

Embedded Systems FS 2012

Solution to Lab 1: Setup, Make, Assembler, LEDs

Discussion Date: 07. March 2012

Introduction

The *BTnode* is an autonomous wireless communication and computing platform based on a Bluetooth radio and a microcontroller. It serves as a demonstration platform for research in mobile and ad-hoc connected networks (MANETs) and distributed sensor networks. The BTnode has been jointly developed at ETH Zurich by the Computer Engineering and Networks Laboratory (TIK) and the Research Group for Distributed Systems. Currently, the BTnode is primarily used in the NCCR-MICS research projects.

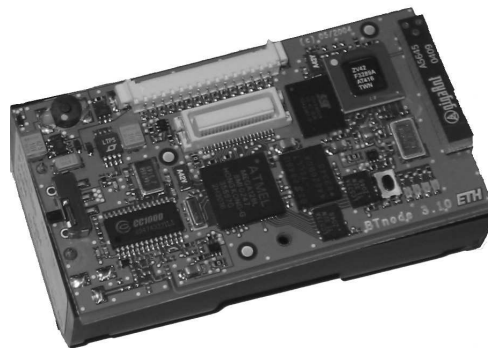


Figure 1: The BTnode rev3

In addition to its Bluetooth radio, the latest BTnode revision (rev3) also features a low-power radio identical to the one used on the Crossbow/Berkeley Mica2 Motes, allowing it to interact with both Mica2-based nodes and previous, Bluetooth-only revisions of the BTnode. Both radios can be operated simultaneously or be independently powered off completely when not in use, considerably reducing the idle power consumption of the device.

BTnodes run an embedded systems OS from the open source domain, called Nut/OS. Nut/OS is designed for the Atmel ATmega128 microcontroller (which is used on the BTnodes) and intentionally kept very simple. According to the Nut/OS description, it features:

- Non preemptive cooperative multi-threading
- Events

- Periodic and one-shot timers
- Dynamic heap memory allocation
- Interrupt driven streaming I/O

In order to use Nut/OS on the BTnodes, a set of BTnode-specific drivers have been added, and in particular a Bluetooth stack for its on-board Bluetooth radio. These three pieces form together the BTnut system software or short BTnut.

More information about the BTnode is available here:

<http://www.btnode.ethz.ch/>

Goals of this Session

In this session, we will start with setting up the development environment consisting of the Eclipse IDE and the BTnode hardware development kit. Then, your first real task will be to upload and run the existing `bt-cmd` application. The `bt-cmd` application opens a command-line terminal on a serial console that allows you to use the Bluetooth radio of the BTnode. For the remainder of this session, you will learn more about programming the BTnode using simple register-based IO. In contrast to BTnut, this is the bare hardware and therefore not very comfortable, but it gives you an inside view to what is actually going on inside the processor.

Task 1: Setup of the Development Environment

Task 1.1: Hardware Setup



Figure 2: The BTnode development kit. The minimal set of tools consists of the three items on the very right: a BTnode, a USB programming board, and a USB cable. Additionally, there are an ISP (In-System-Programmer) and a serial cable (left half).

To setup the hardware, please follow these steps:

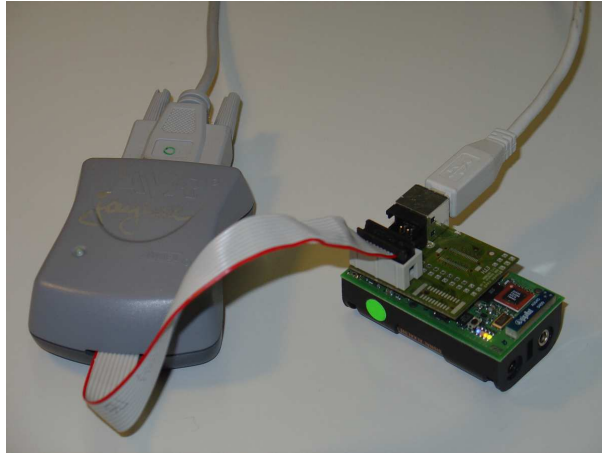


Figure 3: BTnode with connected USB cable and ISP with serial cable. **Attention:** The USB programming board can be inserted in two ways. Make sure that the edges of the USB programming board are aligned with the edges of the BTnode.

- Open your development kit and check that it includes at least all parts that are shown on Figure 2.
- Insert the USB programming board. The edges of the programming board must be aligned with the edges of the BTnode. Compare your setup with Figure 3 by looking on the location of the connectors relative to the BTnode.
- Connect the ISP both to the BTnode programming board as shown and your PC using the serial 9-pin cable.
- Connect the BTnode programming board as shown and your PC with the USB console cable.

Task 1.2: Software Installation

During this lab, we will use Eclipse for editing source code, a Linux terminal for running tasks like compiling software, make for scripting the binary build process, and minicom for connecting to the serial console of the BTnode. A cross-compiler and a linker for the ATmega128 microcontroller are provided by the GNU AVR toolchain. All mentioned components are already installed on your lab PC.

Relevant sources for the Embedded Systems lab are located at the following URL:

<http://www.tik.ee.ethz.ch/tik/education/lectures/ES/lab/lab-exercises.zip>

Setup your software environment:

- Open a Linux terminal
- Use `wget URL` to download the software sources to your home directory
- Start Eclipse
- Let Eclipse create a workspace directory, if you do not already have one. From now on, this location is called `$WORKSPACE`.
- Check that you have opened the C/C++ perspective, switch to this perspective if not
- Import the downloaded archive by running `File > Import > General > Existing Projects into Workspace > Select Archive File`
- You have now imported the Eclipse project `es`. From now on, the location `$WORKSPACE/es/btnut` is called `$BTNUTDIR`.

Task 2: Exploring the BTnode

We will now explore the possibilities of the BTnode with the already existing `bt-cmd` demo application. The `bt-cmd` demo application is a brief example of how to use the Bluetooth radio and protocol stack.

First, we have to install `bt-cmd` on the BTnode:

- Use your Linux terminal to navigate to `$BTNUTDIR/app/bt-cmd`
- Run `make btnode3` to compile the binary image
- Upload the binary image to the BTnode by running `make btnode3 upload`

Now, we use the `minicom` terminal application to connect to the command-line terminal running on the BTnode. The BTnode console is usually connected to `/dev/ttyUSB0` of your Linux machine. The serial terminal session must be configured to `57.6k`, `8N1`, no flow control.

- Run `minicom usb0` in the Linux terminal
- Press the reset button on the BTnode (see Figure 4)
- Hit `Tab` twice to get a selection of possible commands (see Figure 5)
- You can leave Minicom by pressing `Ctrl A-X` (press `Ctrl + A` followed by entering `X`)

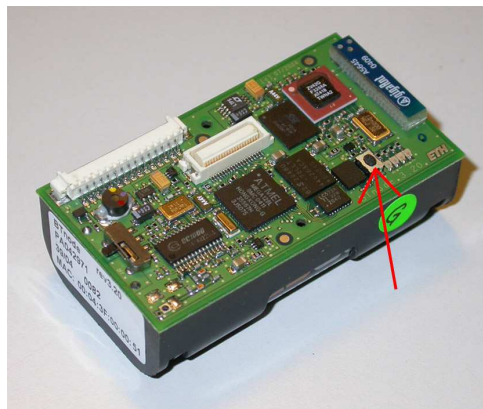


Figure 4: BTnode reset button

```
bt - bluetooth radio commands
led - toggle LED patterns
bat - get the battery status
nut - show OS system information
log - BTnut logging features
```

Figure 5: There are NutOS/BTnode and Bluetooth specific commands (if called without arguments they will show hints on the correct syntax, where applicable).

You are now ready to solve the following questions:

- a) Enable the four LEDs on the BTnode using the `led` command
- b) Try to locate different BTnodes by issuing `bt inquiry sync`.

Task 3: Blink application

Our first own application for the BTnode will be Blink. As the name says, it does nothing more than periodically toggling the LEDs on the BTnode. The reason for this choice is that for any further work with the BTnodes, we need some kind of feedback from the programs we implement. In general, LEDs offer a valuable feedback when programming embedded devices that often have no screen.

Task 3.1: Accessing LEDs on the BTnode

The LEDs are an off-chip resource. Unfortunately, accessing the LEDs is a bit tricky and requires some knowledge about the design of the BTnode: The address bus of the ATmega128 is 16 bit wide and it is mapped to the ports A (lower 8 bits) and C (upper 8 bits). You can see this on the “BTNODE CORE” page of the attached BTnode schematics. The address bus is mainly needed to access the external SRAM (AMIC_LP62S2048), but at the same time it is also connected to a latch (TI_SN74LVC573A) multiplexing the LEDs and other outputs (see “BTNODE POWER I/O” page of BTnode schematics). To set or clear LEDs, the bits that determine whether the LEDs should be on or off have to be put on the address bus. Then the latch is enabled, i.e. it samples the value on the address bus. After a while, the latch is disabled, i.e. it holds the previously sampled value. The following C code and the corresponding Assembler program do exactly this:

<pre>BTNUTDIR/ex/lab1-blinkc/blink.c #include <avr/io.h> void init(void) { // enable external memory bus MCUCR = 1<<SRE; // set latch pin as output DDRB = 1<<DDB5; } int main(void) { volatile char * pointer; char dummy; init(); // compute the pseudo-address that // contains the values for the LEDs pointer = (char *) (((short)0x2) << 8); // force the compiler to write this // pseudo-address to the address-bus dummy = *pointer; // now enable the latch PORTB = 1<<PB5; // wait a moment - volatile keeps the // compiler from removing this line asm volatile ("nop" ::); // disable the latch, i.e. hold the value PORTB &= ~(1<<PB5); return 0; }</pre>	<pre>BTNUTDIR/ex/lab1-blinkasm/blink.S #include <avr/io.h> .global main init: ; enable interface for external memory ldi r24, 0x80 ; SRE out _SFR_IO_ADDR(MCUCR), r24 ; set latch pin as output sbi _SFR_IO_ADDR(DDRB), DDB5 ret main: rcall init ; set pseudo-address on bus lds r24, 0x0200 ; set latch sbi _SFR_IO_ADDR(PORTB), PB5 ; wait for one cycle nop ; clear latch cbi _SFR_IO_ADDR(PORTB), PB5 ret</pre>
---	---

Figure 6: C and Assembler code of first Blink application.

Take a look at the different accesses to hardware registers. First, the SRE (extern SRAM enable) bit in the MCUCR (MCU General Control Register) register must be set to tell the processor that there is an

external SRAM bus. Before the latch can be used, the control line for controlling the write access must be configured. This is done by defining pin 5 as an output in the data direction register DDRB. While this is the minimum setup for accessing our off-chip LEDs, more initialization code is necessary for running more complex functions.

The full instruction set of the AVR microcontroller can be accessed at

http://www.btnode.ethz.ch/pub/uploads/Documentation/avr_instructionset.pdf

You can access existing source code templates under `$BTNUTDIR/ex/lab1-blinkasm` and `$BTNUTDIR/ex/lab1-blinkc`. Run `make` in these directories to compile the source code and `make upload` to upload the compiled code to the BTnode.

Please work on the following tasks:

- Enable the blue LED on the BTnode. Read the BTnode schematics to find out which address must be set on the address bus. You need to change only one line of code in both source code examples.
- In the `$BTNUTDIR/ex/lab1-blinkasm` folder, run `avr-objdump -d blink.S.o`. Open another terminal and run `avr-objdump -d blink.S.elf` in the same folder. Compare the both outputs. Which parts are equal, how do both outputs differ? Why?
- Compare the output of `avr-objdump -d blink.S.o` from the `$BTNUTDIR/ex/lab1-blinkasm` folder with the output of `avr-objdump -d blink.c.o` from the `$BTNUTDIR/ex/lab1-blinkc` folder. Actually, both pieces of code should fulfill the same function. Why is the output of `avr-objdump -d blink.c.o` larger?

Solution a)

For enabling the blue LED, the address 0x0100 must be used.

```
lds    r24, 0x0100

pointer = (char *) (((short)0x1) << 8);
```

Solution b)

The output of `avr-objdump -d blink.S.o` is equal to the written Assembler code. In contrast, `blink.S.elf` is already fully linked and also contains an interrupt vector that defines to which address the program counter has to be set when a specific interrupt occurs. The added instructions are provided by the AVR libraries.

Solution c)

The compiled C program needs already more space compared to the bare Assembler code. First, the C compiler added code for initializing the stack pointer. Second, the return value is saved to registers before the function finally returns.

Task 3.2: Period Toggling of BTnode LEDs

Our LEDs are now turned on, but they do not blink. Implement a blinking LED in both C and Assembler so that the LED periodically changes its state.

- Implement periodic toggling of the BTnode LEDs both in your C and your Assembler program.
- Estimate the period in which the LED is toggled in the Assembler version. Calculate the period length as accurate as possible by analyzing the Assembler code. BTnodes are running at 8 MHz, use the AVR Instruction Set Manual for reference on execution times.

<pre style="margin: 0;">\$BTNUTDIR/ex/lab1-blinkc/wait.c void waitabit() { int i, j; for(i = 0; i <= 1200; i++) { for(j = 0; j <= 1200; j++) { asm volatile ("nop" :::); } } }</pre>	<pre style="margin: 0;">\$BTNUTDIR/ex/lab1-blinkasm/wait.S #define outerwait r17 #define innerwait r18 #define middlewait r19 loop: ; do something rcall waitabit rjmp loop waitabit: ldi outerwait,0xB4 waitouterloop: ldi middlewait,0x96 waitmiddleloop: ldi innerwait,0x64 waitinnerloop: subi innerwait,0x01 brne waitinnerloop subi middlewait,0x01 brne waitmiddleloop subi outerwait,0x01 brne waitouterloop ret</pre>
--	--

Figure 7: C and Assembler example for busy waiting

c) The given Assembler code for busy waiting is based on three nested loops. What could be done to use less loops but still keeping the code size small? Think about the location of the counter variables in the example.

Solution b)

The following execution times are given from the AVR Instruction Set Manual:

brne (taken)	2 cycles
brne (not taken)	1 cycle
ldi	1 cycle
rcall	3 cycles
ret	4 cycles
subi	1 cycle

This yields the following formula:

$$\begin{aligned}
 C_{inner} &= l_{inner} \cdot (1 + 2) - 1 \\
 C_{middle} &= l_{middle} \cdot 1 + l_{middle} \cdot C_{inner} + l_{middle} \cdot (1 + 2) - 1 \\
 C_{outer} &= l_{outer} \cdot 1 + l_{outer} \cdot C_{middle} + l_{outer} \cdot (1 + 2) - 1 \\
 C_{total} &= C_{outer} + 3 + 4 \\
 &= l_{outer} \cdot (3 + (l_{middle} \cdot (3 + l_{inner} \cdot 3))) + 6
 \end{aligned}$$

Here, C_{inner} etc. denote the number of cycles, while l_{inner} etc. describe the number of iterations in the specific loop level. For the given parametrization from the source code, this results in 8.181.546 cycles or approximately 1,02 seconds for each call of the wait function.

Solution c)

In the given code, counter variables are stored in 8 bit registers. With two loops, the maximum number of iterations is $255 \cdot 255 = 65025$. This is too small for waiting approximately 8 million cycles.

Counter variables larger than 8 bit could be stored in the RAM. However, more Assembler instructions are necessary for working with those counters.

Solution source code

```
#include <avr/io.h>

void init(void) {
    // enable external memory bus
    MCUCR |= 1<<SRE;
    // set latch pin as output
    DDRB |= 1<<DDB5;
}

void write_led(short value) {
    volatile char * pointer;
    char dummy;

    // compute the pseudo-address that contains the values for the LEDs
    pointer = (char *) (((short)value) << 8);
    // force the compiler to write this pseudo-address to the address-bus
    dummy = *pointer;
    // now enable the latch
    PORTB |= 1<<PB5;
    // wait a moment - volatile keeps the compiler from removing this line
    asm volatile ("nop" :::);
    // disable the latch, i.e. hold the value
    PORTB &= ~(1<<PB5);
}

void waitabit() {
    int i, j;
    for(i = 0; i <= 1200; i++) {
        for(j = 0; j <= 1200; j++) {
            // wait a moment - volatile keeps the compiler from removing this line
            asm volatile ("nop" :::);
        }
    }
}

int main(void)
{
    init();

    for(;;) {
        write_led(0x1);
        waitabit();
        write_led(0x0);
        waitabit();
    }

    return 0;
}
```

Figure 8: Code of extended Blink C application

```

#include <avr/io.h>

#define outerwait r17
#define innerwait r18
#define middlewait r19

.global main

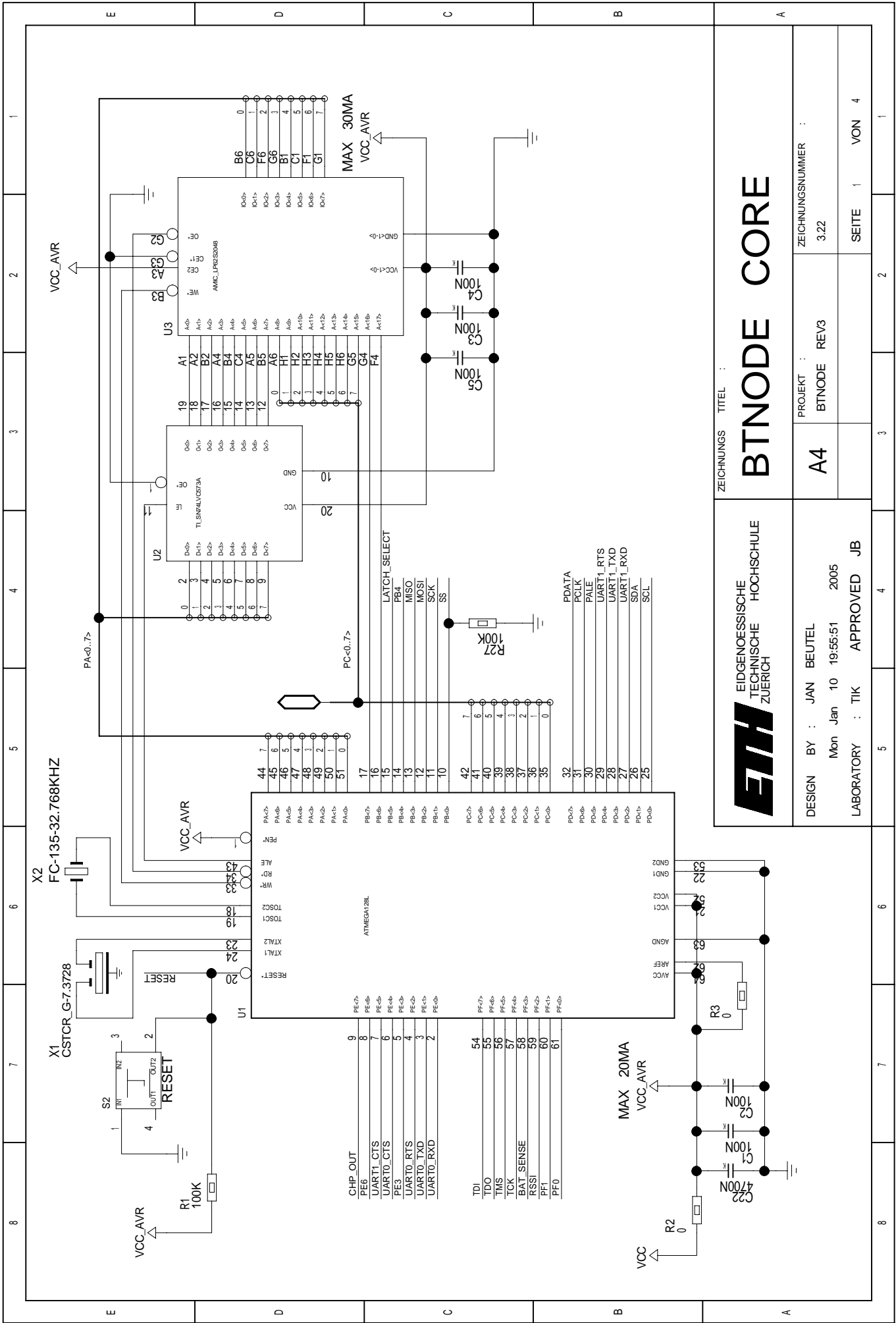
init:
    ; enable interface for external memory
    ldi    r24, 0x80 ; SRE
    out    _SFR_IO_ADDR(MCUCR), r24
    ; set latch pin as output
    sbi    _SFR_IO_ADDR(DDRB), DDB5
    ret

main:
    rcall    init
loop:
    lds    r24, 0x0100
    sbi    _SFR_IO_ADDR(PORTB), PB5
    nop
    cbi    _SFR_IO_ADDR(PORTB), PB5
    rcall    waitabit
    lds    r24, 0x0000
    sbi    _SFR_IO_ADDR(PORTB), PB5
    nop
    cbi    _SFR_IO_ADDR(PORTB), PB5
    rcall    waitabit
    rjmp    loop

waitabit:
    ldi    outerwait,0xB4
waitouterloop:
    ldi    middlewait,0x96
waitmiddleloop:
    ldi    innerwait,0x64
waitinnerloop:
    subi    innerwait,0x01
    brne    waitinnerloop
    subi    middlewait,0x01
    brne    waitmiddleloop
    subi    outerwait,0x01
    brne    waitouterloop
    ret

```

Figure 9: Code of extended Blink Assembler application



ZEICHNUNGS TITEL :

BTNODE CORE

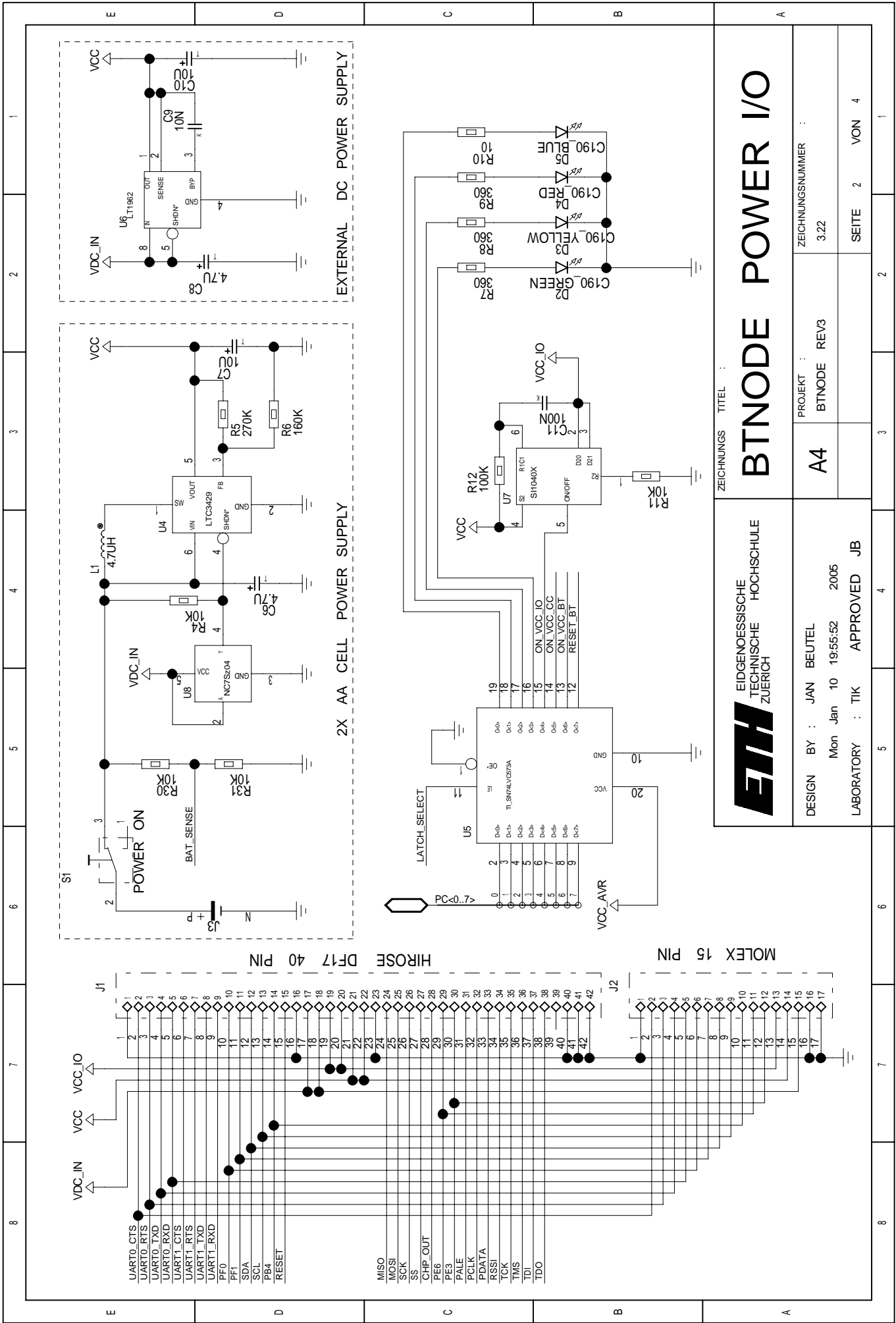
EIDGENÖSSISCHE
TECHNISCHE HOCHSCHULE
ZÜRICH

PROJEKT :
BTNODE REV3

ZEICHNUNGSNUMMER :
3.22

DESIGN BY : JAN BEUTEL
Mon Jan 10 19:55:51 2005

LABORATORY : TIK APPROVED JB



ZEICHNUNGS TITEL :

BTNODE POWER I/O

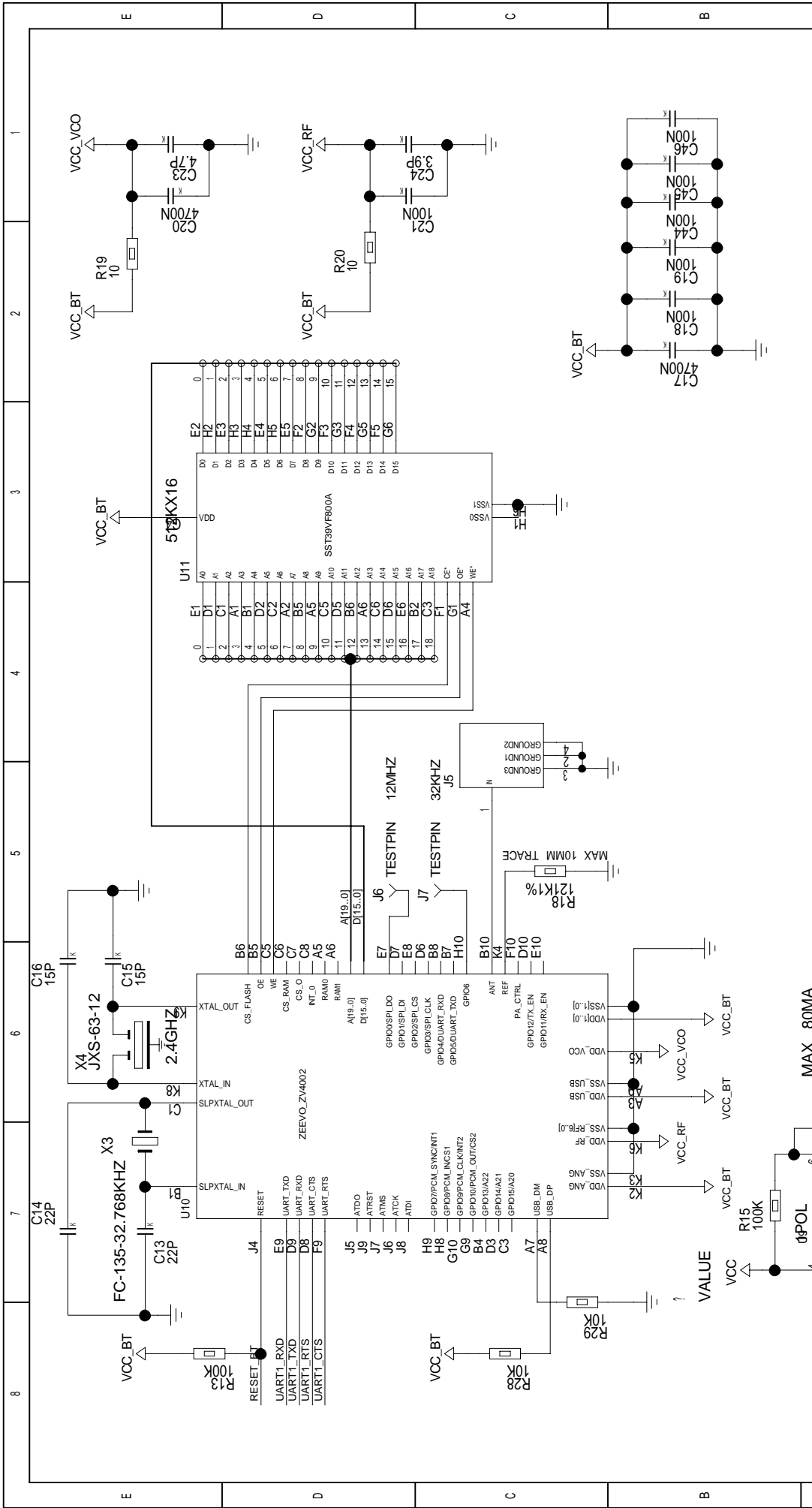
PROJEKT : BTNODE REV3
ZEICHNUNGSNUMMER : 3.22

DESIGN BY : JAN BEUTEL
Mon Jan 10 19:55:52 2005

LABORATORY : TIK APPROVED JB

EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE ZÜRICH

SEITE 2 VON 4



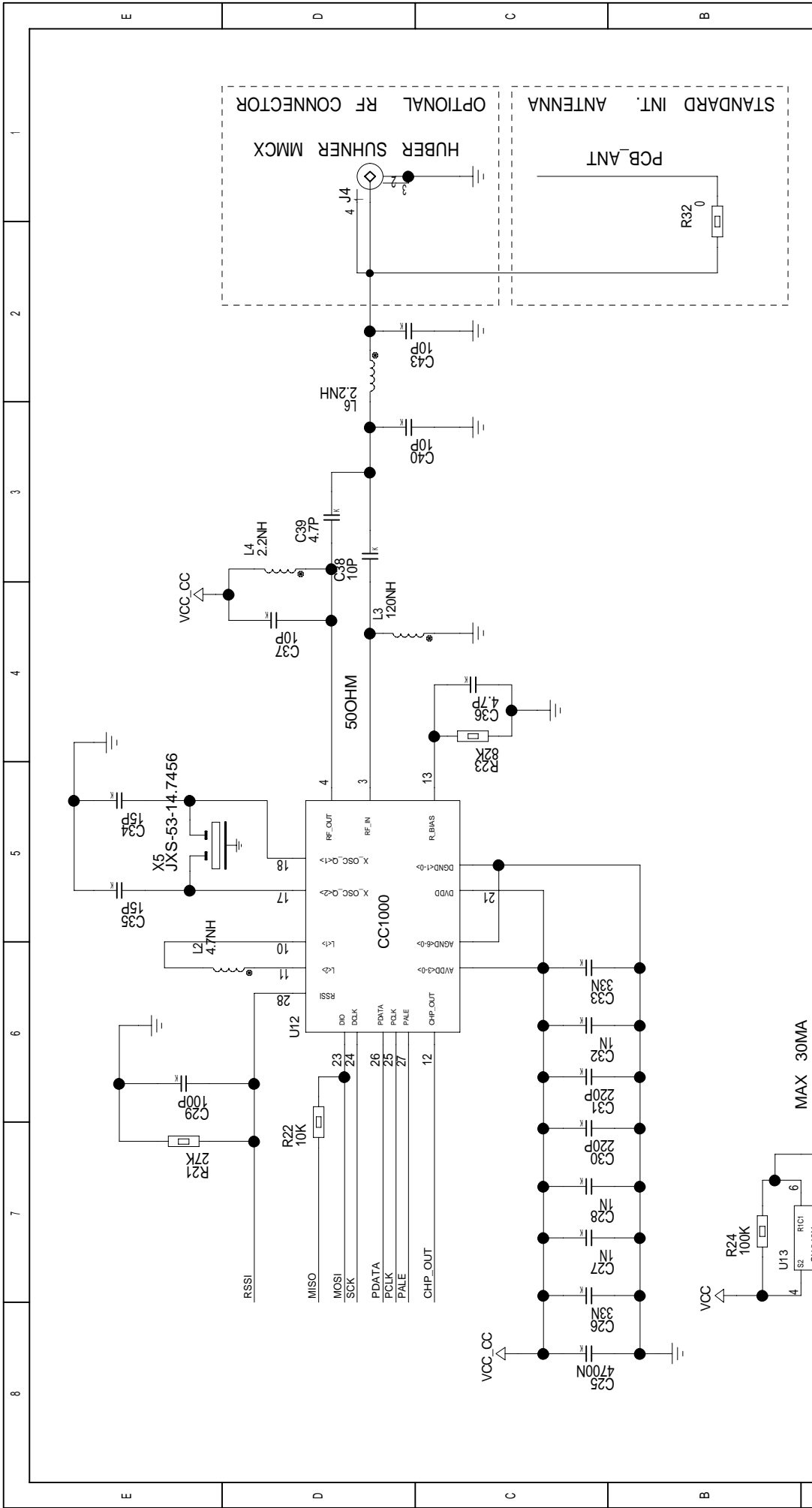
ZEICHNUNGS TITEL :


BTNODE BLUETOOTH

DESIGN BY : JAN BEUTEL
 Mon Jan 10 19:55:52 2005

LABORATORY : TIK APPROVED JB

A4	PROJEKT : BT NODE REV 3	ZEICHNUNGSNUMMER : 3.22	SEITE 3 VON 4
----	----------------------------	----------------------------	---------------



 EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE ZÜRICH		ZEICHNUNGS TITEL : <h1 style="margin: 0;">BTNODE LP RADIO</h1>	
DESIGN BY : JAN BEUTEL Mon Jan 10 19:55:53 2005 LABORATORY : TIK	PROJECT : BTNODE REV3	ZEICHNUNGSNUMMER : 3.22	SEITE 4 VON 4