Writing hardware

Concurrent and Distributed Systems

2018-12-21 Niels Bohr Institute

Kenneth Skovhede

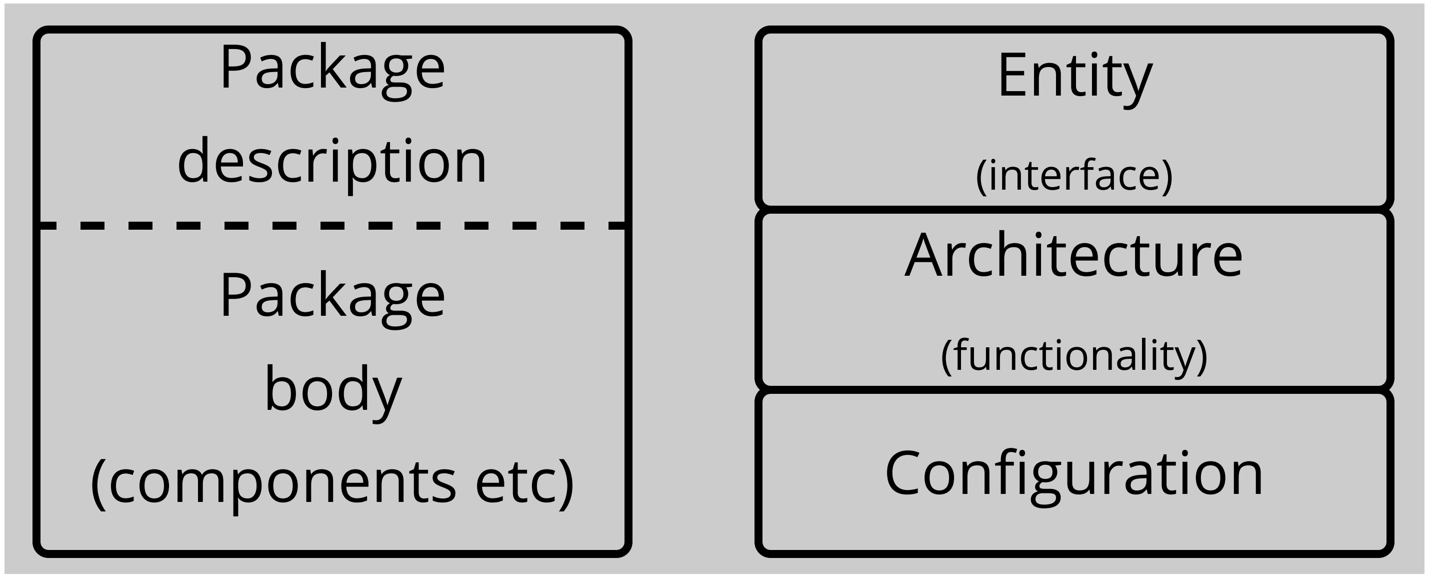
VHDL

Modelling tool for large scale systems

Mostly used to describe logic for FPGA or ASIC

From 1983 - based on Ada

VHDL structure



VHDL data types

Basic Types

- BIT (0,1)
- BOOLEAN (True, False)
- INTEGER (-inf ..., -1, 0, 1, ... inf)
  - NATURAL (0,1,2,3,4,...)
  - POSITIVE (1,2,3,4,...)

Array Types

```
type BIT_VECTOR is array (natural range<>) of BIT;
```

- BIT_VECTOR (natural range<>)
  - SIGNED (positive range<>)
  - UNSIGNED (positive range<>)

VHDL data types

std_logic / std_ulogic

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
```

- 'U': uninitialized. This signal hasn't been set yet.
- 'X': unknown. Impossible to determine this value/result.
- '0': logic 0
- '1': logic 1
- 'Z': High Impedance
- 'W': Weak signal, can't tell if it should be 0 or 1.
- 'L': Weak signal that should probably go to 0
- 'H': Weak signal that should probably go to 1
- '-': Don't care.

VHDL data types

std_logic_vector

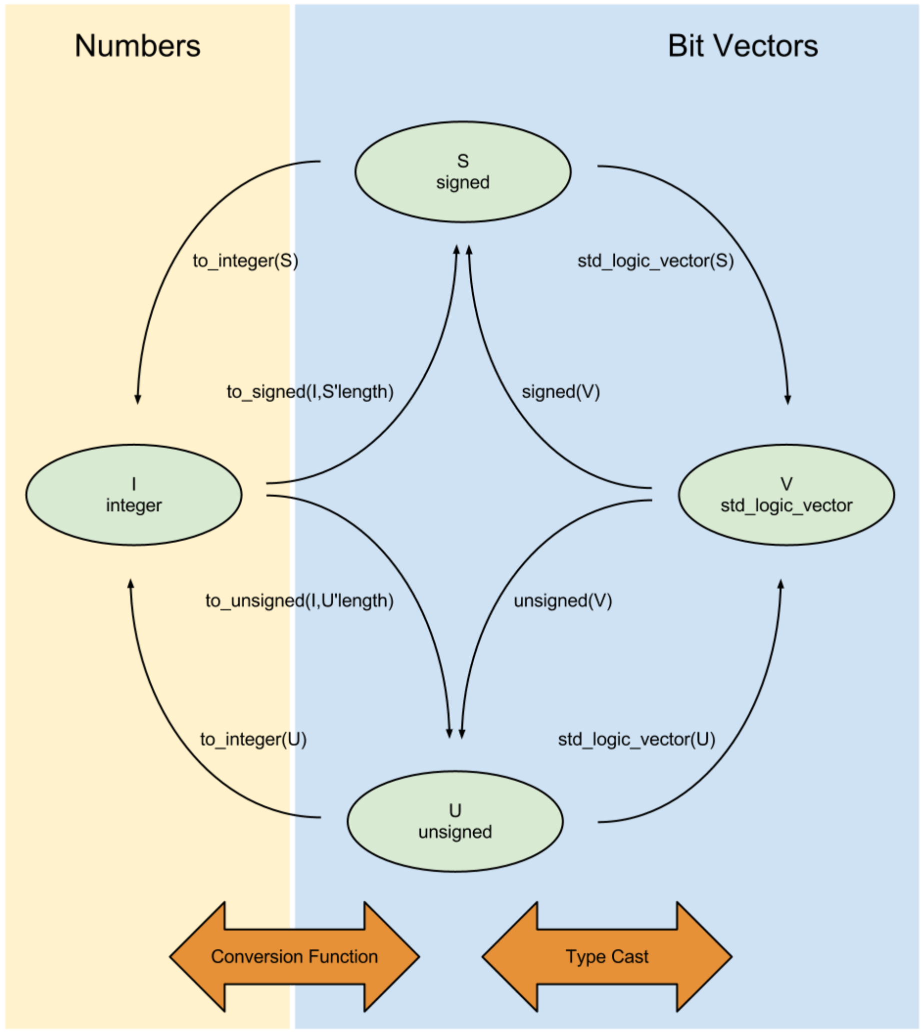
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
```

```
signal a : std_logic_vector(0 to 3);
signal b : std_logic_vector(3 downto 0);

a(0) <= '1';
a(1) <= '0';
a(2) <= '0';
a(3) <= '1';

b <= "1010";

-- not allowed: a <= b

for i in b'range loop
   a(i) <= b(i);
end loop;
```

VHDL data types

signed / unsigned / integer / std_logic_vector

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
```



VHDL data types

time units

```
type Time is range --implementation defined-- ;
  units
     fs;             -- femtosecond
     ps  = 1000 fs;  -- picosecond
     ns  = 1000 ps;  -- nanosecond
     us  = 1000 ns;  -- microsecond
     ms  = 1000 us;  -- millisecond
     sec = 1000 ms;  -- second
     min = 60   sec; -- minute
     hr  = 60   min; -- hour
  end units;
```

```
signal clock: std_logic;

while True loop
    clock <= '0';
    wait 4ns;
    clock <= '1';
    wait 4ns;
end loop;
```

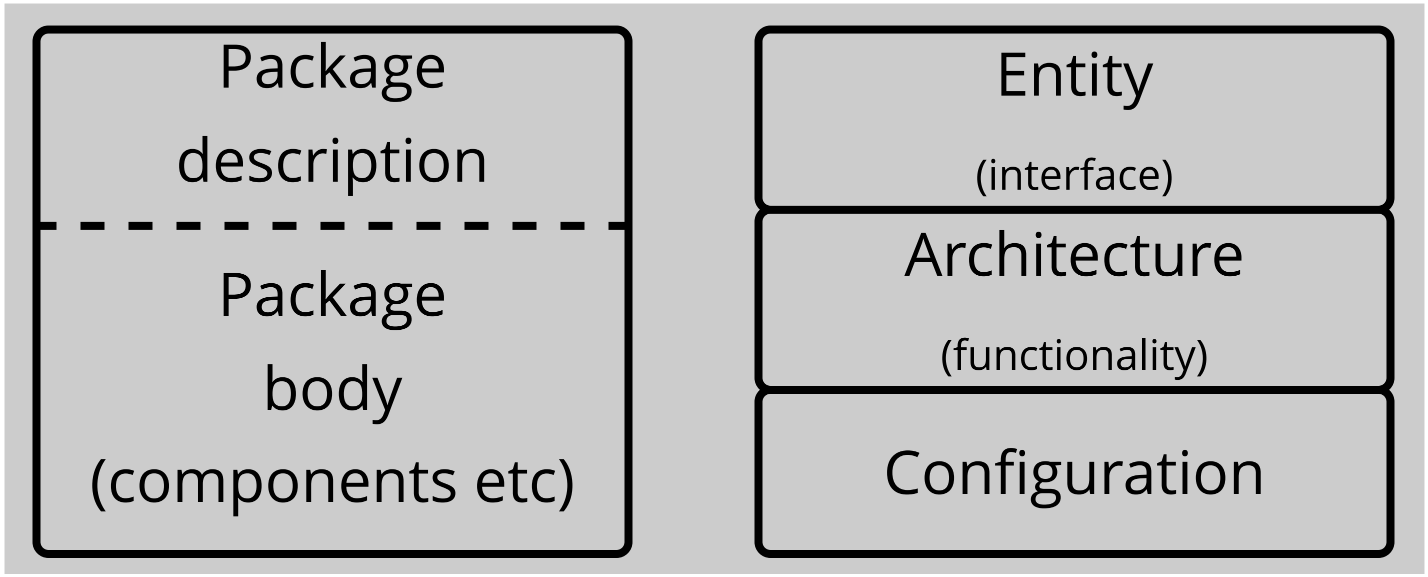
VHDL data types

Enumerations

```
type FSM_States is (Init, Read, Decode, Execute, Write);
type Test is ('0', '1', L, H);
```

```
TYPE State_type IS (A, B, C);
SIGNAL State : State_Type;

...

CASE State IS
        WHEN A =>
                IF P='1' THEN
                        State <= B;
                END IF;
        WHEN B =>
                IF P='1' THEN
                        State <= C;
                END IF;
        WHEN C=>
                IF P='1' THEN
                        State <= B;
                ELSE
                        State <= A;
                END IF;
        WHEN others =>
                State <= A;
END CASE;
```

VHDL structure



VHDL Example

```
entity Adder is
    port(
        A : in std_logic_vector(31 downto 0);
        B : in std_logic_vector(31 downto 0);
        C : out std_logic_vector(31 downto 0);
        CLK : in Std_logic;
        RST : in Std_logic
    );
end Adder;

architecture RTL of Adder is
begin
    process(CLK, RST, A, B)
    variable num : UNSIGNED(31 downto 0);
        begin
        if RST = '1' then
            num := TO_UNSIGNED(0, 32);
            C <= STD_LOGIC_VECTOR(TO_UNSIGNED(0, 32));
        elsif rising_edge(CLK) then
            num := TO_UNSIGNED(A) + TO_UNSIGNED(B);
            C <= STD_LOGIC_VECTOR(num);
        end if;
    end process;
end RTL;
```

```vhdl
entity Counter is
    port(
        Reset : in std_logic;
        Valid : in std_logic;
        Data : out std_logic_vector(31 downto 0);
        CLK : in Std_logic;
        RST : in Std_logic
    );
end Counter;

architecture RTL of Counter is
begin
    process(CLK, RST, Reset, Valid)
    signal count: UNSIGNED(31 downto 0);
    variable nextcount: UNSIGNED(31 downto 0);
        begin
        if RST = '1' then
            nextcount := TO_UNSIGNED(0, 32);
            count <= TO_UNSIGNED(0, 32);
            Data <= STD_LOGIC_VECTOR(TO_UNSIGNED(0, 32));
        elsif rising_edge(CLK) then
            nextcount := count;

            if Reset = '0' then
                nextcount := TO_UNSIGNED(0, 32);
            elsif Valid = '1' then
                nextcount := nextcount + TO_UNSIGNED(1, 32);
            end if;
            Data <= std_logic_vector(nextcount);
            count <= nextcount;
        end if;
    end process;
end RTL;
```

Connecting designs

```
entity TopLevel is
  port(
    Reset : in std_logic;
    Count0 : in std_logic;
    Count1 : in std_logic;
    Data0 : out std_logic_vector(31 downto 0);
    Data1 : out std_logic_vector(31 downto 0);
    DataSum: out std_logic_vector(31 downto 0);
    RST : in Std_logic;
    CLK : in Std_logic
  );
end TopLevel;

architecture RTL of TopLevel is
    signal tmp0 : std_logic_vector(31 downto 0);
    signal tmp1 : std_logic_vector(31 downto 0);
begin
    Counter0: entity work.Counter
    port map (
        Reset => Reset,
        Valid => Count0,
        Data => tmp0,
        CLK => CLK,
        RST => RST
    );

    Counter1: entity work.Counter
    port map (
        Reset => Reset,
        Valid => Count1,
        Data => tmp1,
        CLK => CLK,
        RST => RST
    );

    Data0 <= tmp0;
    Data1 <= tmp1;
    DataSum <= std_logic_vector(unsigned(tmp0) + unsigned(tmp1));
end RTL;
```

Testbench

```vhdl
entity Counter_tb is
end;

architecture TestBench of Counter_tb is
  signal CLOCK : Std_logic;
  signal StopClock : BOOLEAN;
  signal RST : Std_logic;
  signal Reset : std_logic;
  signal Count0 : std_logic;
  signal Count1 : std_logic;
  signal Data0 : std_logic_vector(31 downto 0);
  signal Data1 : std_logic_vector(31 downto 0);
  signal DataSum: std_logic_vector(31 downto 0);
begin
  uut: entity work.TopLevel
  port map (
    Reset => Reset,
    Count0 => Count0,
    Count1 => Count1,
    Data0 => Data0,
    Data1 => Data1,
    DataSum => DataSum,
    RST => RST,
    CLK => CLOCK
  );

  Clk: process
  begin
    while not StopClock loop
      CLOCK <= '1';
      wait for 5 NS;
      CLOCK <= '0';
      wait for 5 NS;
    end loop;
    wait;
  end process;

  TraceFileTester: process
  begin

     ...

  end process;
end architecture TestBench;
```

Testbench

```vhdl
wait until rising_edge(CLOCK);

RST <= '0';
Count0 <= '0';
Reset <= '1';

wait until rising_edge(CLOCK);

Reset <= '0';

wait until rising_edge(CLOCK);

assert to_integer(unsigned(Data0)) = 0 report "Failed after reset on Data0"
assert to_integer(unsigned(Data1)) = 0 report "Failed after reset on Data1"
assert to_integer(unsigned(DataSum)) = 0 report "Failed after reset on DataSum"

wait until falling_edge(CLOCK);

Count0 <= '1';

wait until rising_edge(CLOCK);

assert to_integer(unsigned(Data0)) = 1 report "Failed after reset on Data0"
assert to_integer(unsigned(Data1)) = 0 report "Failed after reset on Data1"
assert to_integer(unsigned(DataSum)) = 1 report "Failed after reset on DataSum"

...
```

Verilog

Merged with SystemVerilog in 2008

SystemVerilog has extended verification models

From 1984 - based on C, but interpreted

Counter

```
module Counter (rst, clk, enable, data);

input rst;
input clk;
input enable;

output [31:0] data;
reg [31:0] counter;

always @ (posedge clk or posedge rst)
  if (rst)
    counter = {32{1'b0}};
  else
  if (enable == 1'b1)
    begin
        counter = counter + 1'b1;
    end

assign data <= counter;

endmodule
```

Testbench

```verilog
`include "counter.v"
module counter_tb();
reg clock, reset, enable;
wire [31:0] counter;

initial begin
  $display ("time\t clk reset enable counter");
  $monitor ("%g\t %b    %b      %b       %b",
          $time, clock, reset, enable, counter);

  // Reset state
  clock = 1;
  reset = 0;
  enable = 0;

  // Timing tests
  #2 reset = 1;
  #2 reset = 0;
  #2 enable = 1;
  #2 enable = 0;
  #2 $finish;
end

// Simple clock
always begin
  #1 clock = ~clock;
end

// Wire up counter to testbench
my_counter U_counter (
clock,
reset,
enable,
counter
);

endmodule
```

Other approaches

High level synthesis

Register-level design

PyRTL

```python
def one_bit_add(a, b, carry_in):
    assert len(a) == len(b) == 1  # len returns the bitwidth
    sum = a ^ b ^ carry_in  # operators on WireVectors build the hardware
    carry_out = a & b | a & carry_in | b & carry_in
    return sum, carry_out

def ripple_add(a, b, carry_in=0):
    a, b = pyrtl.match_bitwidth(a, b)
    if len(a) == 1:
        sumbits, carry_out = one_bit_add(a, b, carry_in)
    else:
        lsbit, ripplecarry = one_bit_add(a[0], b[0], carry_in)
        msbits, carry_out = ripple_add(a[1:], b[1:], ripplecarry)
        sumbits = pyrtl.concat(msbits, lsbit)
    return sumbits, carry_out

# instantiate an adder into a 3-bit counter
counter = pyrtl.Register(bitwidth=3, name='counter')
sum, carry_out = ripple_add(counter, pyrtl.Const("1'b1"))
counter.next <<= sum

# simulate the instantiated design for 15 cycles
sim_trace = pyrtl.SimulationTrace()
sim = pyrtl.Simulation(tracer=sim_trace)
for cycle in range(15):
    sim.step({})
sim_trace.render_trace()
```

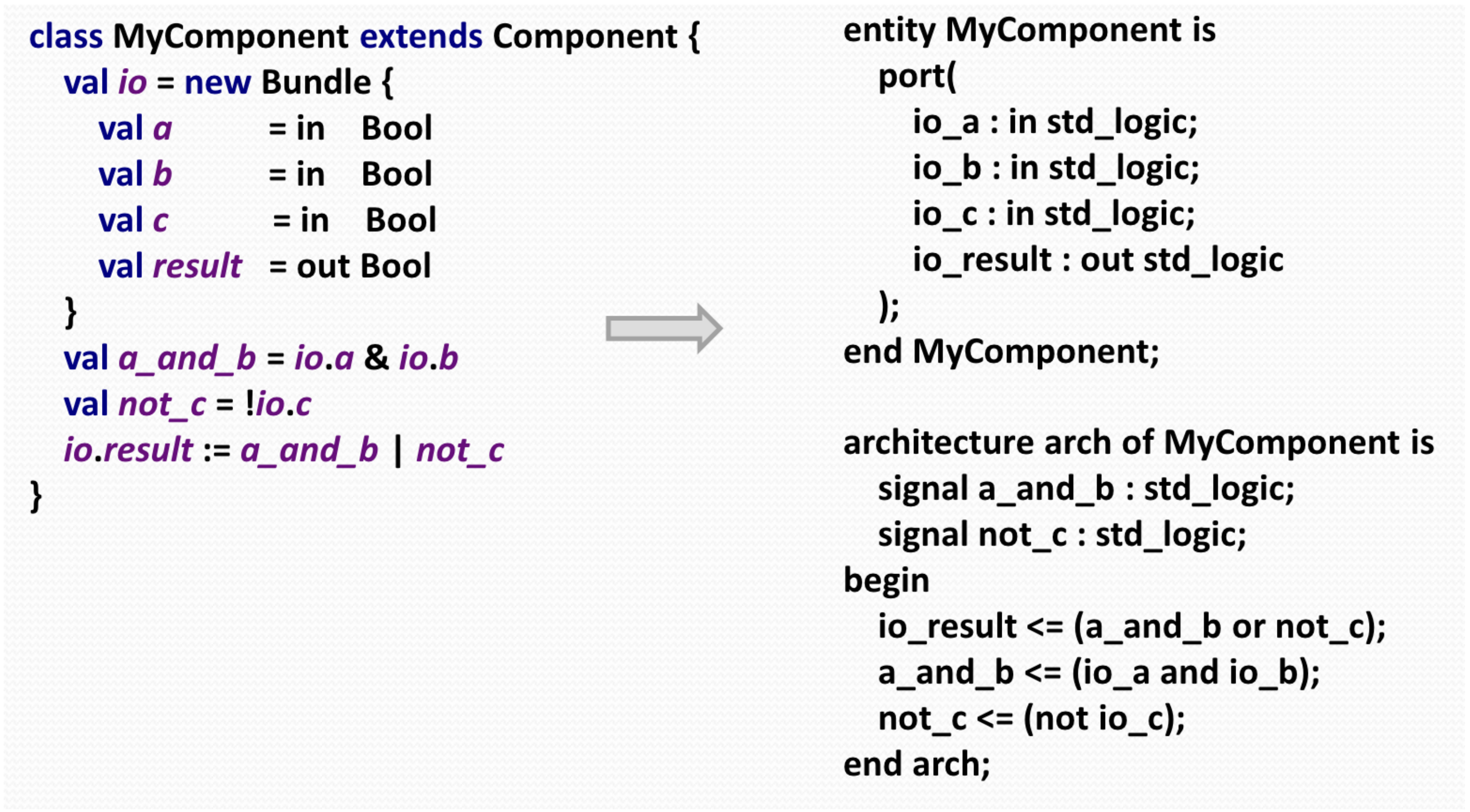
From https://ucsbarchlab.github.io/PyRTL/

MyHDL

```python
from myhdl import *

ACTIVE = 0
DirType = enum('RIGHT', 'LEFT')

def jc2_alt(goLeft, goRight, stop, clk, q):
    @instance
    def logic():
        dir = DirType.LEFT
        run = False
        while True:
            yield clk.posedge
            # direction
            if goRight == ACTIVE:
                dir = DirType.RIGHT
                run = True
            elif goLeft == ACTIVE:
                dir = DirType.LEFT
                run = True
            # stop
            if stop == ACTIVE:
                run = False
            # counter action
            if run:
                if dir == DirType.LEFT:
                    q.next[4:1] = q[3:]
                    q.next[0] = not q[3]
                else:
                    q.next[3:] = q[4:1]
                    q.next[3] = not q[0]

    return logic
```

From http://www.myhdl.org/docs/examples/jc2.html

SpinalHDL

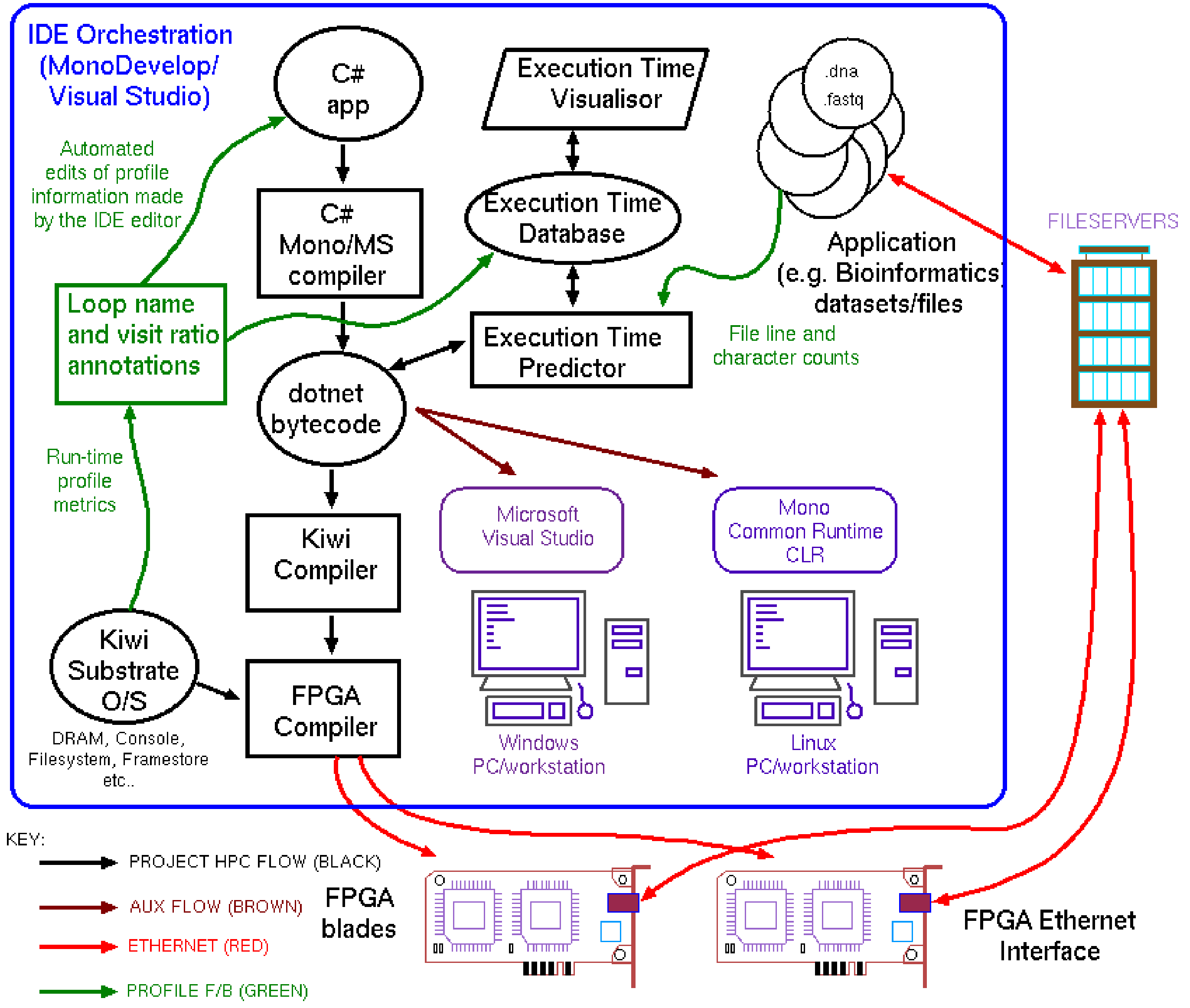


```
class MyComponent extends Component {
  val io = new Bundle {
    val a       = in   Bool
    val b       = in   Bool
    val c       = in   Bool
    val result  = out Bool
  }
  val a_and_b = io.a & io.b
  val not_c = !io.c
  io.result := a_and_b | not_c
}
```

```
entity MyComponent is
  port(
    io_a : in std_logic;
    io_b : in std_logic;
    io_c : in std_logic;
    io_result : out std_logic
  );
end MyComponent;

architecture arch of MyComponent is
  signal a_and_b : std_logic;
  signal not_c : std_logic;
begin
  io_result <= (a_and_b or not_c);
  a_and_b <= (io_a and io_b);
  not_c <= (not io_c);
end arch;
```

From https://spinalhdl.github.io/SpinalDoc/

Kiwi



From https://www.cl.cam.ac.uk/~djg11/kiwi/

HLS

LegUp: http://legup.eecg.utoronto.ca/

Vivado HLS / OpenCL:

https://www.xilinx.com/support/documentation-navigation/design-hubs/dh0012-vivado-high-level-synthesis-hub.html

PySME

```python
from sme import Network, Function, External, Bus, SME, Types
t = Types()

class Producer(Function):
    def setup(self, ins, outs):
        self.map_outs(outs, "out")
        self.v1 = 0 # type: t.u7
        self.v2 = 0 # type: t.u7

    def run(self):
        self.out["val1"] = self.v1
        self.out["val2"] = self.v2
        self.v1 += 1
        self.v2 += 1
        if self.v1 > 100:
            self.v1 = 0
            self.v2 = 0

class Mul(Function):
        def setup(self, ins, outs):
                self.map_ins(ins, "valbus")
                self.map_outs(outs, "mulbus")

        def run(self):
                self.mulbus["res"] = self.valbus["val1"] * self.valbus["val2"]

class SomsOps(Network):
        def wire(self):
                mulbus = Bus("MulBus", [t.u14("res")])
                mulbus["res"] = 0
                self.tell(mulbus)
                prod = Producer("Producer", [], [valbus])
                self.tell(prod)
                mul = Mul("Mul", [valbus], [mulbus])
                self.tell(mul)
```

SMEIL

```
proc id(in inbus)
    bus idout {
        val: i32 = 0;
    };
{
    idout.val = inbus.val;
}

// plusone proc
proc plusone(in inbus)
    bus plusout {
       val: i32 = 0;
    };
{
    trace("Wrote value {}", inbus.val);
    plusout.val = inbus.val + 1;

}

network plusone_net() {
    instance plusone_inst of plusone(id_inst.idout);
    instance id_inst of id(plusone_inst.plusout);
}
```

From https://github.com/truls/libsme

SMEIL

```python
from sme import *

class Id(SimulationProcess):
    def setup(self, ins, outs, result):
        self.map_outs(outs, "out")
        self.map_ins(ins, "inp")

    def run(self):
        print("Got val", self.out["val"])
        result[0] = self.out["val"] + 1
        self.out["val"] = self.inp["val"]
        self.out["valid"] = True

@extends("addone.sme", ['-t', 'trace.csv', '--force'])
class AddOne(Network):
    def wire(self, result):
        plus_out = ExternalBus("plusone_inst.plusout")
        id_out = ExternalBus("idout")
        p = Id("Id", [plus_out], [id_out], result)
        self.add(plus_out)
        self.add(id_out)
        self.add(p)


if __name__ == "__main__":
    sme = SME()
    result = [0]
    sme.network = AddOne("AddOne", result)
    sme.network.clock(100)
    print("Final result was ", result[0])
```

From https://github.com/truls/libsme