HPLC measurements. Particular thanks go to Maren Klett for her relentless help and assistance during cell culture experiments. I am very thankful to Dr. Adam Williamson for his precious advices and help during writing the thesis.

Special thanks to Dr. Michael Gebinoga and Maria Gebinoga for their unceasing support and encouragement for me and my family during our time in Ilmenau.

Last but not the least, I want to thank to my family. Especially to my wife Jana for her love and supportive home environment and also to my parents for their unwavering love and prayers.

My sincere gratitude to all of you for your support!