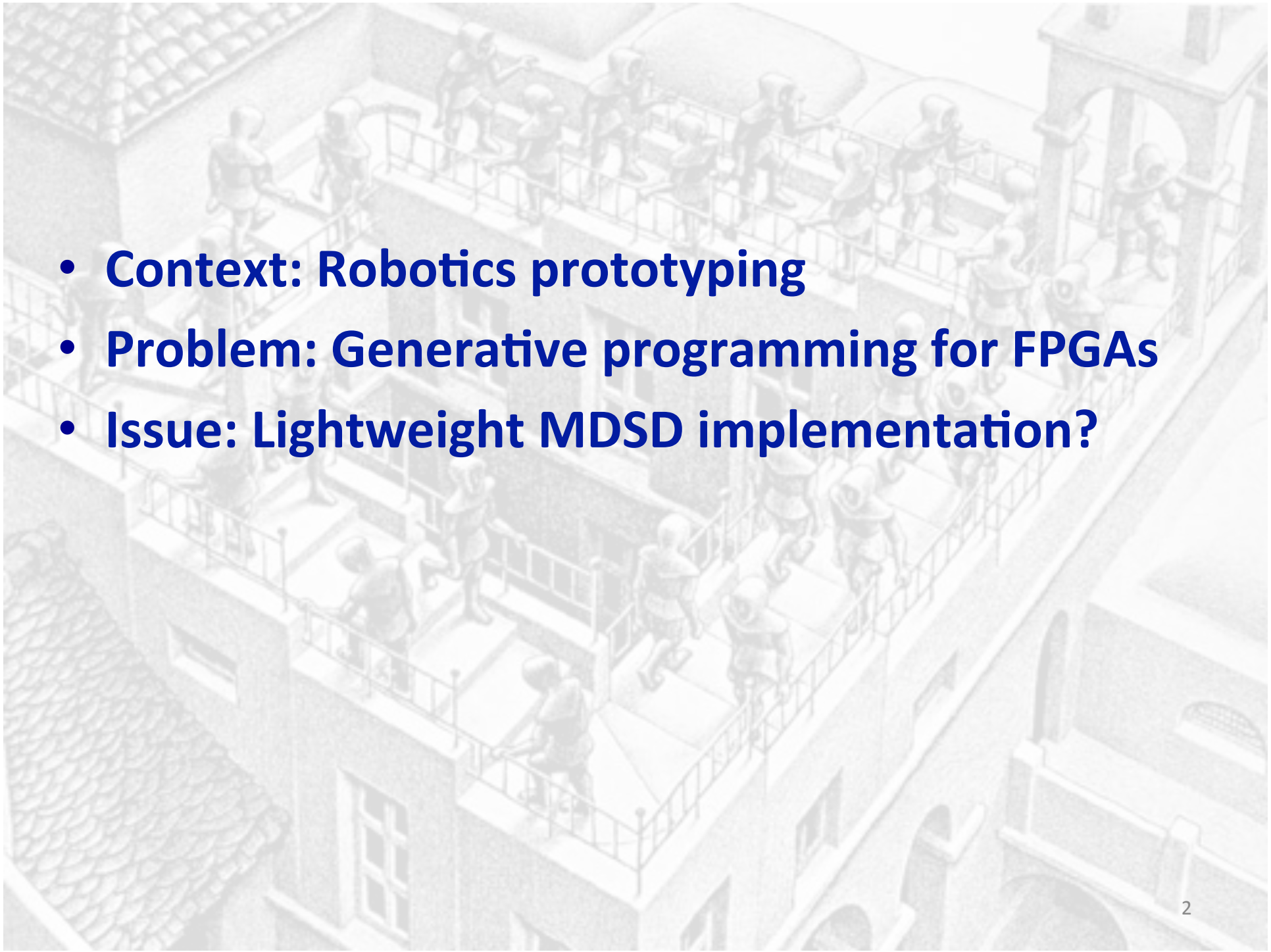
Automatic Generation of an Electronics-to-Middleware Interface Layer *(Work-in-progress)*

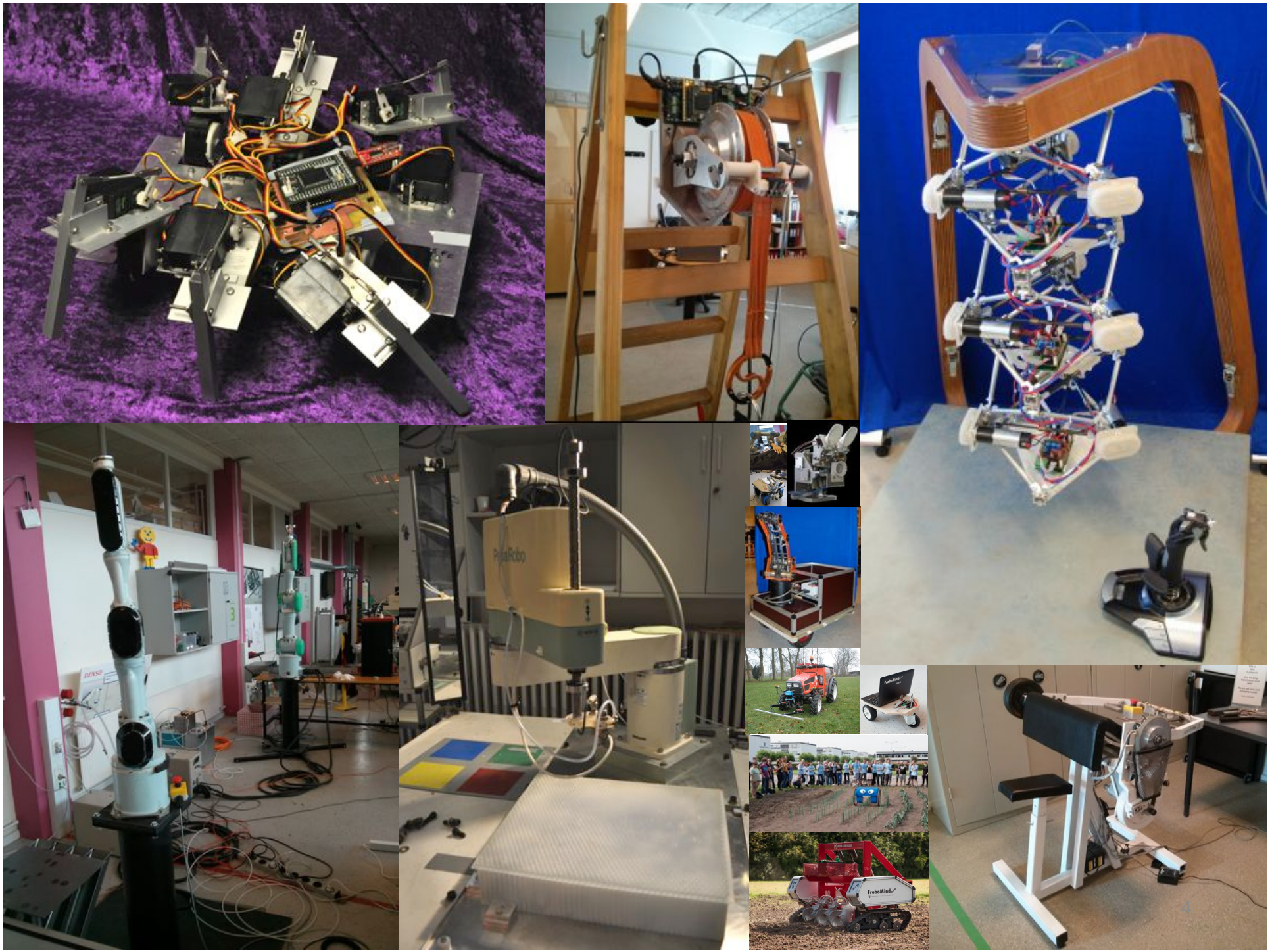
Ulrik Pagh Schultz

(with: Anders Lange, Anders Sørensen)

University of Southern Denmark

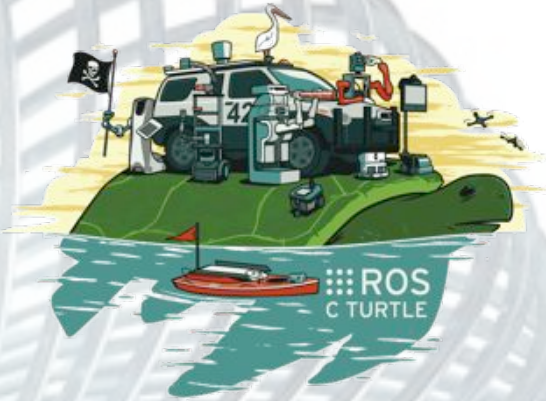


- Context: Robotics prototyping
- Problem: Generative programming for FPGAs
- Issue: Lightweight MDSD implementation?


FrobaRobo
FrobaMind

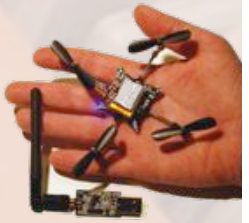
Flexible, reusable solutions?

- **General:**
  - control algorithm
  - high-level software
- **Specific:**
  - mechanics, actuators, sensors (...)
  - low-level software
- **In-between:**
  - digital electronics
  - CPU



FPGAs in robotics

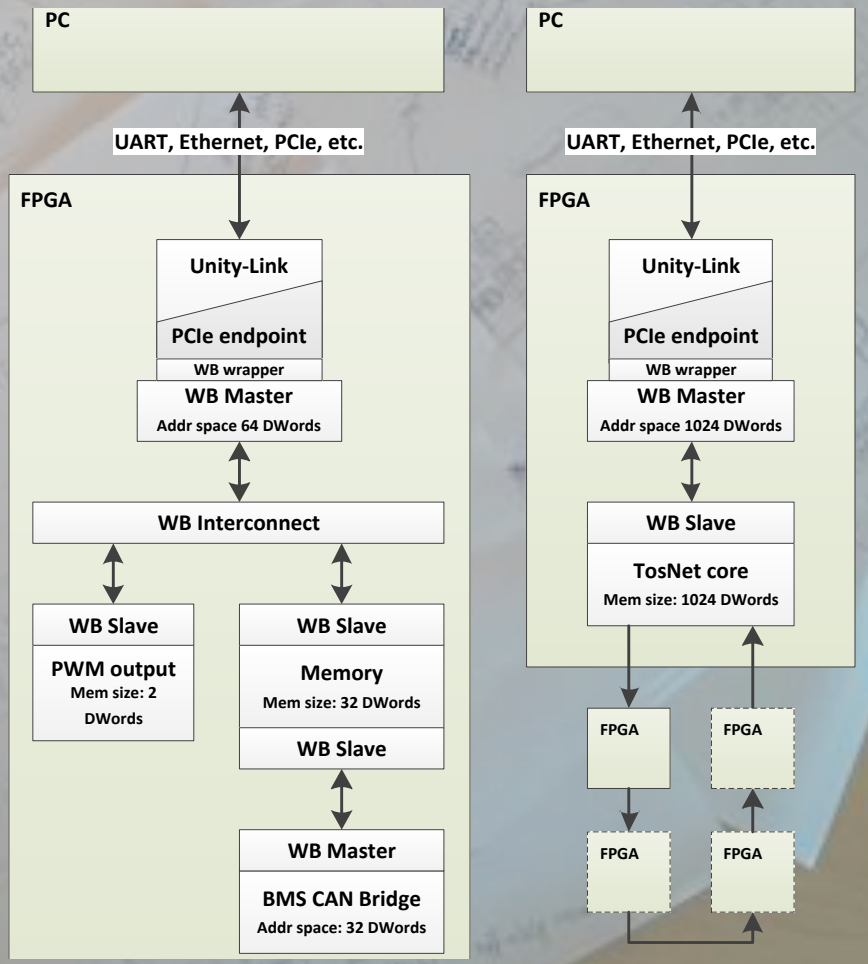
- Flexible digital electronics: very interesting
- Massively parallel data crunching: not as interesting (...)
- "Perfect" real-time behavior: useful
  - motor control, sensing, communication, ...
  - per-functionality digital logic (parallelism)
  - flexibility through state machines, DSPs, ...



Difficult to use! (...)

Unity: Generative low-level robotics architecture

- Key issue: interface high-level software to low-level electronics
- Approach: specify low-level interface, generate all required gateware and software
- Platform: FPGAs, real-time network, modular electronics
- Status: framework works reliably, generator is work-in-progress



(Details)

Unity: Generative low-level robotics architecture

- **Key issue: interface high-level software to low-level electronics**
- **Approach: specify low-level interface, generate all required gateware and software**
- **Platform: FPGAs, real-time network, modular electronics**
- **Status: framework works reliably, generator is work-in-progress**

(Details)

```
interface ftdi_uart_if {
     rx: inputport
     tx: outputport
}
...
   fpga1.pin[A12] = ft232h.tx
   fpga1.pin[B6] = leds.bit(0)
...
node led_blinker {
 board: XC6LX45(
     ft232h=ftdi_uart_if)
  led: public outputport(7)
  configuration {
   board.ft232h = unity_serial(
     serial=board.ft232h,
     rate=mbaud(6))
   board.leds = led
  }
}
```

Unity: Generative low-level robotics architecture

- **Key issue: interface high-level software to low-level electronics**
- **Approach: specify low-level interface, generate all required gateware and software**
- **Platform: FPGAs, real-time network, modular electronics**
- **Status: framework works reliably, generator is work-in-progress**

(Details)

Generic components:

- CPUs, DSPs, state machines
- configured

Specific components:

- motor controller, hardware interface
- implemented in VHDL

Interconnects:

- shared memory, data buses, ...
- mostly implicit, but could be explicit

Unity: Generative low-level robotics architecture

- **Key issue: interface high-level software to low-level electronics**
- **Approach: specify low-level interface, generate all required gateware and software**
- **Platform: FPGAs, real-time network, modular electronics**
- **Status: framework works reliably, generator is work-in-progress**

```
component A {
 VHDL instantiation
}
component B {
 statemachine specification
}
component C {
 control equation for DSP
}
component D {
 C program
 real-time configuration (*)
}
```

- Use extensible, modular grammar system?
- Shooting sparrows with cannons?

(*): Lange et al: HartOS - A hardware implemented RTOS for hard real-time applications

A step further: Migrating software components into a real-time context

- Concretely, ROS:
  - C++ implementation → softcore with HartOS (*)
  - publish/subscribe → shared memory
  - RPC (aka “service”) → FIFO+address/data bus
- Interface dependencies specified

- Gateware equivalent
- Requires static component structure and thread timing

Concrete approach: "GPCE(*)"

- **In both cases: MDSD**
  1. **declare requirements, populate metamodel, transform, generate VHDL code**
  2. **simplify code generation by using fixed architecture populated by configuring VHDL components**

```
interface ftdi_uart_if {
  rx: inputport
  tx: outputport
}
interface ft232_if: ftdi_uart_if | ...;
board XC6LX45 {
  interface {
    ft232h: ft232_if
    leds: outputport(7)
    clk200Mi: inputport(freq=200000000)
  }
  fpga1: fpga("sp6lx45-3")
  configuration {
    ft232h = configuration.ft232h
    fpga1.pin[A11] = clk200Mi
    fpga1.pin[C14] = ft232h.rx
    ...
```

Toolchain?

Step 1: PyParsing

```
lbrace,rbrace,lpar,rpar,colon = map(Suppress,'{}():')
...
usp_member = Group(ident('name') + colon
 + usp_port_type)('member')
usp_extends = colon + delimitedList(ident,delim='|')
usp_interface = Literal('interface') + ident('name')
 + Optional(usp_extends)('extends') + lbrace
 + Group(ZeroOrMore(usp_member))('members') + rbrace
```

```
interface ftdi_uart_if {
  rx: inputport
  tx: outputport
}
interface ft232_if: ftdi_uart_if | ...;
board XC6LX45 {
  interface {
    ft232h: ft232_if
    leds: outputport(7)
    clk200Mi: inputport(freq=200000000)
  }
  fpga1: fpga("sp6lx45-3")
  configuration {
    ft232h = configuration.ft232h
    fpga1.pin[A11] = clk200Mi
    fpga1.pin[C14] = ft232h.rx
    ...
```

```
[['interface', 'ftdi_uart_if', [],
  [['rx', 'inputport'],
    ['tx', 'outputport']]],
 ['interface', 'ft232_if', ['ftdi_uart_if', ...]]
 ...
```

Step 2: ?

PyNABL?

NABL* (presumably, not tried)

```
namespaces interface member
rules
  Interface(i,_,_):
    defines unique interface i
    scopes member
  Member(m,_)
    defines unique member m
  InterfaceExtends(i):
    refers to interface i
    imports member (transitive)
```

PyNABL (work-in-progress, ...)

```
ub = Binder()
uinterface = ub.namespace("interface")
umember = ub.namespace("member")

uinterface.name = Definition()
uinterface.members = Scope(umember)
uinterface.extends =
  References(uinterface).
  importsTransitiveScope(umember)
umember.name = Definition()
```

(Details)

*: Konat et al: Declarative Name Binding and Scope Rules, SLE 2012

Lightweight toolchain?!

- Parser and model builder: internal Python DSL
- Model to model: it's a functional language (...)
- Model to code: Django templates (...)
- Infrastructure: python modules / unix

Claim: we need more lightweight and easy-to-use MDSD toolchains!