FAKULTÄT FÜR !NFORMATIK

A Platform for Teaching and Researching Distributed Real-Time Systems

Diplomarbeit

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von Alexander Kössler
Matrikelnummer 0325498

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuer:
Privatdoz. DI Dr.techn. Wilfried Elmenreich

Wien, 17. März 2009 _____ _____

(Unterschrift Verfasser) (Unterschrift Betreuer)

für meine Eltern

Acknowledgements

I would like to express my gratitude to my advisor, Wilfried Elmenreich, who guided me very well through this work. I especially want to thank him for the patience and faith he had in me during the last 2.5 years, and not only during office hours.

Special thanks applies to Christian Trödhandl for his all-time constructive comments and contributions to this work and to Bettina Weiss for proofreading.

So long, and thanks for all the LaTeX support to Martin Biely who made this thesis not only look good with regard to content, but also with regard to the style. It is impossible to weight his expertise in gold.

Thanks does also apply to my parents who in fact made all this possible.

I would also like to thank Johanna for her patience and her love during the last years. I know it was not always easy with me.

Finally, I would like to thank the following people (listed in alphabetical order) for their support and/or motivation during my studies: Andi, ECS staff, Emp, Gernot, RTS staff, the person who invented coffee, Thomas, Traude, and everyone else who should be listed here but I unfortunately forgot.

Alexander Kößler, in February 2009

Zusammenfassung

Heutzutage sind eingebettete Computersysteme allgegenwärtig. Aus diesem Grund steht das Programmieren von Mikrocontrollern und das Entwickeln von eingebetteten Computersystemen im Studienplan der Informatikstudien vieler Universitäten weltweit. Auch an der *Technischen Universität Wien* bietet das *Institut für Technische Informatik* mit Embedded Systems Engineering eine Lehrveranstaltung mit hohem Praxisanteil, in der Studenten fortgeschrittene Fähigkeiten im Programmieren von Mikrocontrollern erlangen sollen. Die erste Generation der dafür entwickelten Übungshardware stellte die Übungsleiter aber bald vor ein großes Problem hinsichtlich Wartung und Aktualisierung. Die verwendeten Mikrocontroller veralteten und die mechanische Stabilität des Laboraufbaus war nicht mehr gegeben. Das sind nur zwei der Gründe, warum nach einer Weiterentwicklung der Übungshardware verlangt wurde.

Diese Arbeit umfasst Entwurf und Entwicklung einer neuen universellen Plattform für die Lehrveranstaltung Embedded Systems Engineering (ESE). Nach eine Darstellung verschiedener Abhaltemethoden einer praktischen Lehrveranstaltung (LVA) über eingebettete Computersysteme werden diese bezüglich ihrer Nutzbarkeit in ESE diskutiert. Der Entwurfsprozess der neuen Hardware wird dokumentiert und ihre Besonderheiten, wie zum Beispiel die Unterstützung in praktischen Prüfungen oder die Möglichkeit für Fernarbeit, werden vorgestellt. Zum Schluss werden die Ergebnisse des praktischen Einsatzes der letzten drei Jahre mit ESE Studenten vorgestellt, eine Fallstudie, die mit freiwilligen Studenten in Priština gemacht wurde, evaluiert und eine Zusammenfassung über Forschungsthemen, die von der Plattform unterstützt wurden, gegeben.

Abstract

Nowadays, embedded systems are present in a very wide range of devices in everyone's life. Therefore, microcontroller programming is in the syllabus of computer engineering courses at many universities. Therefore, the *Department of Computer Engineering* at the *Vienna University of Technology* offers *Embedded Systems Engineering (ESE)*, a hands-on course teaching advanced microcontroller programming and distributed real-time communication between multiple microcontroller nodes. Students were offered several embedded systems development boards in a laboratory. Soon, course instuctors were faced with big problems concerning the maintenance and upgrade of the first generation hardware, which raised the desire for a more suitable replacement.

This thesis covers the design and development of a new versatile platform for the ESE course. Starting from a survey about different methodologies how such an embedded systems course can be held and their evaluation with respect to suitability for the ESE course, the thesis describes the design process of developing the new lab hardware with features for hands-on exams and remote access capabilities. Our experience over the last three years with ESE students, a case study of the remote access with voluntary students from Priština, and a summary of supported research topics from the department conclude the thesis.

Contents

List of Figures

1

Introduction

Calculators, microwave ovens, washing machines, automotive and aircraft controlls, they all have one thing in common: nowadays they contain embedded systems[Elm09]. Small microcontrollers, that handle all kinds of tasks starting from simple calculations up to fault-tolerant distributed control loops. Therefore, teaching the programming of embedded systems and advanced microcontroller programming are in the syllabus of computer engineering studies of many universities. The department of *Computer Engineering* at the *Vienna University of Technology* also offers two courses for this purpose. *Microcontroller programming* is an undergraduate course teaching students well-founded skills in programming microcontrollers using Assembler as low-level language and C as high-level language. Based on this, in the course *Embedded Systems Engineering (ESE)* students are taught advanced microcontroller and peripheral programming and communication among a set of distributed microcontrollers. This thesis describes different approaches how such courses can be structured and presents a platform which fulfills not only the needs of teaching ESE, but also the requirements of various research topics around microcontroller programming like interfacing a real-time sensor network or automated code generation.

1.1 Motivation and Objectives

In summer 2006, we decided to redesign the hardware used in the hands-on lab course *Embedded Systems Engineering*. The one used before was built up of many small printed circuit boards (PCB) mounted on a wooden baseplate and interconnected via a set of wires. It consisted of four microcontroller node PCBs with several add-on boards like a

seven-segment add-on or a bargraph add-on, connected via pin headers and several wire connections. However, the manner the hardware was designed led to many problems:

For example, because of the wiring it was much easier to reset a single microcontroller by unplugging the power supply to the node PCB than to reset it via a command line tool which controlled the programmer. Thus, with every unplug – plug cycle students caused the mounting to loose up more, which ultimately led to microcontroller nodes which were detached from the baseplate. Furthermore, the mechanical loosening of the nodes caused connection problems at the sockets whereon the communication between the microcontroller nodes got unreliable and the connection to the peripheral devices like displays or LEDs failed many times.

Another problem of the old hardware came from the way the hardware was designed. Since it consisted of several small boards, schematics were available for those parts only, but not for the overall system. Therefore, the understanding of the system as a whole got very difficult and it turned out that several students who finished the course had little or no knowledge about the connections between the parts.

Furthermore, the old hardware used two different microcontroller models with different pinout and different oscillator frequencies. This made it very hard for the students to write generic code for them, because every timing had to be calculated depending on the microcontroller node for which it was compiled and also the pinout had to be conditioned to the target node. The worst problem occurred when students had to write generic drivers for integrated hardware like timers or analog to digital converters. Here it happened that depending on the target platform different registers had to be initialized with different values making it very time consuming and error-prone to write generic source code.

Programming of the nodes was done by an SPI programmer developed in-house, which had the drawback that for every new microcontroller type that should be programmed, the firmware had to be adapted to identify the new type and to allow its programming. The main problem was the lack of documentation, which made it very hard to add new devices to the ones supported by the programmer after the developer left the department. Additionally, by supporting only SPI as programming mode, the hardware could not provide any JTAG debugging features, as for example step-by-step debugging or setting breakpoints.

Because of the previously mentioned programmer's support problem of new devices, the used microcontroller devices were outdated and hence their hardware features like on-board Random Access Memory (RAM) or Electrically Erasable Programmable Read Only Memory (EEPROM), or the number of timers, were minimal. This led to restrictions in the complexity of tasks the students could be assigned.

For these reasons, we decided to design a new hardware which should avoid all these problems. This new hardware should not only replace the old one in the hands-on part of

the course, but it should also allow a later expansion of the course to include an additional remote working alternative. Therefore, we defined requirements the system has to fulfill and worked out a concept for a platform that satisfies our needs.

Apart from these practical reasons, this new approach also allows us to observe the effects of different systems on students and staff. From our point of view, it was interesting to observe how different system architectures affect the students' understanding of the system. As mentioned, the old hardware consisted of several PCBs, so although students sometimes did not realize how they worked together, they could at least identify the components easily. In our new approach, all components are on one PCB. While we expected this architecture to improve students' understanding of the interactions between the components, we were not certain if it would not overwhelm beginners and thus lead to a decrease in overall student understanding of the system.

Another aspect we were interested in was the effect and acceptance of the remote workplace option. This allows providing students a 24/7 work environment, which we considered particularly attractive to students working in a job during the regular lab opening hours, but we did not know how much actual demand there was.

From the point of view of the staff, we were of course interested in creating a system which is easy to handle and maintain, and which also leads to a reduction of the workload of the staff.

1.2 Content and Structure of the Thesis

This thesis documents the design of the new lab platform and its validation for different purposes. The author of this thesis designed the PCBs of the lab hardware, organized its production and supported assistant professors and students as a Tutor in the Embedded Systems Engineering Lab course for 3 years during which this platform was used for programming excercises and evaluations. He also set up an experiment for remote lab access between Vienna, Austria and Priština, Kosovo. Moreover, the author acted as system integrator by extending the lab platform with the results from several bachelor's thesis (eg., remote access, debugging, etc.).

The thesis is structured as follows: Chapter 2 gives an overview about our requirements for a platform used both for teaching ESE and for research on distributed real-time systems. In Chapter 3, we describe the possibilities for teaching state-of-the-art embedded real-time systems and we discuss the advantages and disadvantages of hands-on labs, hardware simulation, take-home lab kits, and remote lab access. Our developed solution is introduced and described in detail in Chapter 4. In Chapter 5 we describe four showcases our platform should master and the results we got after a mid-term evaluation. Finally,

Chapter 6 summarizes the key results of the presented work and gives an outlook on open questions, possible improvements, and further research topics.

2

Requirements

The goal of this work was to develop a versatile platform that can be used for teaching students advanced microcontroller programming in the course Embedded Systems Engineering at the Vienna University of Technology. Additionally, this platform should also be usable in the research of distributed real-time systems at the department of Computer Engineering. Before we started working on the development, we identified 11 requirements for our platform. These include hardware requirements like programming without replugging cables as well as more general requirements like production costs or robustness which are described more in detail in the following.

2.1 State-of-the-Art Hardware

One of the reasons looking for a new platform for our lab course was because in the old one, the used microcontroller units (MCU) were outdated, leading to memory problems when writing larger embedded applications. Thus, state-of-the-art MCUs should allow to implement interesting tasks without the fear of getting a stack overflow when using more than five nested function calls. The platform should also provide various high level interfaces such as SPI and a versatile bus system, as well as the standard digital and analog interfaces. Time-critical interfaces like Input Capture (IC) and Pulse-Width Modulation (PWM) should be available too, allowing much more interesting tasks and more versatile usage.

2.2 Teaching Distributed Systems

In order to teach distributed embedded programming, we need multiple MCUs on the platform, which can interact with different peripherals and with each other. The old hardware platform featured four MCUs, and in our opinion that is the bare minimum one needs to provide interesting and challenging distributed tasks to the students. Also, all the peripherals have to be split among the processing nodes, for example to allow the implementation of a distributed fan control. We furthermore want to program systems with a direct interface to the hardware. Thus, embedded Linux systems or similar are out of question. We want the students to program the hardware directly and to meet the challenge of limited processing power and especially limited memory.
When teaching distributed systems in which the processing nodes are communicating via message passing, one thing is inevitable: a bus system between the nodes. This bus system should not limit the possibilities, i.e., it should not only support a single protocol, but should also provide enough flexibility to implement different protocols, for example protocols similar to Controller Area Network (CAN) or Time Triggered Protocol Class A (TTP/A) [Kop02]. Also, an ordinary serial communication between two nodes should be possible to implement very easily. Another requirement to the bus system would be to use a simple bus driver, so students do not need to read a bunch of manuals to set up a simple peer to peer communication between two nodes.

2.3 Teaching Real-Time Systems

When teaching distributed real-time systems hands-on, the used hardware has to fulfill some special needs. In this regard, the bus system[1] is one of the most important things. If the bus system does not provide real-time capabilities, one cannot implement a distributed real-time application on top of it. Therefore, and because one of the tasks in our course is the implementation of a real-time protocol, one requirement to our bus system is the possibility of real-time communication. The additional requirement that the hardware has to run our chosen real-time field bus protocol TTP/A is not really hard to fulfill, since its architectural requirements [Trö02] are rather low. To make the real-time challenges aware to the students, the system should allow some example real-time applications like implementing a control loop for the fan speed.

2.4 Flexibility

Since the platform should be used in research too, the new hardware should be flexible in every possible way. If more processing power is needed, it should be easily possible to

[1] The term "bus system" here includes the hardware bus and the line drivers used for accessing it.

change the used MCUs against ones with more processing power. Needs for external RAM or a hardware real-time clock should be as easy to fulfill as the change of the manufacturer of the used microcontroller unit (MCU).

In the scope of research it is often necessary to work with special sensors or actuators. Without much effort, it should be possible to connect additional nodes to the platform which deal with the special hardware and, in terms of a smart transducer [EHK01, OMG03], provide clean interfaces to the processing cluster. Therefore, additional sensors like infrared or acceleration sensors should be easy to attach, and extensions like an Ethernet port, or a gateway to higher level real-time protocols, e.g. TTP/C [TTA03], should be simple to realize.

To implement more complex distributed tasks or algorithms without having to attach a lot of external nodes, it should be easily possible to connect multiple platforms to form a large cluster of nodes.

2.5 Robustness/Maintenance

The "life" in an embedded systems lab is very rough – especially for the lab hardware. Every day dozens of students work on it, sometimes eat and drink next to it, and do not always handle the lab hardware with care. A lab hardware might also have to suffer lots of aggressions if the student's solution does not work correctly. This all implies the requirement for an enhanced physical robustness. One of the reasons looking for a new platform for our lab course was because the old one was not integrated on a single circuit board, and therefore loose cables from countless unplug – plug cycles by students led to regular connection problems and loosened mountings on the wooden ground plate. Because of this, the old hardware was very sensitive to touch and movement.

Clearly the new hardware should not have these drawbacks. Spilled out water should not immediately lead to total destruction of the hardware, and it should survive moderate shock without serious damage. Another issue is theft; in case of a platform which consists of multiple parts, ideally every single part has to be protected against theft.

Apart from these issues, long service intervals and easy replacement of defect components is also part of the platform's robustness. MCUs have a limited amount of programming cycles; if the limit has been reached, it should be easy and cheap to replace them.

2.6 Lab Working Places

For the lab course ESE, we need at least 16 fully equipped lab working places during the course. In addition to this, it should be possible to extend the lab working places by remote working places. The physical size of a lab workstation is of interest too, because

the smaller one lab working place is, the more can be arranged in a lab of the same size. Therefore, the lab places should be kept simple and compact, which means we prefer a single monolithic design instead of a system with parts spread out and tethered by a lot of bus and power cables.

2.7 Cost

Of course, the costs for new educational hardware should be kept as low as possible. The cheaper one lab place is, the more workstations can be made available to the students. The cost of the development platform can be divided in hardware costs and software costs. Software costs are dominated by licensing costs, which can be held low by using free and open source software. We further reckon self-designed hardware whose cost can be decomposed into the part costs, which can be minimized by using cheap standard industry parts, and assembly costs, which can be minimized by using as much surface mounted technology as possible, allowing automated mounting.

2.8 Development Software Support

The used software should be as much standard software as possible, because software written by students or by department staff has one main drawback: the programmer sooner or later is out of reach, which results in a big maintenance problem. Therefore, in deference to maintainability and cost factor, the used software should be established open source software which is well maintained by the community. When taking into account our lab system, the software has to run on Linux, because our lab environment is completely Linux based. In the ideal case, the complete tool chain starting from operating system to editor, compiler, programmer, and also the debugger should run on Linux and should be free of charge. For reasons of flexibility, however, the software should support other platforms too, at least Cygwin[2] in a Windows environment.

2.9 Programming without Replugging Cables

As described in Section 2.3, we want to teach distributed systems, which means that we have multiple MCUs that should be programmed. Because we also want to enable the possibility of remote working, the programming solution must not require the replugging of any programming cable.

[2]Cygwin is a Linux-like environment for Microsoft Windows. It allows the integration of Windows system ressources with applications and data of the Linux-like environment by introducing an Linux Application Programming Interface (API) emulation layer. Cygwin is available at `http://www.cygwin.com/`.

2.10 Support for Hands-on Exams

On the one hand, examinations where students have to complete a task on a live microcontroller system are a good way to check the student's ability to use their knowledge in a practical way. Both, teachers and students have often stated the desire for such real-world examinations. However, after implementing such examinations in the first place, we found out that they often have been a source of stress and hectic rush for all involved persons, especially at the end when many students want feedback for their programmed solutions concurrently. The new platform should therefore mitigate these problems by providing automated feedback to the students during the examination, such that they are informed whether their solution fulfills the specification or not. Therefore, the platform should be capable of evaluating the students' program output in the value and in the time domain.

If this evaluation is also the basis for grading the result, it is necessary to ensure that the student cannot fake a correct result. Therefore, the exam evaluation should provide a sufficient level of security.

2.11 Remote Working

When talking about remote working, this includes not only remote programming, but also remote monitoring and debugging, which is by far more difficult, but should reduce the lab load and enhance flexibility of the working times of the students. Especially for students living far away from the university, or for students with jobs besides their study, it is hard to come and work in the lab during regular opening hours. Therefore, more flexible working times could help them finish their studies in time. The possibility of remote working also provides a scalable extension to lab places.

3

State-of-the-Art in Teaching Embedded Systems Courses

This chapter deals with the different options how undergraduate and graduate courses on embedded systems programming can be conducted. Such courses can be categorized in four parts. In Section 3.1 we describe how a course can be held using a hands-on lab for solving exercises. Section 3.2 shows the pros and cons of using simulated hardware in an embedded systems course. In Section 3.3 the option of using take home hardware is discussed, and Section 3.4 describes the advantages and disadvantages of using a remote access to the lab. Every section comes with references to courses using the introduced models. Of course a combination of different models is possible and most of the referenced courses in fact do use at least two of the options together.

There are a lot of papers dealing with the question of which approach is better suited [NMN03, WHL05, CNEC04, CF97], but in our opinion, this heavily depends on the requirements of the course. In many cases a combination of several approaches is the most suitable.

3.1 Hands-on Lab

Using a hands-on lab [Bac05] has many advantages. First, students get in touch with real hardware which allows them to "get together" with real world problems like ground bounces, accidental time effects, or race conditions of which, in our opinion, students of computer engineering should be aware. Another advantage of hands-on labs is the accessibility of wiring, and the availability of helpful measurement equipment like oscilloscopes

or logic analyzers. To reduce hardware costs, the measurement equipment can also be shared among the lab-places. When thinking of the used software, hands-on labs clearly have the advantage that it can be maintained centrally and held consistently. Every workstation runs the same operation system, and the same programs are installed. No things like "... but the programmer does not work on my Mac ..." can occur. Another big advantage is the available support through tutors on site. No email, forum post, or similar communication is needed to ask questions.

Of course there are also disadvantages of hands-on labs, for example most labs have time restrictions when students may get access to them. Additionally, it is very hard and time consuming for students who do not live in the city of the university to take part in such a course. Another well-known fact is that in the days before exercise deadlines the amount of students working in the lab is noticeably increasing. The lab has therefore either to be dimensioned to these peak-load situations, leading to a bad average usage, or one has to schedule the access in peak-load times by issuing restrictions on particular time slots.

3.2 Hardware Simulation

The main advantages of hardware simulation are that students can work at home and therefore are completely free to decide when and where they want to solve their exercises. There is no physical lab needed, which reduces the cost needed for rooms and equipment and therefore it does not make a big difference whether 10, or 100 students attends a course. Another big advantage of hardware simulation is that students get a better insight into their program, because of the very good debugging possibilities, for example step-by-step execution. The students have full access to the internal state of the processor and can inspect all registers and memory contents at any time.

Unfortunately, most good simulation solutions come with very high licensing costs, although there are also some which are free of charge, for example *Scilab*[1]. However, simulation can never be as exact as real hardware since the accuracy of processed values is bounded. Another problem comes with different hardware of the students, since the performance of the used simulator heavily depends on the student's processor speed. Also things like the used operating system or special hardware drivers have to be taken into account.

Although simulation is way different from hands-on labs, it can help students to find errors that could not be found that easily without simulation. For example, [BPM00] are using simulation in addition to hands-on labs. The students first solve their tasks and simulate their programs in a simulator. Afterwards, the students download their programs

[1] Available at `www.scilab.org`.

to hardware. One problem was found by the students when they attempted to locate an external device at address 0x7FFF. Switching the address lines from all 1's to mostly 0's created a ground bounce which caused a disturbance at the address-latch enable line. This ground bounce problem came from the used microcontroller and was corrected in the next revision, but without previous simulation the students would have spent much more time for localization of the problem.

3.3 Take-Home Lab Kits

The main advantage of take home hardware, also called lab kits, is that students can work on real hardware not only in the lab, but also at home. Such lab kits have several requirements to meet: First, they must not include expensive software licenses, or even better should use open source software only to avoid any licensing problems. Any software registration/licensing process is time-consuming and may cause trouble if not done correctly. Second, a lab kit has to be easy to install for students and must not require extensive knowledge of operating system functionality. For example, it must not happen that certain registry tweaks are necessary to run the lab kit on a student's PC. Third, the lab kit should be independent of the student's choice of operating system or the student's PC hardware. Therefore, it should be no problem whether a student uses a Windows or Linux PC, or even a Mac, and in the best case even how old the PC might be.

[ETW06] achieve the requirements by using a Linux system based on Knoppix [Kno00], which is included on a bootable CD and does not need any configuration. The complete tool chain is pre-installed and only consists of open source software. One of the major disadvantages using lab kits is, however, that no expensive hardware can be handed out to students, for example no storage oscilloscopes or logic analyzers. Whenever expensive hardware is needed, it is better to leave it in the lab, where at least a minimum of supervision can be ensured.

3.4 Remote Lab Access

During the last years, remote lab courses got quite common. There are countless examples of such courses in literature [MLM03, TPA06, NMN03, CHMM04, PP00, MČŽ00, GCARVA+01, TPE06]. Remote labs have been seen as the ideal addition to hands-on labs. They are reducing the lab load by providing remote access to lab hardware and to the lab equipment. Additionally, they are removing time and space constraints for accessing the lab by providing 24/7 access to it while the students are still working on real hardware. There are several ways how the communication between the working student and the lab environment can be accomplished. When the feedback is given as a live stream of a video camera like in [MLM03], there is no other possibility to interact with the hardware than

programming it and running the programmed code. Additionally, a student may view the screen of an oscilloscope, but is not able to change its configuration, which however sometimes is not a drawback but an advantage. Another possibility of giving feedback to the students is by monitoring the interesting data, like in [MČŽ00] or [TPE06]. Here, monitoring hardware is used to collect all interesting information about the development hardware and all physical process data. The data is sent to the students and visualized. By adding additional hardware for dynamic configuration like in [GPPD05] to change the hardware configuration, students can also interact with the lab hardware and experiment as they would in the lab. This solution, however, has the drawback that it is quite expensive, not really space efficient, and usually there is no way to integrate the control of lab equipment like a logic analyzer. Remote lab access has the drawback that every student needs to have access to the Internet with a reasonable bandwidth. This might not be a problem in our case, but may be one in other regions.

4

System Architecture

The survey on requirements in Chapter 2 combined with the different ways a course on embedded systems programming in Chapter 3, turned out a combination of hands-on lab and remote lab access as ideal for Embedded Systems Engineering (ESE). A summary of the facts can be found in Table 4.1. Additionally, it showed that there exists no

requirement	hands-on lab	hardware simulation	take home lab kits	remote lab access
state-of-the-art hardware	✓			✓
teaching distributed systems	✓	✓		✓
teaching real-time systems	✓	✓		✓
flexibility	✓	✓	✓	✓
robustness	✓			✓
lab working places	✓	✓	✓	✓
cost	✓			✓
software support	✓	✓		✓
replugging cables	✓	✓	✓	✓
hands-on exams	✓		✓	
remote working		✓	✓	✓

Table 4.1. Summary of requirements vs. course types

commercial-off-the-shelf system that would fulfill all our needs. Therefore, we decided to create our own lab platform. After research of the requirements, we developed the concept of our *ESE–Environment* containing the *ESE–Board* as development platform

and providing the *ESE–Toolchain*. This chapter will introduce the system architecture of the ESE–Board and will give information about how we dealt with the upcoming problems.

First the design of the basic board, which is the version of the ESE–Board that is currently used in our labs, is introduced in Section 4.1. To allow remote monitoring some special extensions have been made to the basic board. They are described in Section 4.2 together with an update of the software requirements. In Section 4.3 we describe how our platform is able to evaluate hands-on exams automatically, and in Section 4.4 we introduce our remote workplace approach, which is based on the monitoring extensions from before.

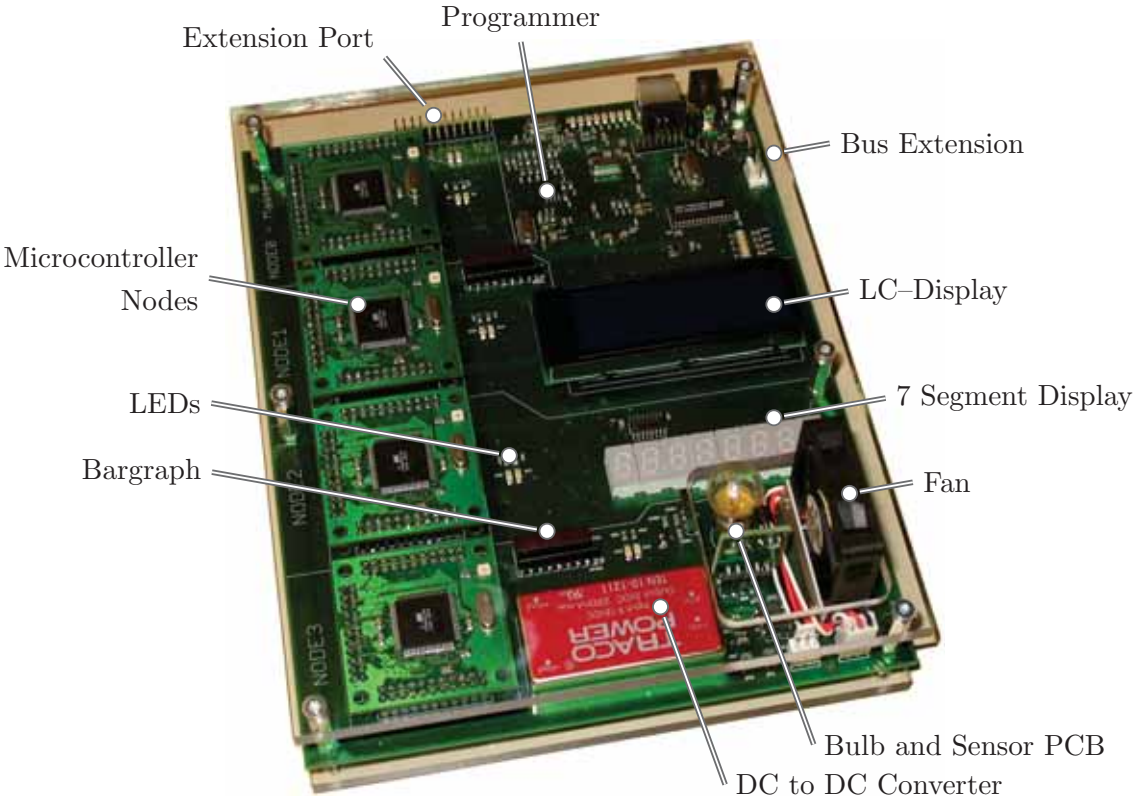


Figure 4.1. The ESE–Board

The platform consists of four nodes interconnected by a serial bus-system. As shown in Figure 4.2, every node has different peripherals connected to it. Node 0 has 2 LEDs for debugging, a Universal Asynchronous Receiver Transmitter (UART) for transmitting data to the connected PC, and some monitoring inputs for observing some chosen signals of other nodes. The monitoring inputs are for evaluating exam tasks and will be explained in detail in Chapter 4.3. Node 1 also features 2 LEDs for debugging and in addition an 8-Bit bargraph display for debugging and simple visualization of values. Furthermore, an LC–Display is connected via Serial Peripheral Interface (SPI), and the display's contrast, the brightness of the background light, and an additional bulb can be adjusted via PWM.

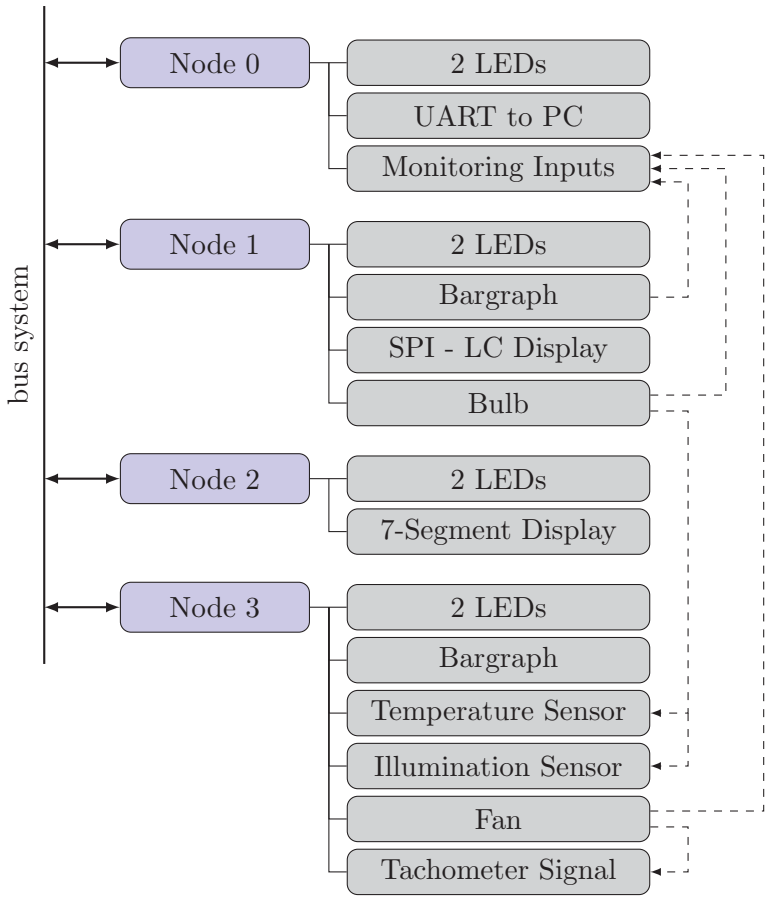


Figure 4.2. Basic concept of the peripherals

Node 2 also has 2 LEDs and a 7-segment display connected. Node 3 is the sensor node and has 2 LEDs, a bargraph display, a temperature sensor, an illumination sensor, and a fan connected to it. The fan generates an index signal, which can be read back from the node to determine its current speed. The currently used programming mode can be seen in Figure 4.3. The overall system is called "ESE–Board" and is shown in Figure 4.1.

4.1 Basic Board

The basic board is used in the lab as hands-on hardware only. It does not support remote monitoring, but it provides all the functionality that is needed by the students to solve their tasks directly in the lab. Of course, it supports the automatic exam evaluation and thus is perfectly suited for every task inside the lab.

4.1.1 Hardware

The concept of the hardware peripherals is shown in Figure 4.2. The solid lines represent hardware connections, the dashed lines can be seen as external dependencies (for example, the bulb influences the temperature and the illumination sensor). The four nodes can be programmed by a single programmer, while the programming of multiple nodes is possible due to a multiplexer[1]. The different programming possibilities are presented in the following paragraphs.

Programmer

During the work on this thesis, several programming concepts reaching from a simple serial boot loader to a generic Extensible Markup Language (XML) programmer were implemented. Their advantages, disadvantages, and special features are discussed in the following:

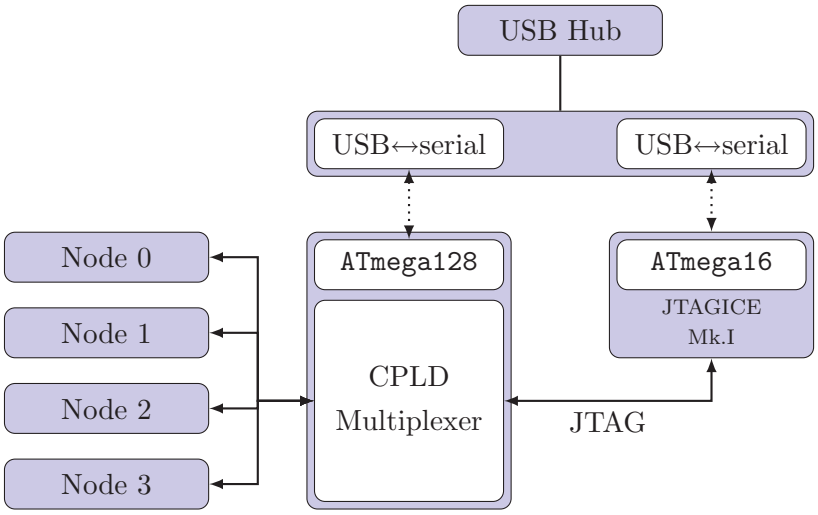


Figure 4.3. Programming concept using JTAGZeusProg

JTAGZeusProg The JTAGZeusProg was developed together with the ESE–Board and is the programmer originally designed for the board. The concept of the JTAGZeusProg can be seen in Figure 4.3. The programmer consists of a XILINX Complex Programmable Logic Device (CPLD) device containing the switch circuitry for switching the JTAG signals between up to 8 microcontroller nodes. An ATMEL `ATmega128` microcontroller is controlling the CPLD declaring which device has to be programmed. The programming itself

[1] JTAG Daisy Chain programming is not used, because it is not supported by every version of the microcontroller nodes. For example, previous versions of the ATMEL `ATmega128` microcontroller do not support chaining.

is done by a Joint Test Action Group (JTAG) In–Circuit–Emulator clone running on an Atmel `ATmega16`. Both the `ATmega128` and the `ATmega16` are connected to the host PC via a 2-channel FTDI *FT*2232 Universal Serial Bus (USB) to serial converter. Via JTAG, the programmer supports debugging via GDB such as single step execution, breakpoints, and memory evaluation. In addition to programming and debugging, this programmer is able to emulate virtual push buttons which can be accessed by the user via a command line tool. This feature was implemented to eliminate the need for mechanical components, which cannot be operated if one is accessing the board via Internet from home. The virtual push buttons can be 'pressed' by software and be released automatically after 10ms like a "virtual button", or can be held and released manually like a "virtual switch". The virtual push buttons are connected to an external interrupt input of the microcontroller nodes to allow detection of button presses and releases without polling.

Since the used programming device is pretty outdated, it does not support newer devices like the Atmel `ATmega1280` family, or the `2560` family, which led to problems during the work on the remote monitoring part. Another drawback of this programmer is the limitation to Atmel microcontroller devices.

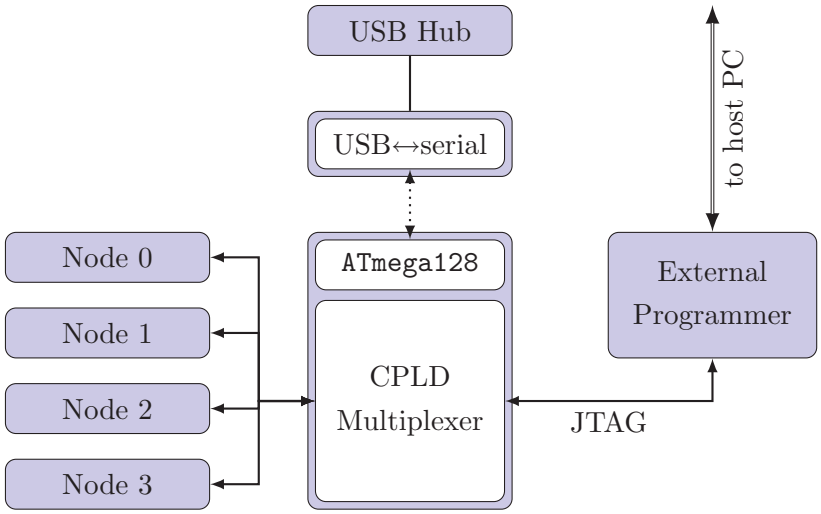


Figure 4.4. Programming concept with external programmer

External JTAG Programmer Figure 4.4 shows the concept how an external JTAG programmer can easily be included in the toolchain. This is of interest if the external programmer provides special functions which are needed. Switching between the microcontroller nodes can still be done by the CPLD or, if switching is not wanted, the CPLD can be reprogrammed to connect the microcontroller nodes to a Daisy–Chain.

Bootloader When working on the remote monitoring part of the platform, the first problems with the JTAGZeusProg, as described above, came up. ATMEL `ATmega1280` were chosen as monitoring nodes, but they were not supported by the JTAGICE Mk.I clone. Therefore, M. Hofer and T. Mair configured a boot loader to work with those devices and enable programming them [HM07]. The bootloader programming mode, however, is only interesting for the monitor nodes because they have an UART connected to the development system.

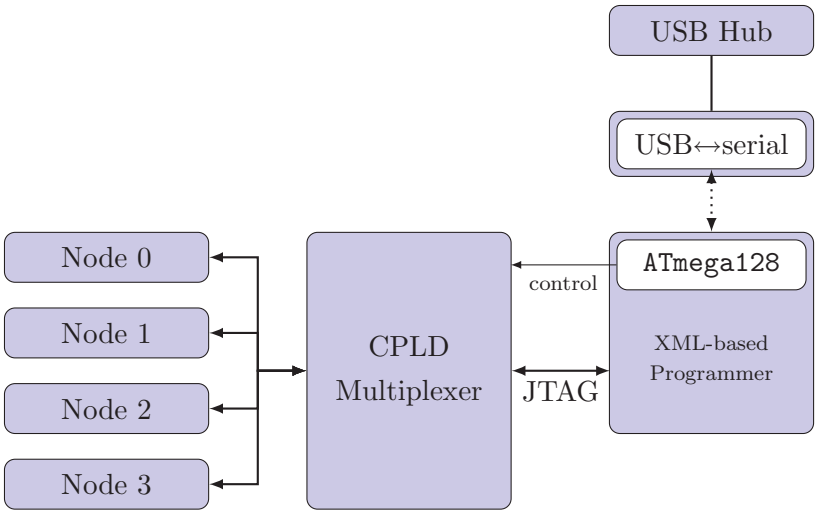


Figure 4.5. Programming concept using the XML-based programmer

Modular, XML-Based JTAG Programmer for Embedded Devices The modular XML-based JTAG programmer is the last development related to the ESE–Toolchain. It was done as bachelor thesis by M. Schmölzer [Sch08] and enables JTAG programming for various devices. The key feature is the XML configurable programming algorithm which allows adding support for new devices without recompiling the programmer or its firmware. The programmer is designed to work for all currently available microcontrollers with a JTAG programming interface, as well as other devices like FPGAs or memories. Currently, the programmer does not support debugging via the JTAG interface, but interfaces are designed to easily integrate such features in the future. By now the programmer is tested on development boards and it is expected to be used in the next course on all ESE–Boards.

Microcontroller Nodes

Since flexibility is one of our main requirements, we do not focus on a single microcontroller type or model. Instead we have chosen to place every microcontroller node on its own Printed Circuit Board (PCB) connected to the ESE–Board by a socket with a

defined hardware interface. This makes it very easy to replace the microcontrollers, for instance if the currently used devices are outdated, more processing power or a design with external RAM is needed by the application, or simply the limit of programming cycles of a microcontroller has been reached. Therefore, the pinout of the socket is also not matched to a particular microcontroller but has been structured functionally (see Figure 4.6) and only describes the principal functional groups of the pins. The disadvantage is that this functional interface is leading to misunderstandings by the students, because often they are not used to reading more than one schematic, which is required here to find out the respective microcontroller pin that is used to enable a particular Light Emitting Diode (LED). Experience has shown the need to emphatically explain more than one time how the mapping between the functional interface and the real microcontroller pins can be found out.

Designing a new CPU–Board requires that the functional pin out is not violated. A detailed pin out diagram which shows all the groups in detail can be found in the appendix in Section C.2.

In our lab hardware we are currently using ATMEL `ATmega128` devices for all four microcontroller nodes, because they provide all functionality that is needed to perform our tasks and they are very easy to program. Furthermore, there exists a very good toolchain for those devices which is available completely as open source software.

Peripherals

LEDs Each CPU board has a green and a red LED for status indication or very simple debugging (see Figure 4.7). The LEDs are driven active-low and are connected consistently

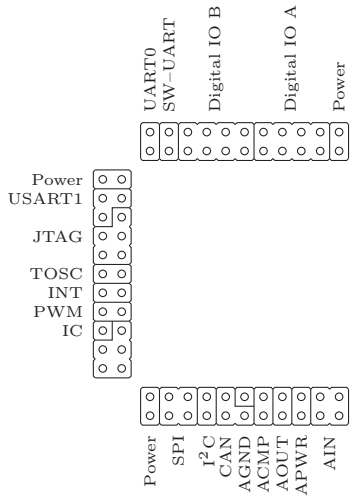


Figure 4.6. Functional layout of the socket interface

to the same ports through all microcontroller nodes by default. Therefore, it is very easy to re-use the code written for a specific node on another node even without changing the IO settings for the LEDs. This is especially advantageous when one is programming the core of a communication protocol running on all nodes and using the LEDs for status indication like used in TTP/A.

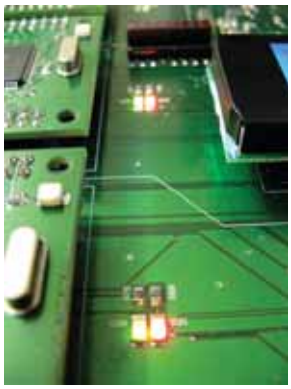


Figure 4.7. LEDs on the ESE–Board

Bargraph The bargraphs are connected to 8-bit IO ports which allows various tasks other than debugging, for example software PWM tasks like a "Knight Rider" light as well as counter visualization tasks can be performed to provide variantly and challenging tasks for students to solve.

There are bargraphs connected to node 1 and node 3. While the bargraph at node 1 is connected on all pins of the digital IO port, the one at node 3 has pin 4 (count started with 0) disconnected. This is because with our choice of microcontroller nodes, this pin would be shared with the input capture pin of the fan index signal resulting in a flickering led whenever the fan is running. However, this is the only time the chosen microcontroller concerns the board layout. All other peripherals are not affected by such things and the functional interface is perfectly preserved.

Figure 4.8. Bargraph connected to node 1

Light Bulb The light bulb connected to node 1 is driven active high and can be used to influence the illumination and temperature sensors connected to node 3. It can be used as actuator in a control circuit, for example when controlling the temperature. It is not connected directly to the microcontroller, but driven by a transistor amplifier with two stages using a $BD140$ power transistor as output stage. The bulb however only requires $5V$ and therefore is connected to the $5V$ power supply of the board. It is recommended to use $5V$ bulbs, since a $9V$ or $12V$ bulb hav shown not produce enough light and heat. Figure 4.9 shows the bulb with the sensor PCB containing an illumination and a temperature sensor side by side.

Figure 4.9. Light bulb with sensor PCB

LC–Display The LC–display (shown in Figure 4.10) is connected to node 1 and has various interfaces. The background illumination and the contrast of the display are controlled via PWM, which allows the use of different displays without the need of recalibrating the hardware such as the contrast voltage. This is especially important since the manufacturer of the used display provides several different types of displays with different sizes and they are all pin compatible. Therefore, it is possible to change the display to a smaller or larger one, or even a graphic display which is also available with the same pin out. This enables swapping to graphic displays in the future.

The display itself is controlled via SPI and connected to the hardware SPI controller of the microcontroller node. There are many implementation modes of the display possible, such as using/not using a frame buffer inside the microcontroller, or using hardware/software SPI.

Since the Liquid Crystal Display (LCD) is certainly one of the most complex peripheral devices on the ESE–Board, it is a good idea to provide additional information to the students. One big problem of the display programming that comes with the advantage of replaceable displays is that the contrast has to be set up by the students. It happened that students lost days configuring their SPI controller, while the controller worked fine all the time, but the contrast was not configured. So the display did show everything

Figure 4.10. LC–Display

correctly, but the students could not see it. The hint that the contrast should be in the range of 30% is very useful and will allow students to focus on the real problems, although it is sure that those students had learned their lesson and will never make such a mistake again.

Programming the SPI communication is another challenge, no matter whether the hardware controller is used or not. If the display does not show anything, it is possible that either the SPI communication does not work, or the display is not configured right. Therefore, it is a good idea to provide a minimal set of instructions that should bring a blinking cursor on the display if they were transmitted to the display correctly. Only after getting this minimal example right, students should proceed to program more complex display tasks; if they directly start with their own tasks, the debugging will cost a lot more time if something went wrong.

Seven–Segment Display The seven–segment display connected to node 2 is a great exercise for multiplexing since it consists of eight seven-segment digits connected to a digital IO port of the microcontroller where the segments of the digits are connected in parallel. The selection which digit is active is done by using a transistor for switching the common anode of the active digit to VCC while the common anodes of the inactive digits are left unconnected. The transistors of the digits are controlled by a $74LS138$ 1-of-8 binary demultiplexer which uses three inputs and demultiplexes them to eight lines.

Using all digits together requires multiplexing and switching between them fast enough. Figure 4.11 shows the string "`HELLO 42`" written to the seven–segment display. Various exercises concerning different refresh rates and multiplexing frequencies are possible.

Temperature Sensor As one can see in Figure 4.9, the temperature sensor is not mounted directly on the ESE–Board but on a small extension PCB together with the illumination sensor. This sensor–PCB is plugged into a socket board connector on the main board to be easily replaceable in case of needing a different sensor. The currently used

Figure 4.11. Seven–segment display connected to node 2

temperature sensor is a temperature–dependent Zener diode ($LM335$) with an accuracy of 1°C without any calibration. The output of this temperature sensor is directly proportional to the absolute temperature at $10\frac{mV}{K}$, which allows direct measurement in °C when using the ADC's differential inputs with $2,7316V$ on the negative line, corresponding to the offset of 273,16°C, and the temperature sensor on the positive one.

Illumination Sensor The illumination sensor is, like the temperature sensor, mounted on the sensor–PCB. Currently a Light Dependend Resistor (LDR) with a resistance range between 300Ω and 20kΩ is used. The illumination sensor can be used in exercises to detect if the bulb is switched on or off, triggering the start of something else. For example, students have to measure the temperature progress after switching on the bulb without direct communication between the sensor node and the node controlling the bulb.

Fan The fan used on the ESE–Board is a $12V$ brushless fan (shown in Figure 4.12), driven by a non-inverting operational amplifier circuit, which amplifies the low-pass filtered PWM signal from node 3 by a factor of 2, resulting in a maximum supply voltage of $10V$. The index signal of the fan is fed back to the microcontroller node's Input Capture pin, where it can be used to determine the real speed of the fan. The fan can be used in exercises for cooling the bulb and the resulting temperature variation can be measured by the temperature sensor. It may also be used to build up a closed control loop to control its speed.

As the operational amplifier chip provides two gates, a second PWM–to–analog conversion is done and connected to a second, currently unused, connector. This output can be used for external peripherals needing analog voltage.

1–Wire Bus The 1–wire bus, connecting the four microcontroller nodes the students may program, is a simple serial bus using a standard TTL tri–state–buffer as line driver. This means that every node that wants to write 0 on the bus may do this, all other buffers are high impedance. Therefore 0 is the dominant state of the bus and 1 is the recessive

Figure 4.12. Fan connected to node 3

one. There are several protocols that may be used on the board reaching from simple *RS*232 point to point communication over a CAN-like multinode communication protocol with bitwise bus arbitration in the header to time-triggered protocols like TTP/A.

If more than four processing nodes are needed, the bus can easily be expanded by the bus extension interface, shown in Figure 4.13, which allows not only to connect several ESE–Boards together, but since there is also a power supply available on the interface, to easily add additional stand alone nodes as well. This is especially of interest if gateway nodes to other communication protocols are needed, e.g. gateways to an Ethernet network or gateways to a Time Triggered Protocol Class C (TTP/C) cluster, or nodes with special hardware are needed, e.g. nodes with built in acceleration sensor, pressure sensor, or infrared sensors.

Figure 4.13. 1–wire bus extension connector

UART The UART connected to node 0 is the main communication interface between the ESE–Board and the host system. Students should send measurement and debugging

data to the lab PCs and should receive data for processing. TTP/A uses this interface for its monitoring software. Communication is provided by a Future Technology Devices International (FTDI) USB–to–serial converter FTDI $FT232BM$ which allows transmission signaling by two LEDs shown in Figure 4.14. All USB ports of the ESE–Board are collected by a $TUSB2046B$ USB hub and transmitted to the host PC via a single USB connection.

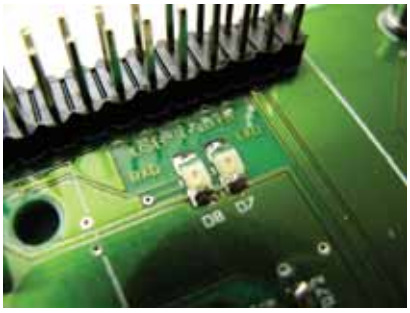


Figure 4.14. UART traffic LEDs of node 0

4.1.2 Software

The basic board needs the current avr toolchain containing the avr-gcc cross-compiler, and the avr-libc libraries. The avr-binutils linker is needed to generate the target binary. Those tools are provided by various Linux distributions as binaries in the package management. The software needed for programming depends on the chosen programmer. Using the standard on-board JTAGZeusProg, *avarice* is needed for programming and the JTAGZeusProg client executable is needed for switching between the microcontroller nodes. If the newer XML programmer is used, avarice is not needed, but the JTAGProg executable and the device description files instead.

4.2 Monitoring Extension

This section deals with the hardware and software changes that are necessary to enable monitoring on an ESE–Board. It does not deal with the real monitoring feature, which is described in Section 4.4.

4.2.1 Hardware

The principle of the monitoring hardware is shown in Figure 4.15. As one can see, it is quite similar to the programming concept shown before in Figure 4.3, in fact only the four monitoring nodes, called A–D, and their serial interfaces to the host system were added.

The monitoring nodes are placed in between the ESE–Board and the microcontroller nodes like in a sandwich shown in Figure 4.16. This configuration allows monitoring all

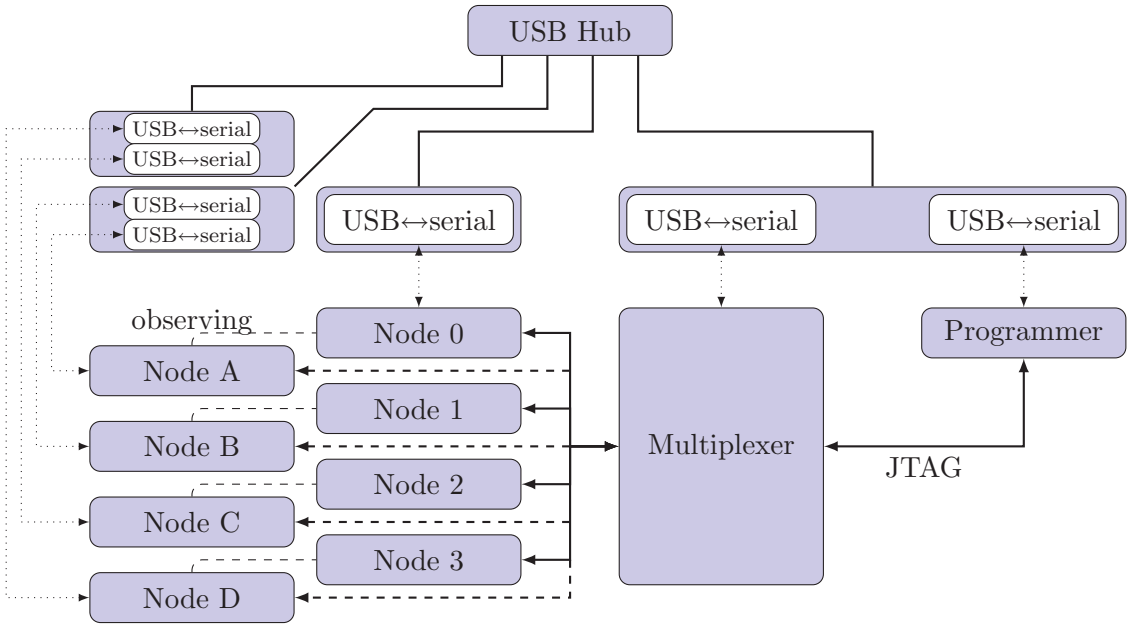


Figure 4.15. Basic concept of the monitoring hardware

signals coming from and going to the microcontroller node, while the space requirements remain nearly the same.

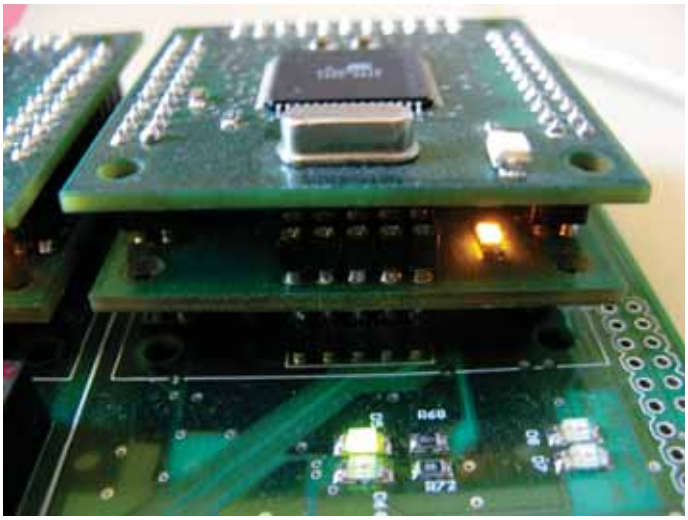


Figure 4.16. Monitoring node between ESE–Board and microcontroller node

The monitoring nodes are connected to each other by an own bus system which is, in principle, the same as the bus between the microcontroller nodes, and therefore also has the same characteristics. This bus system will be used in the remote workplace part to build up a monitoring network for distributed data acquisition of the microcontroller

nodes programmed by the students. Every monitoring node also has a UART interface to the host PC. This is achieved by two additional FTDI *FT*2232 dual–channel USB–to–serial converters. The additional pins used for the monitoring nodes are provided by an extension of the socket shown in Figure 4.17. Therefore it is assured that, even if no monitoring node is present at the board, the microcontroller node is working. This ensures that boards providing the monitoring extension can also be used for regular lab usage, for example in cases where there is a lack of lab hardware.

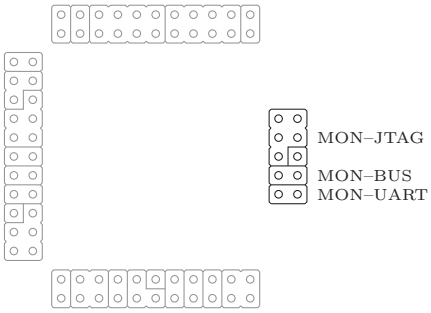


Figure 4.17. Socket interface – monitoring extension

Currently, ATMEL `ATmega1280` microcontrollers are used as monitoring nodes, because they provide enough IO to connect all signals that have to be monitored. Unfortunately, as already pointed out, they are not supported by the JTAGICE Mk.I clone used by the JTAGZeusProg. Therefore, another programming concept of the ones describes in Section 4.1.1 has to be used. During the development of the monitoring firmware, the bootloader solution was used to program them, after the XML based programmer was finished, it was used.

4.2.2 Software

The monitoring extension has the same software requirements as the basic board described in Section 4.1.2. Additionally, if one is using the JTAGZeusProg for programming the microcontroller nodes, a bootloader is needed to program the monitoring nodes, because the currently used monitoring nodes are not supported by the JTAGZeusProg. If another programmer is used, e.g. the XML based one described in Section 4.1.1, no further software is needed.

4.3 Exam Evaluation

One of the main advantages of our platform is the ability to check exam solutions for correctness and provide this information to the students in real-time during the exam. This is very useful, as it helps saving the time needed for ‘manual’ exam evaluation and

helps the students to appraise themselves (see [KE06] for further details). A detailed description of the whole showcase can be found in Section 5.2.

4.3.1 Workflow

A typical exam workflow is shown in Figure 4.18. After some preparation time, students are working on their development systems to solve the exam exercises. When they think they have finished their work, they can check their solution for correctness by executing a checker program which sends a check request to the hardware. For security reasons, this request also includes some information about the student calling the program, for example the student ID. If the hardware receives such a check request, it starts the evaluation of the student's program by monitoring its output and if necessary stimulating its inputs. This evaluation can be done periodically too, which means the hardware evaluation node continuously evaluates the student's program, which will result in a much reduced response time of the check requests. After the evaluator has finished and decided whether the solution is correct, wrong, or the sample solution (described in the software section below), the hardware returns an encrypted response to the host system. This response contains the number of the exercise and whether it is solved correctly or not. The check program displays the result on the screen to inform the student about success or failure of the evaluated solution. If the student's solution was correct, the check program stores the result locally to enable later collection of all the success information and do automatic grading. A detailed concept is shown in Figure 4.19, and the parts are explained below in more detail.

4.3.2 Hardware

The hardware part of the exam evaluation consists of the monitoring inputs, which can be connected to the target nodes as needed. There are various signals from all nodes that can be connected to the evaluator, such as analog signals, digital signals, timed signals like IC or PWM, and the bus system. Additionally, for providing direct visual feedback without the need of the client software, the evaluator node uses the two LEDs attached for signaling the evaluation state. The red LED indicates that the solution was evaluated to wrong, while the green LED indicates the solution was evaluated to correct. If both LEDs are turned on, the example solution was identified.

4.3.3 Software

The software part of the automated exam evaluation system can be split into 3 parts. The PC client software (called "checker process" in Figure 4.19), which interacts with the student, the evaluator running on the evaluation node, which checks the student's solution

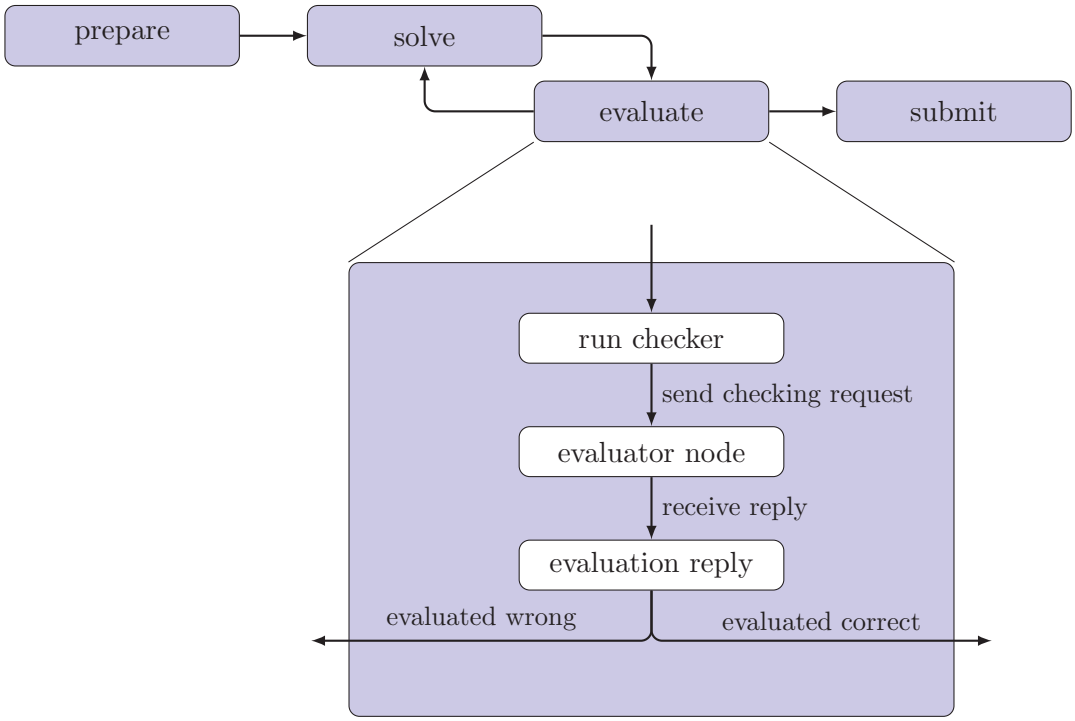


Figure 4.18. Exam workflow

for correctness, and the sample solution, which is provided to the students to allow the detection of hardware faults. Needless to say, all three parts of this system are crucial for the function of the whole system.

Client Software

This part of the exam evaluation tool provides a command line front end for the students and gives them visual feedback of the evaluation of their solution.

It also stores the success of the evaluation encrypted local on the hard drive, so that at the end of the exam all the information of the students' solutions can be collected and used for automated grading. The PC client needs to run a Linux operating system and has to be able to communicate with the evaluator firmware via a serial port interface. Further requirements are given only by the used encryption, but the system itself has no further dependencies.

Evaluator

The evaluator is the key element in the automated exam evaluation system. It does black box testing of the student programs, for which it needs information about the correct temporal and value behavior of a correct solution, and checks the program currently running

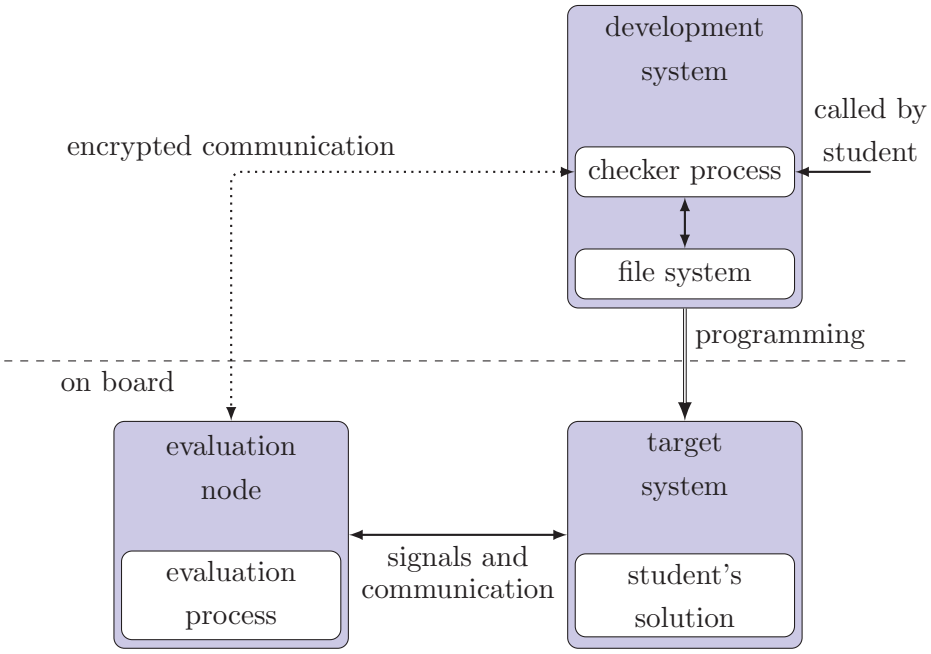


Figure 4.19. Detailed concept of the automated exam evaluation

on the target nodes against the correct solution by monitoring the output and, if necessary, stimulating the inputs to create test cases. Since the system-under-test runs on a different hardware than the evaluator, timing tests must take a possible clock drift between both nodes into account. Therefore, the evaluator checks timing results within a correctness window. The same mechanism is necessary to evaluate results in the analog voltage domain. In contrast, digital outputs can be checked using "hard" evaluations. Because the evaluator firmware tends to get very complex, we have built a generic framework, which provides the core functionality every evaluator needs, like implementing the approximate timing evaluation, a generic state machine, and the functions for the communication with the development system. By now there are template implementations of various types of exercise tasks, which makes it very easy to implement new exams in the future.

In the actual implementation, the value of the window range of correct solutions can be changed easily by editing a single parameter in the evaluator program. This parameter reflects the maximum of the relative difference to the correct solution against which the students' solution is evaluated.

The communication between the evaluator and the development system should be encrypted to prevent students from trying to hack the system instead of solving the exam tasks. The encryption algorithm can range from basic symmetric encryption by a shared key to high level asymmetric encryption like the public key encryption. Due to limited

resources available for the evaluator running on a microcontroller node, simple encryption algorithms are recommended.

Example Solution

It is very useful to provide an example solution to the students to allow them to check that the hardware on which they are solving their exam tasks is working properly. In microcontroller programming, there is always the possibility of a hardware fault or side effects from other software. When providing an example solution, the numbers of questions like 'does the hardware work properly' or 'I think there is a hardware fault' will decrease dramatically, because every student has the possibility to check whether the hardware works by flashing the sample. Therefore it is more than crucial that this solution really works. The example solution has to provide visual features to distinguish it from students' solutions without the need of an evaluator. This can be done easily by switching on a led which should be off in the real solution. Additional, the example solution should provide a feature to allow the evaluator program to distinguish between the example solution and the student one, otherwise automatic evaluation cannot be done.

4.4 Remote Workplace

The Remote Workplace (RWP) is rather extensive, so most parts were outsourced as practicals to other students. Therefore, we are just giving a brief overview here. The interested reader is invited to get further information by reading the references [HM07] [Luc08].

The idea of the remote workplace scenario is described in the identically named Section 5.3, and the concept is shown in Figure 4.20. The remote workplace part should reduce the load of the lab and should provide the students with more flexibility in their work times. Students should be able to connect via ssh to a remote workplace server and solve their exercises. There are several ESE–Boards connected to this server, and if a student wants to try out his work, he gets a board assigned and may test his program. Feedback is provided through a visualization tool which also connects to the server, receives the board state, and displays it as plugin system.

The remote workplace section consists of four parts, namely the board administration (described in Section 4.4.1) where the hardware is managed, the node monitoring network (Section 4.4.2) where the monitoring information is collected, the server process, described in Section 4.4.3, which provides all board data via a single User Datagram Protocol (UDP) or Transmission Control Protocol (TCP) port, and the Virtual Dashboard (VDB) (Section 4.4.4) which runs on the client PC, connects to the server process, and does the visualization of the data.

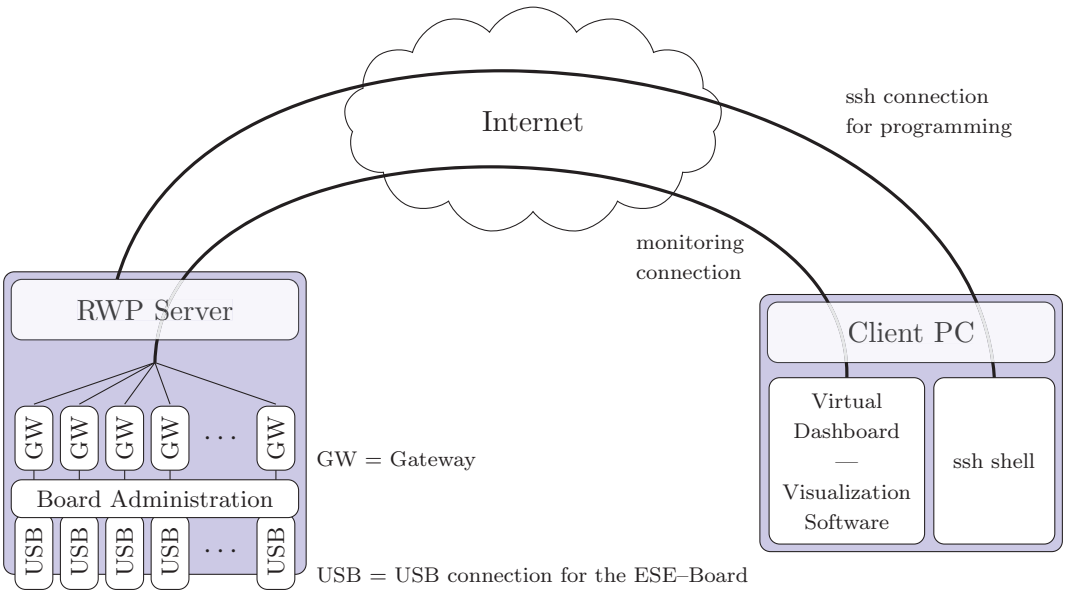


Figure 4.20. Remote workplace concept

4.4.1 Board Administration

The board administration handles all the ESE–Boards that are connected to the remote workplace server. It creates a pool of boards and manages the access to them. Whenever a new board is connected, it is detected by the hotplug-system. The new board is added to the board pool and the permissions are set correctly. If a board is removed during operation, this board is removed from the board pool and all open connections to it are gracefully closed. This behavior provides a very high level of maintainability, since there is no need to reboot the remote workplace server when there is a hardware fault in one of the boards, because the USB connections and the administration software allow hot plugging.

If a student is working via a remote workplace and needs hardware to test his solution, he just has to run the ‘make’ command via ssh on the remote workplace server. The special adopted makefile executes a binary named ‘acquire’, which itself connects via a UNIX socket to a running server process. The server process is running with root privileges to manage the board pool and does linearization of the incoming acquire requests. It looks for unused hardware and, if needed, it frees hardware assignments of students which have not used the hardware for a certain amount of time. A state flow diagram of the acquire procedure can be seen in Figure 4.21.

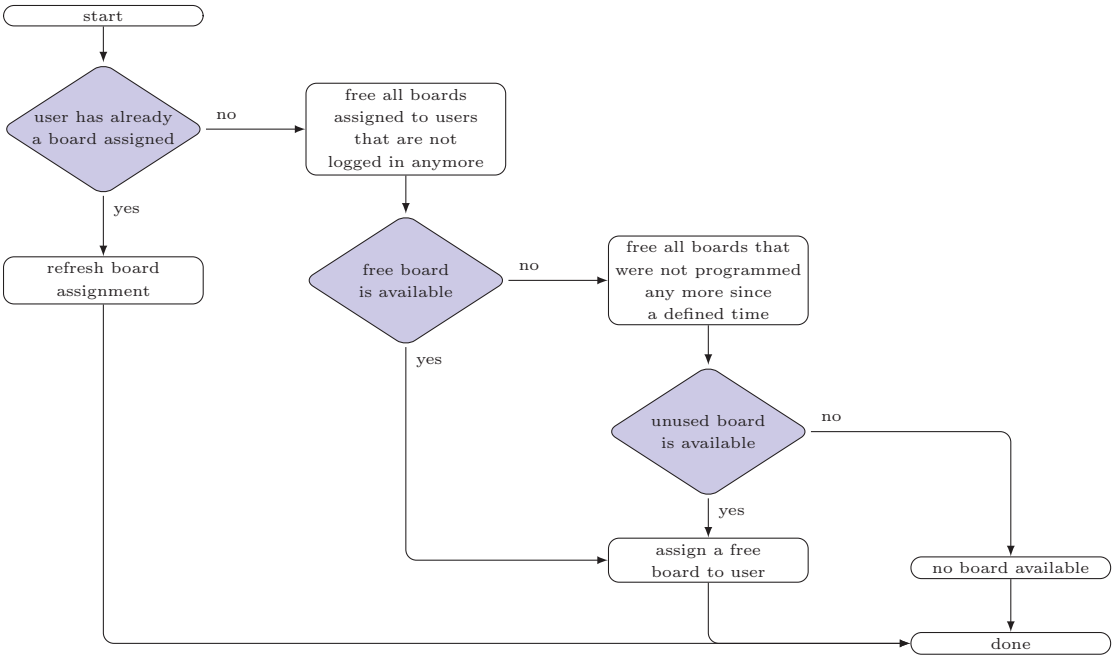


Figure 4.21. State flow to acquire a board

4.4.2 Node Monitoring Network

For monitoring the programming nodes, the remote workplace board has to provide the hardware monitoring extension described in Section 4.2. The monitoring nodes collect all IO data from the students' programs and send them through the monitoring network to the gateway node. This node sends the collected data to the gateway process running on the remote workplace server. As long as the bandwidth of a single serial connection is enough, we recommend using only one serial line per board, even if every monitoring node has its own serial connection to the server. This on-board data collection reduces organization effort at the server compared to merging the data of multiple serial communications at the server. However, if more bandwidth is needed between the monitoring network and the server process, another serial connection can be used in addition. The original firmware for the nodes of the monitoring network was written by Martin Hofer and Thomas Mair as their bachelor thesis [HM07] and was extended by Werner Luckner with the bus analyzer functionality [Luc08] to allow remote monitoring of the 1–wire bus system connecting the students' nodes.

4.4.3 Gateway Process

As depicted in Figure 4.20, for every ESE–Board connected to one of the USB interfaces, one server process collects all the data from the board. The gateway is started via a UDEV

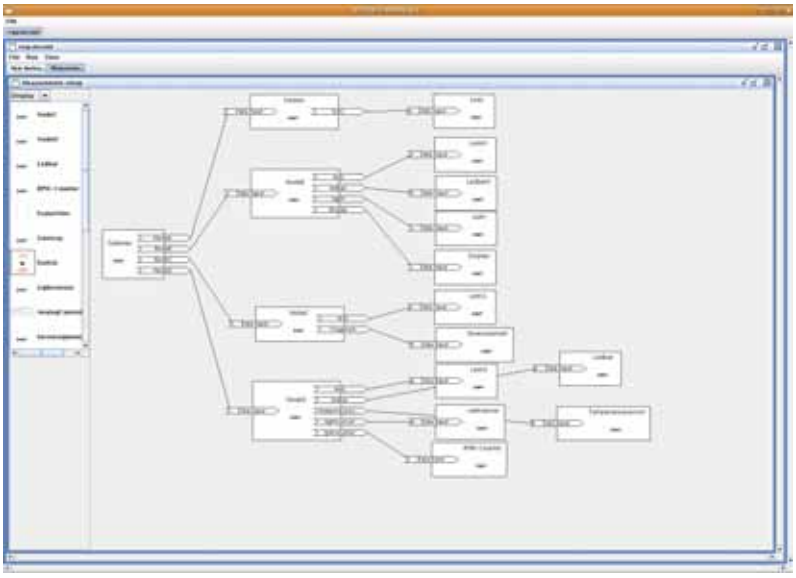


Figure 4.22. Measurement setup of the Virtual Dashboard

rule whenever a board is connected to the remote workplace server and stopped if the board is unplugged. Every gateway must have its own port number so the client PC can connect to it. We recommend a fixed base value for the port number, e.g. 30000, with the board id from the corresponding ESE–Board added. Therefore the complete data from one ESE–Board is collected and provided to the client via a single TCP or UDP Port. Which protocol is used depends on the requirements of the connection, and has to be consistent with the settings of the Virtual Dashboard.

4.4.4 Virtual Dashboard

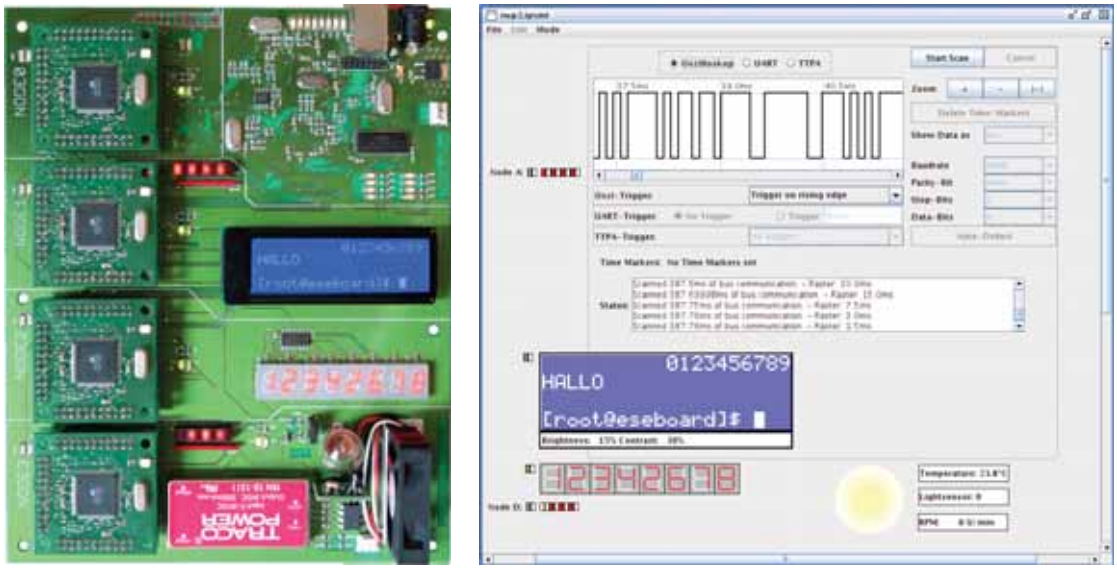
The Virtual Dashboard (VDB) is the client application of the remote workplace part. It is written in Java and provides the visualization of the data coming from the remote ESE–Board. The Virtual Dashboard is designed to be easily extendable via a plugin system in a way that new instruments only need to be copied into the plugin directory and can be used. The VDB concept is the one of a switchboard. It consists of an *Instrument Panel* and a *Measurement Setup*. The *Measurement Setup* (an example is shown in Figure 4.22) can be viewed as the backside of a switchboard, where all the wiring is done no matter where the instruments are placed. Therefore, it just defines the logical connection between data sources and visualization instruments.

The *Instrument Panel* of the Virtual Dashboard corresponds to the front side of the switchboard where all the instruments can be seen. Therefore the virtual ESE–Board can be built in a way that every instrument like LCD or bargraph is nearly at the same place as in the real hardware version. The Instrument Panel also allows the usage of images for

customizing visuals. Figure 4.23 shows a comparison between the real hardware and an Instrument Panel of the Virtual Dashboard.

In 2008 Werner Luckner extended the Virtual Dashboard by a bus analyzer plugin which is able to monitor the real time bus available for communication between the students' nodes. This plugin simplifies the development of communication between the nodes, as it not only provides simple logic evaluation of the bus state, but is also capable of a semantic interpretation of the data. It can find UART frames and display the transmitted values in various number systems, automatically detect the baud rate and transmission settings of serial communication, and is even aware of the TTP/A round structure and therefore can analyze TTP/A communication.

More information about the plugin system of the Virtual Dashboard and the features of the bus analyzer can be found in [HM07] and [Luc08].



(a) Real world (b) Client PC

Figure 4.23. Real board compared to visualization via Virtual Dashboard

5

Showcases

We have identified four showcases which illustrate the possible uses of the ESE–Environment. First there is the normal usage as platform in our lab course, which is explained in detail in *Showcase I*. The usage of the platform for automatically evaluating practical exams in the course is described in *Showcase II*. *Showcase III* explains the possibility for students to participate in our course even if they do not have physical access to the laboratory, and *Showcase IV* deals with the research prospects of the platform.

5.1 Showcase I – Students in Course

Embedded Systems Engineering (ESE) is a course at the Vienna University of Technology. ESE is mandatory in the bachelor study of computer engineering (where it is scheduled in the 5^{th} term) and, if not already heard, it is also mandatory in the master study of computer engineering. The main goals of the lab course are:

1. teaching students advanced skills in microcontroller programming
2. experimenting with common access strategies on a shared communication media
3. giving students a deeper understanding of problems in real-time communication
4. teaching problems of distributed programming
5. teaching advanced peripherals programming

Every year about 70 – 80 students take this course. ESE is held mainly as hands-on course, as described in Section 3.1. In the lab there are two lab rooms available with overall 18 development systems, each consisting of a Dell Precision 390 Workstation

running CentOS 5.2 which is connected to a ESE–Board. Four Tektronix TDS 2024B[1] are provided to the students to make measurements at the hardware.

5.1.1 Embedded Systems Engineering – A Course Overview

The Embedded Systems Engineering course is split into two major parts. The first part teaches peripherals programming. As exercise, students have to program driver modules for the processor's peripherals (UART, timer, ...) and various of the on-board IO components (LC–display, fan, sensors, ...). These driver modules should be programmed in a generic way to enable their use later on in the course. This not only saves a lot of time in the further tasks, but also teaches generic modular programming for better source re-usability. An exam finalizes the first part, where the students have to show their theoretical background knowledge by answering some multiple-choice questions in the theory part, and where they have to solve two practical programming tasks under certain time pressure. More detailed information about the process of the exams in ESE is given in Section 5.2. By the end of the first part, every student has become familiar with the ESE-Board, the basic setup, its handling and its functions, and thus every student should be prepared for the following part. The first part usually takes about five weeks and comes along with tutorials and lectures.

In the second part, the students form teams of three persons to solve two successive exercises which are a bit more extensive. In the first group exercise, the students should learn the difficulties of networked embedded systems programming by implementing a MAC-layer protocol. Afterwards they have to implement a small distributed application which utilizes the developed protocol. The main focus in this exercise is clearly the protocol implementation, where the students should perceive the problems of timing, synchronicity, and also some real-time problems in a distributed communication via a shared communication medium. The students have to implement different line encodings (Return to Zero (RZ), Non Return to Zero (NRZ), Manchester) and different Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA) protocol types (Token Ring, CAN derivates, collision avoidance by different precharge times).

In the second and last exercise of this part and the course, the students have to familiarize themselves with a given real-time communication protocol (TTP/A) [Kop02] which was developed in-house. TTP/A is a time-triggered field bus protocol which provides real-time capabilities. The groups should build a control application on top of this protocol. The aim of this exercise is to demonstrate to the students how easily communication can be accomplished, if a working protocol stack is available. Another goal is to teach the difficulties of becoming acquainted with a given protocol stack. After the students

[1] The Tektronix TDS 2024B is a four channel, 200MHz, 2GS/s, digital storage oscilloscope. More information can be found at `http://www.tek.com/products/oscilloscopes/tds1000_tds2000/`.

have mastered the protocol, they have to implement a closed loop control on top of it, for example a fan speed control loop or a temperature control loop, to learn about different control controllers (e.g., PI–, PID–, two-position controller).

Like the first part, an examination containing a theory and a practical component finalizes the second part and strengthens the students' obtained knowledge.

Students may also do some voluntarily exercises, like hardware design using Eagle, to collect bonus points for better grading.

5.1.2 The ESE–Board in Course

Our ESE–Board has to handle several tasks during a course. During normal course operation, up to 20 students per day have their weekly and supervised time slots. Additionally, up to 10 students (in the week before the exercise deadlines up to 40 students) are working permanently on the targets to solve their exercises during the lab opening hours. Both exams have hands-on parts where the students have to solve programming exercises. Therefore, the boards are in use the whole term nearly every day, which causes the need for robustness and maintainability.

5.1.3 Results

The ESE-Environment is now running the third year without major problems. By now, one CPU–Board had to be replaced by a new one, because the microcontroller got defect. In the first year, on three ESE–Boards the fans lost some blades because students were too lazy to turn them off by software. Instead, they stuck pens and similar things between the blades to stop them. Therefore we provided a "beQuiet–Example" program to the students, which turns off the fans.

Every year about 70–80 students are registered for the course, and about 50 – 60 students do finish the course. On average, every exercise is done in 20 hours work per student. Assuming 60 students are working on the three exercises on the 18 boards that are arranged in the lab, this would result in 200 working hours per board per year, where the exam time is left unconsidered.

5.2 Showcase II – Exam Evaluation

As described in the previous section, in ESE each of the two parts is finished with an examination. The exam details are explained in the following.

5.2.1 Exams in ESE – A More Detailed View

In the week before an exam, students choose a time slot when they want to do the examination. Timeslots are scheduled every 30 minutes starting typically at 8.40am till

about 2pm. In every time slot at most 8 students can do the exam synchronously. Later timeslots are only available if all previous slots are completely filled up.

An exam takes about 2 hours and the students have to go through three stages. First they have to solve a theory part where 12 multiple-choice questions, each offering three answers, have to be solved in 15 minutes on a computer-aided test environment called Generic Test Framework (GTF) [Dei09]. In these questions the students are asked fundamental theory concepts of microcontroller programming, embedded systems engineering, and the development environment used in the laboratory where the embedded systemes. Figure 5.1 shows two students working on the theory part. In every exam timeslot there are at most 8 students doing the theory part concurrently. There are 9 PC's running the test environment, so one PC is available as a spare. The students are not allowed to use any manuals or documentation during this test stage.

Figure 5.1. Students at the theory part of an exam

After the theory part, the students are guided to the next station, where they receive the description of two programming tasks. They have 25 minutes for preparing their solutions on paper. During this time the students have access to the datasheets and schematics, but only as hardcopy, not electronically (which means there is no electronic search function available). Figure 5.2 shows some students preparing for the programming part.

In the last stage, the students have 55 minutes time for coding and debugging their previously prepared solutions on a development system with an ESE–Board connected. Therefore, two lab rooms with 9 PC's each are available. Again one PC per room is left as a spare in case of problems. The PC's are running the same development environment as in the normal lab course but are disconnected from the local network and the Internet. During this part of the test, the students have access to the datasheets and schematics as print out as well as PDF files, where searching can be done more easily. Figure 5.3 shows some students at the programming part of an exam.

Figure 5.2. Students during preparation time

Figure 5.3. Students during hands-on part of the exam

As described in Section 4.3, every programming task the students have to solve during an exam comes together with an example solution. This is necessary to provide the students with the possibility to check whether the hardware works correctly. In addition to the example solution, the exam task comes with an evaluation program that is downloaded automatically to Node 0 and does black box testing of the student's solution by stimulating its inputs and monitoring the corresponding outputs. The evaluator compares the behavior of the student's program against the correct solution and decides wether the solution is correct. If the solution was evaluated to correct, the students may run a submission script which logs the successfully solved exercise. Additionally to the submission script, tutors may also check the output of the evaluator and mark the tasks solved on the task exam cover sheet.

After solving both tasks correctly, or if the time is up, the students have to leave the lab and their exam is finished.

5.2.2 Results

During the last three years 6 exams have been evaluated by the ESE–Environment. Only one exam could not be evaluated fully automatically, not because of problems from the environment, but because due to a mistake the task descriptions did not match the example solutions and the evaluators.

Therefore we recommend to double check every example solution by at least two different persons. A wrong example solution or a not matching task description will lead to confusion during the exam, and it is possible that a student who normally could solve the tasks fails both, just because of the confusion of one example.

During the first three exams, every task was not only evaluated by the ESE–Environment, but was evaluated additionally during the exam by tutors. About $60 \times 2 \times 2 = 240$ tasks have been evaluated per year. Assuming an assistant professor needs only 5 minutes to evaluate a task to right or wrong by inspecting the source code, using automated evaluation means about 20 hours per year more time for research. At every exam, about 3 students complain about hardware faults because their solution does not work correctly. By providing an example solution during the last 3 years, approximately 18 doubts about hardware faults could therefore be answered quickly by a counterquestion about a working example solution.

5.3 Showcase III – Remote Working Students

In fall 2007, we started a case study with some voluntary students at the university of Priština, Kosovo (the location of the cities of the two involved universities are marked in the map shown in Figure 5.4). The goal was to show that we can offer a lab course also at a distant place. The students had to solve the same exercises as in the local course, but instead of sitting in the lab in front of their targets, they were logged in via SSH to the remote workplace server here in Vienna, which had several boards with monitoring extension connected. An assistant professor from Vienna supervised the course in Priština and introduced the students to the hardware and the environment, but all boards physically remained in Vienna. Programming was done via SSH and command line; monitoring was done via the monitoring client communicating with the server by UDP. There were no exams, just the programming tasks to do. It was the first test run to evaluate the remote working ability of the platform, and to see whether the remote part is ready for productive use in the lab course.

As server we used a Dell Optiplex GX260 Workstation PC running CentOS 5.2 Linux as operating system. There were 3 ESE–Boards connected to the server, which should be more than enough when having 5 user accounts for the groups participating. The whole system was secured by an APC-Smart UPS 700 against power outages.

Figure 5.4. Remote working scenario locations

5.3.1 Results

As mentioned in the description of this showcase, we have done a case study with some students in Priština. This study brought out that there is a major problem with the multiuser-ability of the monitoring solution. It previously was designed for the remote monitoring of a single board for only one student working on it. Therefore, it was not necessary to terminate the connection by the server. However, this is exactly what is required if a board is acquired by another student, but another monitoring session is open. Therefore, we learned that the system architecture of the remote working part of the platform needs a redesign with respect to the multiuser management.

Another shortcoming of the monitoring client application ia, that the whole connection settings have to be set manually in a config file. This is bad if the ports change very often as in our current solution. Therefore, the server-client communication should also be extended by a port negotiation in a way that the client always connects to the same server port and the server assigns another port for communication for each client. This would also solve some of the authorization problems we encountered.

The work that is missing to get the remote monitoring working for multiuser-operation is part of a bachelor thesis which is currently in progress. Adapting a quote from Edison,

we have to conclude this showcase with the sentences “We have not failed. We’ve just found a way that won’t work.”[2]

5.4 Showcase IV – Development and Research Platform

Bachelor and master theses in computer engineering often require practical programming work on a microcontroller. For such practical evaluations, students require a platform providing an easy to go solution for starting work on the one hand and support for connecting and instrumenting special hardware on the other hand. As described in the section of the system architecture, the platform therefore provides extension possibilities like interfaces to interact with additional hardware (analog IO, digital IO, and even timed signals can be patched into and, respectively, generated from the ESE–Board). The bus system is extendable too, so it is possible to add special programming nodes, e.g. with special sensors or actuators, or just programming nodes to run larger distributed algorithms. If more nodes are needed, it is possible to connect two or more ESE–Boards together so that 12, 16, or more microcontrollers execute a distributed application. The advantage of this system is that there is not much external wiring for emulating, for example a large sensor network. This is a particular advantage if one must take the assembly to a conference.

5.4.1 Results

During the last three years the board already was part of various scientific experiments. First, there have been several bachelor theses about the platform itself, whose results have been integrated into the platform features. An example is therefore the remote workplace part of the platform. But there were also several bachelor and master theses which did not deal with the platform as such, but used it as testbed for various experiments. Four interesting projects which were supported by the ESE–Platform are described in the following.

Graphical Development Environment for Embedded Systems

In 2006, Philipp Jahn and Thomas Polzer built a graphical development environment which allows graphical programming of embedded systems as their bachelor thesis [JP06]. Among a different hardware setup they also used this platform as target system for their generated code.

[2]The original quote was “I have not failed. I’ve just found 10,000 ways that won’t work.”, but it was adapted to fit our showcase.

Design and Implementation of Interfaces to the Time-Triggered Fieldbus System TTP/A

By 2009, Gernot Klingler wrote his master thesis about the design of gateways between different time-triggered protocols [Kli09]. He used the ESE–Platform for implementation of various communication interfaces to the TTP/A protocol.

A Modular XML-based JTAG Programmer for Embedded Devices

In 2008, Martin Schmölzer developed a versatile JTAG programmer as his bachelor thesis [Sch08]. The JTAG programmer uses XML for specification of programming cycle and type, which makes it a very versatile programming and debugging tool for embedded microcontrollers based on the AVR architecture. It is planned to replace the current programming software of the ESE–Board with this tool.

Implementation of a LIN v2.1 Protocol Stack for Embedded Devices

Currently, based on the Local Interconnect Network (LIN) V1.3 implementation from Atmel [Atm05], a student implements the major parts of the LIN v2.1 protocol [ABD+08]. He uses the ESE–Board as development platform emulating a four node LIN network consisting of one master node (Node 0) and three slave nodes (Nodes 1–3). Futhermore he implements a closed loop control as demo application. This work should be integrated in the ESE–Course as second available protocol stack for the students, besides TTP/A.

6

Conclusion and Outlook

In this thesis we surveyed different approaches how state-of-the-art embedded systems lab courses can be held. We compared them against our requirements and the aim of ESE, a hands-on lab course at the Vienna University of Technology. We found that remote lab access would be a very good addition to the hands-on part of the course.

We designed a new lab hardware to get around certain drawbacks of the old hardware, like outdated microcontroller units or connection problems between them. In addition, the hardware was constructed to reduce maintenance intervals and workload of the staff. One big advantage that was not available prior to this work is the hardware-aided exam evaluation, which dramatically reduces the stress during practical examinations by supporting students and supervising staff. This makes hands-on microcontroller programming exams as relaxed as they could be.

A by-product of this work was a new JTAG programmer that allows programming of various devices. The programming algorithm is specified in an XML configuration file, which allows adding the support for new devices without changing the firmware.

The results of various bachelor theses were integrated into the platform, for example to provide the remote lab access that allows students to work from outside the lab without restrictions concerning the lab opening hours. A monitoring network was designed to supervise the microcontroller nodes the students are programming and to track their IO behaviour. The collected data is sent via a Remote Workplace server to the Virtual Dashboard program, where the ESE–Board is visualized. The integrated bus oscilloscope allows inspection of the on-board serial bus and due to various different display modes, for example the support for UART frames with an automatic detection of the used baud rate and transmission mode, it is even more comfortable than our real oscilloscope.

Furthermore, a remote lab experiment was set up between Vienna, Austria and Priština, Kosovo. In this case study it turned out that the multi-user part of the platform is not optimal and a major revision is needed. Nevertheless, the system worked well in principle and doubts about insufficient bandwidth could be resolved. A student is currently working on a new communication and management system to solve the remaining problems and the course coaches are working on a mode to integrate the remote workplace into the ESE course.

The platform was also designed to support various research topics around distributed real-time systems and networked embedded systems. For this reason, it is easily extendable and can be adopted to a special research task with very little effort. Due to its compact architecture it can be easily taken to conferences or presentations.

By now, the new hardware is used for the third year in ESE and the feedback of the students is positive by all means. As we thought, the availability of an overall schematic of the platform advances the general understanding of the interactions between the sub-parts. Our fears about overburding beginners with a fully integrated printed circuit board of this complexity could also be resolved by providing a detailed and well-structured schematic of the hardware.

6.1 Outlook

During the last years while working on this platform several possible improvements came up. In the following we want to describe the three major drawbacks of the current system and how one could avoid it in a future revision of the hardware and software.

Programming Concept As the programming concept of the ESE–Board changed two times during development without changing the hardware, at a major revision of the ESE–Board one could fit the programming hardware to the currently used programmer, i.e., the XML based generic programmer introduced in Section 4.1.1. We would not recommend to remove the switching CPLD but instead rearrange the programming channels. For compatibility to the actual hardware channel 0 to 3 should be left unaltered to the microcontroller nodes the students are working with, but the monitoring nodes should be chained to use only one channel. Hence, one of the three remaining channels can be merged to the extension interface in order to allow programming the extension nodes in a chain as well. This would need a bit of redesign of the extension nodes currently available, but would greatly simplify their programming.

Board Administration The board administration of the remote workplace server is working, but it comes with some drawbacks. For example, if a student acquires an ESE–Board and currently there is no free board available, but there are several boards assigned

but unused, the board administration frees all idle boards, even if there is just a single one needed. It is recommended to overhaul this part of the platform to free only one board, for instance the one less recently used, and leave the other ones idle without taking them away from other students.

Remote Workplace System Due to the fact that it was originally designed as a single user application and later was integrated into the platform, the currently used solution supporting various boards is not optimal. Presently another student takes care of this problem by redesigning the communication system between the visualization client and the gateway server. This should fix the major problem of the missing authentication in the communication system, which currently leads to problems assigning the monitoring connections to the acquired board. The plan is to overcome this problems by programming a connection proxy which is aware of the boards acquired by the users, and thus is able to route the monitoring traffic to the specific gateway servers.

Bibliography

[ABD+08] Audi AG, BMW AG, DaimlerChrysler AG, Motorola Inc., Volcano Communication Technologies AB, Volkswagen AG, and Volvo Car Corporation. LIN specification v2.1, 2008. Available at `http://www.lin-subbus.org`.

[Atm05] Atmel Corporation. AVR322: LIN v1.3 Protocol Implementation on Atmel AVR Microcontrollers, 2005. Available at `http://www.atmel.com/dyn/resources/prod_documents/doc7548.pdf`.

[Bac05] R. Bachnak. Teaching microcontrollers with hands-on hardware experiments. *Journal of Computing Sciences in Colleges*, 20(4):207–213, 2005.

[BPM00] D. Beetner, H. Pottinger, and K. Mitchell. Laboratories teaching concepts in microcontrollers and hardware-software co-design. In *Proceedings of the 30th Annual Frontiers in Education Conference*, volume 2, pages S1C/1–S1C/5, Kansas City, Missouri, USA, October 2000.

[CF97] C. A. Canizares and Z. T. Faur. Advantages and disadvantages of using various computer tools in electrical engineering courses. *IEEE Transactions on Education*, 40(3):166–171, Aug 1997.

[CHMM04] M. J. Callaghan, J. Harkin, T. M. McGinnity, and L. P. Maguire. Cost-effectiveness issues in remote experimentation. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, volume 5, pages 4700–4704, October 2004.

[CNEC04] J. E. Corter, J. V. Nickerson, S. K. Esche, and C. Chassapis. Remote versus hands-on labs: A comparative study. In *Proceedings of the 34th ASEE/IEEE Frontiers in Education Conference*, pages F1G17–21, Savannah, GA, USA, October 2004.

[Dei09] H. Deinhart. A generic solution for computer based automation of examinations. Technical Report 28/2009, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/2/182-2, 1040 Vienna, Austria, 2009.

[EHK01] W. Elmenreich, W. Haidinger, and H. Kopetz. Interface design for smart transducers. In *IEEE Instrumentation and Measurement Technology Conference*, volume 3, pages 1642–1647, Budapest, Hungary, May 2001.

[Elm09] W. Elmenreich, editor. *Embedded Systems Engineering*. Vienna University of Technology, Austria, Vienna, Austria, 2009.

[ETW06] W. Elmenreich, C. Trödhandl, and B. Weiss. Embedded Systems Home Experimentation. In *2nd IASTED International Conference on Education and Technology (ICET'06)*, pages 11–15, July17–19, 2006.

[GCARVA+01] F. J. González-Castañoa, L. Anido-Rifón, J. Vales-Alonso, M. J. Fernández-Iglesias, M. Llamas Nistal, P. Rodríguez-Hernández, and J. M. Pousada-Carballo. Internet access to real equipment at computer architecture laboratories using the Java/CORBA paradigm. *Computers & Education*, 36(2):151–170, February 2001.

[GPPD05] Y. Guran-Postlethwaite, D. N. Pocock, and D. Dutton. Web-based real electronics laboratories. In *Proceedings of the 2005 American Society for Engineering Education Annual Conference & Exposition*, Portland, OR, USA, June 2005.

[HM07] M. Hofer and T. Mair. Remote Board Monitoring in Embedded Systems Lab Courses. Bachelor's Thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2007.

[JP06] P. Jahn and T. Polzer. Graphical Development Environment for Embedded Systems. Bachelor's Thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2006.

[KE06] A. Kößler and W. Elmenreich. Automated solution evaluation during a practical examination. *Junior Scientist Conference*, 2006.

[Kli09] G. Klingler. Design and Implementation of Interfaces to the Time–Triggered Fieldbus System TTP/A. Master's Thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2009.

[Kno00] K. Knopper. Building a self-contained auto-configuring Linux system on an ISO9660 filesystem. In *Proceedings of the 4th Annual Linux Showcase & Conference*, Atlanta, Georgia, USA, October 2000. USENIX Association.

[Kop02] H. Kopetz et al. Specification of the TTP/A protocol. Research Report 61/2002, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, September 2002. Version 2.00.

[Luc08] W. Luckner. Busanalyzer for Remote Monitoring of an Embedded Systems Lab Course Board. Bachelor's Thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2008.

[MČŽ00] G. Mužak, I. Čavrak, and Mario Žagar. The virtual laboratory project. In *Proceedings of the 22th International Conference on Information Technology Interfaces*, pages 241–246, Pula, Croatia, June 2000.

[MLM03] B. D. Moulton, V. L. Lasky, and S. J. Murray. The development of an enviroment for remote embedded systems: Feedback from students and subsequent enhancements. *World Transactions on Engineering and Technology Education*, 2(1):65–68, 2003.

[NMN03] Z. Nedic, J. Machotka, and A. Nafalsk. Remote laboratories versus virtual and real laboratories. In *Proceedings of the 33rd ASEE/IEEE Frontiers in Education Conference*, pages T3E1–6, Boulder, CO, USA, November 2003.

[OMG03] Object Management Group (OMG). *Smart Transducers Interface V1.0*, January 2003. Specification available at `http://doc.omg.org/formal/2003-01-01`.

[PP00] N. Perić and I. Petrović. Virtual laboratory for automatic control and supervision – challenges and opportunities. In *Proceedings of UN-ECE International Conference on Technology Transfer for Economic Development*, Zagreb, Croatia, June 2000.

[Sch08] M. Schmölzer. A Modular, XML-Based JTAG Programmer for Embedded Devices. Bachelor's Thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/2/182-2, 1040 Vienna, Austria, 2008.

[TPA06] C. S. Tzafestas, N. Palaiologou, and M. Alifragis. Virtual and remote robotic laboratory: Comparative experimental evaluation. *IEEE Transactions on Education*, 49(3):360–369, August 2006.

[TPE06] C. Trödhandl, M. Proske, and W. Elmenreich. Remote Target Monitoring in Embedded Systems Lab Courses using a Sensor Network. *The 32nd Annual Conference of the IEEE Industrial Electronics Society*, Nov. 2006.

[Trö02] C. Trödhandl. Architectural requirements for TTP/A nodes. Master's thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2002.

[TTA03] TTAGroup. *Specification of the TTP/C Protocol*. TTAGroup, 2003. Available at `http://www.ttagroup.org`.

[WHL05] H.-D. Wuttke, K. Henke, and N. Ludwig. Remote labs versus virtual labs for teaching digital system design. In *Proceedings of the International Conference on Computer Systems and Technologies (CompSysTech'05)*, pages IV.2–1–6, Varna, Bulgaria, June 2005.

B

Acronyms

API Application Programming Interface
CAN Controller Area Network
CD Compact Disc
CPLD Complex Programmable Logic Device
CSMA/CA Carrier Sense Multiple Access/Collision Avoidance
EEPROM Electrically Erasable Programmable Read Only Memory
ESE Embedded Systems Engineering
FTDI Future Technology Devices International
GTF Generic Test Framework
IC Input Capture
JTAG Joint Test Action Group
LCD Liquid Crystal Display
LDR Light Dependend Resistor
LED Light Emitting Diode
LIN Local Interconnect Network
LVA Lehrveranstaltung
MAC Media Access Control
MCU microcontroller unit
NRZ Non Return to Zero
PC Personal Computer
PCB Printed Circuit Board
PWM Pulse-Width Modulation

RAM Random Access Memory
RWP Remote Workplace
RZ Return to Zero
SPI Serial Peripheral Interface
TCP Transmission Control Protocol
TTP/A Time Triggered Protocol Class A
TTP/C Time Triggered Protocol Class C
UART Universal Asynchronous Receiver Transmitter
UDP User Datagram Protocol
USB Universal Serial Bus
VDB Virtual Dashboard
XML Extensible Markup Language

C

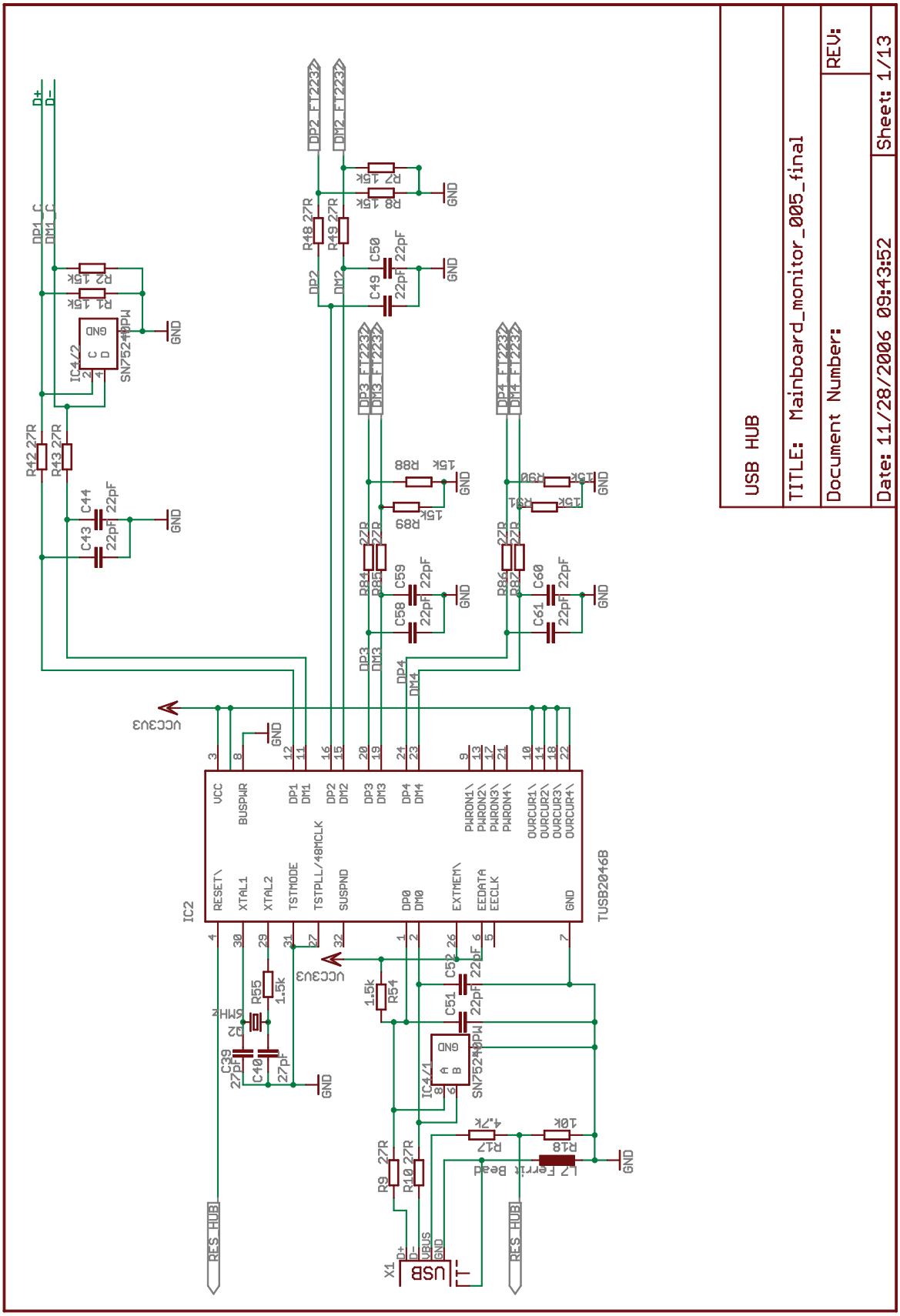
Schematics

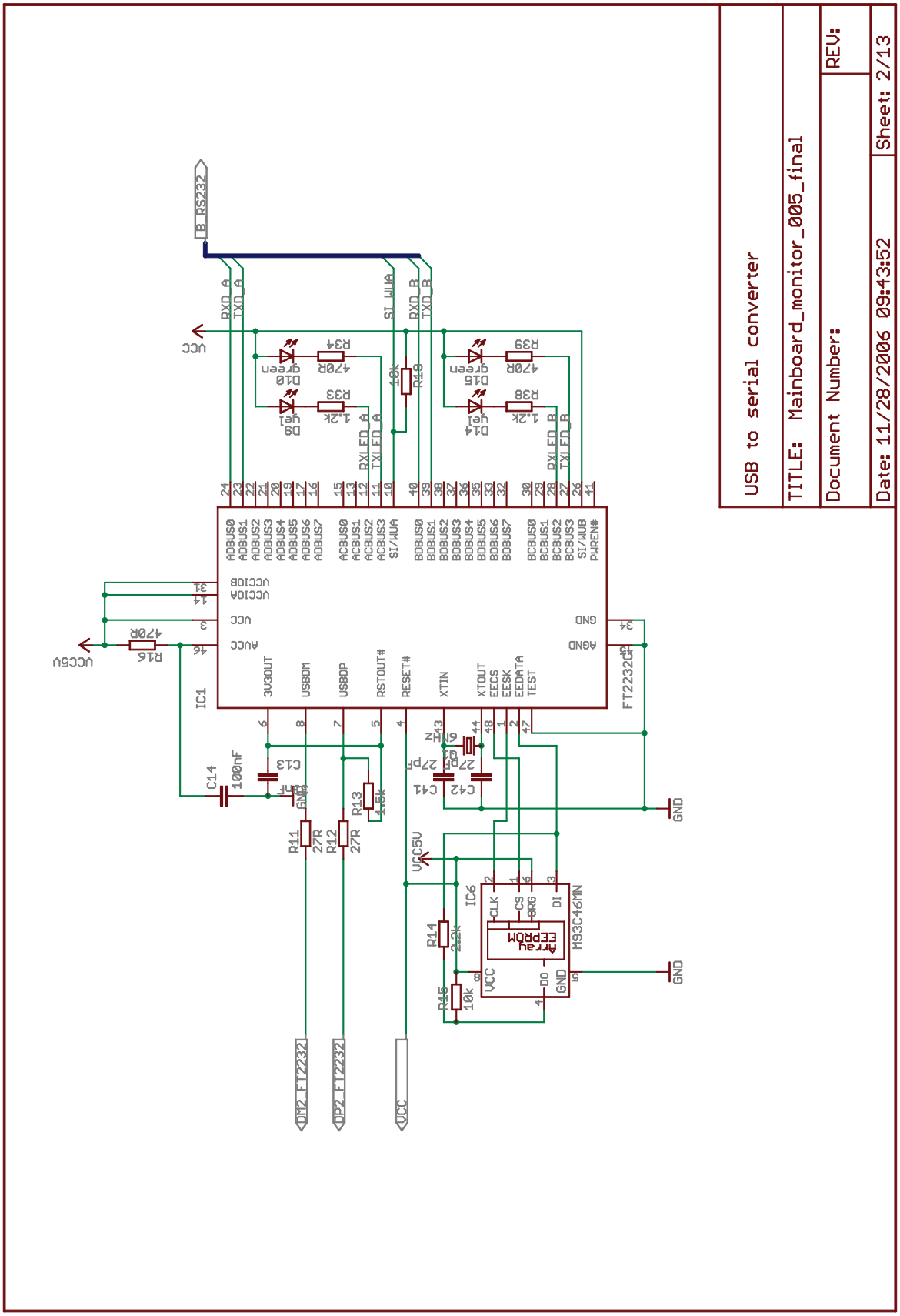
C.1 Monitoring Board

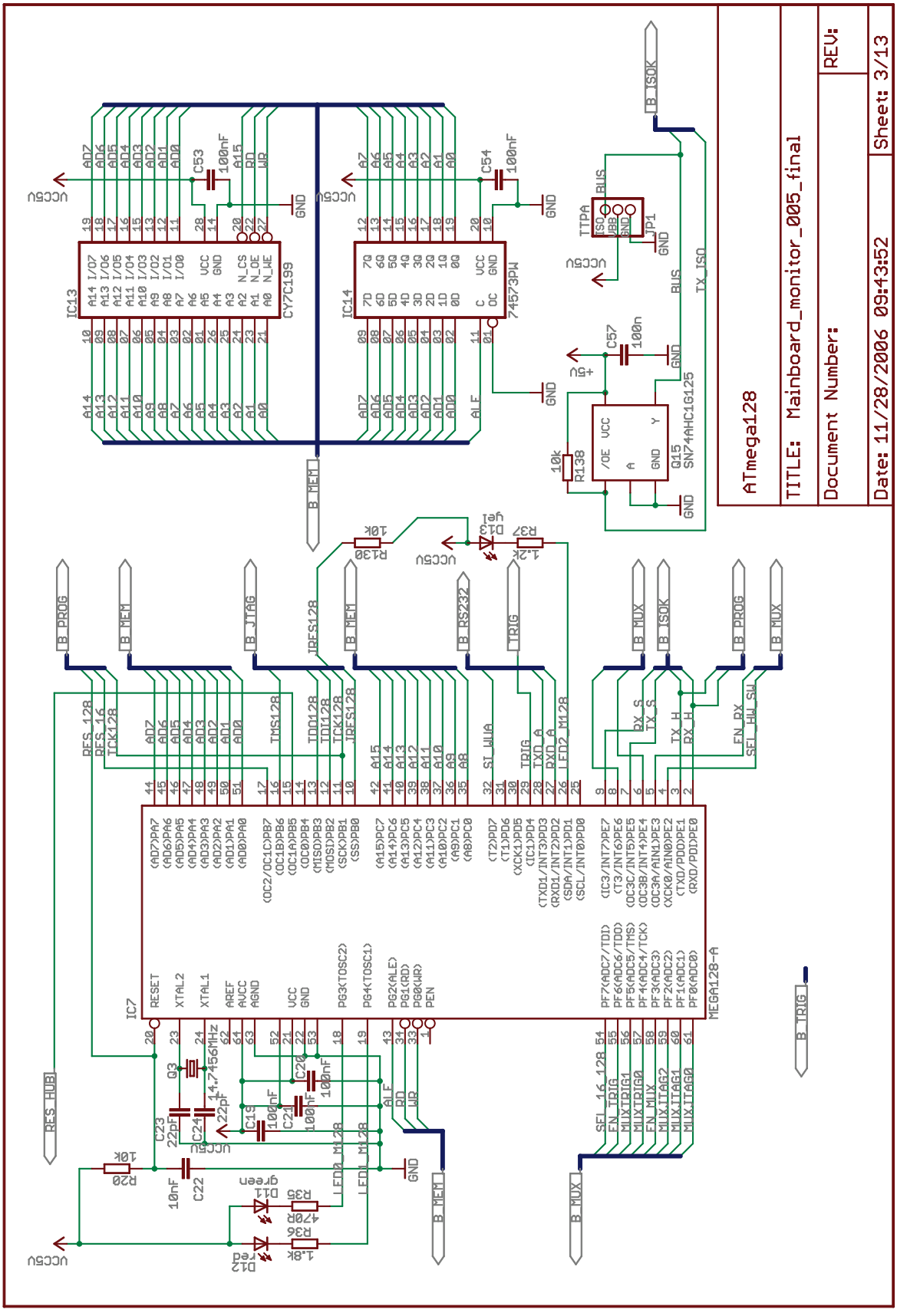
In this Section the schematics for the ESE–Board featuring remote monitoring, exam evaluation, and lab usage are shown.

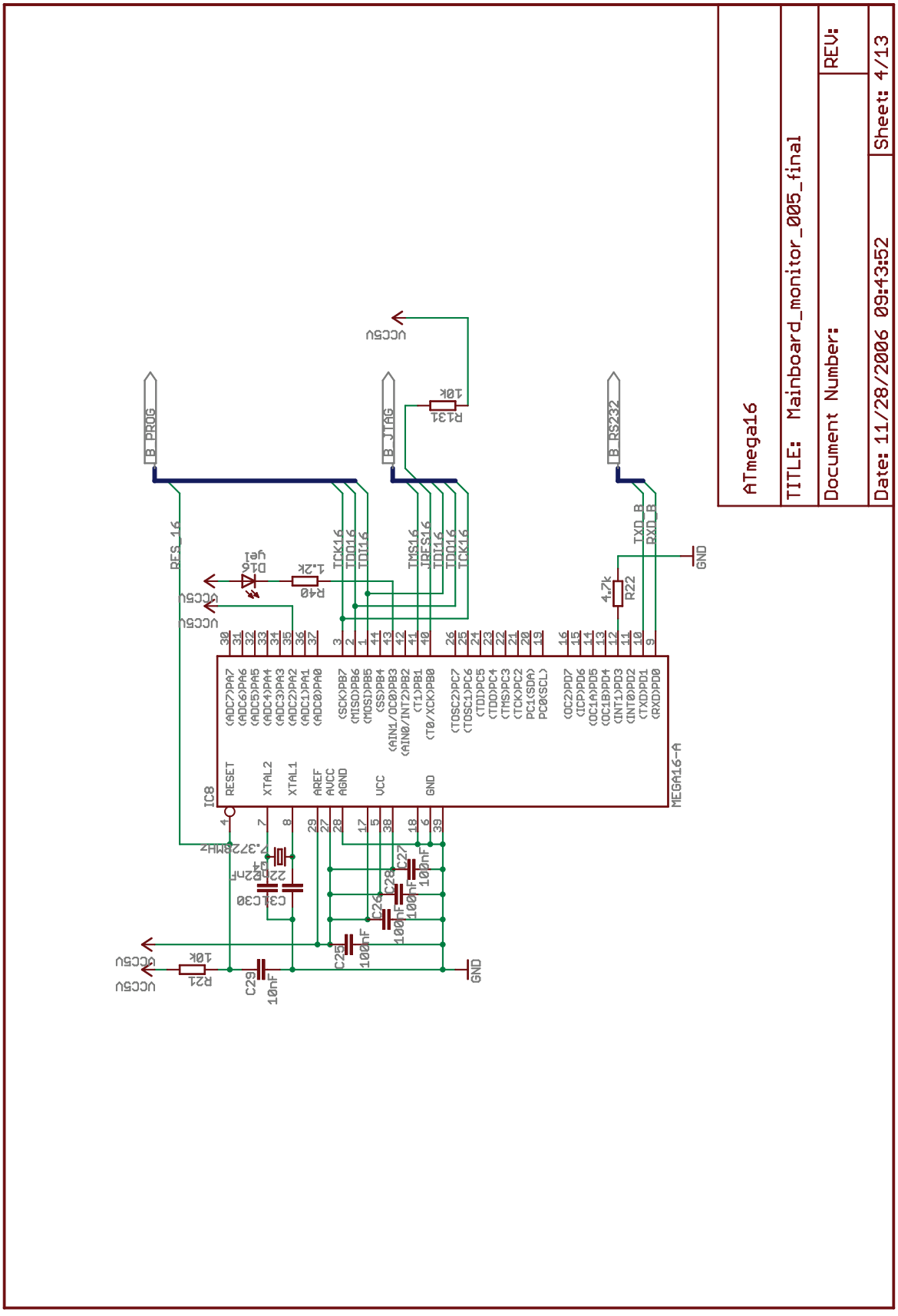
On the sheets 1–5 the programmer is shown containing the two microcontrollers and the CPLD for multiplexing the JTAG lines. Sheet 6 shows the power supply of the ESE–Board. Sheet 7–11 shows the sockets for the microcontroller nodes 0–3 with their peripherals and the monitoring addition for the sandwitch nodes. Sheet 12 and 13 shows the USB to serial converter of the monitoring nodes to the development system.

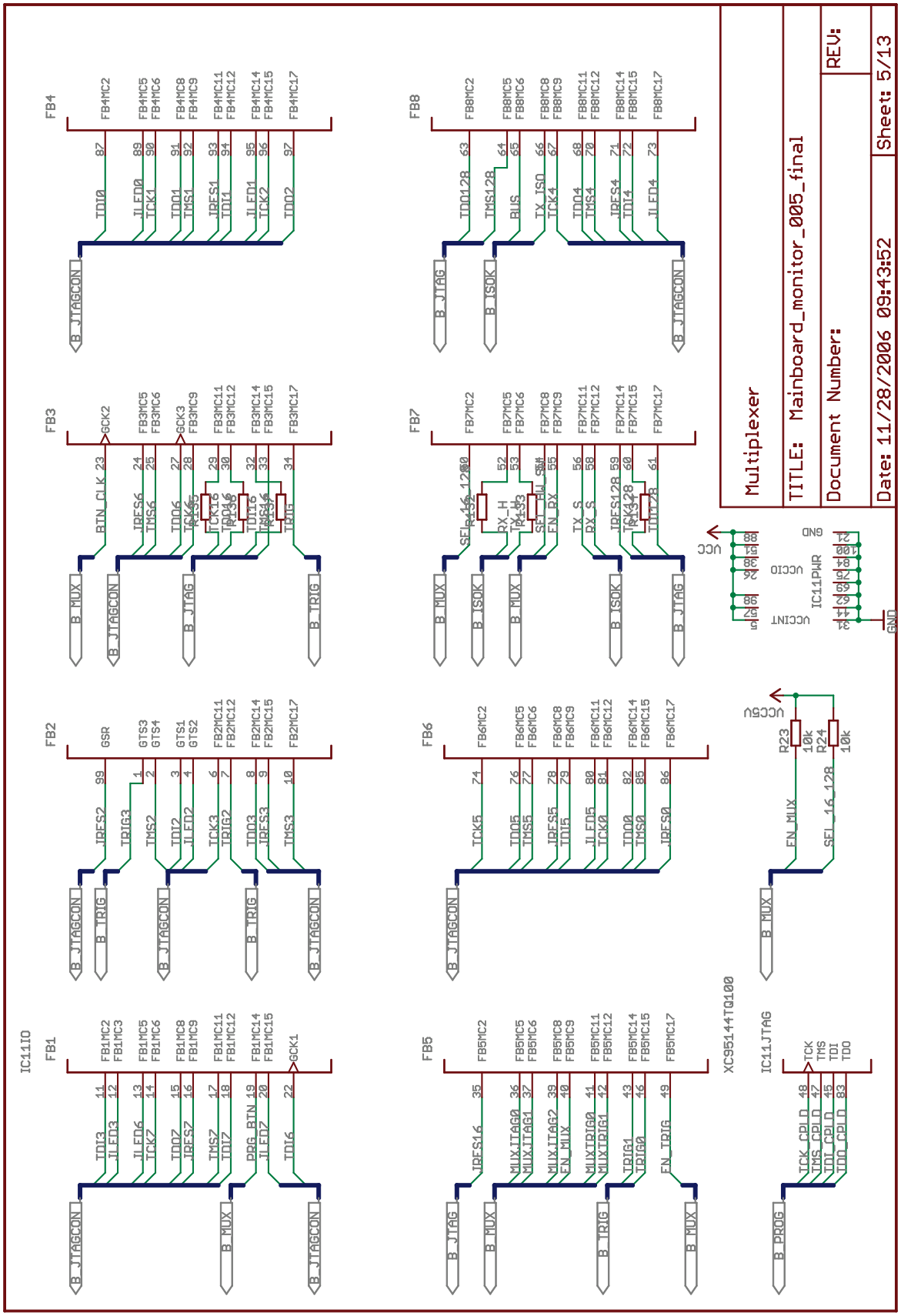
The second schematic shows the sensor PCB.

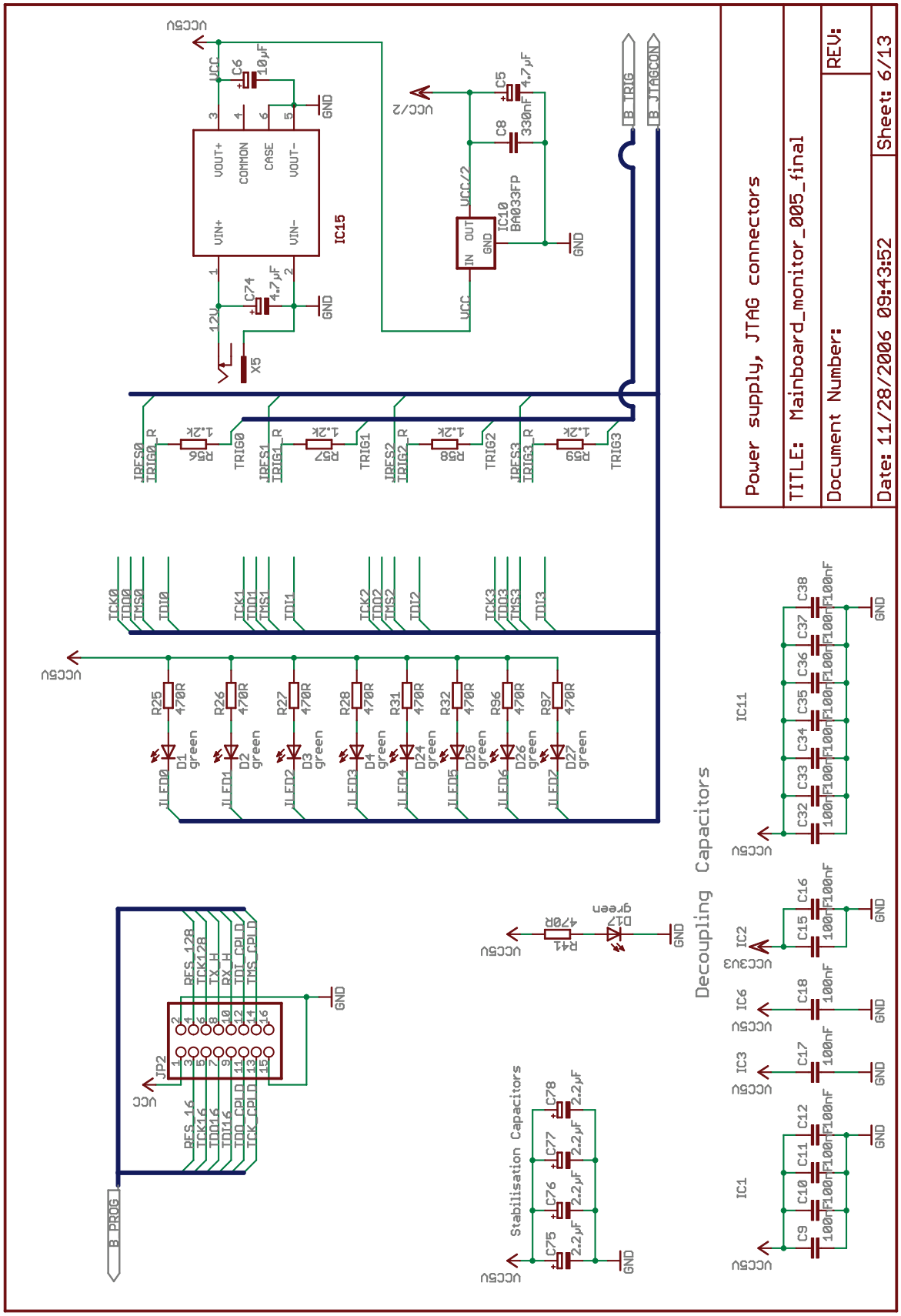

USB HUB
TITLE: Mainboard_monitor_005_final
Document Number:
REV:
Date: 11/28/2006 09:43:52
Sheet: 1/13
IC2
TUSB2046B
SN75240PW
IC4/1
IC4/2
X1
USB
RES HUB
RES HUB
VCC3V3
VCC3V3
GND
DP1_C
DM1_C
D+
D-
DP2_FT232
DM2_FT232
DP3_FT232
DM3_FT232
DP4_FT232
DM4_FT232
R42 27R
R43 27R
R48 27R
R49 27R
R54 1.5k
R55 1.5k
R9 27R
R10 27R
R17 4.7k
R18 10k
C39 27pF
C40 27pF
C43 22pF
C44 22pF
C49 22pF
C50 22pF
C51 22pF
C52 22pF
C58 22pF
C59 22pF
C60 22pF
C61 22pF
Q2 6MHz
L2 Ferrit Bead

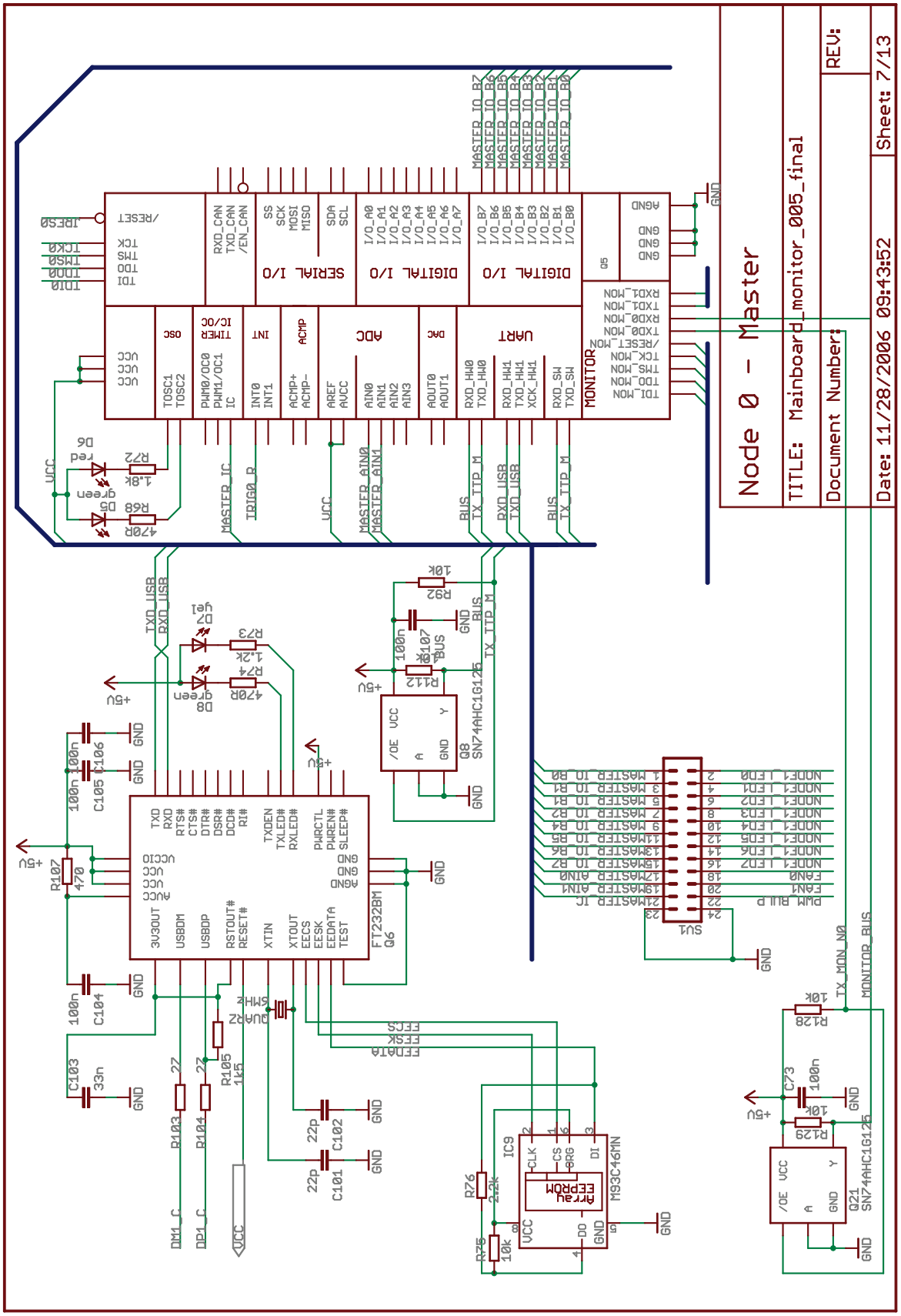

USB to serial converter
TITLE: Mainboard_monitor_005_final
Document Number:
REV:
Date: 11/28/2006 09:43:52
Sheet: 2/13
B_RS232
RXD_A
TXD_A
SI_WUA
RXD_B
TXD_B
VCC
R34
470R
green
D10
R33
1.2K
D9
R49
10k
R39
470R
green
D15
R38
1.2K
D14
RXLED_A
TXLED_A
RXLED_B
TXLED_B
IC1
FT232GP
ADBUS0
ADBUS1
ADBUS2
ADBUS3
ADBUS4
ADBUS5
ADBUS6
ADBUS7
ACBUS0
ACBUS1
ACBUS2
ACBUS3
SI/WUA
BDBUS0
BDBUS1
BDBUS2
BDBUS3
BDBUS4
BDBUS5
BDBUS6
BDBUS7
BCBUS0
BCBUS1
BCBUS2
BCBUS3
SI/WUB
PWREN#
VCCIOA
VCCIOB
VCC
GND
AVCC
AGND
3V3OUT
USBDM
USBDP
RSTOUT#
RESET#
XTIN
XTOUT
EECS
EESK
EEDATA
TEST
VCC5V
R46
470R
C14
100nF
C13
4n7
R13
4.5k
R11
27R
R12
27R
C43
6MHz
C41
27pF
C42
27pF
GND
IC6
M93C46MN
CLK
CS
DI
DO
ORG
VCC
GND
Atmel
EEPROM
R14
2.2k
R45
10k
DP2_FT232
DP2_FT232
VCC

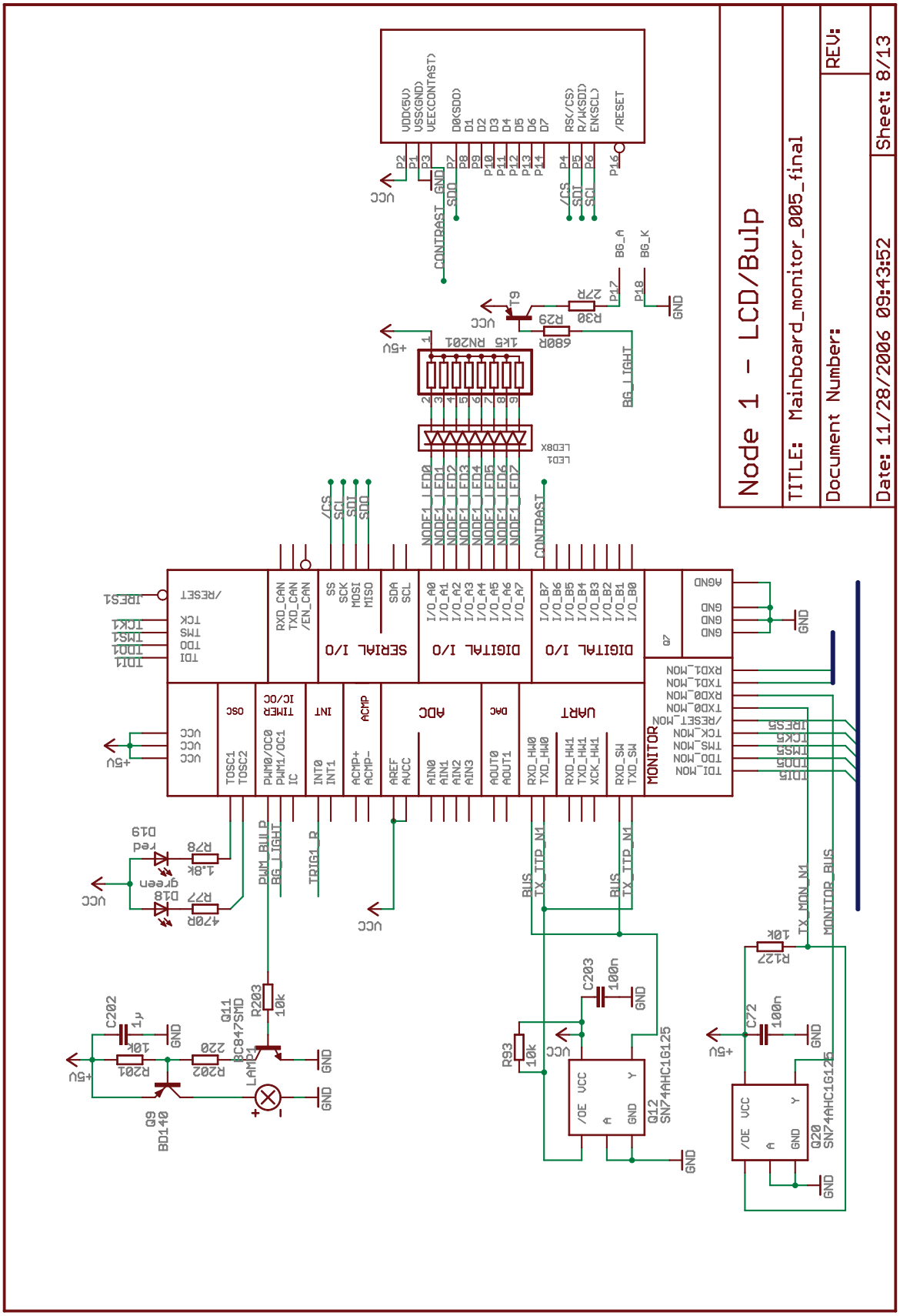

ATmega128
TITLE: Mainboard_monitor_005_final
Document Number:
REV:
Date: 11/28/2006 09:43:52
Sheet: 3/13
IC13
CY7C199
IC14
74573PW
IC7
MEGA128-A
Q15
SN74AHC1G125
B_MEM
B_ISOK
B_PROG
B_JTAG
B_RS232
B_MUX
B_TRIG
TX_ISO
RES_HUB
VCC5U
GND

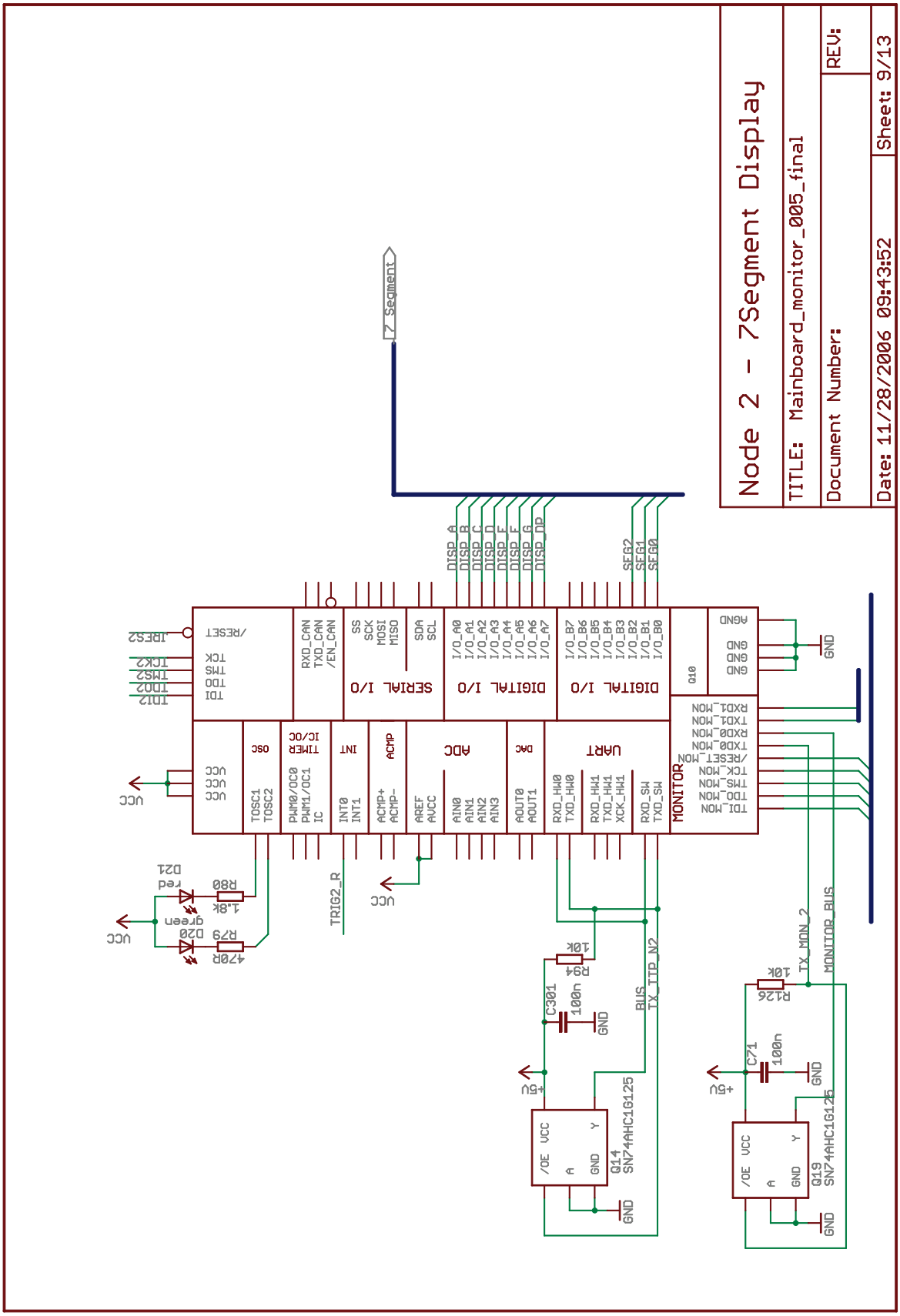

ATmega16
TITLE: Mainboard_monitor_005_final
Document Number:
REV:
Date: 11/28/2006 09:43:52
Sheet: 4/13
IC8
MEGA16-A
B_PROG
B_JTAG
B_RS232
VCC5V
GND
R131 10K
R40 1.2K
R22 4.7k
R21 10K
C29 10nF
C25 100nF
C24 100nF
C28 100nF
C27 100nF
C31 22pF
C30 22pF
Y4 7.3728MHz
TXD_R
RXD_R
TCK16
TDO16
TDI16
TMS16
TDES16

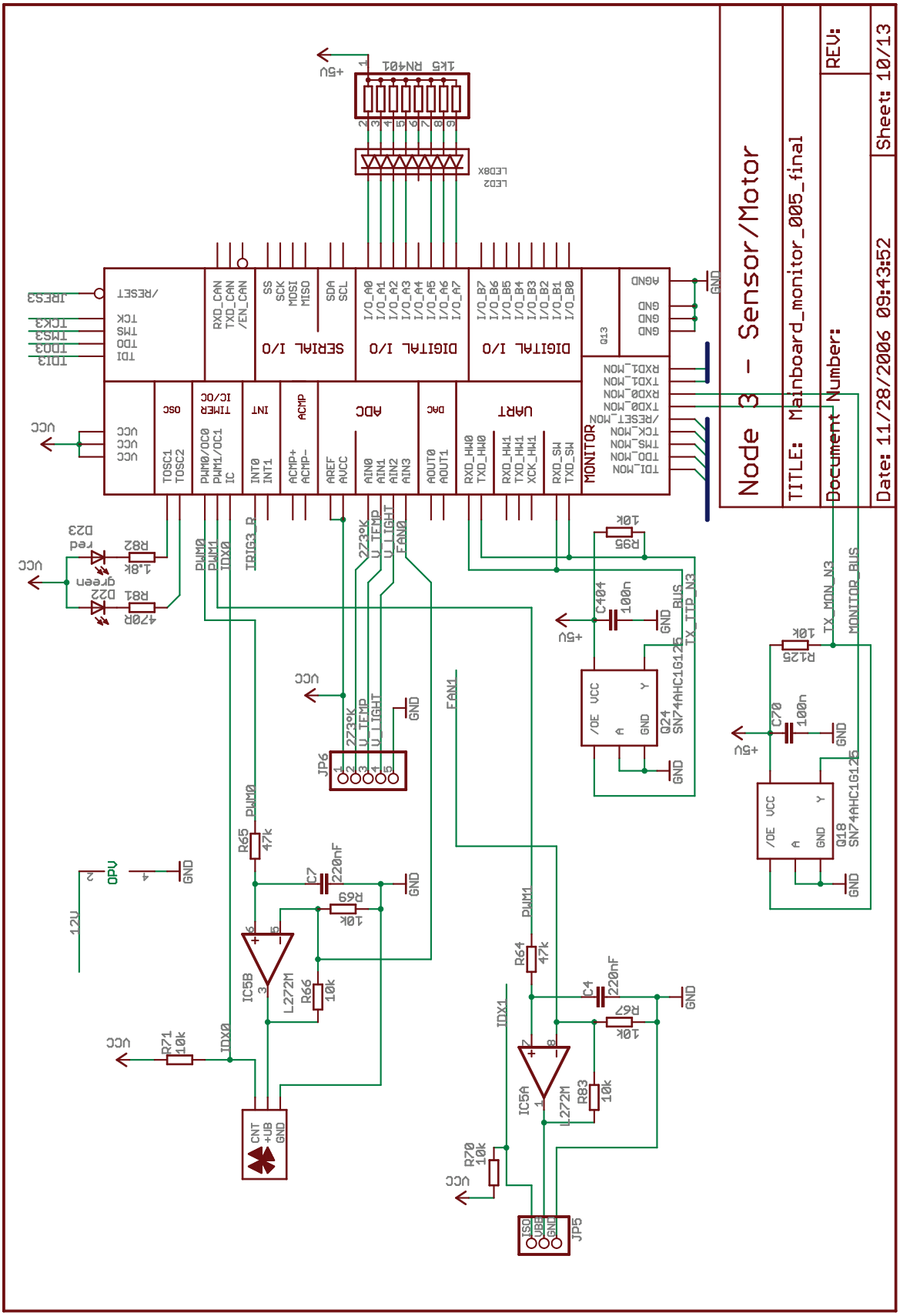

Multiplexer
TITLE: Mainboard_monitor_005_final
Document Number:
REV:
Date: 11/28/2006 09:43:52
Sheet: 5/13
IC11IO
XC95144TQ100
IC11JTAG
IC11PWR
FB1
FB2
FB3
FB4
FB5
FB6
FB7
FB8
B_JTAGCON
B_TRIG
B_MUX
B_JTAG
B_ISOK
B_PROG
VCC
VCC5V
GND
VCCINT
VCCIO
FN_MUX
SEL_16_128
R23
10k
R24
10k
TCK
TMS
TDI
TDO

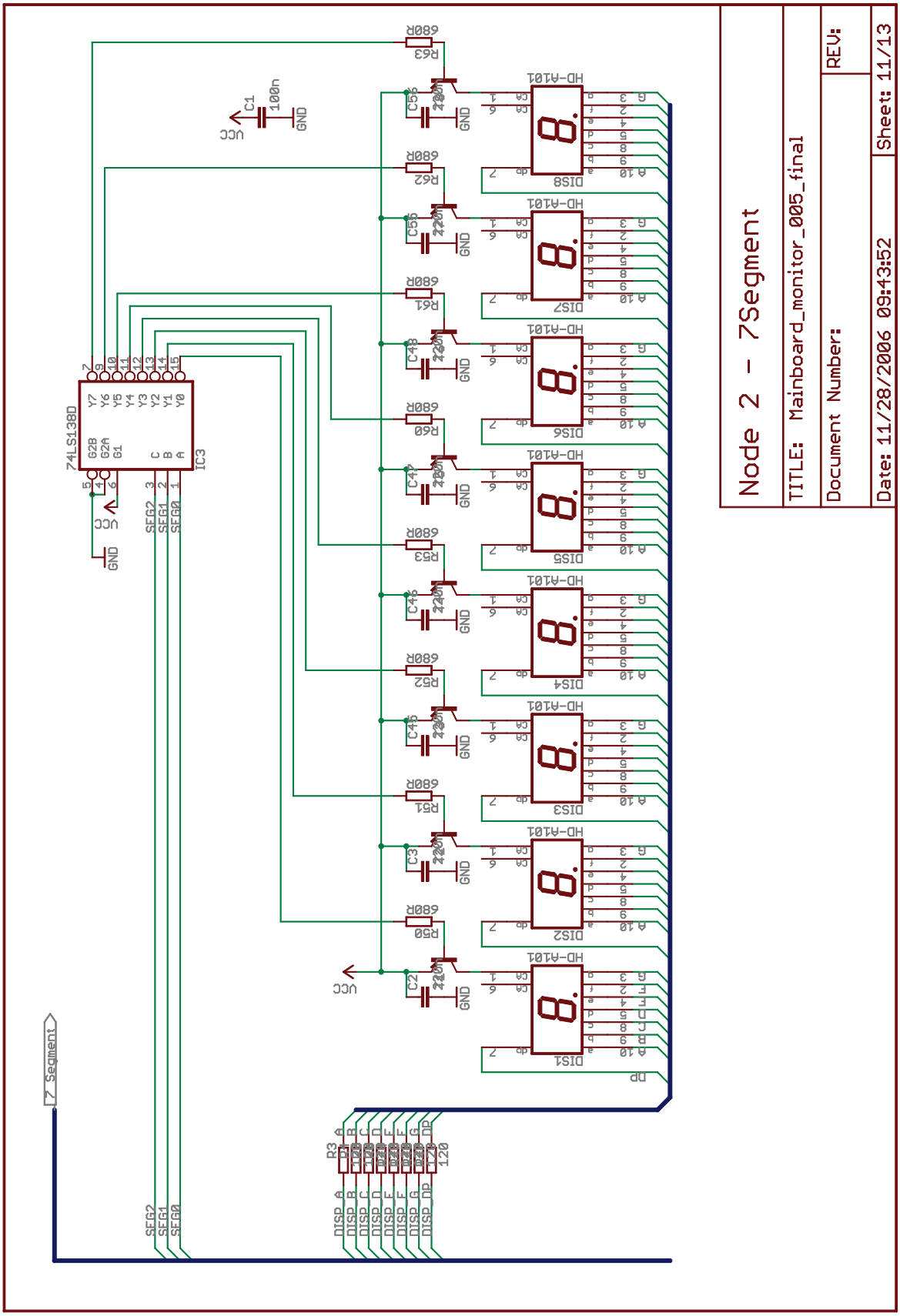

Power supply, JTAG connectors
TITLE: Mainboard_monitor_005_final
Document Number:
REV:
Date: 11/28/2006 09:43:52
Sheet: 6/13
B_TRIG
B_JTAGCON
B_PROG
IC15
IC10 BA033FP
JP2
D1 green
D2 green
D3 green
D4 green
D24 green
D25 green
D26 green
D27 green
D17 green
Decoupling Capacitors
Stabilisation Capacitors
IC1
IC2
IC3
IC6
IC11

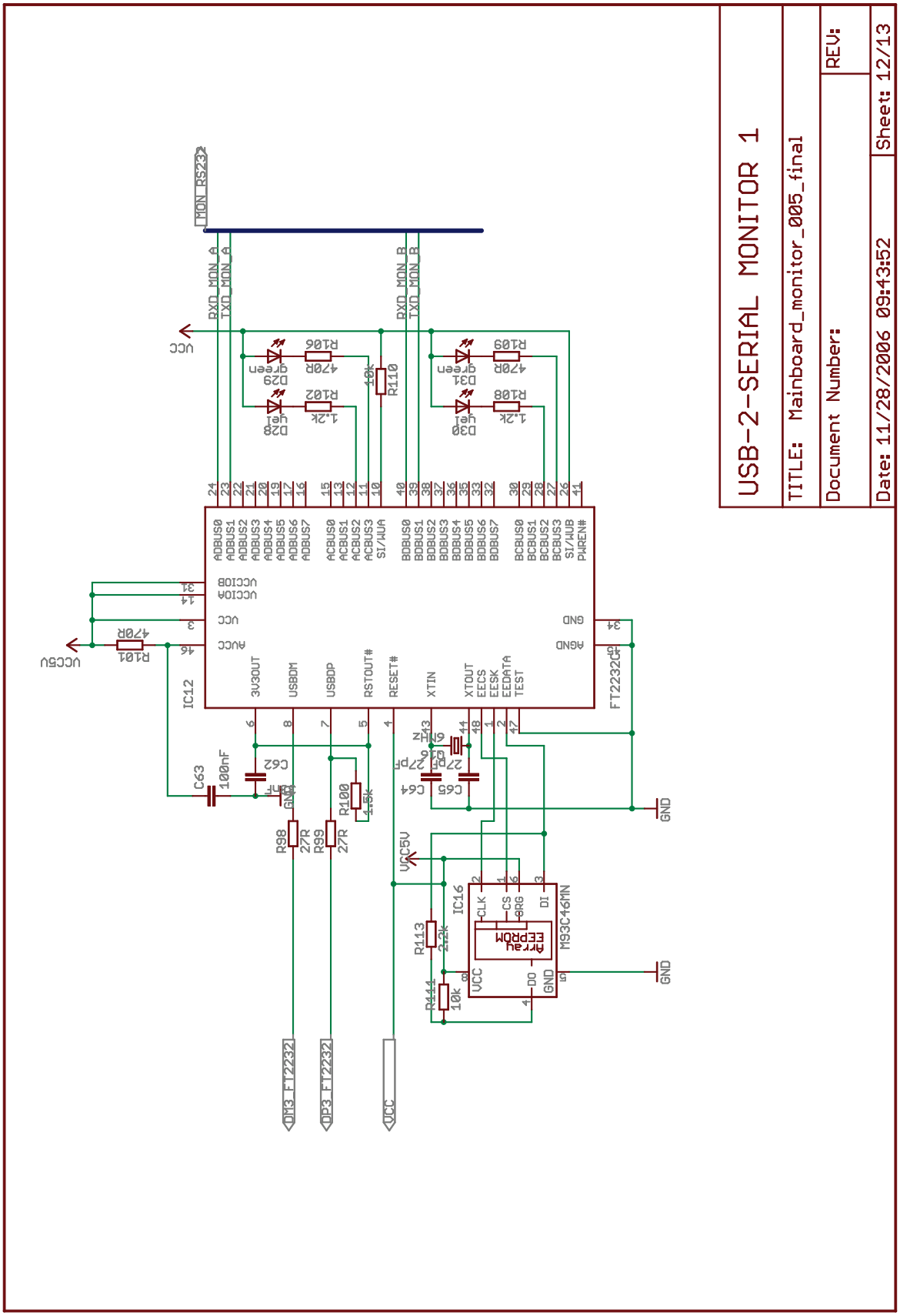

Node 0 - Master
TITLE: Mainboard_monitor_005_final
Document Number:
REV:
Date: 11/28/2006 09:43:52
Sheet: 7/13
SERIAL I/O
DIGITAL I/O
DIGITAL I/O
UART
ADC
MONITOR
FT232BM
SN74AHC1G125
M93C46MN
EEPROM

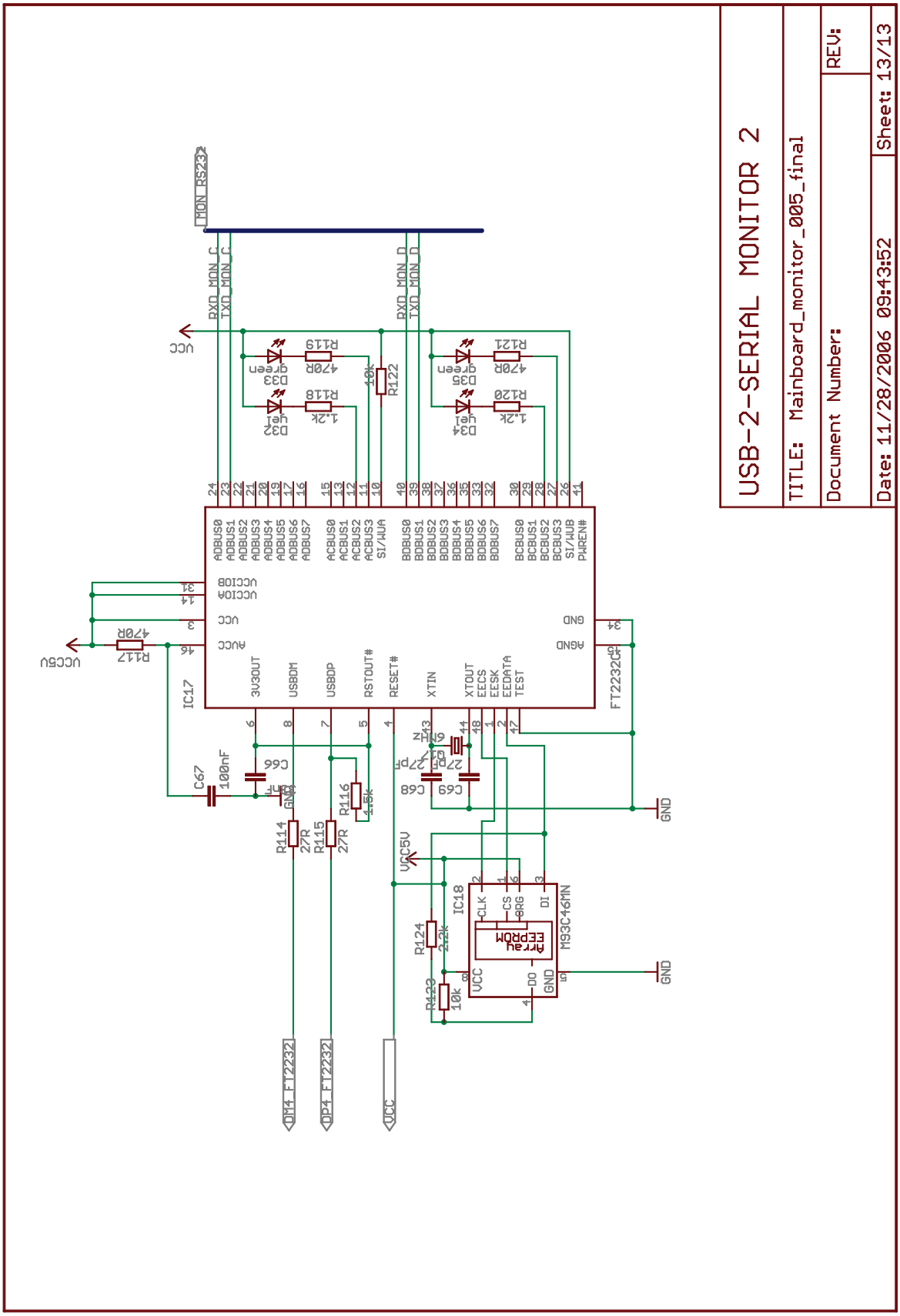

Node 1 - LCD/Bulp
TITLE: Mainboard_monitor_005_final
Document Number:
REV:
Date: 11/28/2006 09:43:52
Sheet: 8/13
MONITOR
DIGITAL I/O
SERIAL I/O
ADC
UART
TIMER
MONITOR_BUS
TX_MON_N1
BG_LIGHT
PWM_BULP
CONTRAST
SN74AHC1G125

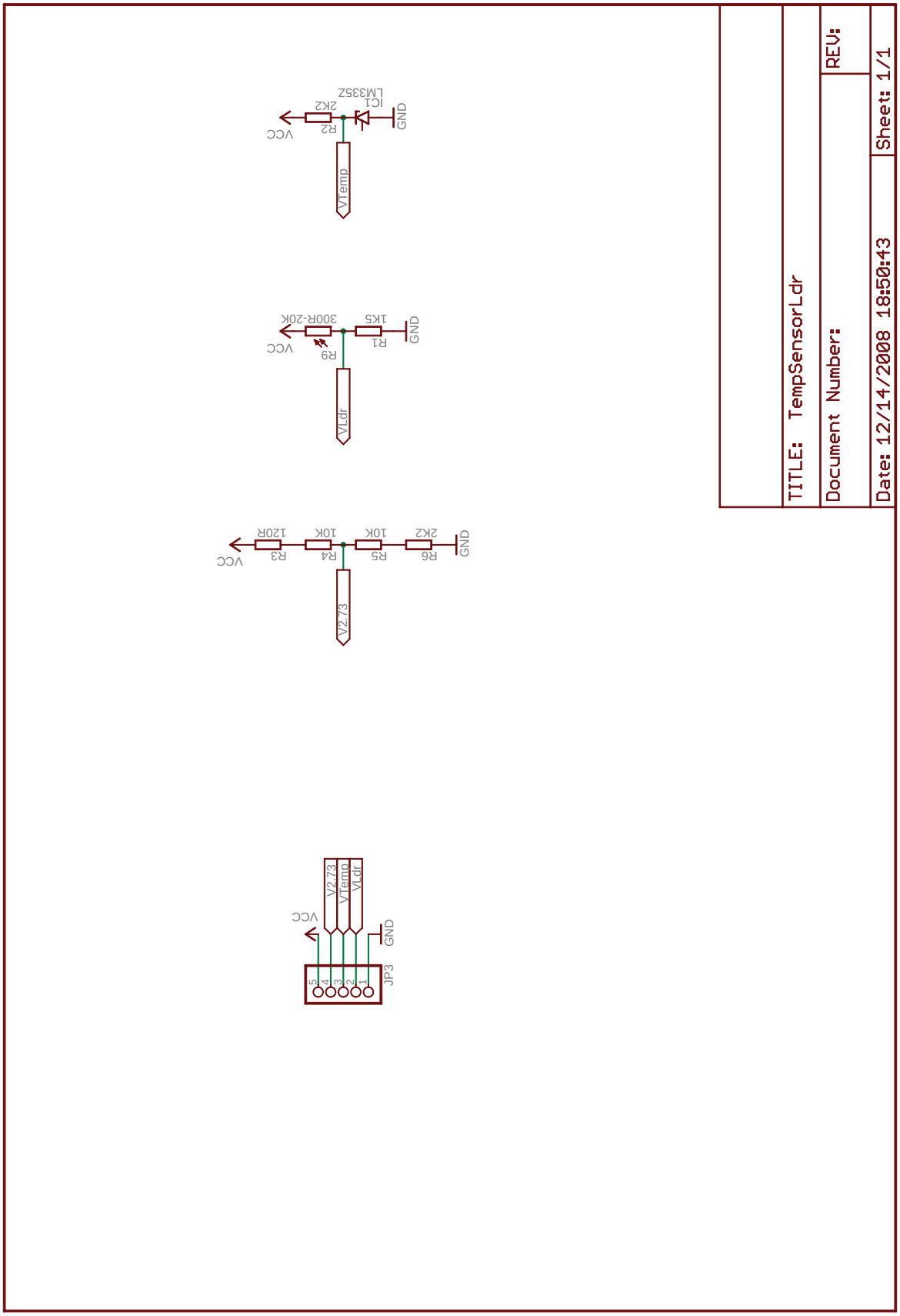

Node 2 - 7Segment Display
TITLE: Mainboard_monitor_005_final
Document Number:
REV:
Date: 11/28/2006 09:43:52
Sheet: 9/13
7_Segment
SERIAL I/O
DIGITAL I/O
DIGITAL I/O
MONITOR
UART
ADC
DAC
OSC
TIMER
INT
ACMP
MONITOR_BUS
TX_MON_2
TX_TTP_N2
TRIG2_R
SN74AHC1G125
GND
VCC
+5V


Node 3 - Sensor/Motor
TITLE: Mainboard_monitor_005_final
Document Number:
REV:
Date: 11/28/2006 09:43:52
Sheet: 10/13
SERIAL I/O
DIGITAL I/O
DIGITAL I/O
MONITOR
UART
ADC
ACMP
INT
TIMER
OSC
MONITOR_BUS
TX_MON_N3
TX_TTP_N3
FAN1
IRIG3_B
PWM0
PWM1
IDX0
IDX1
U_TEMP
U_LIGHT
FAN0
IC5A
IC5B
L272M
SN74AHC1G125
CNT
+UB
GND
VCC
+5V
12V


Node 2 - 7Segment
TITLE: Mainboard_monitor_005_final
Document Number:
REV:
Date: 11/28/2006 09:43:52
Sheet: 11/13
74LS138D
IC3
HD-A101
DIS8
DIS7
DIS6
DIS5
DIS4
DIS3
DIS2
DIS1
680R
R63
R62
R61
R60
R53
R52
R51
R50
100n
C1
C2
C3
C4
C5
C6
C7
C8
VCC
GND
DP
7 Segment
SEG2
SEG1
SEG0
R3
120
DISP_A
DISP_B
DISP_C
DISP_D
DISP_E
DISP_F
DISP_G
DISP_DP

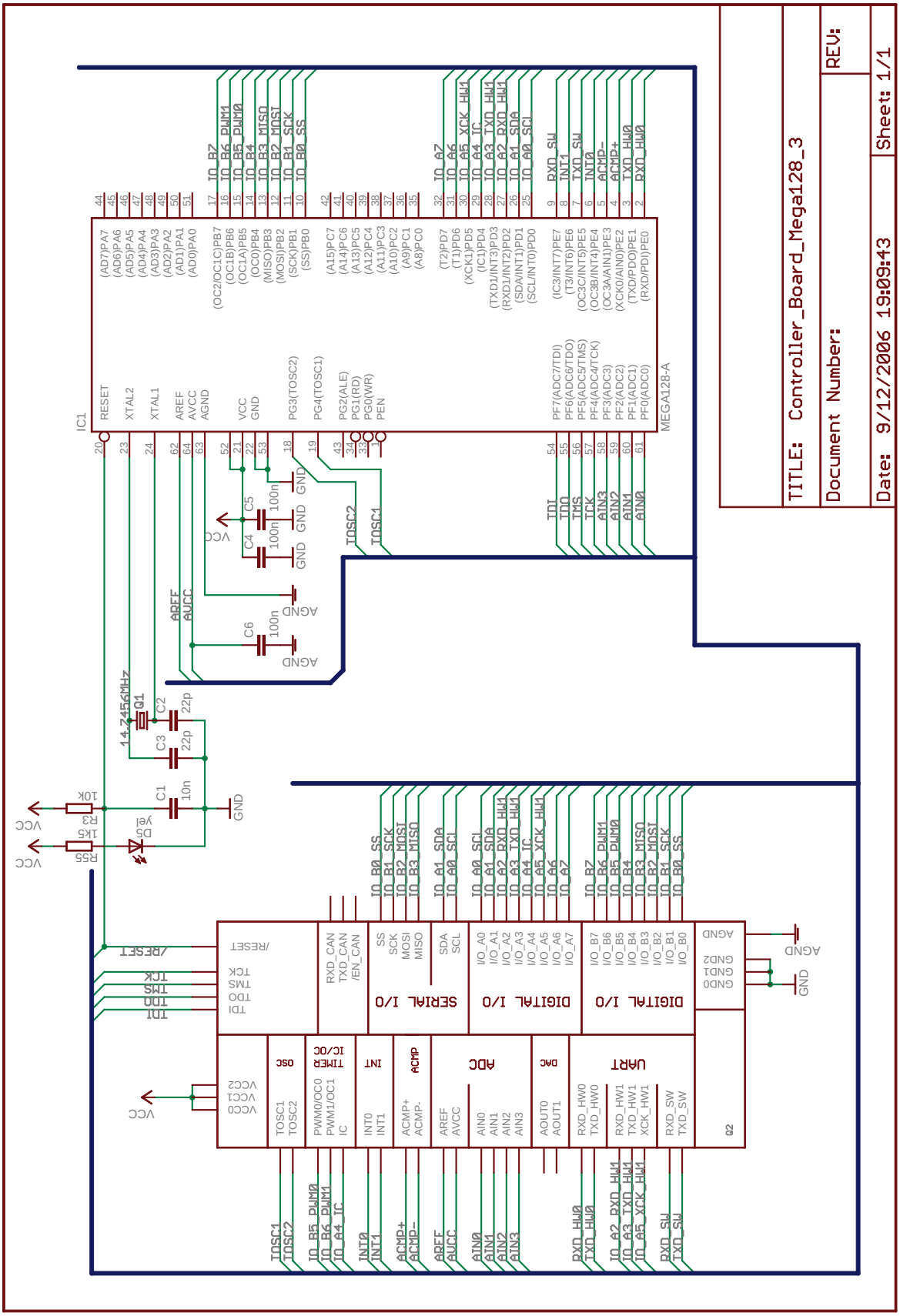

USB-2-SERIAL MONITOR 1
TITLE: Mainboard_monitor_005_final
Document Number:
REV:
Date: 11/28/2006 09:43:52
Sheet: 12/13
MON_RS232
RXD_MON_A
TXD_MON_A
RXD_MON_B
TXD_MON_B
VCC
R106
R102
R110
R109
R108
D29
D28
D31
D30
470R
1.2k
10k
green
IC12
FT2232C
R101
C63
100nF
C62
C64
C65
27pF
R100
4.5k
R98
R99
27R
VCC5V
GND
IC16
M93C46MN
EEPROM
R113
2.2k
R114
10k
DM3_FT2232
DP3_FT2232

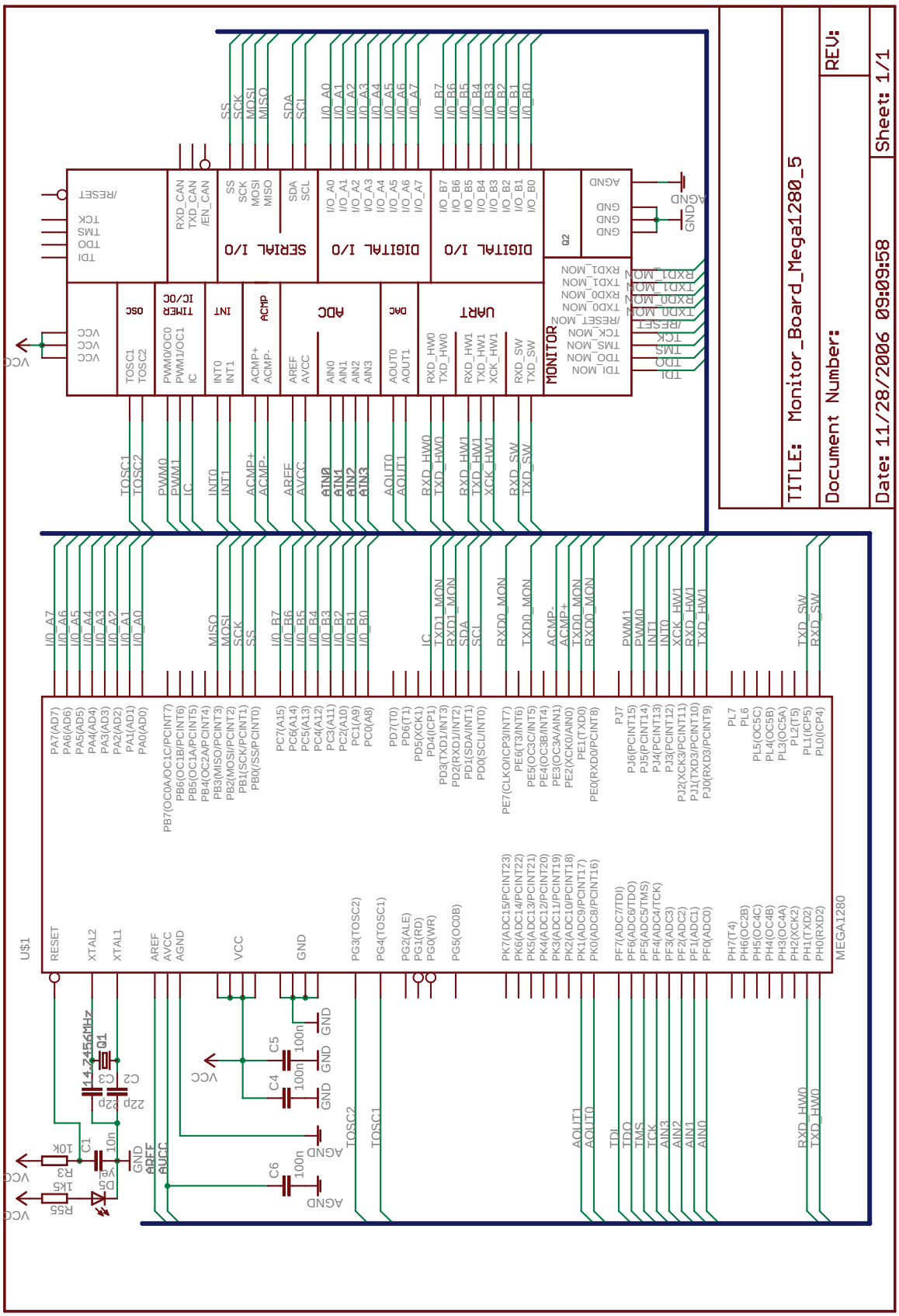

USB-2-SERIAL MONITOR 2
TITLE: Mainboard_monitor_005_final
Document Number:
REV:
Date: 11/28/2006 09:43:52
Sheet: 13/13
MON_RS232
RXD_MON_C
TXD_MON_C
RXD_MON_D
TXD_MON_D
VCC
R119
470R
green
D33
R118
1.2K
red
D32
R122
10k
R121
470R
green
D35
R120
1.2K
red
D34
IC17
FT2232C
ADBUS0
ADBUS1
ADBUS2
ADBUS3
ADBUS4
ADBUS5
ADBUS6
ADBUS7
ACBUS0
ACBUS1
ACBUS2
ACBUS3
SI/WUA
BDBUS0
BDBUS1
BDBUS2
BDBUS3
BDBUS4
BDBUS5
BDBUS6
BDBUS7
BCBUS0
BCBUS1
BCBUS2
BCBUS3
SI/WUB
PWREN#
VCCIOB
VCCIOA
VCC
AVCC
GND
AGND
3V3OUT
USBDM
USBDP
RSTOUT#
RESET#
XTIN
XTOUT
EECS
EESK
EEDATA
TEST
VCC5V
R117
470R
C67
100nF
C66
NF
R114
27R
R115
27R
R116
1.5k
C68
27pF
C69
27pF
C43
NF
IC18
M93C46MN
CLK
CS
DI
DO
ORG
EEPROM
R124
2.2k
R123
10k
GND
DM_FT2232
DP_FT2232
VCC

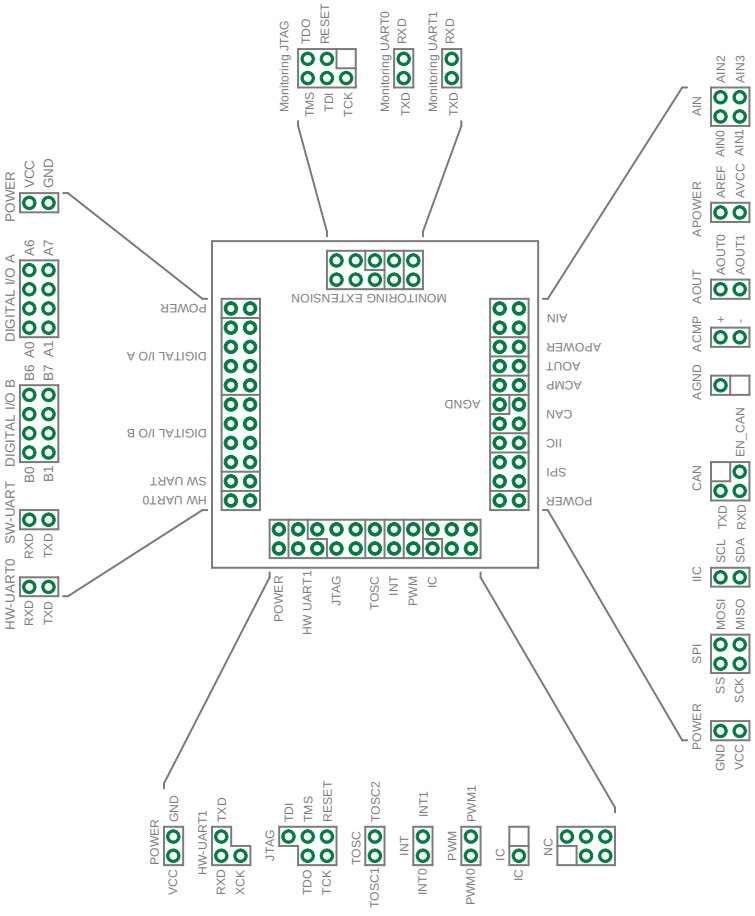

LM335Z
IC1
2K2
R2
VCC
GND
VTemp
300R-20K
1K5
R9
R1
VLdr
120R
10K
10K
2K2
R3
R4
R5
R6
V2.73
V2.73
VTemp
VLdr
VCC
GND
JP3
5
4
3
2
1
TITLE: TempSensorLdr
Document Number:
REV:
Date: 12/14/2008 18:50:43
Sheet: 1/1

C.2 CPU Board Interface

This section contains the schematic for an Atmel ATmega128 microcontroller board, which forms the microcontroller node programmed by the students. The second schematic describes the monitoring node which, in the remote workplace showcase, monitors the students' programs and sends the IO-information to the gateway. At last, the pinout of the socket for the microcontroller nodes and for the monitoring nodes is shown and all signals are named.

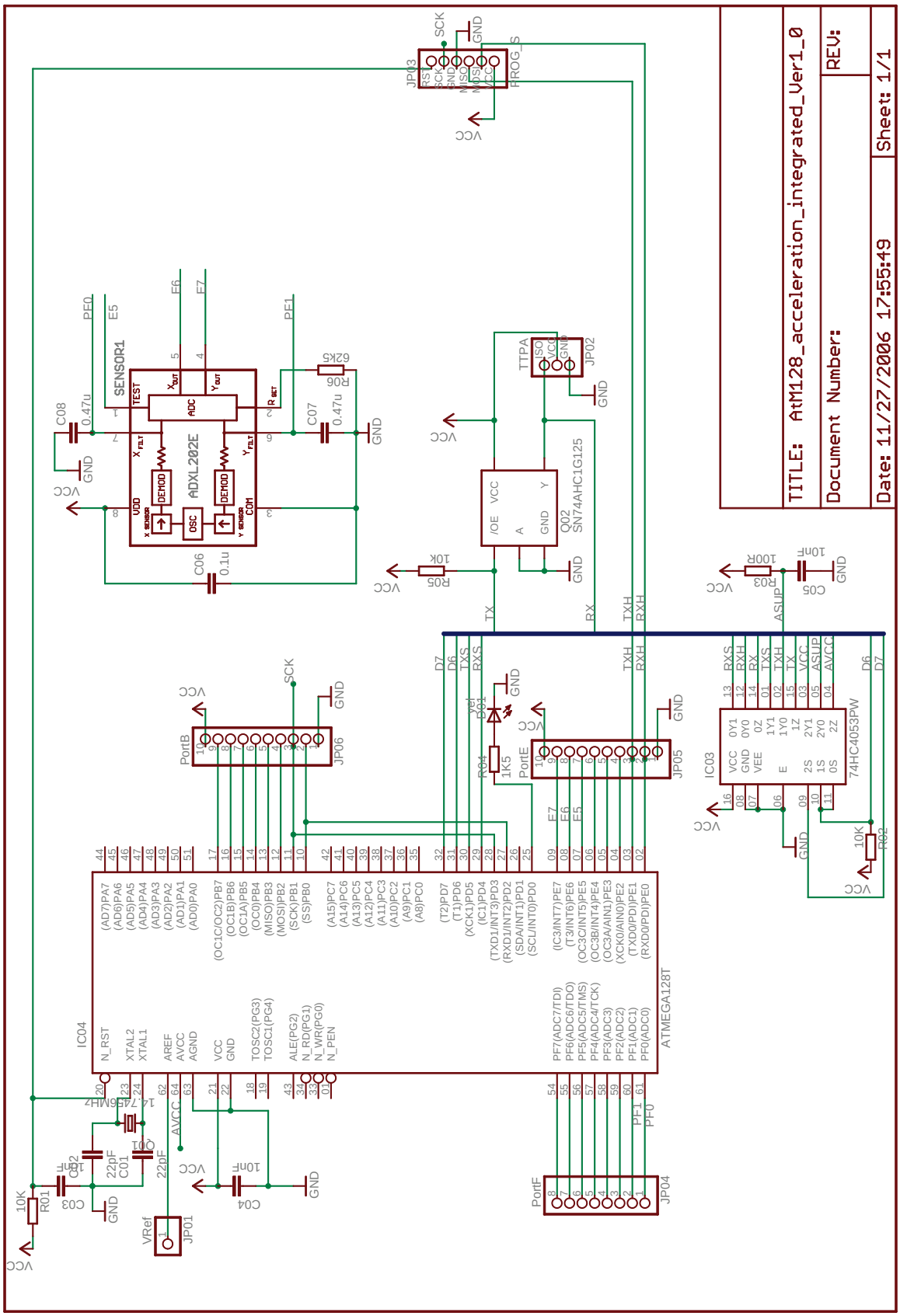

TITLE: Controller_Board_Mega128_3
Document Number:
REV:
Date: 9/12/2006 19:09:43
Sheet: 1/1
IC1
MEGA128-A
Q1
14.7456MHz
C1
C2
C3
C4
C5
C6
R3
R55
D5
SERIAL I/O
DIGITAL I/O
DIGITAL I/O
UART
ADC
ACMP
INT
OSC
DAC
TIMER IC/OC
Q2


TITLE: Monitor_Board_Mega1280_5
Document Number:
REV:
Date: 11/28/2006 09:09:58
Sheet: 1/1
MEGA1280
U$1
SERIAL I/O
DIGITAL I/O
DIGITAL I/O
MONITOR
UART
ADC
OSC
TIMER
INT
ACMP
DAC
14.7456MHz
Q1
C1
C2
C3
C4
C5
C6
R3
R55
D5
10K
1K5
10n
22p
100n
VCC
GND
AGND
AREF
AVCC


Monitoring JTAG
TMS
TDO
TDI
RESET
TCK
Monitoring UART0
TXD
RXD
Monitoring UART1
TXD
RXD
MONITORING EXTENSION
POWER
VCC
GND
DIGITAL I/O A
A6
A7
A0
A1
DIGITAL I/O B
B6
B7
B0
B1
SW-UART
RXD
TXD
HW-UART0
RXD
TXD
POWER
DIGITAL I/O A
DIGITAL I/O B
SW UART
HW UART0
AGND
AIN
APOWER
AOUT
ACMP
CAN
IIC
SPI
POWER
AIN
AIN2
AIN3
AIN0
AIN1
APOWER
AREF
AVCC
AOUT
AOUT0
AOUT1
ACMP
+
-
AGND
CAN
TXD
RXD
EN_CAN
IIC
SCL
SDA
SPI
MOSI
MISO
SS
SCK
POWER
GND
VCC
POWER
HW UART1
JTAG
TOSC
INT
PWM
IC
POWER
VCC
GND
HW-UART1
RXD
TXD
XCK
JTAG
TDI
TDO
TMS
TCK
RESET
TOSC
TOSC1
TOSC2
INT
INT0
INT1
PWM
PWM0
PWM1
IC
IC
NC

C.3 Stand Alone Acceleration Sensor

This sections holds the schematic of an extension node for the ESE–Board. The extension node was designed, because a special acceleration sensor had to be integrated into a TTP/A network. It can be used as base for developing new extension nodes.


TITLE: AtM128_acceleration_integrated_Ver1_0
Document Number:
REV:
Date: 11/27/2006 17:55:49
Sheet: 1/1
SENSOR1
ADXL202E
IC04
ATMEGA128T
IC03
74HC4053PW
Q02
SN74AHC1G125
JP01
JP02
JP03
JP04
JP05
JP06
PortB
PortE
PortF
VCC
GND
SCK
TX
RX
TXH
RXH
ASUP