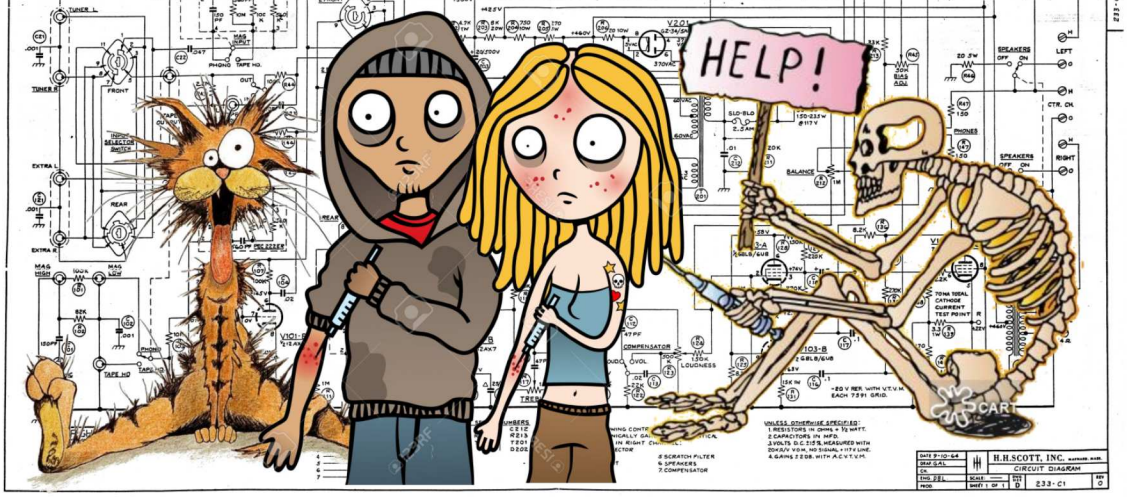
Schematics: the Heroin of Electronic Design

Dr. Dave Vandenbout
XESS Corp.

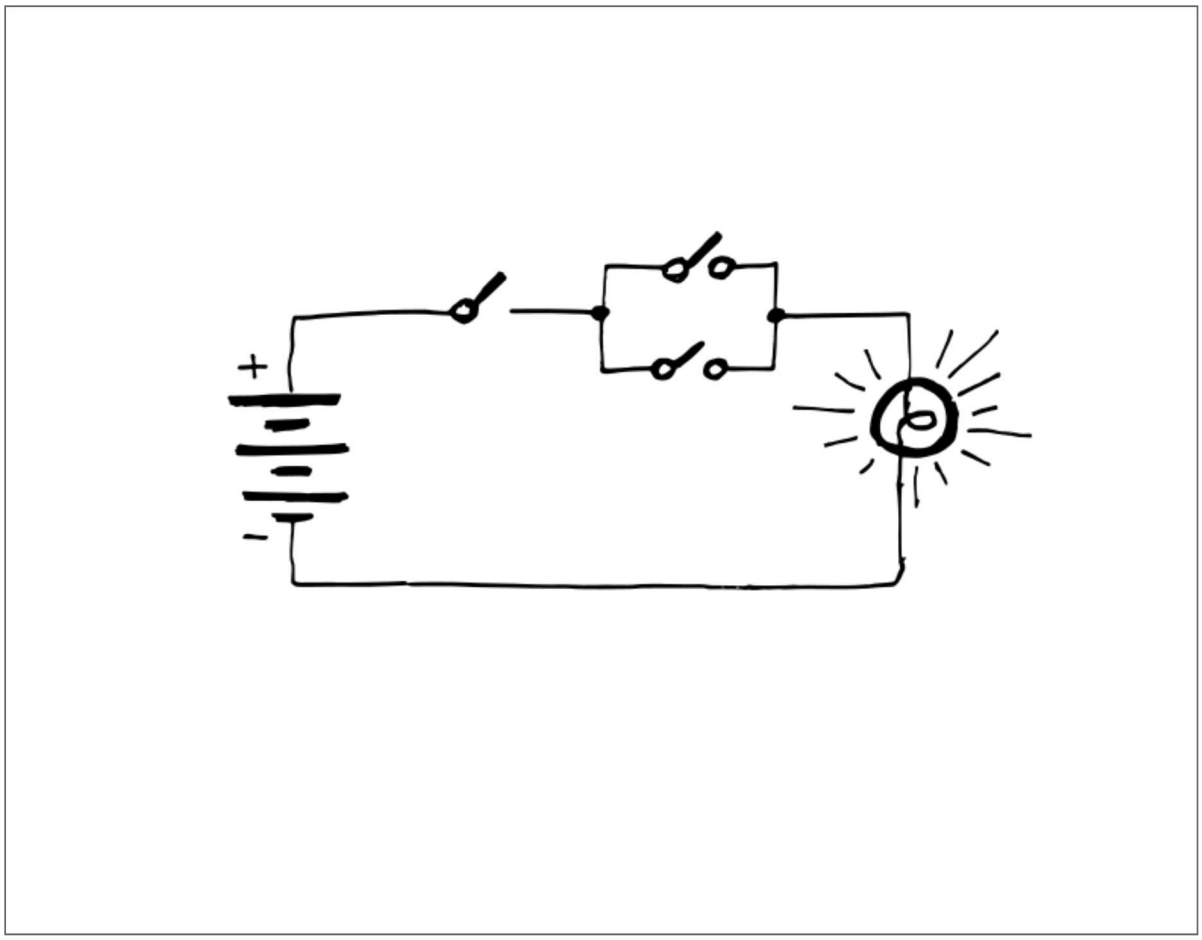


AN UNINTERESTING SLIDE

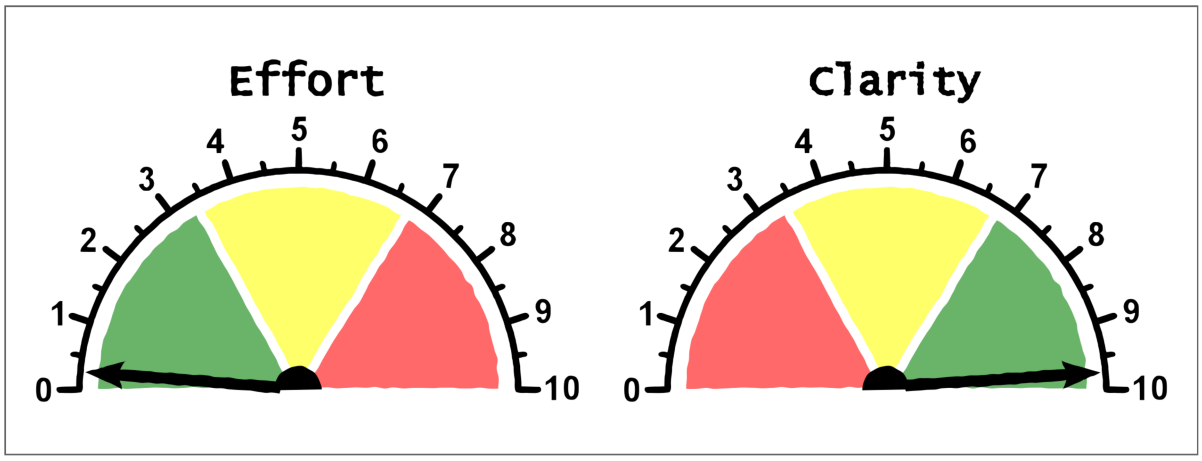
- An uninteresting bullet point.
- Hopefully, things improve…

1968

SEVENTH-GRADE PROJECT

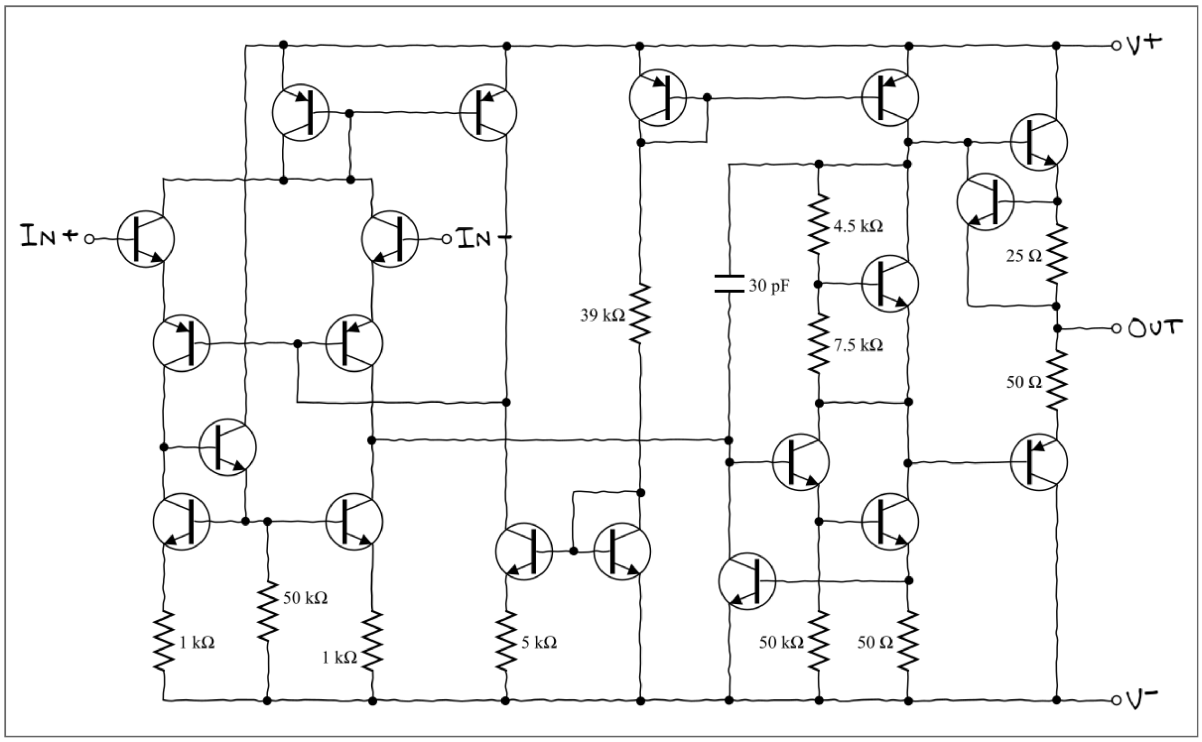


EFFORT VS. CLARITY

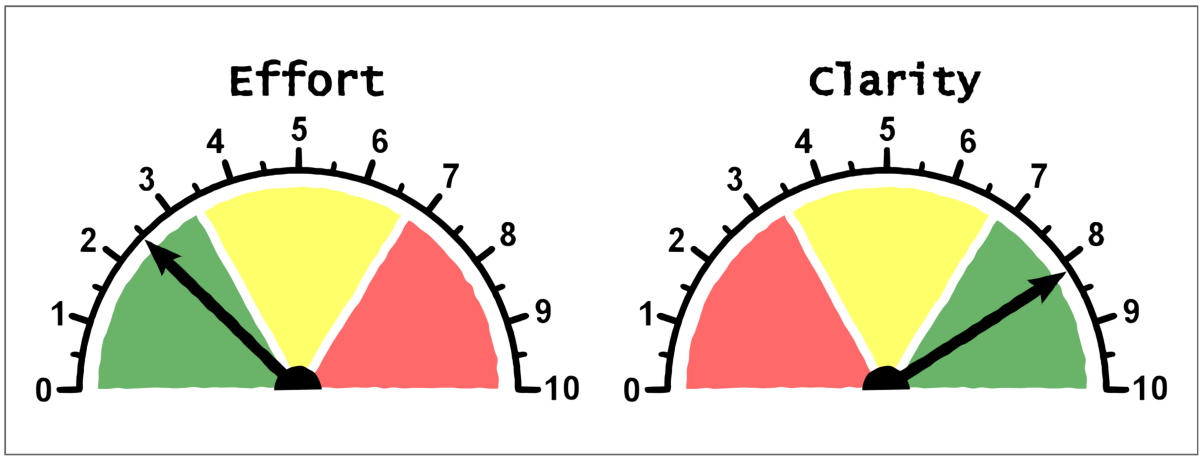


1975

OPAMP

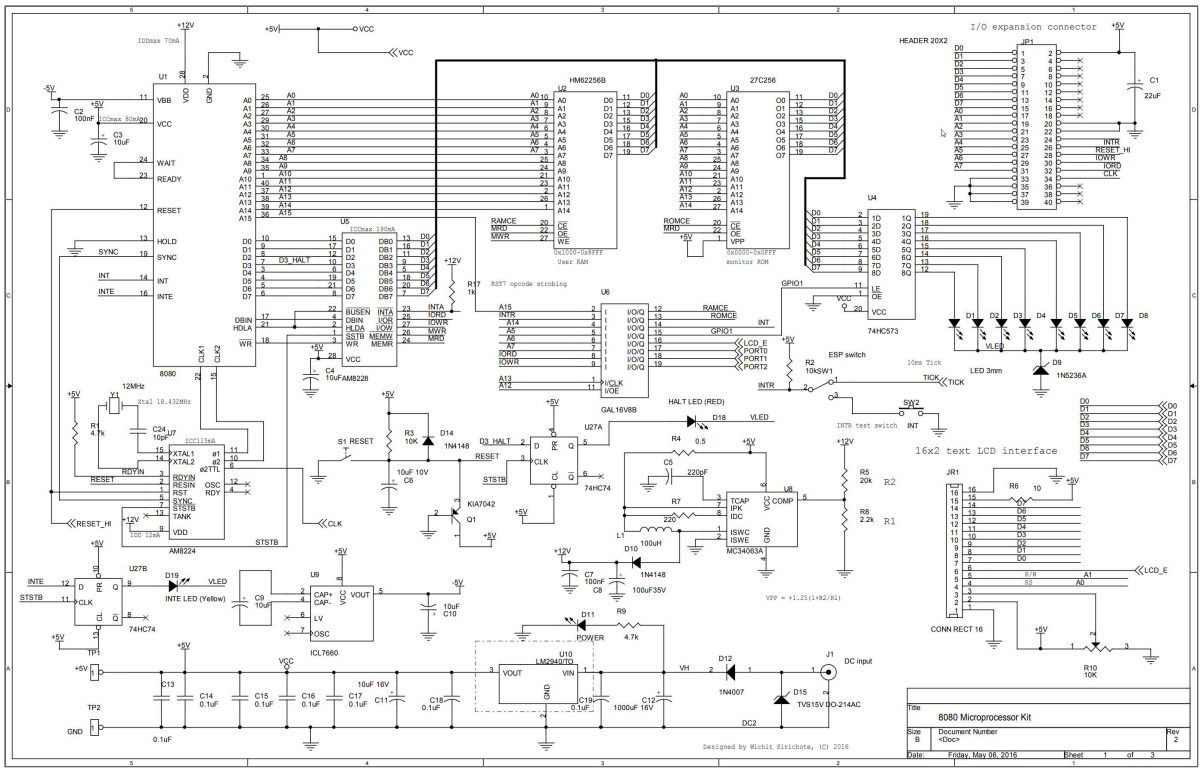

V+
IN+
IN−
4.5 kΩ
25 Ω
30 pF
39 kΩ
OUT
7.5 kΩ
50 Ω
50 kΩ
1 kΩ
1 kΩ
5 kΩ
50 kΩ
50 Ω
V−

EFFORT VS. CLARITY

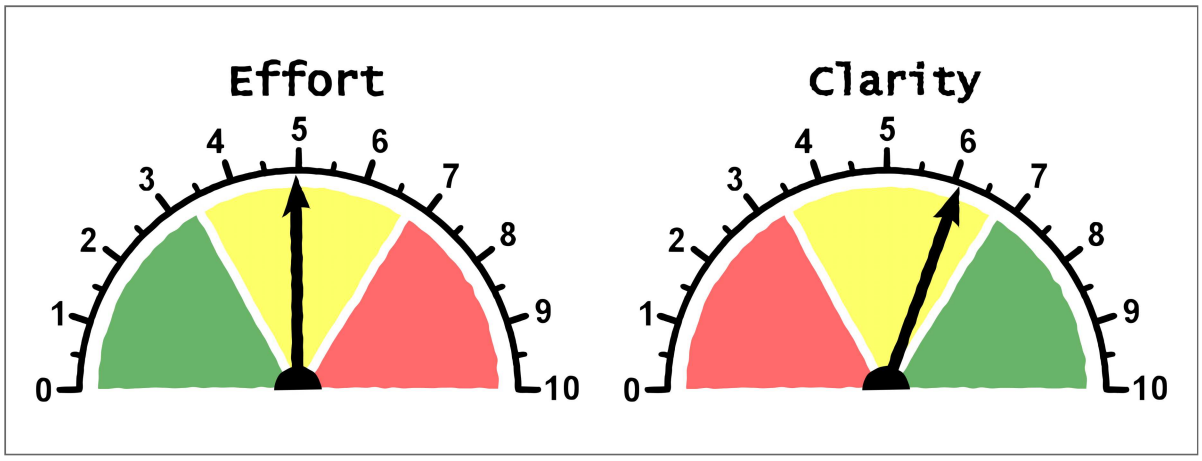


1981

MICROPROCESSOR

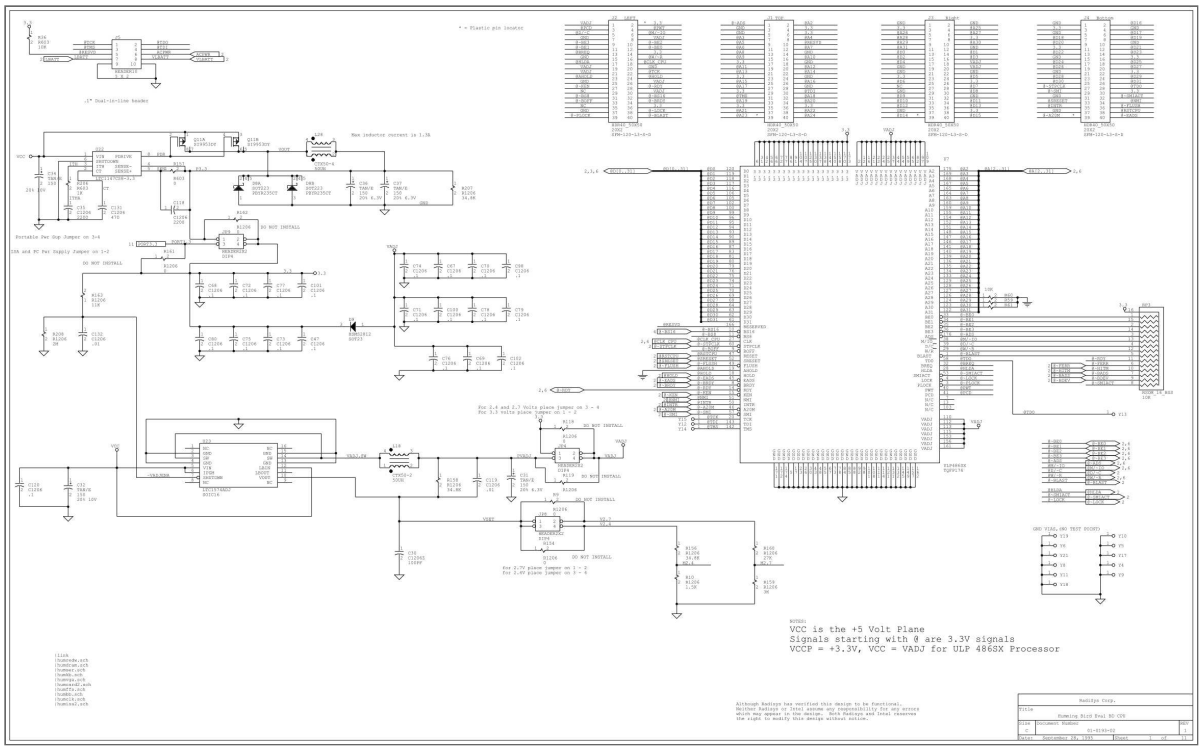


EFFORT VS. CLARITY

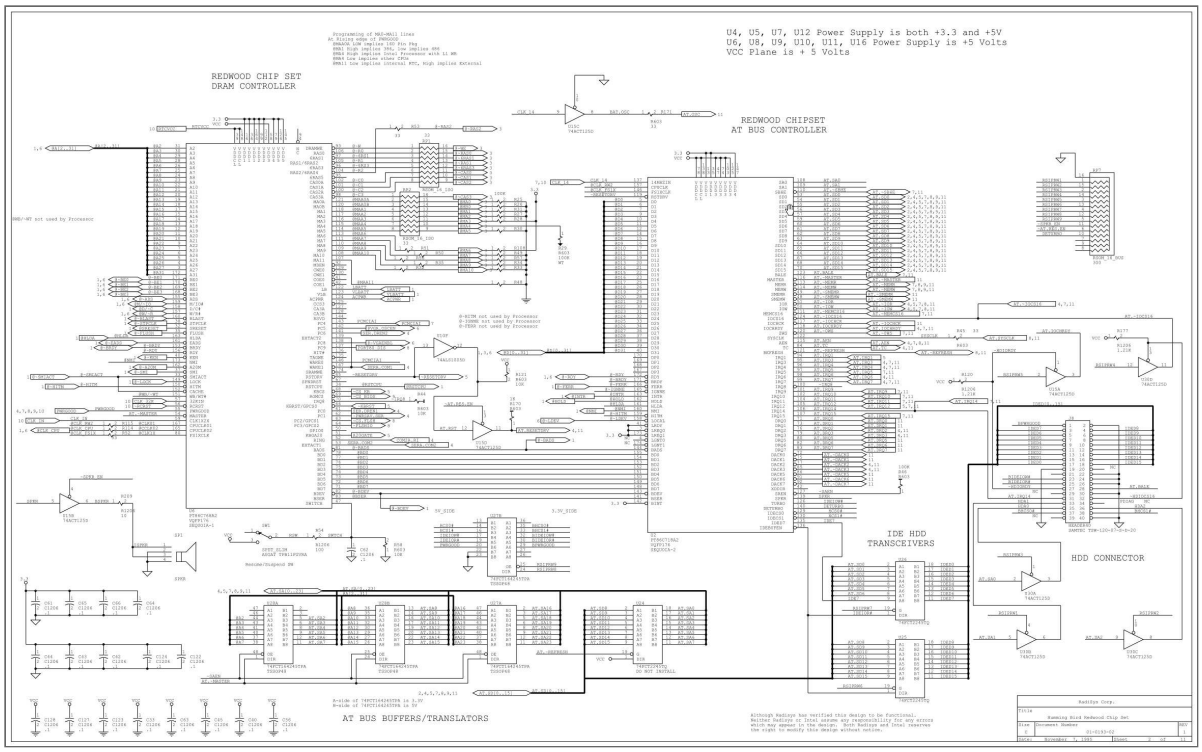


1995

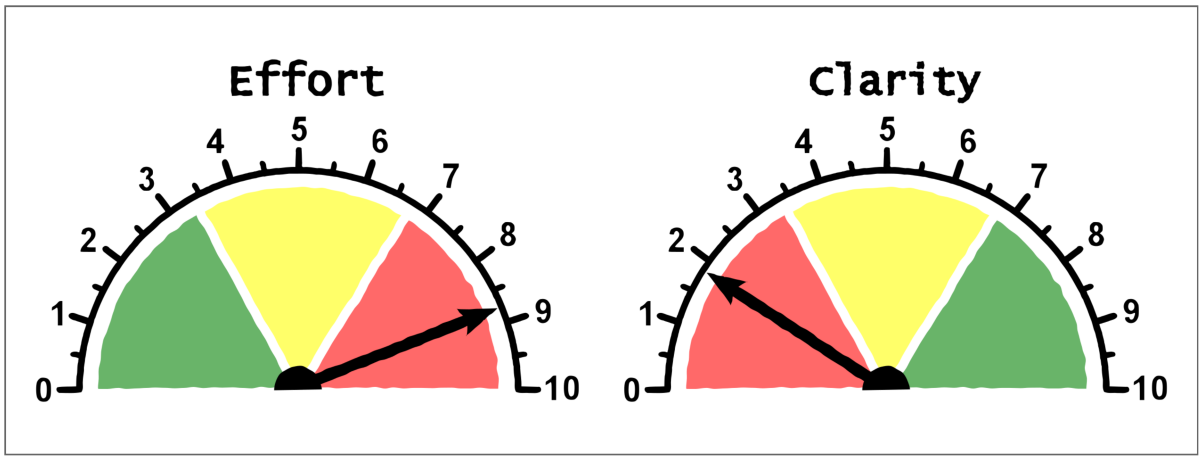
80486 MOTHERBOARD



80486 MOTHERBOARD (CONT'D)

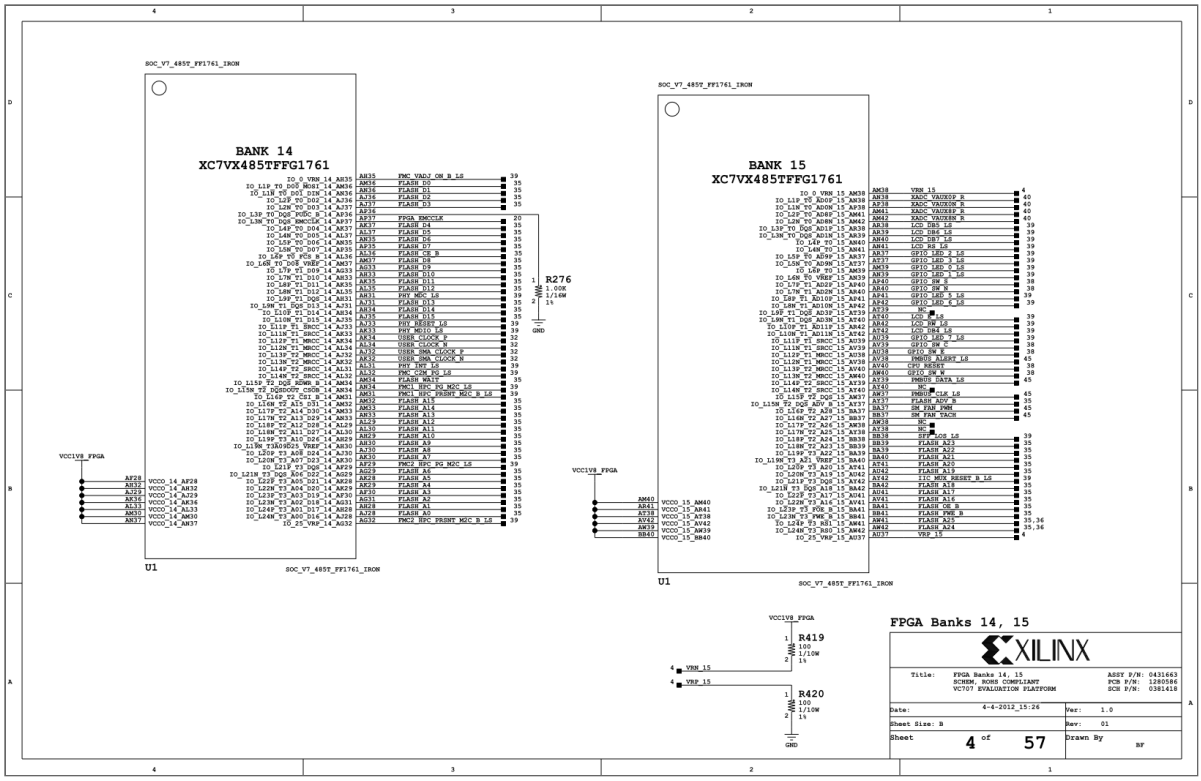


EFFORT VS. CLARITY

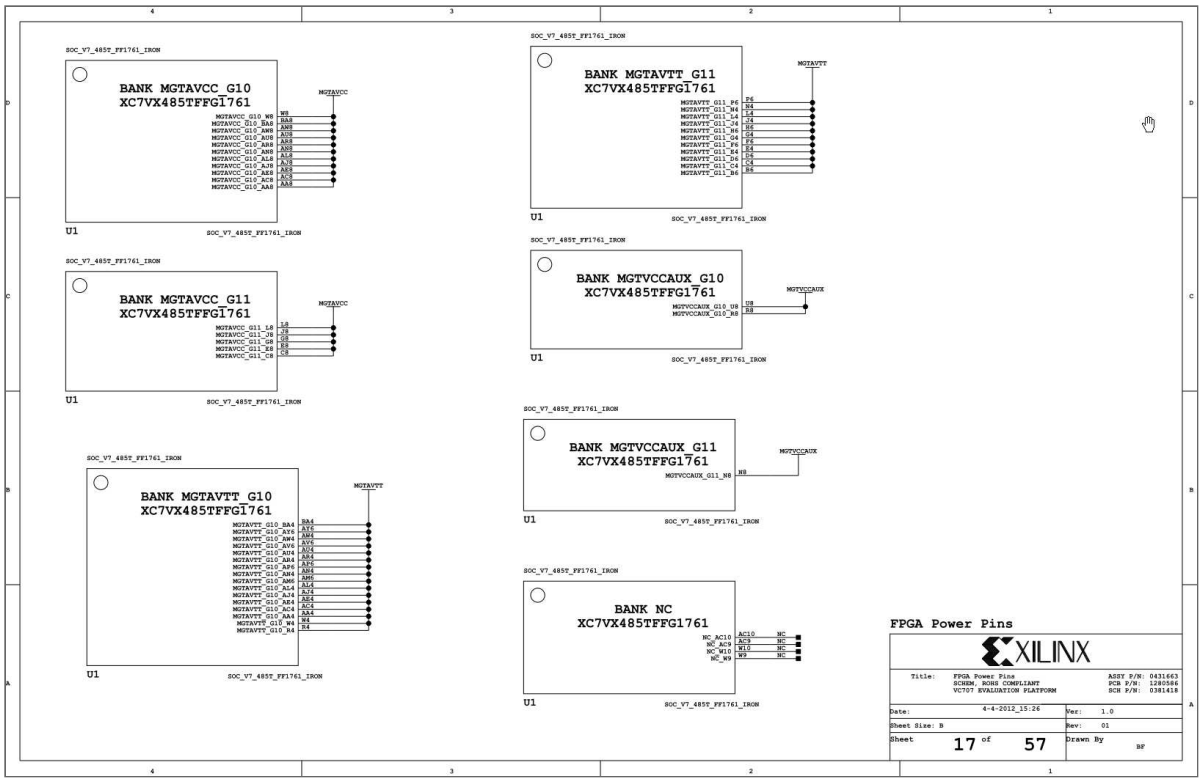


2000 AND BEYOND

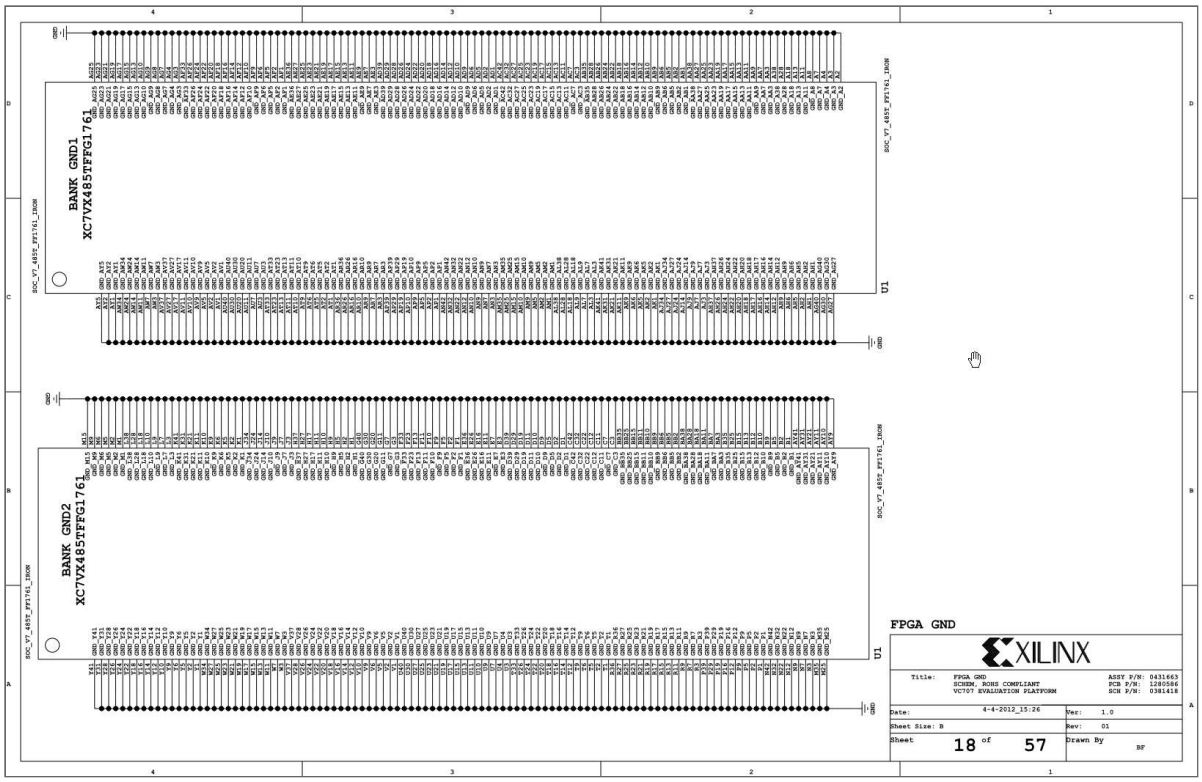
SOCS, FPGAS, OH MY!



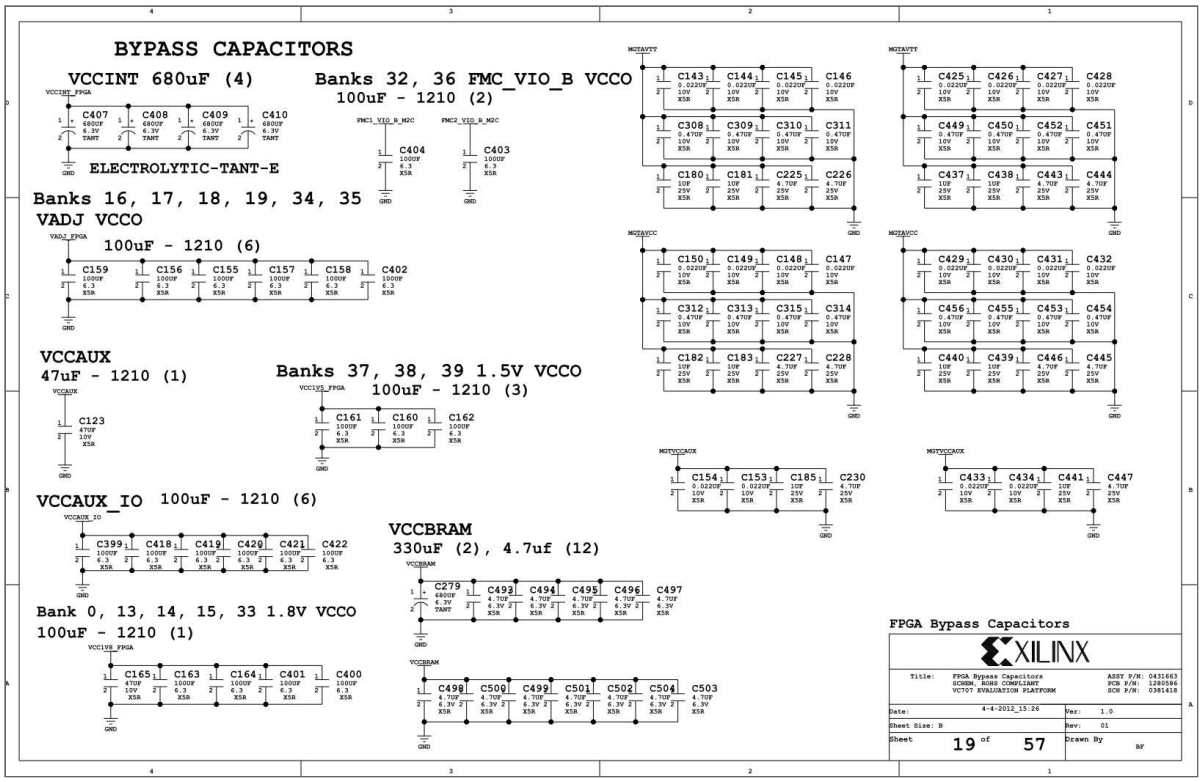
SOCS, FPGAS, OH MY! (CONT'D)



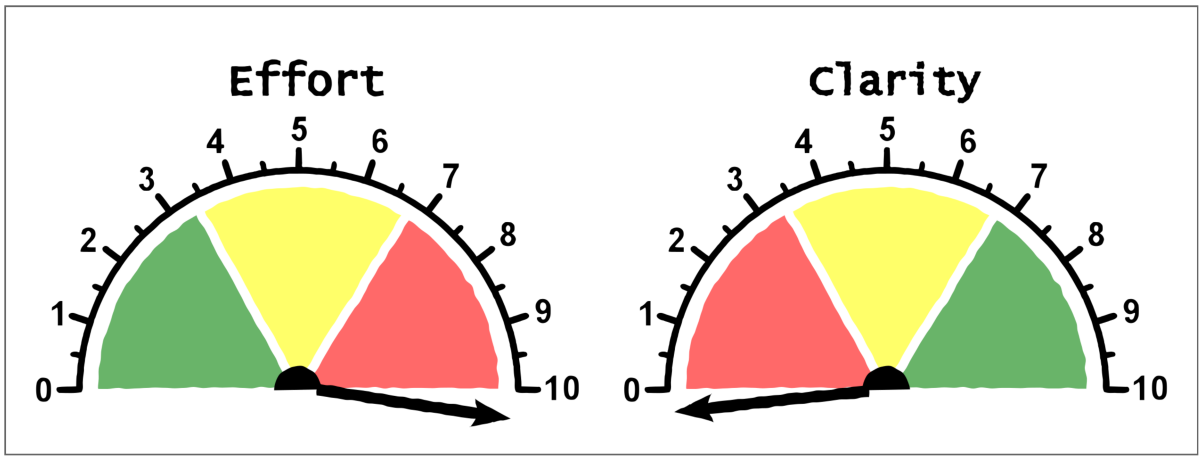
SOCS, FPGAS, OH MY! (CONT'D)



SOCS, FPGAS, OH MY! (CONT'D)



EFFORT VS. CLARITY



THE PROGRESSION

- Started small and things worked great!
- Used a bit more, still OK.
- Used even more. Marginal results.
- Using heavily. No longer sure what's going on…

2016

forum.kicad.info/t/pcbnew-without-eeschema-again/3306

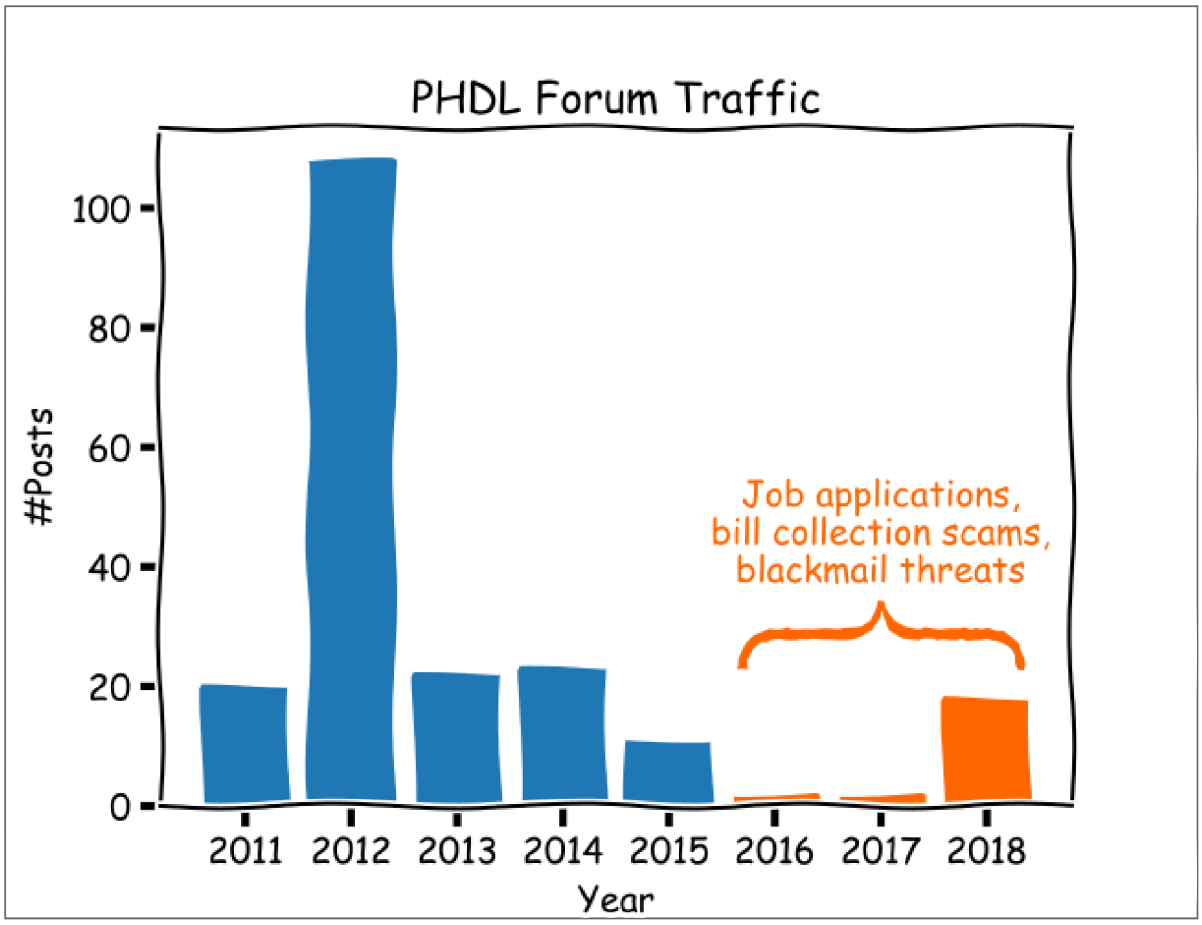
"I'd like to create … netlist information from non-graphical tools…"

- Typing is faster than drawing
- Computers handle text efficiently
- Every OS has a text editor
- Scripts can automate tasks
- Searching is straightforward

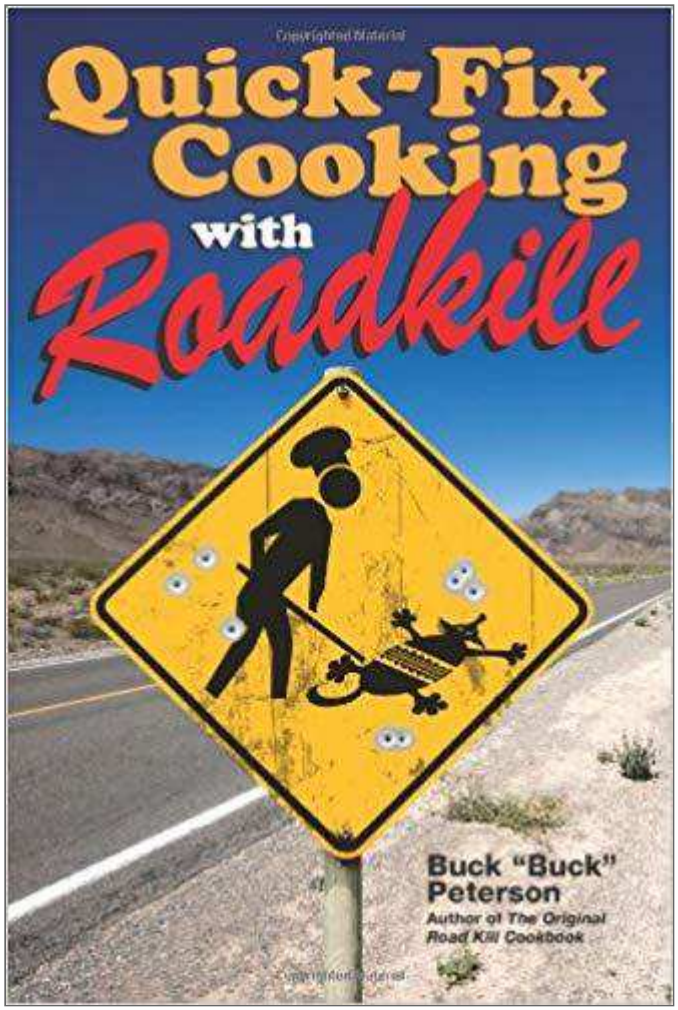
IT’S BEEN DONE BEFORE

- PHDL
- BYU
- Java

BUT PHDL DIED



TASTY AND DOESN'T FIGHT BACK!



PHDL - THE TASTY BITS

- Concise, repetitive part instantiation
- Concise, bulk connectivity
- Encapsulation
- Hierarchy

WHY DID PHDL DIE?

- Very problem-specific
- Lacked general-purpose capabilities
- No ecosystem (libraries, utilities)
- No user community

MYHDL

- Built upon a general-purpose language: Python
- Adds objects for describing digital hardware
- Provides a functional simulation engine
- Outputs VHDL or Verilog for synthesis

PHDL + MYHDL → SKIDL

- Use Python for general-purpose computing
- Add *part* and *net* objects
- Provide methods for connecting them
- Output a netlist to PCBNEW

BASIC SKIDL OBJECTS

- `Pin`: a terminal with a number, name, I/O type, ...
- `Part`: a bag of `Pins`
- `Net`: a connection between one or more `Pins`
- `Bus`: an array of `Nets`

START WITH A PART

```
>>> uc = Part('MCU_Microchip_PIC10', 'PIC10F220-IP')
```

```
>>> uc
```

```
PIC10F220-IP (PIC10F222-IP): 512W Flash, 24B SRAM, PDIP8
    Pin U1/2/VDD/POWER-IN
    Pin U1/3/GP2/BIDIRECTIONAL
    Pin U1/4/GP1/BIDIRECTIONAL
    Pin U1/5/GP0/BIDIRECTIONAL
    Pin U1/7/VSS/POWER-IN
    Pin U1/8/GP3/INPUT
```

CREATE NETS

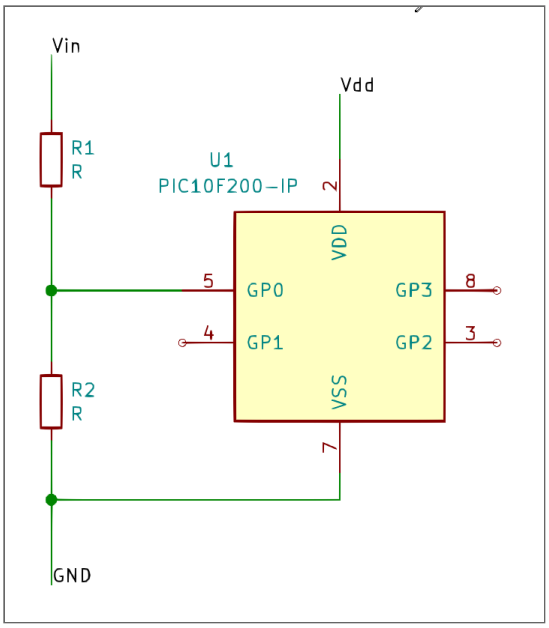
- `n = Net()`
- `n = Net('my_net')`
- `b = Bus('my_bus', 5)`

CONNECT PINS AND NETS

- `[ ]` to access part pins and bus nets
- `+=` to connect stuff
- Pin to net: `n += uc[5]`
- Pin to pin: `uc[3] += uc[4]`
- Bus to part:
  `b[3:0]+=uc['gp3,gp2,gp1,gp0']`
- Bus to part: `b[3:0] += uc['gp[3:0]']`

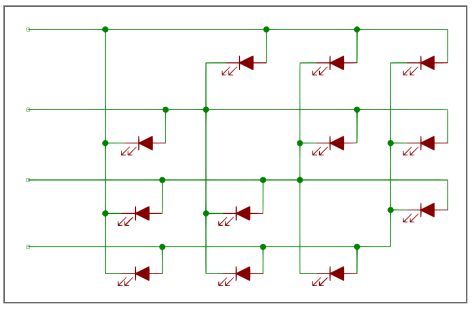
EXAMPLE #1

```
vdd = Net('Vdd')
gnd = Net('GND')
vin = Net('Vin')

r1 = Part('device','R',value='4.7K')
r2 = r1(value='2.2K')
vin += r1[1]
r1[2] += r2[1]
r2[2] += gnd

uc = Part('MCU_Microchip_PIC10',
          'PIC10F220-IP')
uc['VDD'] += vdd
uc['VSS'] += gnd
uc['gp0'] += r2[1]
```



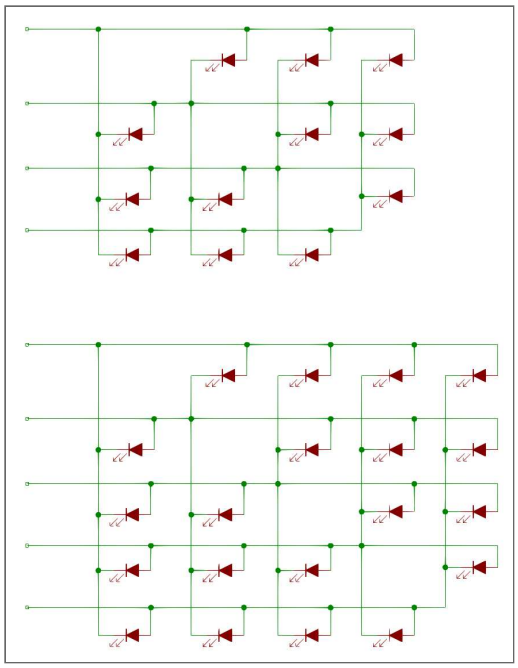
EXAMPLE #2

```
b = Bus('chplx', 4)
for hi in b:
    for lo in b:
        if hi != lo:
            led = Part('device','LED')
            hi += led['A']
            lo += led['K']
```



EXAMPLE #3

```
@subcircuit
def chplx_leds(b,
        ledt=Part('device',
                  'LED',TEMPLATE)):
    for hi in b:
        for lo in b:
            if hi != lo:
                led = ledt()
                hi += led['A']
                lo += led['K']

b1, b2 = Bus('B1',4), Bus('B2',5)
chplx_leds(b1)
chplx_leds(b2)
```

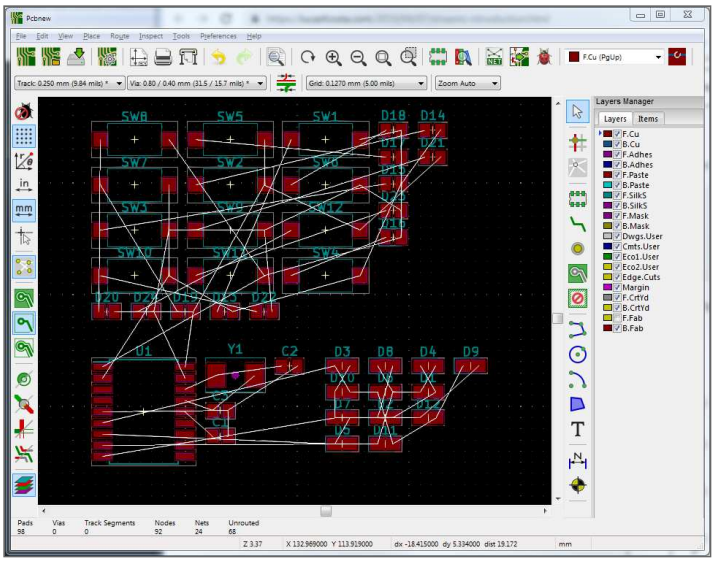


EXAMPLE #4

```
vdd, gnd = Net('VDD'), Net('GND') # power & ground nets.

c = Part('Device','C', TEMPLATE)  # capacitor template.

uc = Part('MCU_Microchip_PIC16',
          'PIC16F83-XXSO')        # Microcontroller.
uc['VDD, VSS'] += vdd, gnd        # Attach pwr, gnd to uC.

c_byp = c(value='10uF')           # Add bypass capacitor.
c_byp[1,2] += vdd, vss

xtal = Part('Device','Crystal')   # Crystal.
uc['OSC1, OSC2'] += xtal[1,2]     # Attach crystal to uC.

c1, c2 = c(2, value='10pF')       # Crystal trim caps.
c1[1,2] += xtal[1], gnd           # Connect trim caps.
c2[1,2] += xtal[2], gnd

chplx_leds(uc['RB[3:0]'])         # 12 charlieplexed LEDs.
chplx_sws(uc['RA[3:0]'])          # 12 charlieplexed switches.
```

NETLIST → LAYOUT

```
...
chplx_leds(uc['RB[3:0]'])
chplx_sws(uc['RA[3:0]'])

generate_netlist()
```



FOOTPRINTS?

Where did the footprints come from?

```
uc = Part('MCU_Microchip_PIC16', 'PIC16F83-XXSO',
          footprint=
            "KiCad_V5/Package_SO.pretty:SOIC-18W_7.5x11.6mm_P1.27mm")
```

PARTS?

How did you find the parts?

```
>>> search('pic10')
```

```
WARNING: Could not open directory ''
MCU_Microchip_PIC10.lib: PIC10F220-IMC (512W Flash, 24B SRAM, DFN8)
MCU_Microchip_PIC10.lib: PIC10F320-IMC (512W Flash, 64B SRAM, DFN8)
MCU_Microchip_PIC10.lib: PIC10F220-IOT (512W Flash, 24B SRAM, SOT-23-6)
MCU_Microchip_PIC10.lib: PIC10F204-IMC (512W Flash, 24B SRAM, DFN8)
MCU_Microchip_PIC10.lib: PIC10F320-IP (512W Flash, 64B SRAM, PDIP8)
MCU_Microchip_PIC10.lib: PIC10F204-IOT (512W Flash, 24B SRAM, SOT-23-6)
MCU_Microchip_PIC10.lib: PIC10F220-IP (512W Flash, 24B SRAM, PDIP8)
MCU_Microchip_PIC10.lib: PIC10F320-IOT (512W Flash, 64B SRAM, SOT-23-6)
MCU_Microchip_PIC10.lib: PIC10F200-IP (512W Flash, 24B SRAM, PDIP8)
MCU_Microchip_PIC10.lib: PIC10F204-IP (512W Flash, 24B SRAM, PDIP8)
MCU_Microchip_PIC10.lib: PIC10F200-IOT (512W Flash, 24B SRAM, SOT-23-6)
MCU_Microchip_PIC10.lib: PIC10F200-IMC (512W Flash, 24B SRAM, DFN8)
```

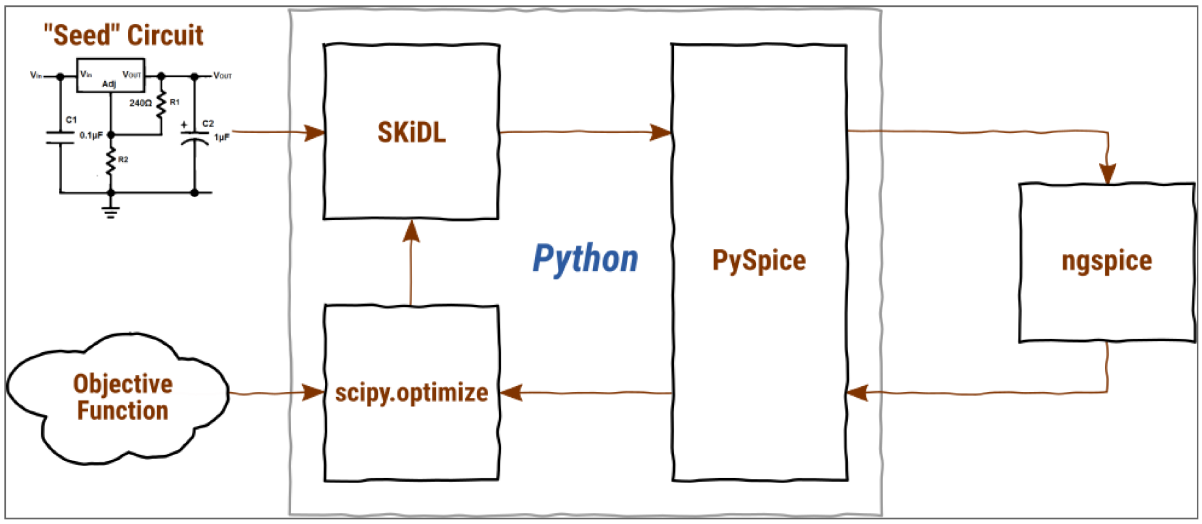
SKIDL FIXES COMMON PROBLEMS

- Trivialities (Junction dot size? Really!?!)
- Repetitive parts (1000 * cap(value="0.1uF"))
- Mistaken disconnections
- Power flags (`diode['K'].drive=POWER`)
- Version control (diff just works)
- Multiple boards in one project

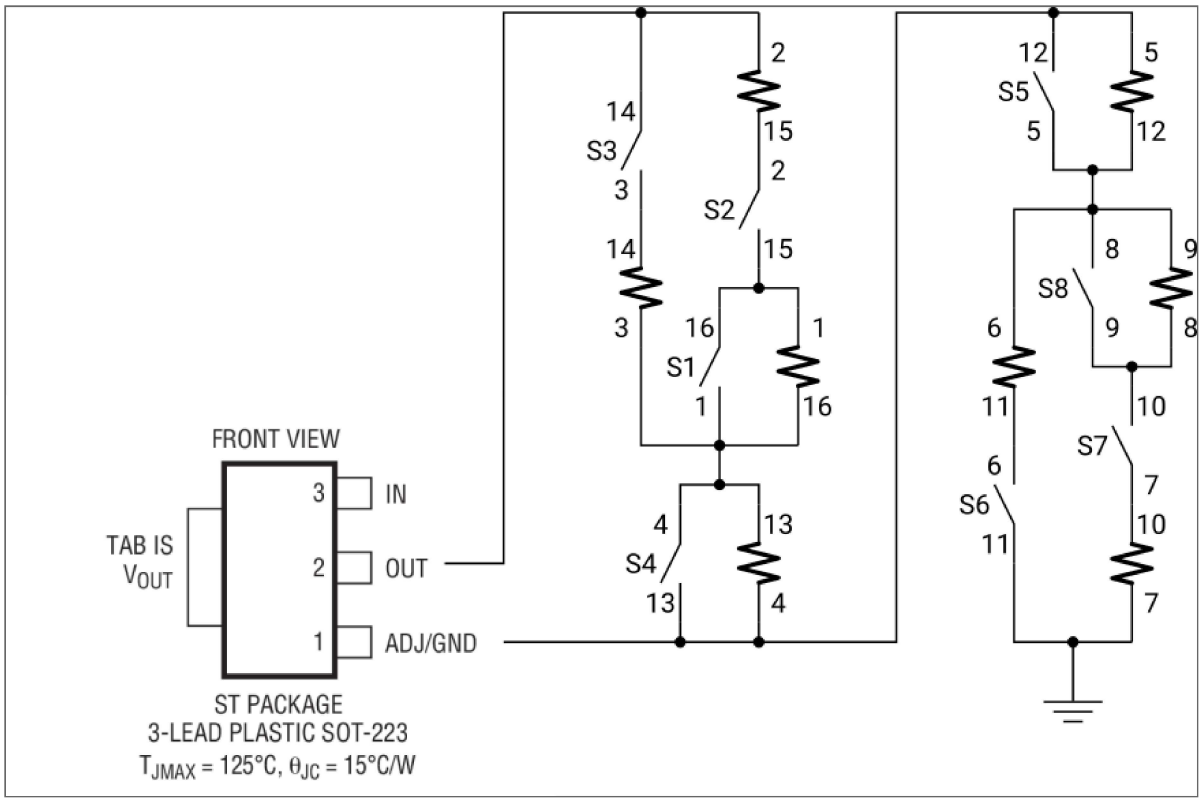
FREEBIES!

- Standard res/cap values (pypi.org/project/eseries)
- Normalizing values (github.com/ulikoehler/UliEngineering)
- `PySpice` - ngspice interface
- `scipy.optimize`

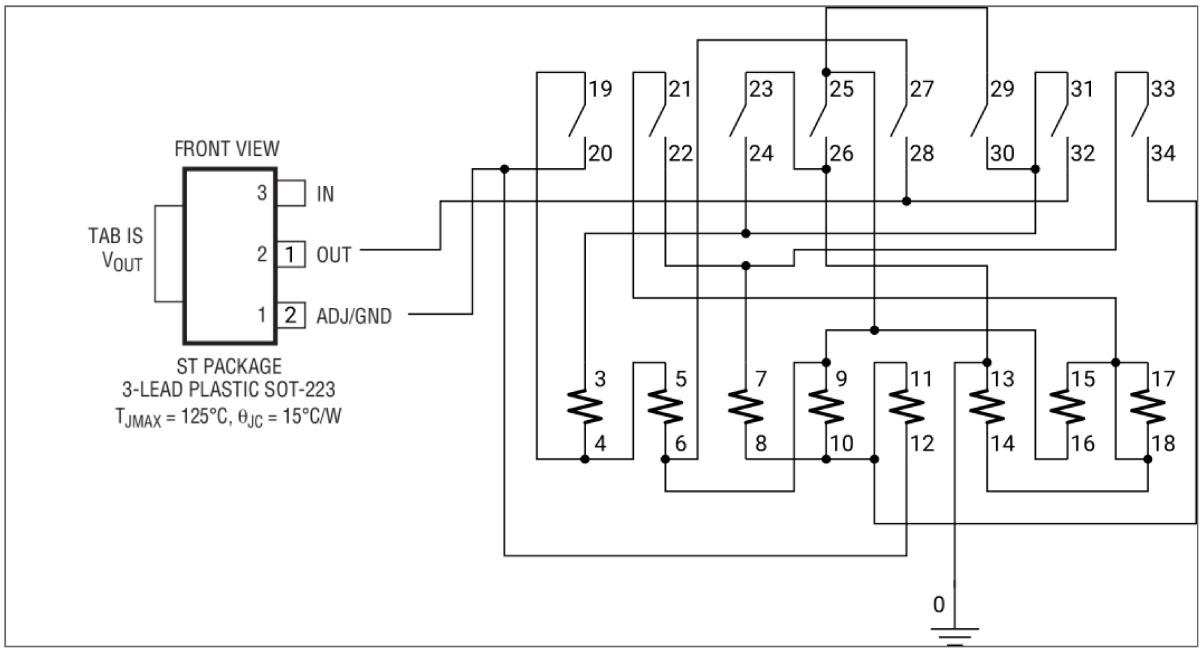
GENERATIVE CIRCUITS



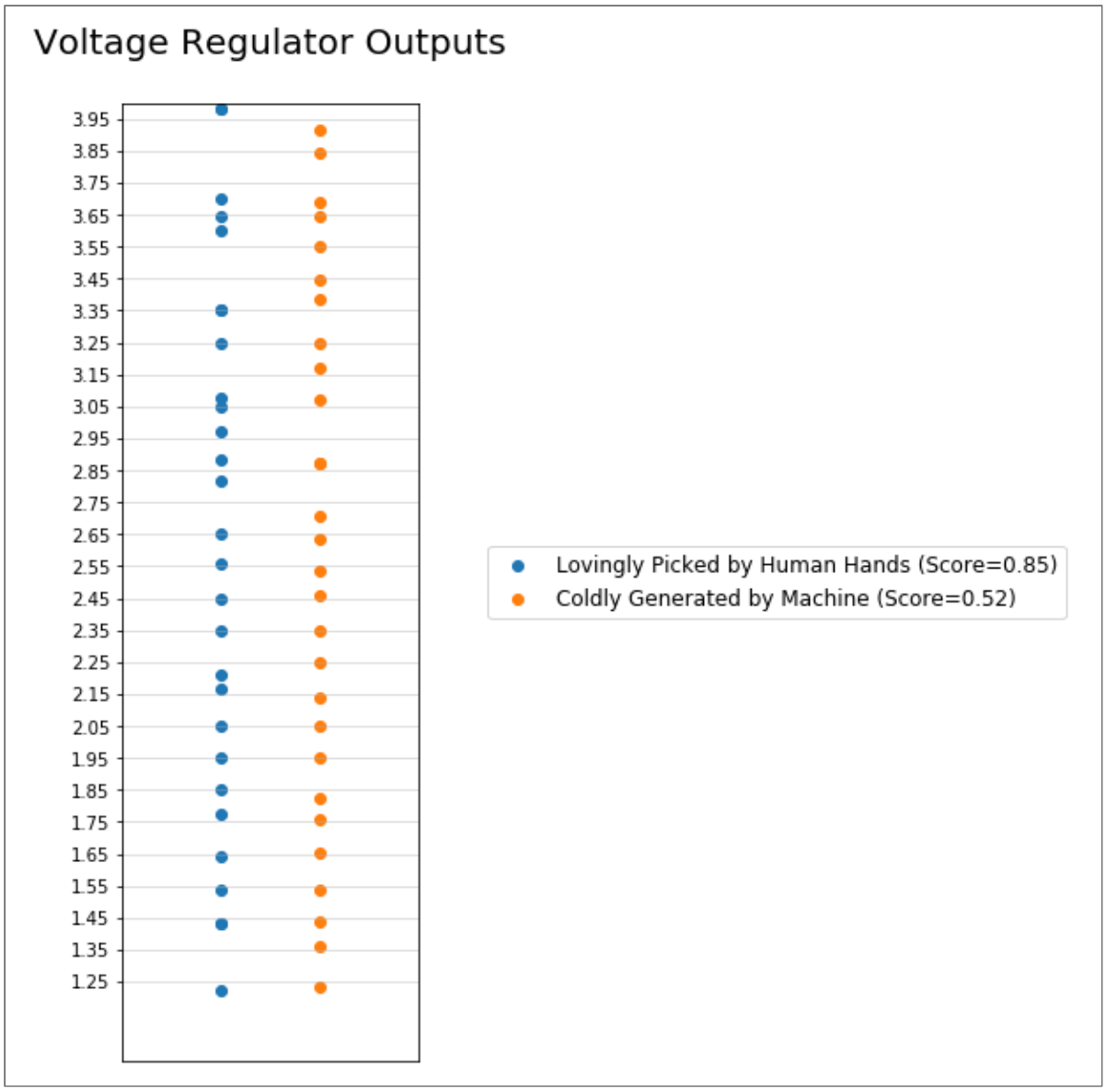
MULTI-VOLTAGE REGULATOR



MACHINE-GENERATED VERSION



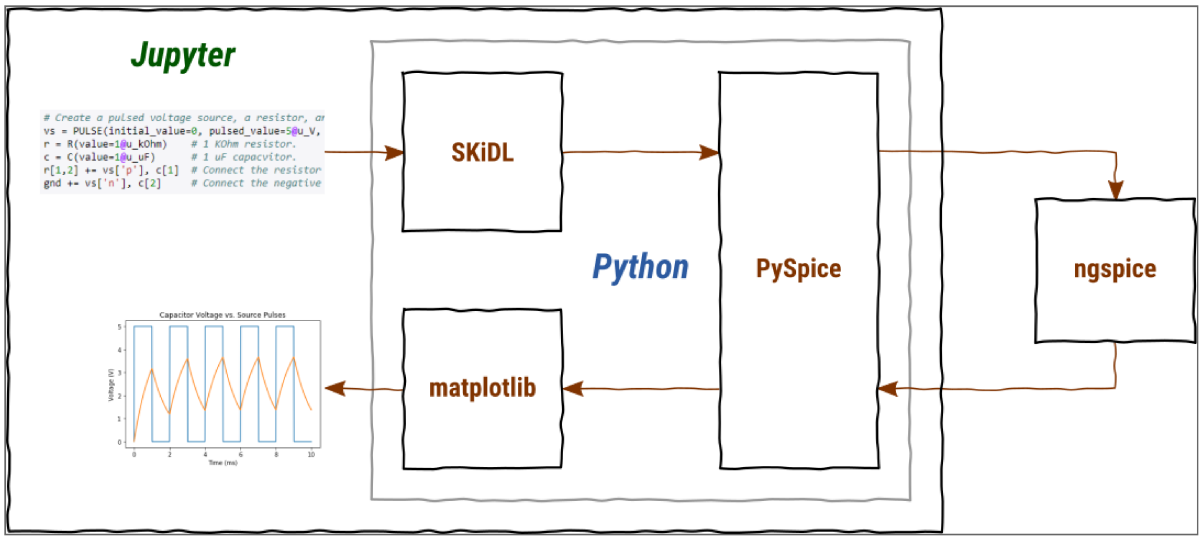
MAN VS. MACHINE



DOCUMENTATION

- Python comments
- Doc strings, Sphinx and autodoc
- Attaching notes to circuits, parts, pins, nets, buses
- Jupyter

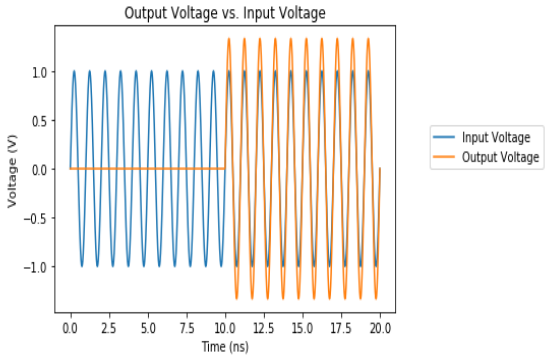
JUPYTER


Jupyter
# Create a pulsed voltage source, a resistor, ar
vs = PULSE(initial_value=0, pulsed_value=5@u_V,
r = R(value=1@u_kOhm)   # 1 KOhm resistor.
c = C(value=1@u_uF)   # 1 uF capacitor.
r[1,2] += vs['p'], c[1] # Connect the resistor
gnd += vs['n'], c[2]   # Connect the negative
SKiDL
Python
PySpice
ngspice
matplotlib
Capacitor Voltage vs. Source Pulses
Voltage (V)
Time (ms)

In [7]:

```
reset()  # Clear out the existing circuitry from the previous example.

# Create a 1 GHz sine wave source, drive it through a 70 ohm ideal transmission line, and terminate it with a 140 ohm resistor.
vs = SINEV(amplitude=1@u_V, frequency=1@u_GHz)
t1 = T(impedance=70@u_Ohm, frequency=1@u_GHz, normalized_length=10.0) # Trans. line is 10 wavelengths long.
rload = R(value=140@u_Ohm)
vs['p'] += t1['ip']  # Connect source to positive input terminal of trans. line.
rload[1] += t1['op'] # Connect resistor to positive output terminal of trans. line.
gnd += vs['n'], t1['in','on'], rload[2] # Connect remaining terminals to ground.

# Simulate the transmission line.
circ = generate_netlist()
sim = circ.simulator()
waveforms = sim.transient(step_time=0.01@u_ns, end_time=20@u_ns)

# Get the waveforms at the beginning and end of the trans. line.
time = waveforms.time * 10**9
vin = waveforms[node(vs['p'])]      # Input voltage at the beginning of the trans. line.
vout = waveforms[node(rload['1'])]  # Output voltage at the terminating resistor of the trans. line.

# Plot the input and output waveforms. Note that it takes 10 nsec for the input to reach the end of the
# transmission line, and there is a 33% "bump" in the output voltage due to the mismatch between the
# 140 ohm load resistor and the 70 ohm transmission line impedances.
figure = plt.figure(1)
plt.title('Output Voltage vs. Input Voltage')
plt.xlabel('Time (ns)')
plt.ylabel('Voltage (V)')
plt.plot(time, vin)
plt.plot(time, vout)
plt.legend(('Input Voltage', 'Output Voltage'), loc=(1.1, 0.5))
plt.show()
```

```
No errors or warnings found during netlist generation.
```



PLAYGROUND RULES

- pip install skidl
- Design examples: xesscorp.github.io/skidl
- Netlist converter: netlist_to_skidl