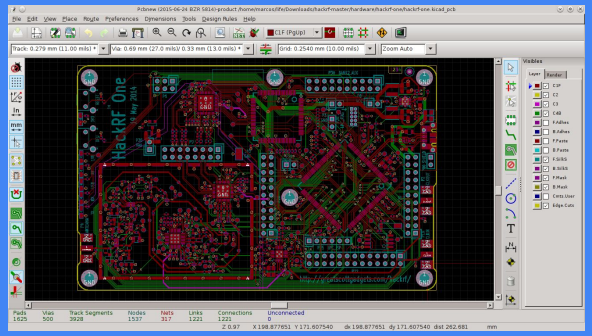
UCLA Rocket Project Electronics Workshop 2

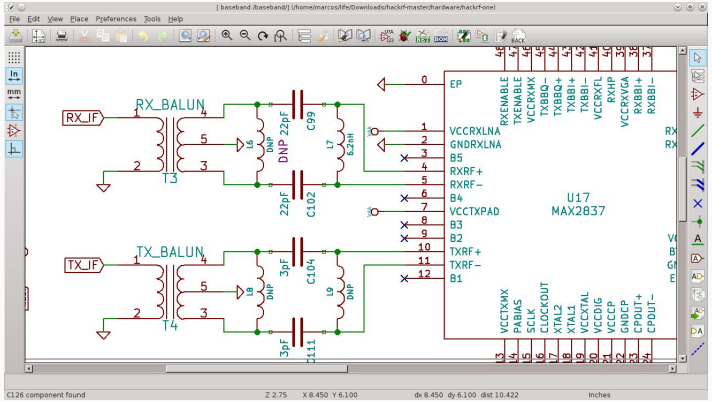
2016/10/14

Next week - KiCad



Electronic design CAD (Computer Assisted Design)

1. Free
2. Popular
3. Surprisingly good
4. Allows custom components well



Introduction to AVR

1. Understanding what a computer is
2. Binary
3. Three possible states recap
4. Pin configuration via Macros
5. Pointers
6. Final Blink program

What is a computer

- Counting machine - consists of (A) many little circuits with functions (B) list of commands
- Can only do what it has physical circuits for
- All computers need to execute a list of commands
- Each command is a choice of which little circuit to use
- All programming languages are translated into this list of commands (circuit choices step by step)

Binary

- Computers DO NOT run on 0s and 1s
- Computers run on a list of commands
- Each command has a number assigned
- We represent numbers using digits 0 to 9
- Computers represent numbers using only digits 0 and 1
- Computers DO NOT run on 0s and 1s

Binary 2

- When typing a human number we can choose from 0 to 9 for each position
- In binary you can only choose between 0 and 1 for each position
- What does our number mean, e.g. 247 = 7 * 1 + 4 * 10 + 2 * 100
- or 247 = 7 * (10^0) + 4 * (10^1) + 2 * (10^2) --> notice the 10 as the *base*
- How much is 0d247? 0d247 = 247 --> 0d is a prefix meaning decimal representation --> same numbers have different representation in different number systems

Binary 3

- How much is 0b001 --> notice 0b as a prefix for binary
- 0b001 = 1 * (2^0) + 0 * (2^1) + 0*(2^2) = 1 * 1 + 0 * 2 + 0 * 4 = 1
- 0b101 = 1* (2^0) + 0 * (2^1) + 1*(2^2) = 1*1 + 0*2 + 1*4 = 1 + 4 = 5
- 0b11110111 = 0d247
- 0b1110 = 0d14
- Plenty of online calculators, no need to do it by yourself

Hexadecimal

- Hex = 6, dec = 10 --> hexadecimal = base 16
- So, numbers 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 are allowed
- change double digits into letters to make it easier (harder?) to read
- So, numbers 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F are allowed
- Hexadecimal is useful for representing binary --> each number in hexadecimal is exactly 4 binary digits --> bits (4 bits)

Binary to hexadecimal

- 0b11010010 --> split into fours 0b1101 0b0010 --> convert fours to decimal 0d13 0d2 --> convert decimal into hexadecimal 0xD 0x2 --> write the hexadecimal number together 0xD2

- Online calculators for this too hexadecimal/decimal/binary

Three possible states of a digital line/pin

- In digital electronics only three possible states of a line/pin/node/end of wire
- HIGH (meaning highest working voltage, 3.3V or 5V)
- LOW (meaning ground or GND, 0V)
- Information (changes between HIGH and LOW in time)

AVR architecture

- AVR is a normal computer --> it will accepts commands, our program --> it will have little circuit for normal number countings --> it will also have circuits for changing levels (HIGH/LOW) of its pins/lines

- Arduino Pro Mini is a an ATmega328p at 3.3V with a processor running at 8MHz

AVR architecture 2

- One of the pins is connected on an Arduino Pro Mini to an LED (LED is a part of the arduino board, but not the AVR)

- We want to toggle (turn on and off in a loop) the LED on the board

AVR architecture 3 - Steps to toggle the LED

- Find out which pin is connected to the LED (search for Arduino Pro Mini schematics and look for LED marker) --> we should get P(letter) and number, so e.g. PD6
- P stands for Port, D stands for the port's name and 6 stands for the pin number in this port, in AVR ports will have 8 pins numbered from 0 - 7

AVR architecture 3 - Steps to toggle the LED

- Learn pin setup steps:

- Step 1: choose whether the pin should be input or output (output for toggling and LED)
- Step 2: change the state (HIGH or LOW) of the pin however you want

AVR architecture 3 - Steps to toggle the LED

- In C code, we can use a macro DDR(port name letter) to choose between input or output
- In C code, we can use a macro PORT(port name letter) to choose between state HIGH or LOW
- Assigning values to DDR(port) and PORT(port) will change the behavior of pins
- DDR(port) and PORT(port) control all 8 pins, so we need to be selective

Changing the state of the desired pin

- DDRB and PORTB are 8-bit numbers, e.g. 0b0100 1110
- each bit corresponds to the desired pin, e.g. pin 0 --> 0b0000 0001
- pin 5 --> 0b0010 0000 --> 0x20 (in hexadecimal)
- SET PIN5 to OUTPUT: DDRB = 0x20 (only pin 5 as output, all others input)
- SET PIN5 to HIGH: PORTB = 0x20
- SET PIN5 to LOW: PORTB = 0x00

Changing **only** the state of the desired pin

- SET PIN5 to OUTPUT: DDRB = (DDRB | 0x20)

  Bitwise OR, preserving all others inputs or outputs

- SET PIN5 to HIGH: PORTB = (PORTB | 0x20)

  Bitwise OR, preserving all other LOWs and HIGHs

- SET PIN5 to LOW: PORTB = (PORTB & ~0x20)

  Bitwise AND, only switching off the desired pin 5

Code example

```
int main(void) {
        DDRB = (DDRB | 0x20);
        while (1) {
                PORTB = (PORTB | 0x20);
                _delay_ms(500);
                PORTB = (PORTB & ~0x20);
                _delay_ms(500);
        }
        return 0;
}
```

Pointers

- Pointers hold the address of something in the memory
- Memory is a big table, so it has cells, a pointer is the (x,y) location of that cell
- In AVR pins are controlled by writing and reading to and from memory at a certain given address --> these addresses do not change and are in the datasheet

Pointers and pins

- To change whether a pin is an input or an output or to give it a HIGH or LOW state, one needs to write to memory where DDRB and PORTB are located (check the AVR datasheet, ATmega328 datasheet here)
- Thus, we don't have to use macros for: DDRD = 0x20
- we can check that DDRD is at location of 0x24 and we can do (*(char*)0x24) = 0x20 (which is a C way of saying: assign value of 0x20 to the memory cell (a register) at location 0x24

Final Blink program

```
int main(void) {

        (*(char*)0x24) = ((*(char*)0x04) | 0x20);

        while (1) {

                (*(char*)0x25) = ((*(char*)0x05) | 0x20);

                _delay_ms(500);

                (*(char*)0x25) = ((*(char*)0x05) & ~0x20);
                _delay_ms(500);

        }

        return 0;

}
```