SKiDL 2.1 CHEAT SHEET

Basic Setup

```
from skidl import *

# Set default tool and library path(optional)
set_default_tool(KICAD9)
lib_search_paths[KICAD9].append('/path/to/my/libs')
```

Parts

```
# Creating part from library
mcu = Part('Microcontroller', 'ATmega328P')

# Creating part with attributes
r = Part('Device', 'R', value='1K', footprint='R_0805')

# Creating multiple copies using templates
r1, r2, r3 = Part('Device', 'R', dest=TEMPLATE)(3)
r1, r2 = 2 * Part('Device', 'R', dest=TEMPLATE)

# Adding part attributes
r.footprint = 'Resistor_SMD.pretty:R_0805_2012Metric'
r.ref = 'R5'
r.fields['Manufacturer'] = 'Vishay'

# Accessing part pins
mcu[1], mcu['VCC']       # Pin by number or name
mcu.VCC                  # Pin as attribute
mcu[1,2,3], mcu[1:3]     # Multiple pins
mcu['VCC,VDD,GND']       # Multiple pin names
mcu['GP[0:2]']           # Sequential pins GP0, GP1, GP2
mcu.p[1], mcu.n['VCC']   # Explicitly by pin number or name

# Pin aliases
mcu['VCC'].aliases += 'PWR'  # Assign alias to VCC pin
mcu['PWR'] += vcc            # Reference VCC pin by alias
```

Nets and Buses

```
# Creating nets
vcc = Net('VCC')
clk = Net()            # Auto-named

# Adding net attributes
vcc.drive = POWER      # Power net for ERC
vcc.do_erc = False     # Skip ERC checking for this net

# Creating buses
data = Bus('DATA', 8)           # 8-bit bus: DATA0..DATA7
mixed = Bus('MIX', 4, vcc, gnd) # Mix of new and existing nets

# Accessing bus lines
data[0]                # First line
data[7:0]              # All lines in reverse
data[1,3,5]            # Lines 1,3,5
```

Connections

```
# Primary rule: ONLY use += for connections!
# Pin-to-net or net-to-pin
vcc += r[1]
r[2] += gnd

# Pin to pin (creates implicit net)
r1[1] += r2[2]

# Multiple connections
vcc += r1[1], r2[1], op['V+']
r[1,2] += vcc, gnd

# Bus connections (same width required)
mcu['PB[0:7]'] += data[0:7]
```

Networks (2-pin parts only)

```
# Serial connection
vcc & r1 & r2 & r3 & gnd

# Parallel connection
vcc & (r1 | r2 | r3) & gnd

# Mixed connections
vcc & ((r1 & r2) | (r3 & r4)) & gnd

# With explicit pins (polarized parts)
vcc & r1 & d1['A,K'] & gnd

# Tee connections
inp & tee(cs & gnd) & l & tee(cl & gnd) & outp
```

Searching and Finding

```
# Command line search
search('opamp low-noise')          # Multiple terms (AND)
search('opamp (dip-8|soic-8)')     # Alternatives (OR)
search('^lm358$')                  # Exact match (regex)
show('Amplifier_Operational', 'LM358')  # Show part details
search_footprints('0805')          # Find footprints
```

Generate Output

```
generate_netlist()                 # KiCad netlist
generate_netlist(tool=KICAD9)      # Specific version
generate_xml()                     # XML for BOM tools
generate_pcb()                     # Direct to PCB
generate_svg()                     # SVG schematic
generate_schematic()               # KiCad schematic (V5)
```

Electrical Rules Check

```
ERC()                                  # Check for errors

# Suppress ERC on specific objects
net.do_erc = False
part.do_erc = False
pin.do_erc = False

# Custom ERC
def check_fanout(net):
    return len([p for p in net.pins if p.func == Pin.funcs.INPUT])

erc_assert('check_fanout(clk_net) <= 10', 'Clock fanout too high')
```

Hierarchy

```
# Method 1: Function with Interface
@SubCircuit
def amplifier(gain=10):
    op = Part('Amplifier_Operational', 'LM358')
    # ... build circuit ...
    return Interface(
        inp=op['INA'],
        out=op['OUT'],
        vcc=op['V+'],
        gnd=op['GND']
    )

# Use it
amp = amplifier(gain=5)
signal += amp.inp
output += amp.out

# Method 2: SubCircuit Class
class VoltageRegulator(SubCircuit):
    def __init__(self, voltage='3.3V'):
        super().__init__()
        reg = Part('Regulator_Linear', 'AMS1117-3.3')
        # ... build circuit ...
        self.vin = reg['VI']
        self.vout = reg['VO']
        self.gnd = reg['GND']

# Use it
vreg = VoltageRegulator()
power_in += vreg.vin

# Method 3: Context Manager
with SubCircuit('power_supply') as psu:
    reg = Part('Regulator_Linear', 'AMS1117-3.3')
    # ... parts created here go in the subcircuit ...
```

Part and Net Classes

```
# Define classes
passive = PartClass("passive", tolerance="5%", temp_rating="85C")
power_nets = NetClass("power", width="0.5mm", clearance="0.3mm")

# Apply to parts/nets
r = Part('Device', 'R', partclasses=passive)
vcc = Net('VCC', netclasses=power_nets)

# Hierarchical assignment
with SubCircuit('analog') as analog:
    analog.partclasses = PartClass("precision", tolerance="0.1%")
    # ... parts created here inherit the precision class ...
```

Part and Net Attributes for Schematics

```
# Part symbol attributes for schematic generation
part.symtx = 'HL'              # Flip horizontal, rotate left
part.symtx = 'VR'              # Flip vertical, rotate right
part[1].symio = 'i'            # Part pin is input
part[2].symio = 'o'            # Part pin is output

# Net attributes for schematic generation
net.netio = 'i'                # Input net
net.netio = 'o'                # Output net
net.stub = True                # Draw as stub
```

Common Patterns

```
# Power distribution
vcc, gnd = Net('VCC', drive=POWER), Net('GND', drive=POWER)
for part in [mcu, sensor, display]:
    part['VCC,GND'] += vcc, gnd

# Bus connections
data = Bus('DATA', 8)
mcu['PD[0:7]'] += data[0:7]

# Pull-up resistors
pullups = 8 * Part('Device', 'R', value='10K', dest=TEMPLATE)
vcc += [r[1] for r in pullups]
data_bus[0:7] += [r[2] for r in pullups]
```

Configuration

```
# Save current settings
skidl.config.store('.')  # Save into .skidlcfg file
```