

ETH Course 402-0248-00L: Electronics for Physicists II (Digital)

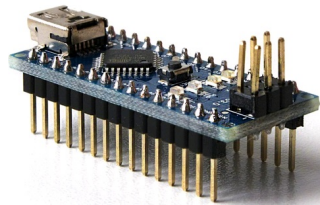
- **Taught by:** Tobi Delbruck, Alejandro Linares
- **Department:** ETH Zurich Department of Physics (D-PHYS)
- **Day/Time:** Weekly on Fridays from 1315 - 1700 in Picaardsaal, ETH Hoenggerberg Campus, [HPT](#) Room C103
- **Breaks:** No class in some weeks, check schedule on wiki.
- **Language:** English.
- **Credits:** 4 credit points.
- **Exam:** There is no exam but students must successfully complete the class exercises. Attendance sheet will be used.
- **Class wiki:** google “dig delbruck”

www.ini.uzh.ch/~tobi/wiki/doku.php?id=dig:start

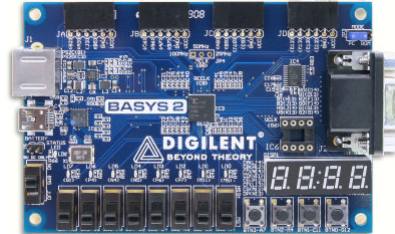
Prerequisites

- **Companion course:** [Electronics for Physicists I \(Analog\)](#), Fall semester, taught by Roland Horisberger.
- The digital course complements the analog course by teaching how to build systems that convert and process analog information.
- You should have had some programming experience, preferably with C. Students (or at least each group of 2-3 students) need a laptop computer, Windows or Linux (but only Windows supported by Tobi). Mac OS can use VM.

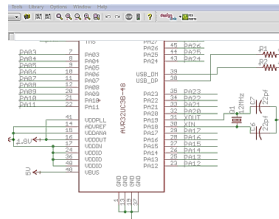
1st half: Embedded systems with microcontrollers



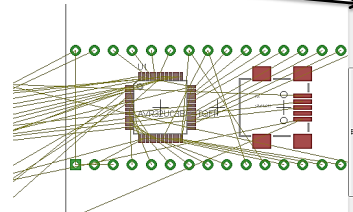
2nd half: Logic design with FPGA



PCB SMD assembly



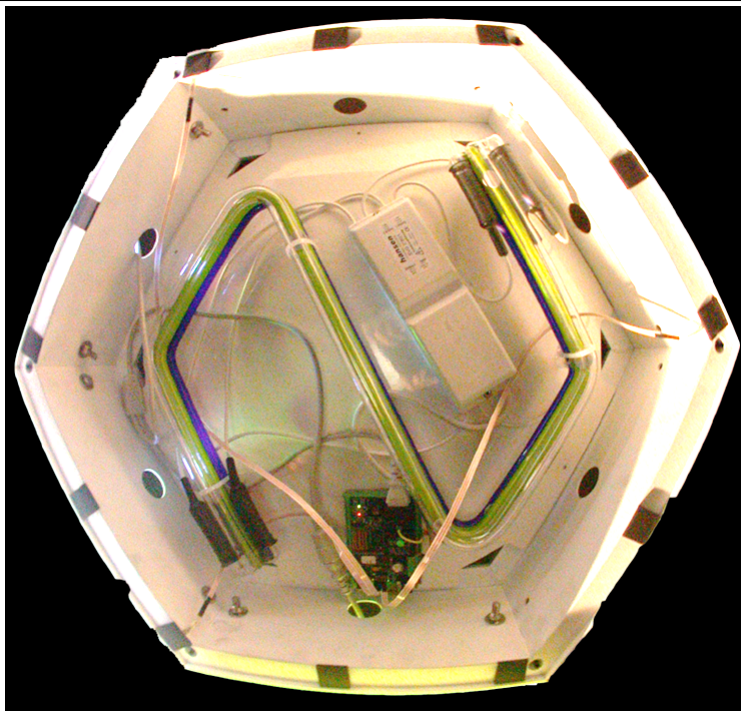
PCB design and layout

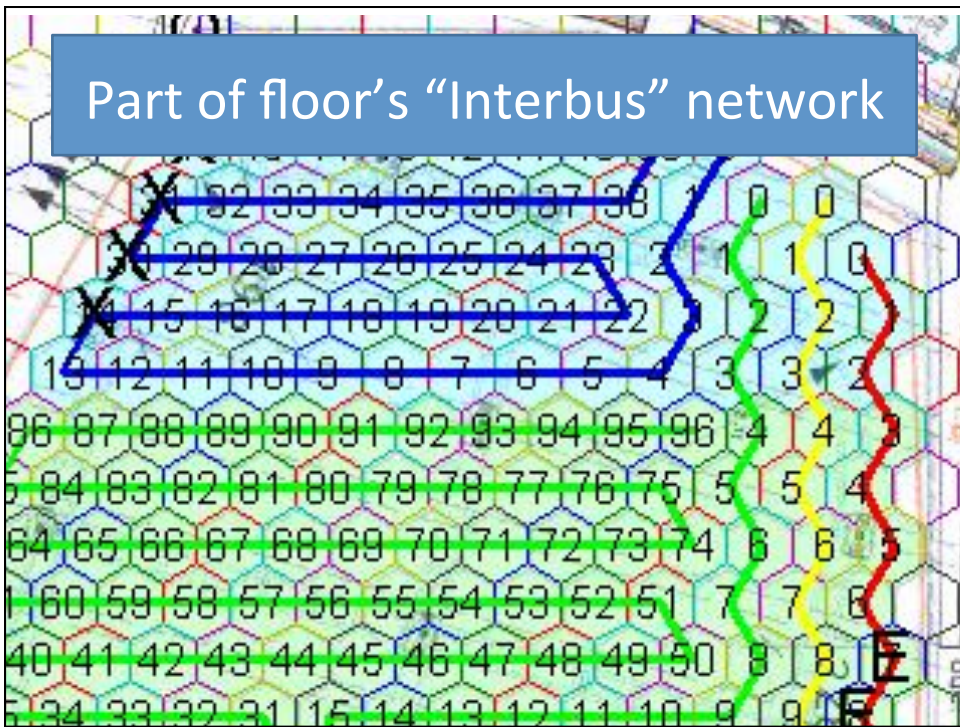
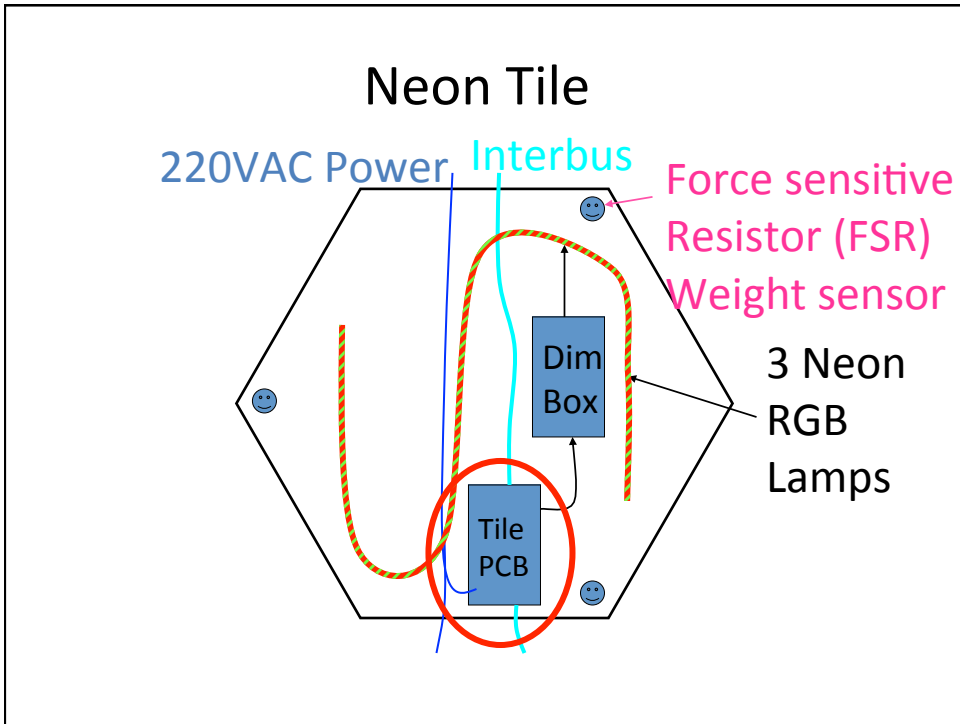


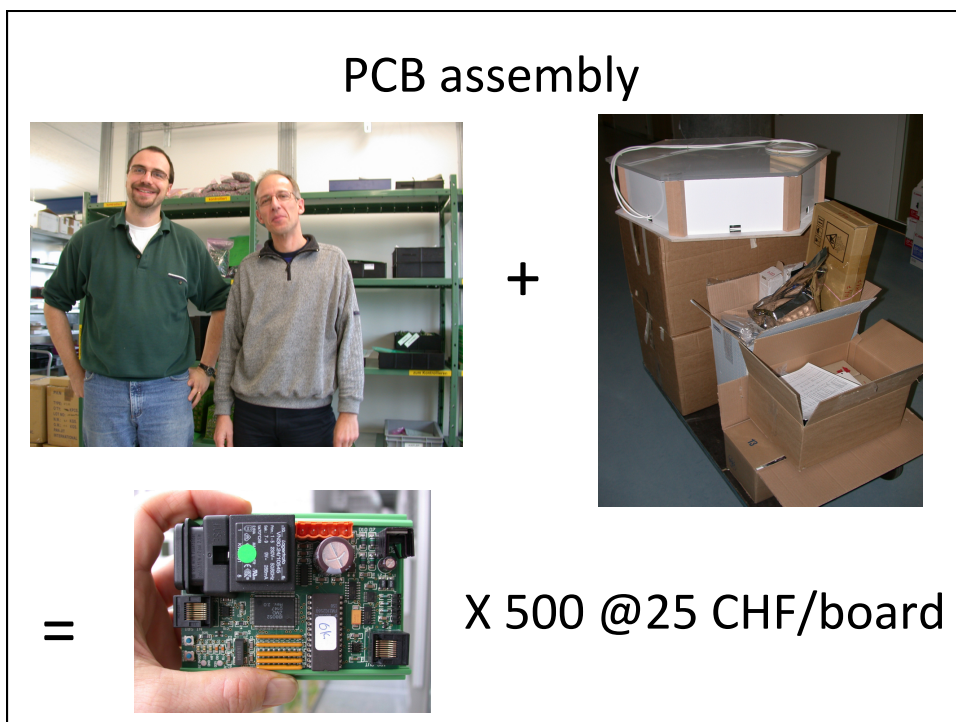
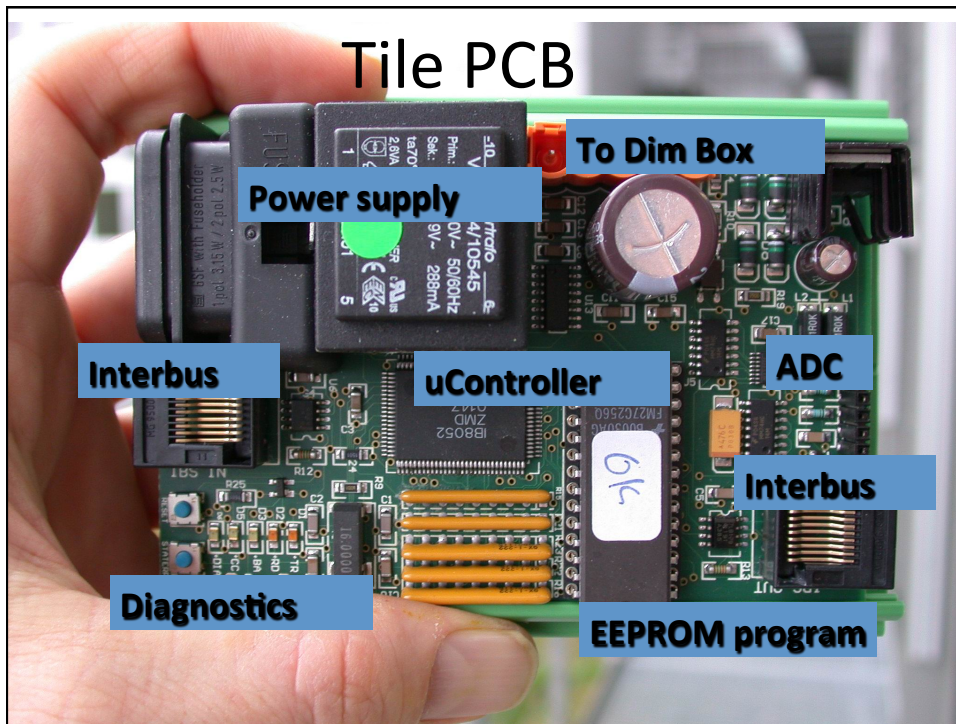
Embedded system design example Ada's luminous tactile floor



Ada's luminous tactile floor

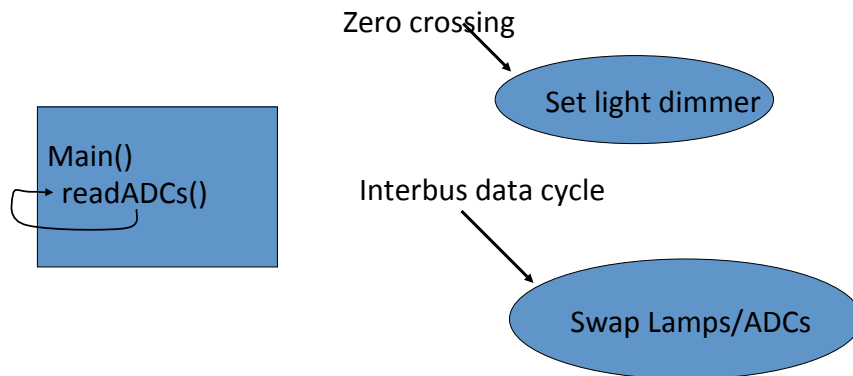






Tile Microcontroller Program

Interrupt service routines (ISRs)



1000 lines C/assembly code running on 8052 derivative
Two demo tiles have been running continuously for >5 years!

The ATmega168P / 328P

- AT = Atmel: Big microcontroller company
- mega: microcontroller family
- 16: 16KB Flash memory / 32: 32KB Flash
- 8: 8-bit architecture
- P: PicoPower Technology. Optional. For low power battery-based applications.

AVR families

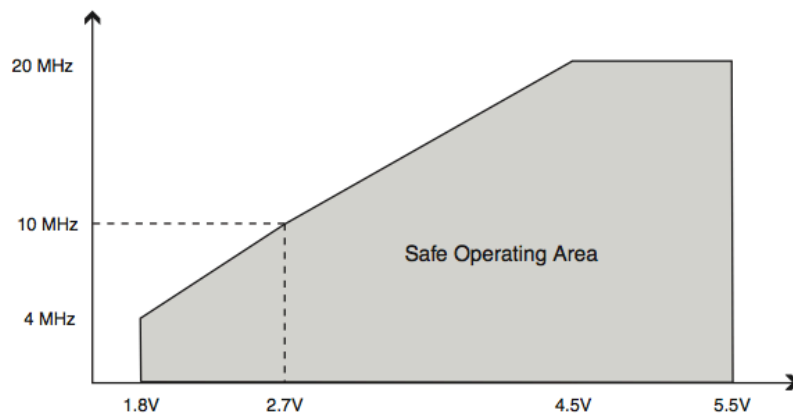
- **tinyAVR** — the *ATtiny* series
 - 0.5–16 kB program memory
 - 6–32-pin package
 - Limited peripheral set
- **megaAVR** — the *ATmega* series
 - 4–512 kB program memory
 - 28–100-pin package
 - Extended instruction set (multiply instructions and instructions for handling larger program memories)
 - Extensive peripheral set
- **XMEGA** — the *ATxmega* series
 - 16–384 kB program memory
 - 44–64–100-pin package (A4, A3, A1)
 - Extended performance features, such as DMA, "Event System", and cryptography support.
 - Extensive peripheral set with [ADCs](#)
- **Application-specific AVR**
 - megaAVRs with special features not found on the other members of the AVR family, such as LCD controller, [USB](#) controller, advanced PWM, CAN, etc.
- **FPSLIC (AVR with FPGA)**
 - [FPGA](#) 5K to 40K gates
 - SRAM for the AVR program code, unlike all other AVRs
 - AVR core can run at up to 50 MHz ^[5]
- **32-bit AVRs**
 Main article: [AVR32](#)
 In 2006 Atmel released microcontrollers based on the 32-bit [AVR32](#) architecture. They include [SIMD](#) and [DSP](#) instructions, along with other audio and video processing features. This 32-bit family of devices is intended to compete with the [ARM](#) based processors. The instruction set is similar to other RISC cores, but it is not compatible with the original AVR or any of the various ARM cores.

ATmega16/328P capabilities

- **System Functions**
 - Power and Clock Manager
 - Low Freq Internal Oscillator
 - Watchdog Timer
 - Real-Time Clock Timer
- **Interrupt Controller**
 - Fixed priority. One level of interruption. Interruptions with flag (can remember) or without. Global Interrupt Enable (I-bit) is disabled during an interrupt service.
- **NO Universal Serial Bus (USB)**
 - This micro hasn't USB. The nano board provide an USB-UART interface from FTDI company.
- One 16-bit **Timer/Counter (TC)** with Auto-Reload and PWM (x2)
- Two 8-bit **Timer / Counter (TC)** with AR and PWM (x2)
- One 8-channel 10-bit **Analog-To-Digital Converter (ADC)**, 76.9ks/s
- SPI, USART, TWI(I2C)

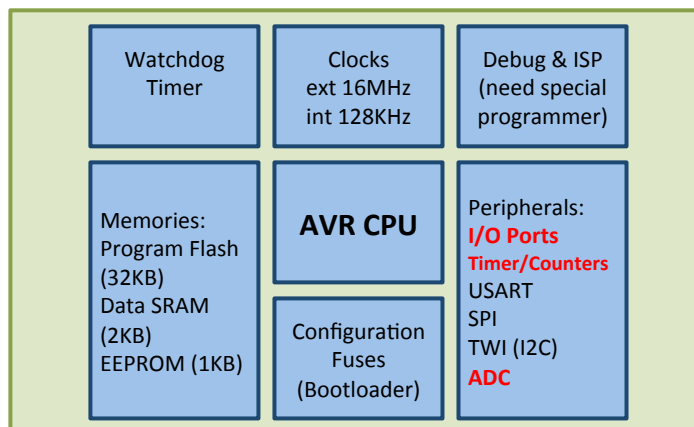
Clocks

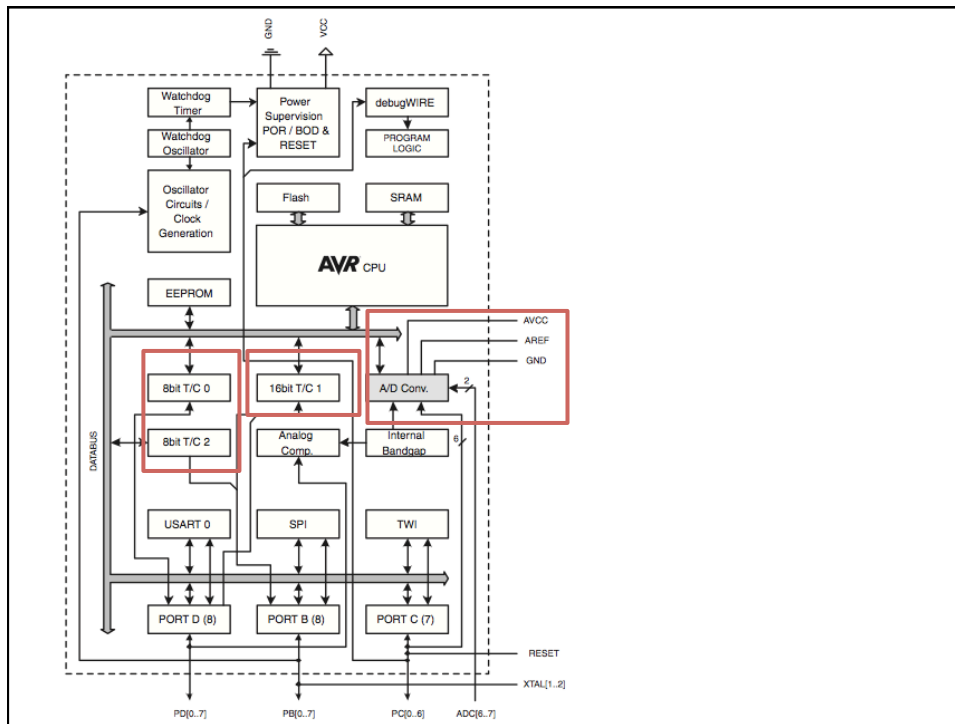
Figure 28-1. Maximum Frequency vs. V_{CC}



Arduino ATmega328P uses a 16MHz Resonator with 5v Vcc, to keep compatibility to ATmega8

ATmega328 simplified block diagram





Atmega168/328P datasheet

- 440 pages
- Useful for architecture comprehension
- Peripheral descriptions
- Not all details are necessary since Arduino bootloader make things easier!!

Arduino Sketch

```

Blink | Arduino 1.0.5
Blink §
/*
 * Blink
 * Turns on an LED on for one second, then off for one second, repeatedly.
 *
 * This example code is in the public domain.
 */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(100);              // wait for a second
  digitalWrite(led, LOW);  // turn the LED off by making the voltage LOW
  delay(500);              // wait for a second
}

Done uploading.
Binary sketch size: 1,084 bytes (of a 30,720 byte maximum)
Binary sketch size: 1,084 bytes (of a 30,720 byte maximum)

10 Arduino Nano w/ ATmega328 on /dev/tty.usbserial-A9615VNN

```

C programming for Arduino

ANSI C STYLE

```

void setup(void);
void loop(void);

void main(void) {
  setup(); //perform one-time initializations
  while(1) {
    loop(); //repeat this over and over
  }
}

void setup(void) {
}
void loop(void) {
}

```

ARDUINO CODE STYLE

```

void setup() {
}

void loop() {
}

```

Any other necessary stuff is included by Arduino IDE during compilation time and it is transparent for the user / programmer (you)

- Managing a LED:
 - Setup: `pinMode(13, OUTPUT); // bitSet(DDRB,5);`
 - Loop: `digitalWrite(13, HIGH); //or LOWz`
 - Managing Serial port:
 - Setup: `Serial.begin(rate); //Serial.end();`
 - Loop: `Serial.println(data);`
`Serial.println(data, data_type); //DEC,BIN,HEX,OCT, BYTE`
`Serial.print(data, data_type);`
`n=Serial.available(); //n=number of bytes (max 128B)`
`b=Serial.read(); // Serial.write(b);`
`b=Serial.peek();`
`Serial.flush()`
- ```
void serialEvent() {}
```

## Standard AVR code

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

#ifdef F_CPU
 #error Must define F_CPU or pass it as compiler argument
#endif

FILE uart_out = FDEV_SETUP_STREAM(uart_putchar, NULL, _FDEV_SETUP_WRITE);
void main(void) __attribute__((noreturn));

void uart_init(){
 # define BAUD 9600
 # include <util/setbaud.h>
 UBRRH0H = UBRRH_VALUE;
 UBRRL0 = UBRRL_VALUE;
 # if USE_2x
 USCR0A |= _BV(U2X);
 # else
 USCR0A &= ~_BV(U2X);
 # endif
 USCR0B |= _BV(TXEN); // enable transmit
}

int uart_putchar(char c, FILE *unused){
 if (c == '\n') // a line feed character also
 uart_putchar('\r', unused); // requires a carriage return
 loop_until_bit_is_set(USCR0A, UDRE0); // wait for UDR0 to be ready
 UDR0 = c; // write character
 return 0; // return
}

void main(void){
 uart_init();
 fprintf(&uart_out, "Hello, UART!\n");
 stdout = &uart_out; // set up stdout
 printf("UART demo...\n");
 for (uint8_t i = 0; i < 10; i++)
 printf("Number %i ", i);
 while(1);
}
```

## Arduino Serial code

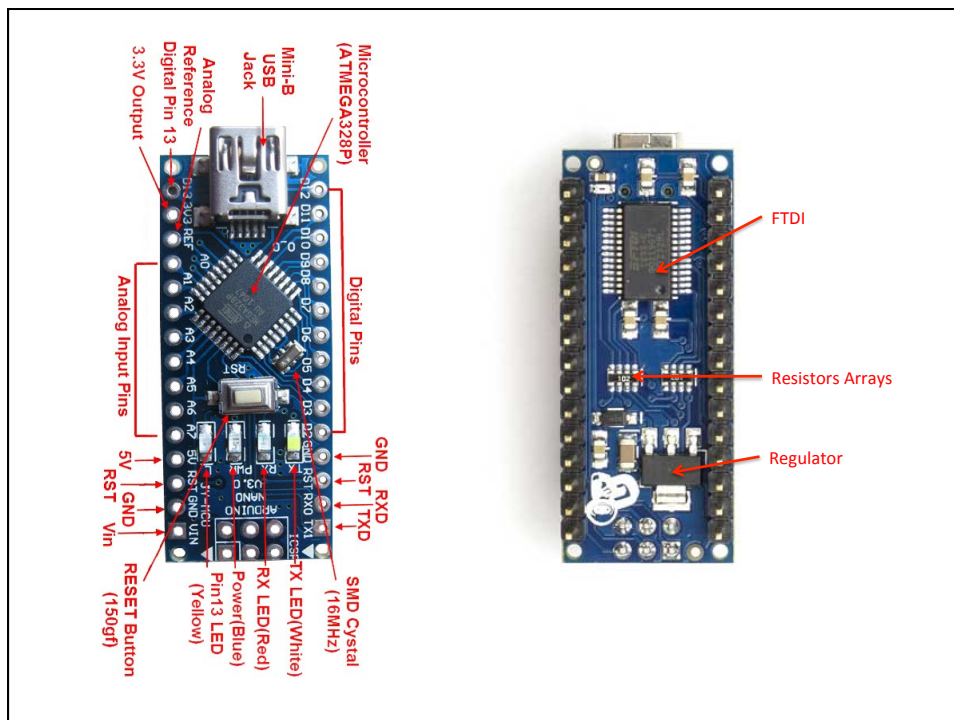
```

int led = 13;

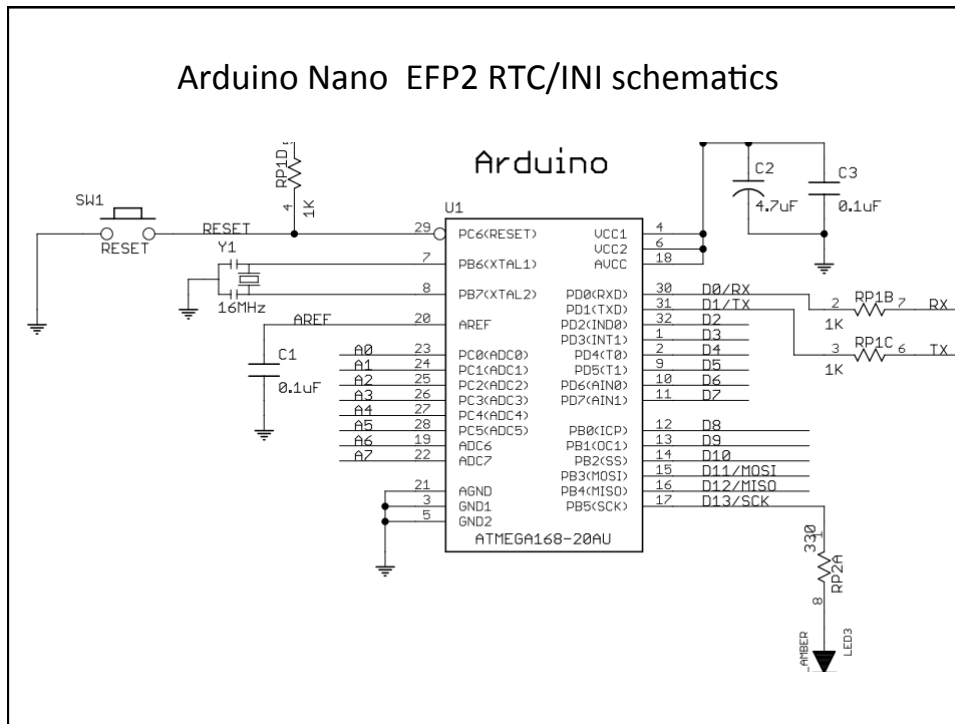
// the setup routine runs once when you press reset:
void setup() {
 // initialize the digital pin as an output.
 pinMode(led, OUTPUT);
 Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {
 digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)
 delay(1000); // wait for a second
 Serial.println("Hello UART");
 digitalWrite(led, LOW); // turn the LED off by making the voltage LOW
 delay(1000); // wait for a second
}

```







## Exercise 1

- Installing and running Arduino tools.
- Opening and understanding basic blinking Example Sketch and run it.
- Add Serial communication to Serial Monitor for “debugging” your Sketch.
- Arranging time for soldering your own Bronze V2 board / Arduino Nano and make it compatible with Sketch (only Nano).
- Exercise 2: probably debugging and repairing your own board.

## Surface mount soldering exercise



## Finding INI=Inst. of Neuroinformatics

