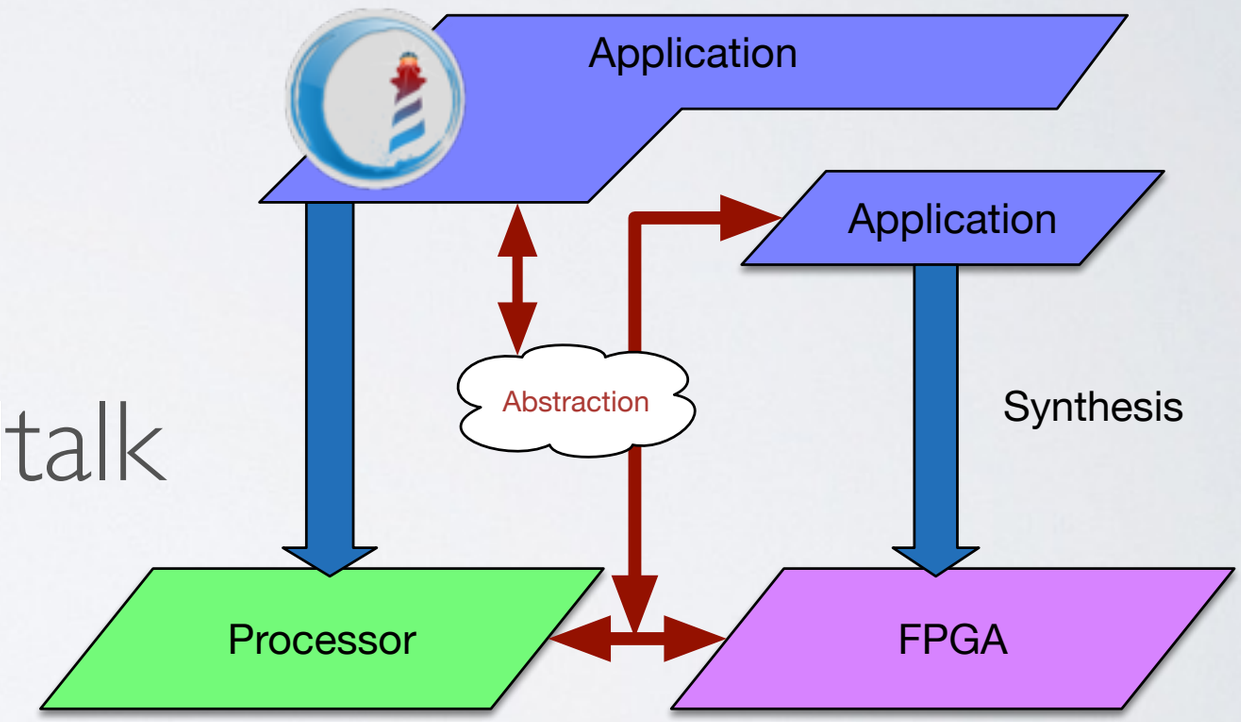
A META MODEL SUPPORTING BOTH HARDWARE AND SMALLTALK-BASED EXECUTION OF FPGA CIRCUITS

Le Xuan Sang[1,2]
Loïc Lagadec[1], Luc Fabress[2],
Jannik Laval[2] and Noury Bouraqadi[2]

[1] Lab-STICC, ENSTA Bretagne
[2] Institut Mines-Telecom, Mines Douai

ABOUT ME

- LE Xuan Sang
- 2014-2017 PHD student
  - Coupling FPGA / Smalltalk in robotic applications

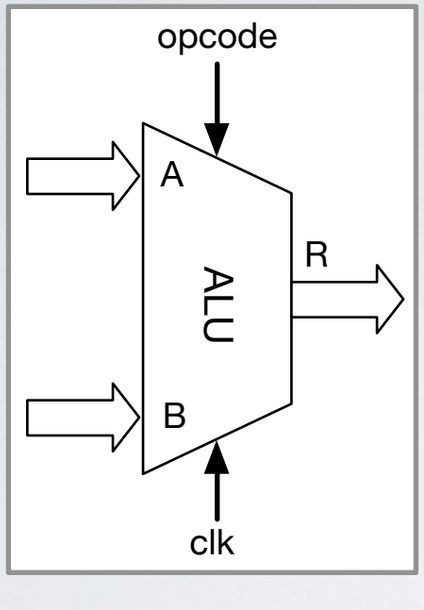


ENSTA Bretagne

MINES Douai

DESCRIPTION OF A CIRCUIT USING VHDL

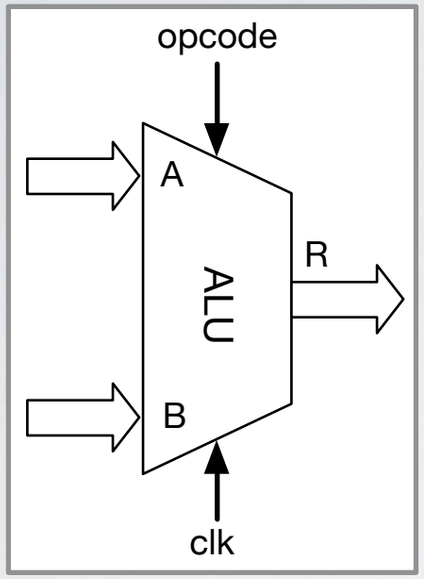
Interface




```vhdl
-- libraries declararion
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL ;
-- Entity declaration
entity SimpleALU is
port (
    clk:in std_logic;
    A,B:in std_logic_vector(31 downto 0);
    opcode:in std_logic;
    R:out std_logic_vector(31 downto 0)
);
end SimpleALU;
-- Architecture
architecture arch of SimpleALU is
    Signal r1,r2,r3:signed(31 downto 0)
                :=(others=>'0');
begin
    r1<=signed(A);
    r2<=signed(B);
    R<=std_logic_vector(r3);
    process(clk)
    begin
        if rising_edge(clk) then
            case opcode is
                when '0'    => r3<=(r1+r2);
                when '1'    => r3<= r1 AND r2;
                when others => (others=>'0');
            end case;
        end if;
    end process;
end architecture;
```

DESCRIPTION OF A CIRCUIT USING VHDL

Interface



Components

- Operated in **parallel**
- Activated when its inputs change
- Signals assignment alway need a **propagation delay** to take effect

```vhdl
-- libraries declararion
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL ;
-- Entity declaration
entity SimpleALU is
port (
    clk:in std_logic;
    A,B:in std_logic_vector(31 downto 0);
    opcode:in std_logic;
    R:out std_logic_vector(31 downto 0)
);
end SimpleALU;
-- Architecture
architecture arch of SimpleALU is
    Signal r1,r2,r3:signed(31 downto 0)
                :=(others=>'0');
begin
    r1<=signed(A);
    r2<=signed(B);
    R<=std_logic_vector(r3);
    process(clk)
    begin
        if rising_edge(clk) then
            case opcode is
                when '0'    => r3<=(r1+r2);
                when '1'    => r3<= r1 AND r2;
                when others => (others=>'0');
            end case;
        end if;
    end process;
end architecture;
```

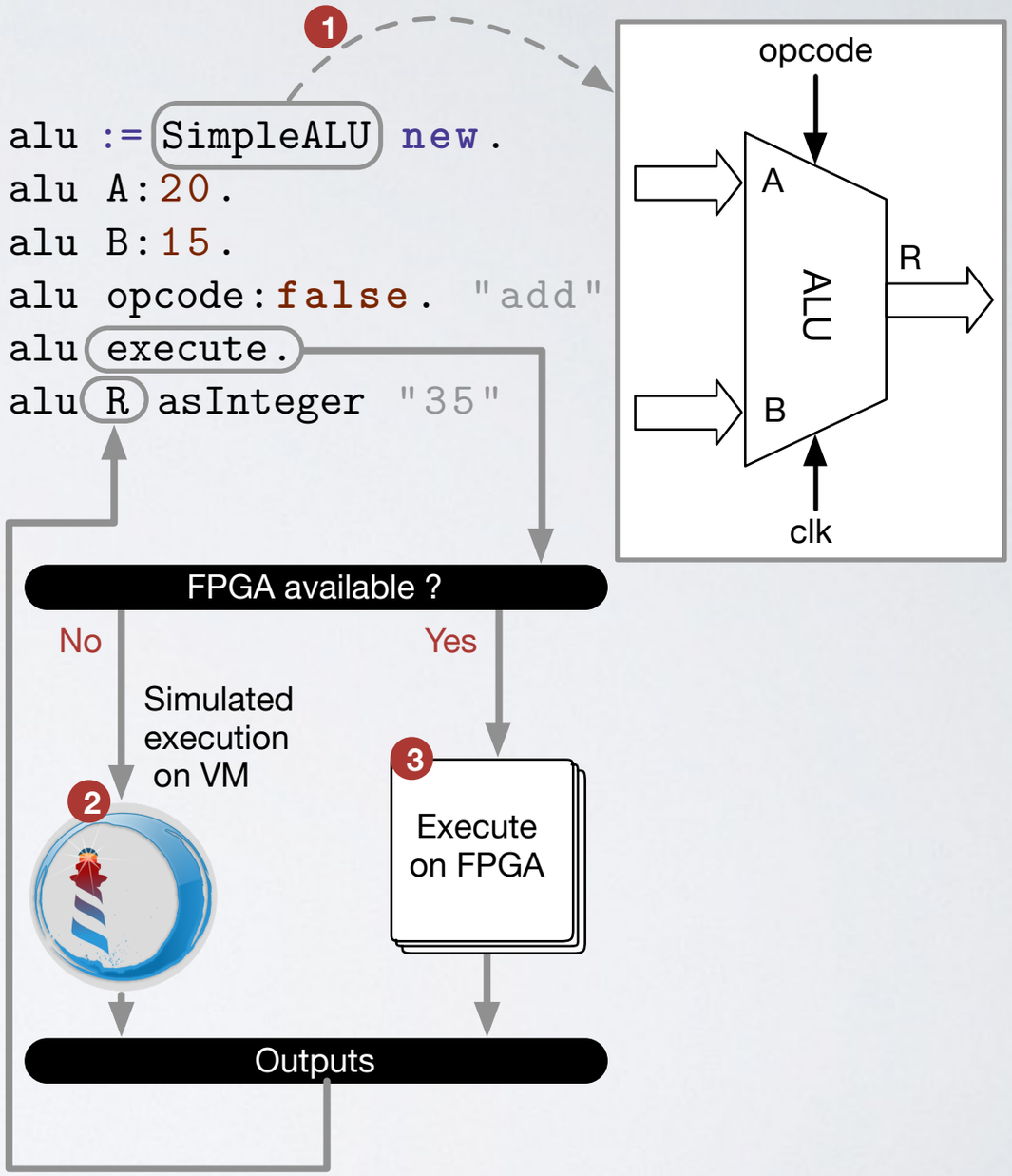
FPGA CIRCUIT MODELLING

3 problems

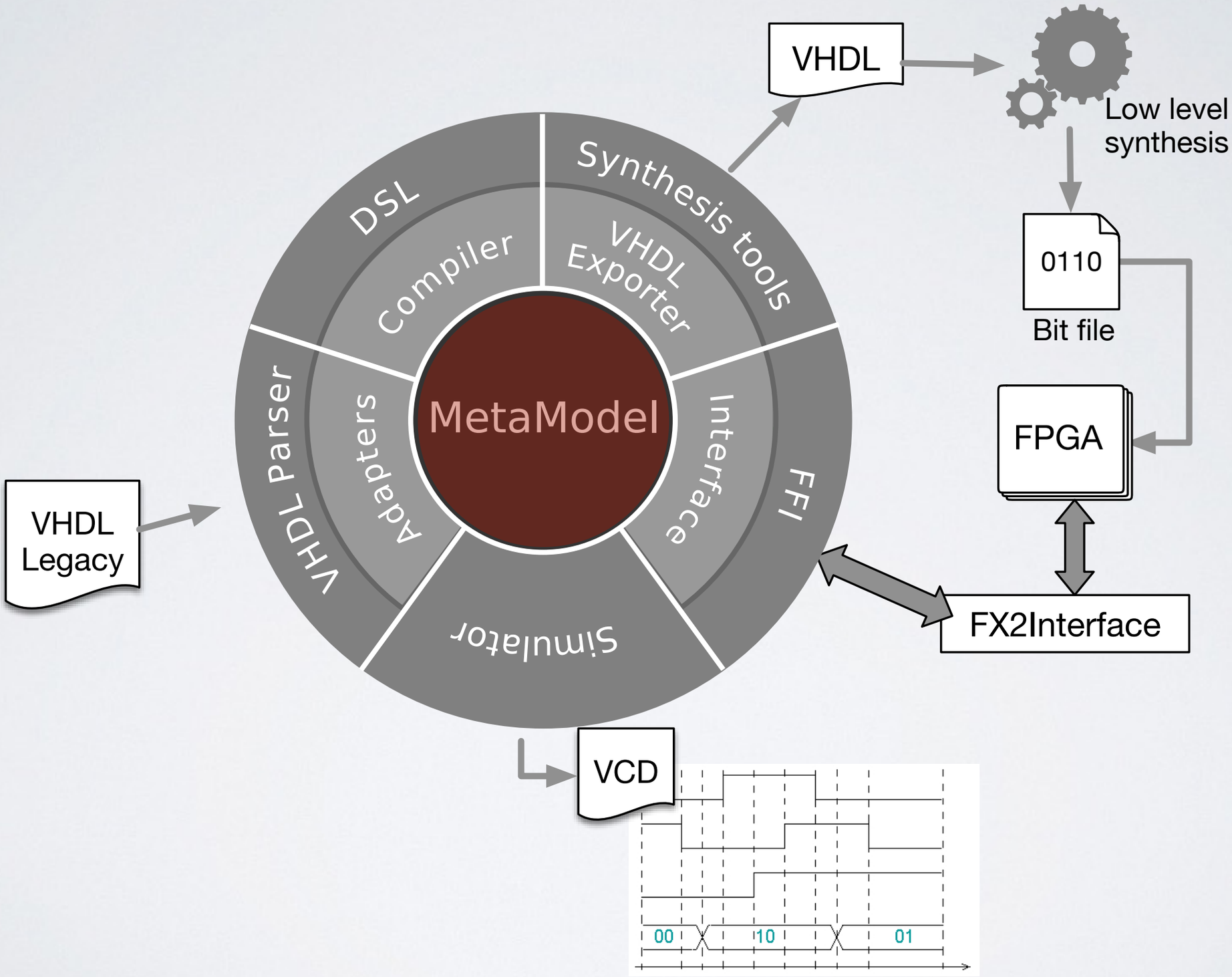
(1) The meta-model must capture both the structure and the behaviour of the circuit

(2) The parallel characteristics + propagation delay of the FPGA circuits must be taken into account

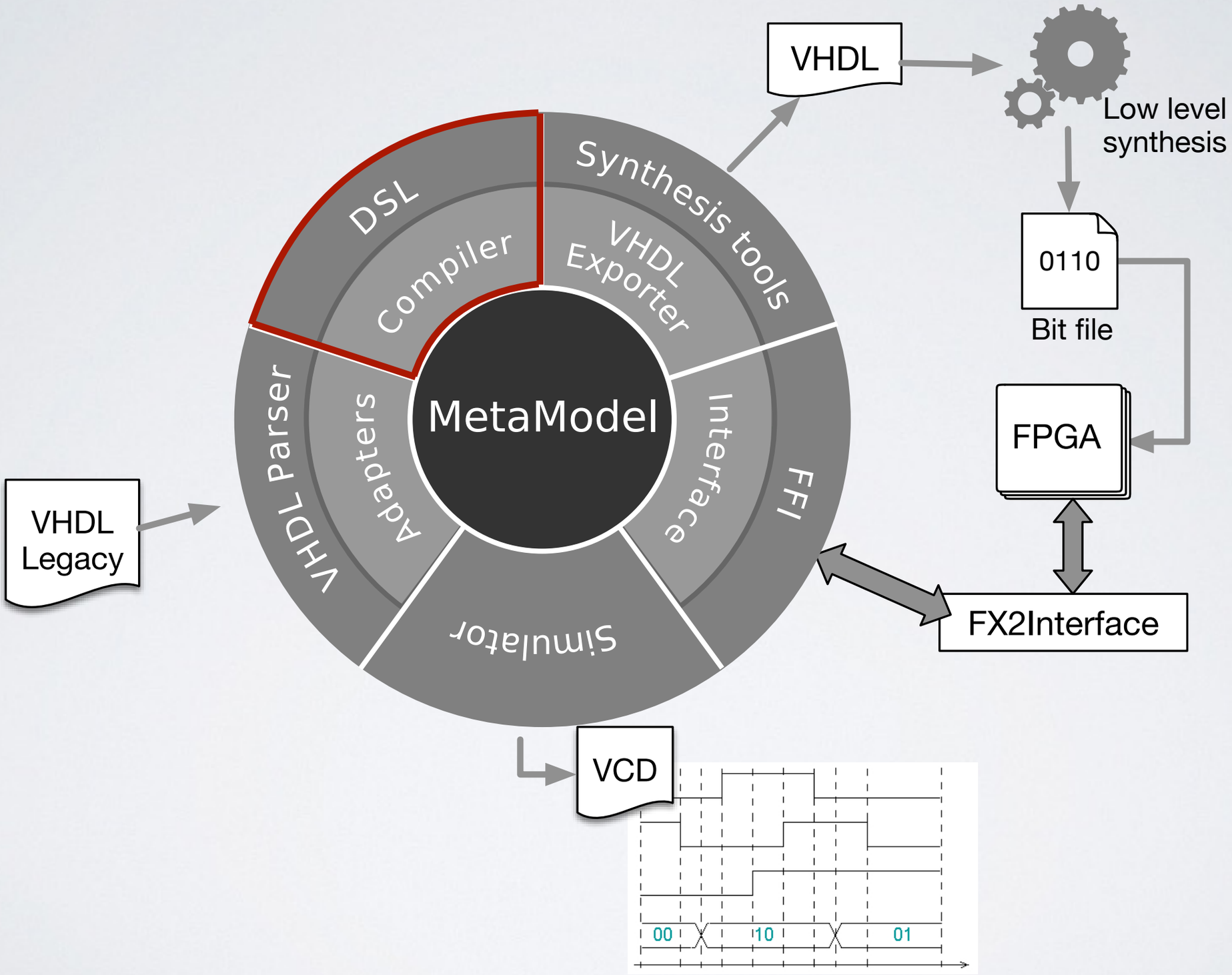
(3) Transparent execution of the circuit on FPGA



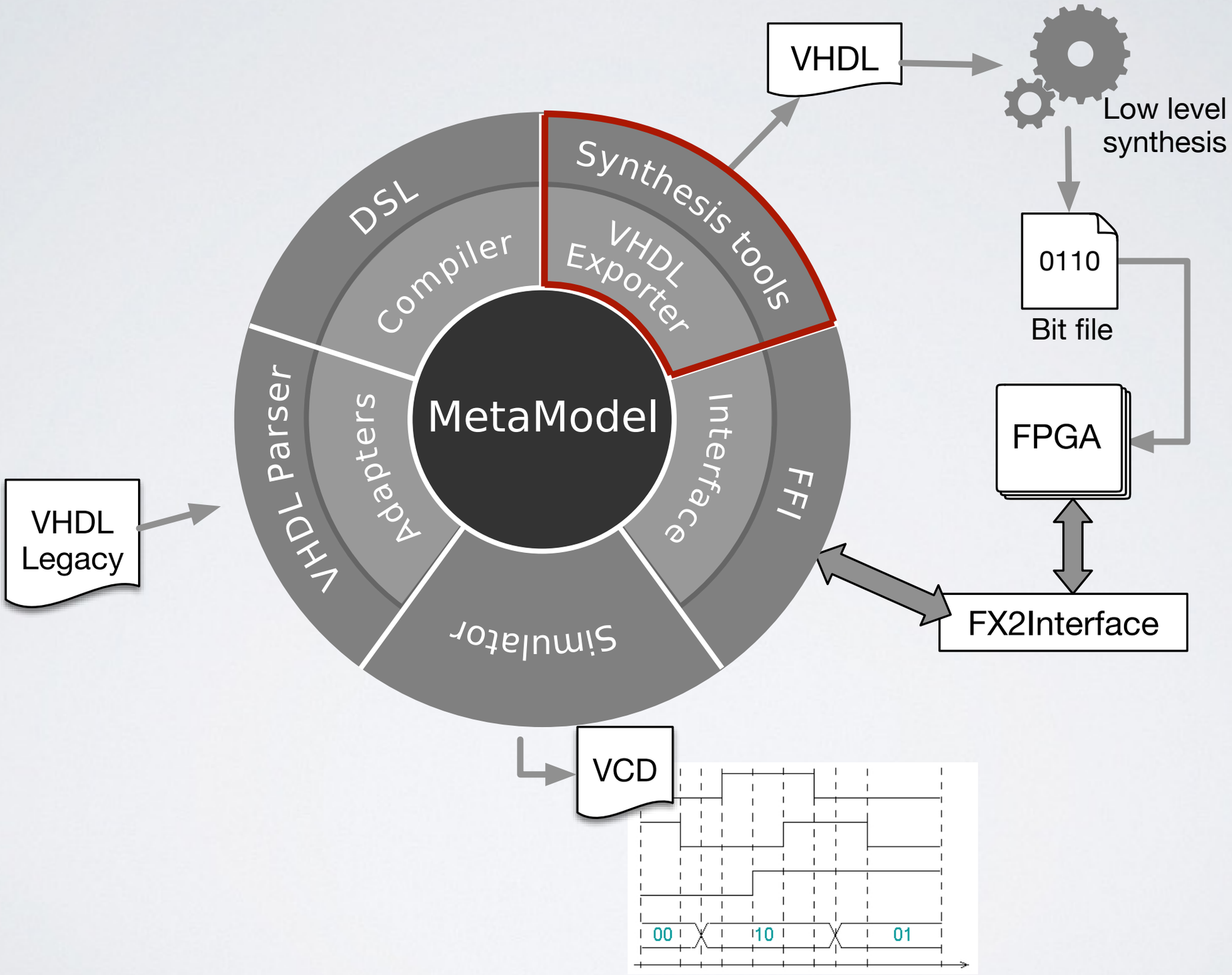
FULL SOLUTION



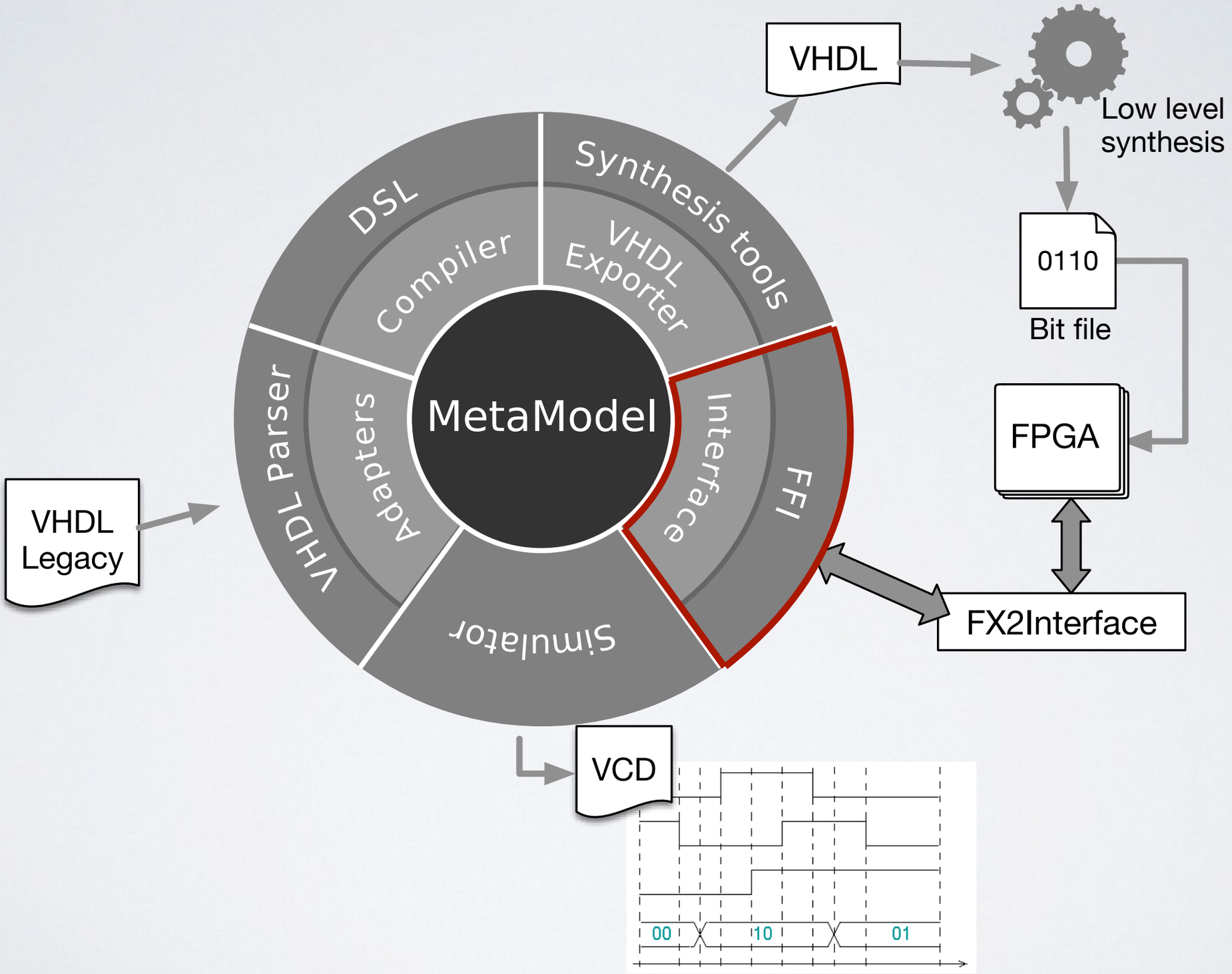
FULL SOLUTION



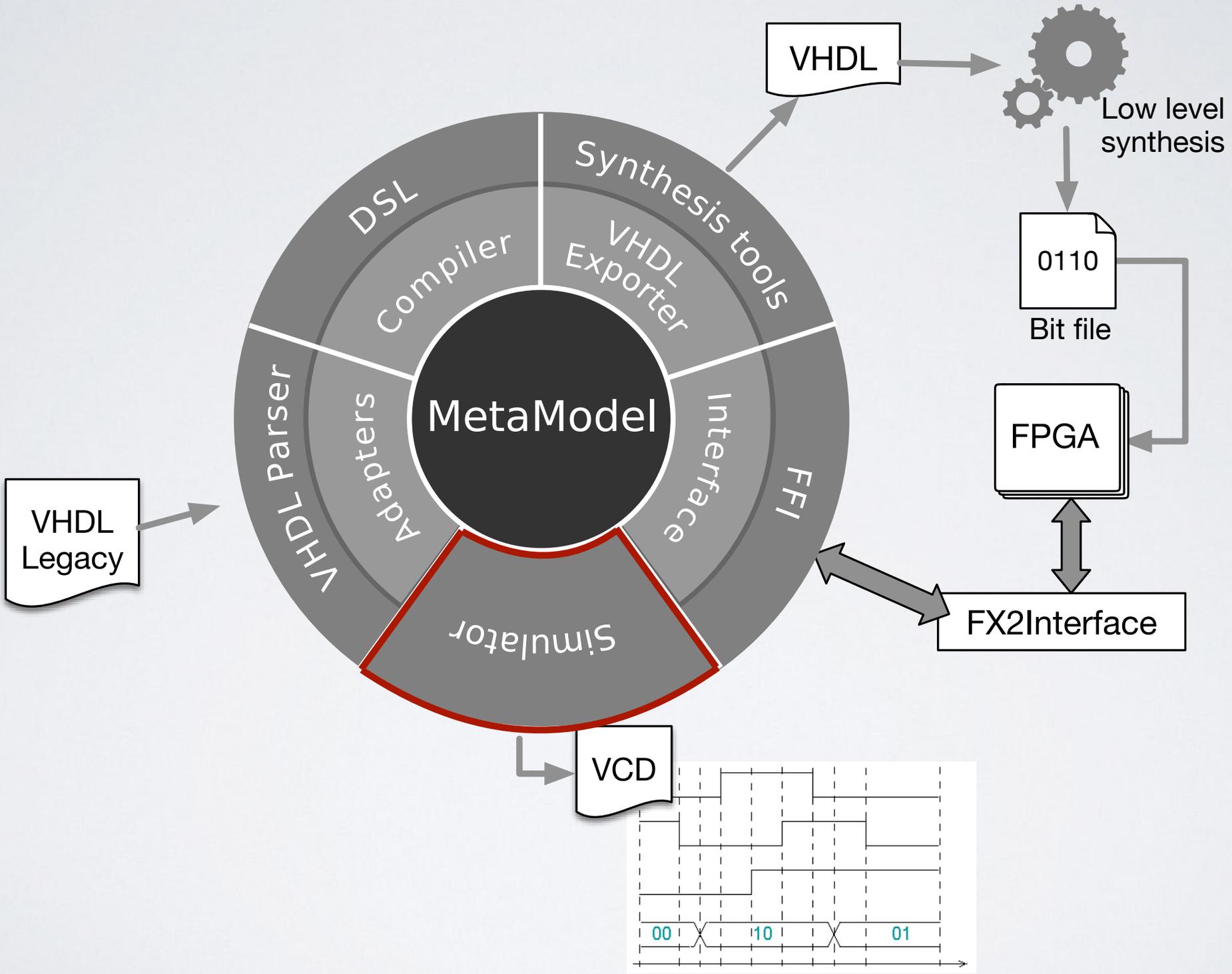
FULL SOLUTION



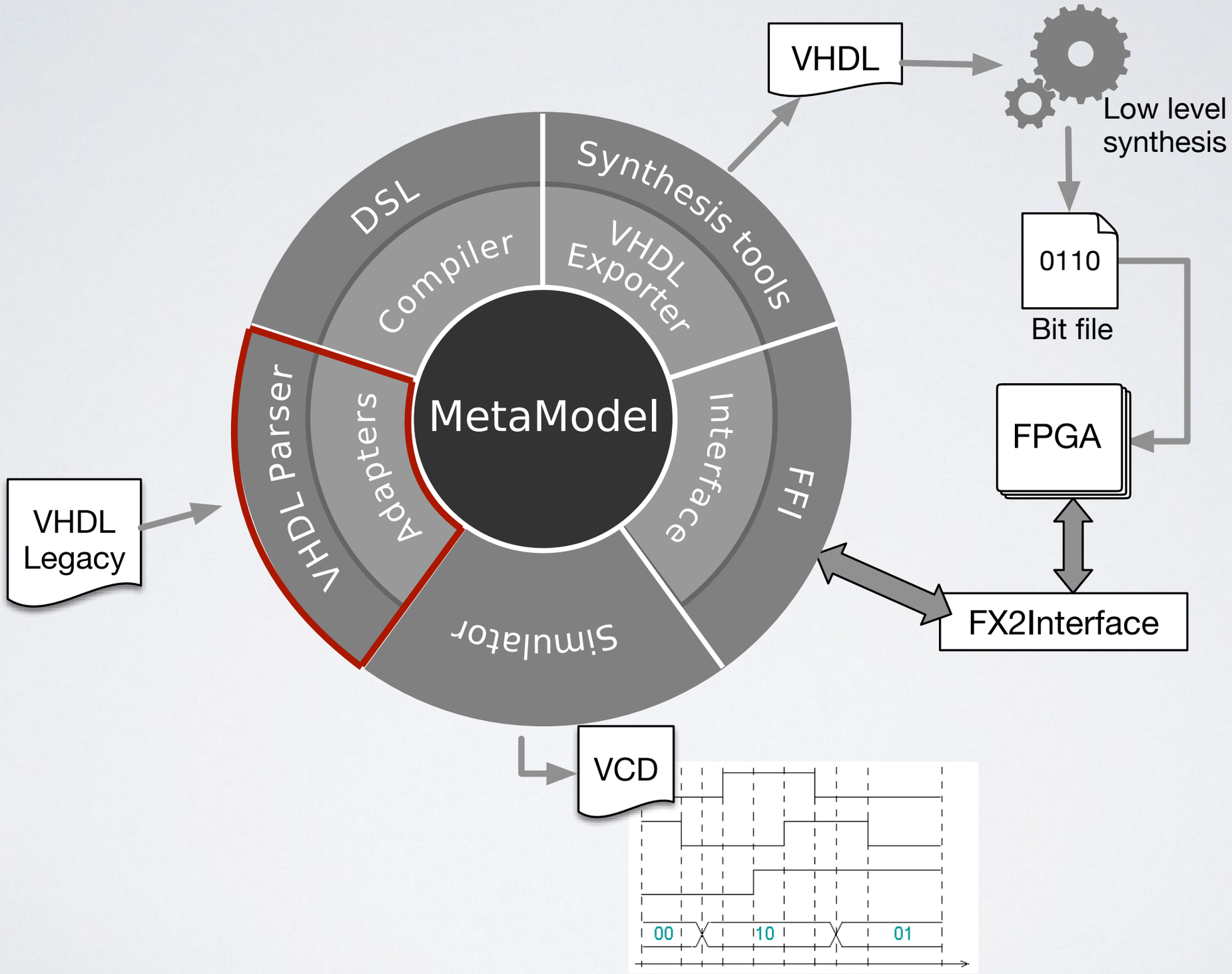
FULL SOLUTION



FULL SOLUTION



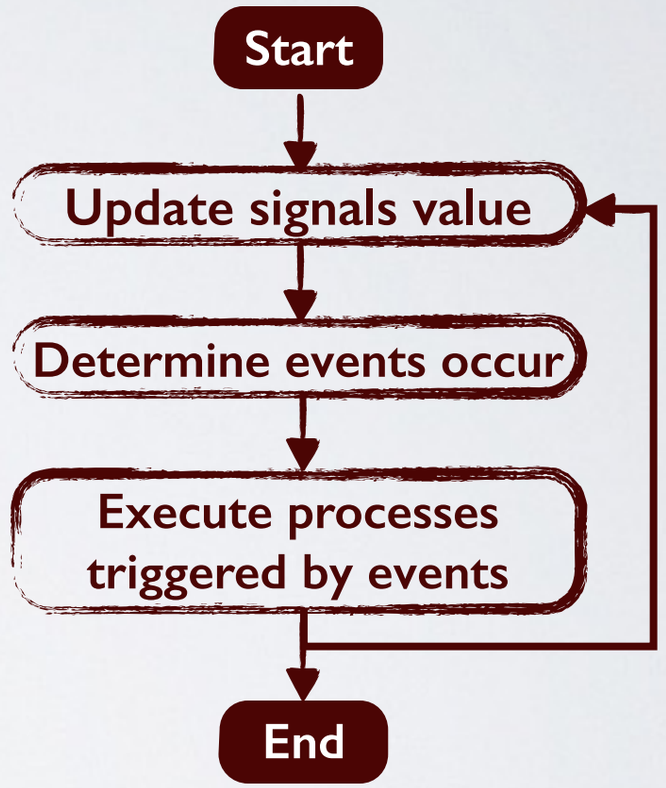
FULL SOLUTION



SMALLTALK-BASED EXECUTION OF THE FPGA CIRCUIT MODEL

EVENT-DRIVEN FOR PROPAGATION EXECUTION

- All parallel components are considered as processes with sensitivity list (event listener)
- The value of signals is changed only after update
- Events occur on signals change
- All processes that have changed signals as inputs will become active
  - Execution of a triggered process may trigger other processes



CIRCUIT MODEL EXECUTION

- Inputs ports are assigned at the first time
- Each circuit has a **done** signal
- The circuit is triggered by the clock **clk**
- The execution is repeated until the **done** signal is asserted

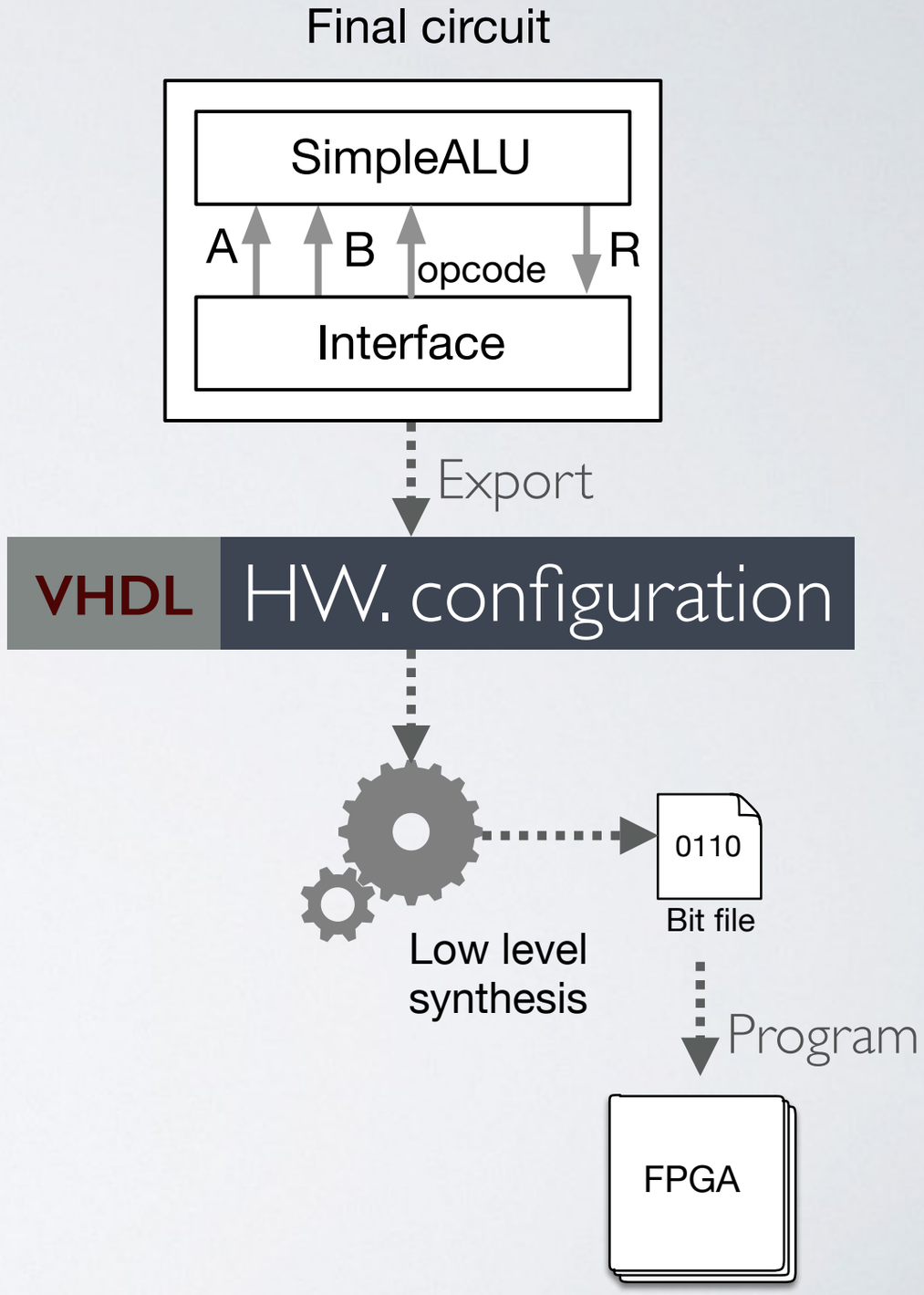
```
alu := SimpleALU new.
alu A:20.
alu B:15.
alu opcode:false.
alu execute.
```

HARDWARE-BASED EXECUTION OF THE FPGA CIRCUIT MODEL

AUTOMATIC DEPLOYEMENT ON FPGA

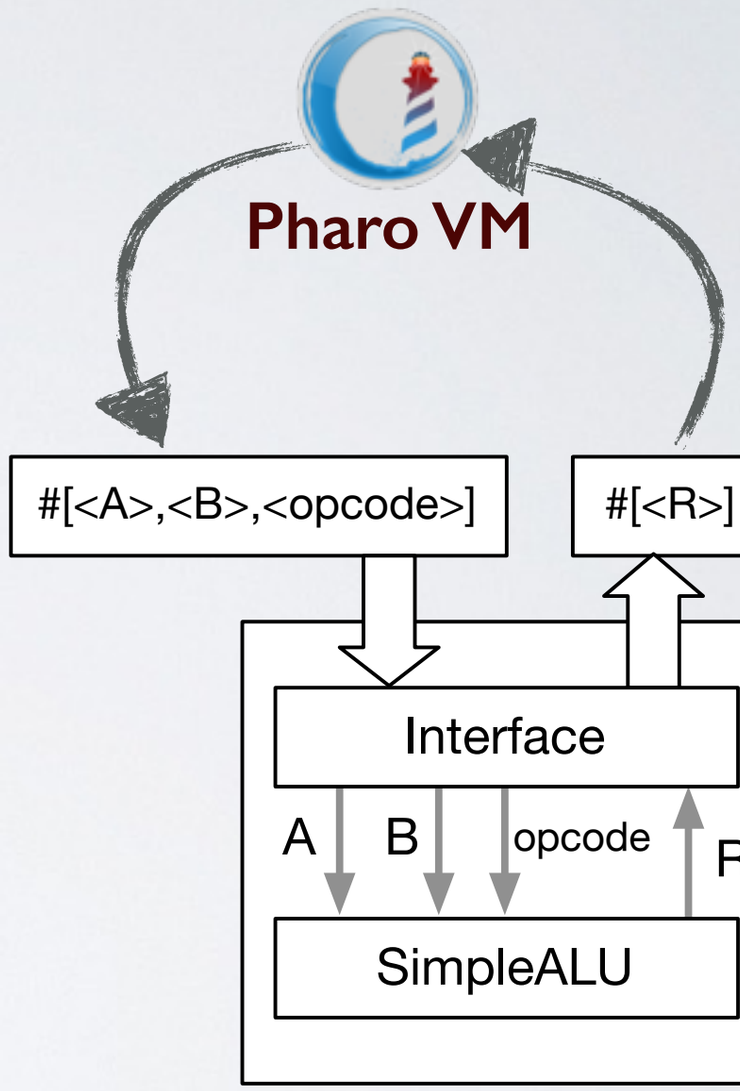
1. Generate automatically the communication interface
2. The VHDL codes is exported from the model
3. The hardware configuration is generated
4. The vendor tool-chain is used for low level synthesis

No effort for developper



HARDWARE BASED EXECUTION OF CIRCUIT MODEL

1. The inputs signals are assigned
2. The input values are encode as ByteArray and sent to FPGA
3. The interface circuit decodes the data it received and sends to the target circuit
4. The outputs of the circuit can be fetched from Smalltalk in a reverse way

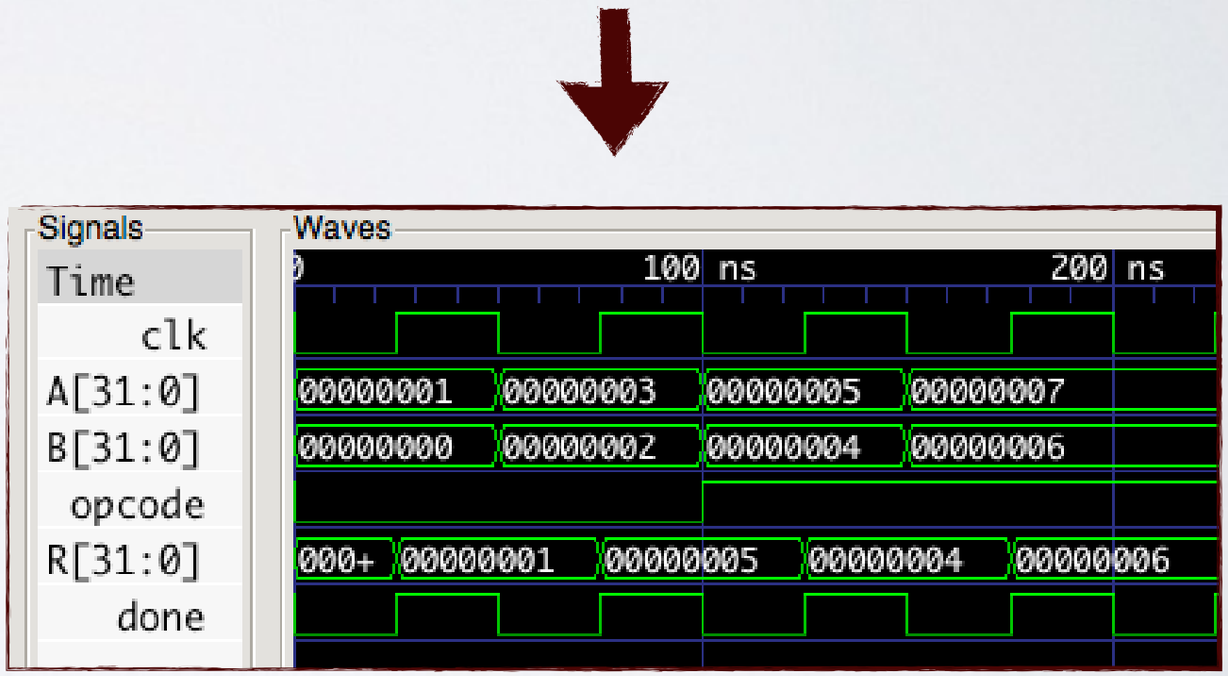


CONTROLLABILITY AND DEBUGGING

POST-MORTEM ANALYSIS OF SIMULATION

- The circuit is executed for a specific amount of time
- The inputs are changed at some specific points of time
- Next input changes are stored in a time queue
- Timing information and signals values are traced continuously into a VCD file

```
alu := SimpleALU new.
queue := {
    #clk clock: 50ns.
    #A change:{1. 3. 5. 7} every:50ns.
    #B change:{0. 2. 4. 6} every:50ns.
    #opcode change:{'0'.'1'} every:100ns.
} asTimeQueueFor:400ns.
stream := WaveFormStream on:'ALU.vcd'.
alu modellingExecution:#execute timeQueue:queue
    dumpOn:stream.
```

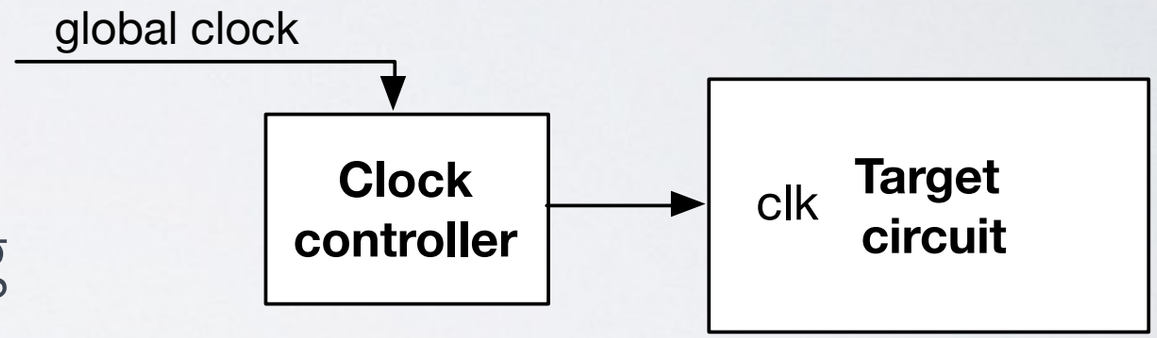


HARDWARE BREAKPOINT

- How can we manually stop the execution of FPGA circuit ?
- Enables the software like debug capabilities on hardware

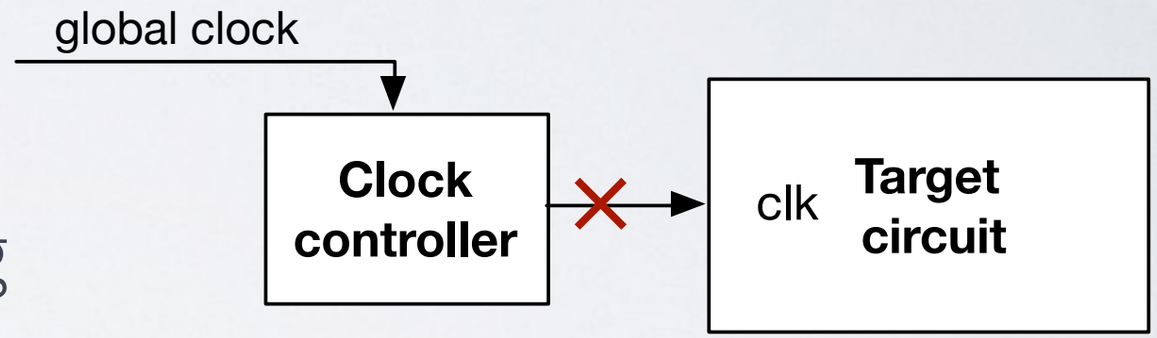
HARDWARE BREAKPOINT

- How can we manually stop the execution of FPGA circuit ?
- Enables the software like debug capabilities on hardware



HARDWARE BREAKPOINT

- How can we manually stop the execution of FPGA circuit ?
- Enables the software like debug capabilities on hardware



STATIC BREAKPOINT

A simple counter

```
.....
SimpleCounter>>execute
    <hdl:#combinational>
    {self clk. self reset. self start} onChange:
    [
        self reset = '1' ifTrue:[
            cnt reset:'0'.
        ] ifFalse:[
            self start = '1' ifTrue:[
                cnt reset:'0'.
                done <- false.
            ] ifFalse:[
                cnt = 100 ifTrue:[
                done <- true.
                ] ifFalse:[
                    cnt <- (cnt +1).]
                cnt = 4 ifTrue:[
                    self halt.
                ]
            ]
        ]
    ]
dout <- cnt.
```

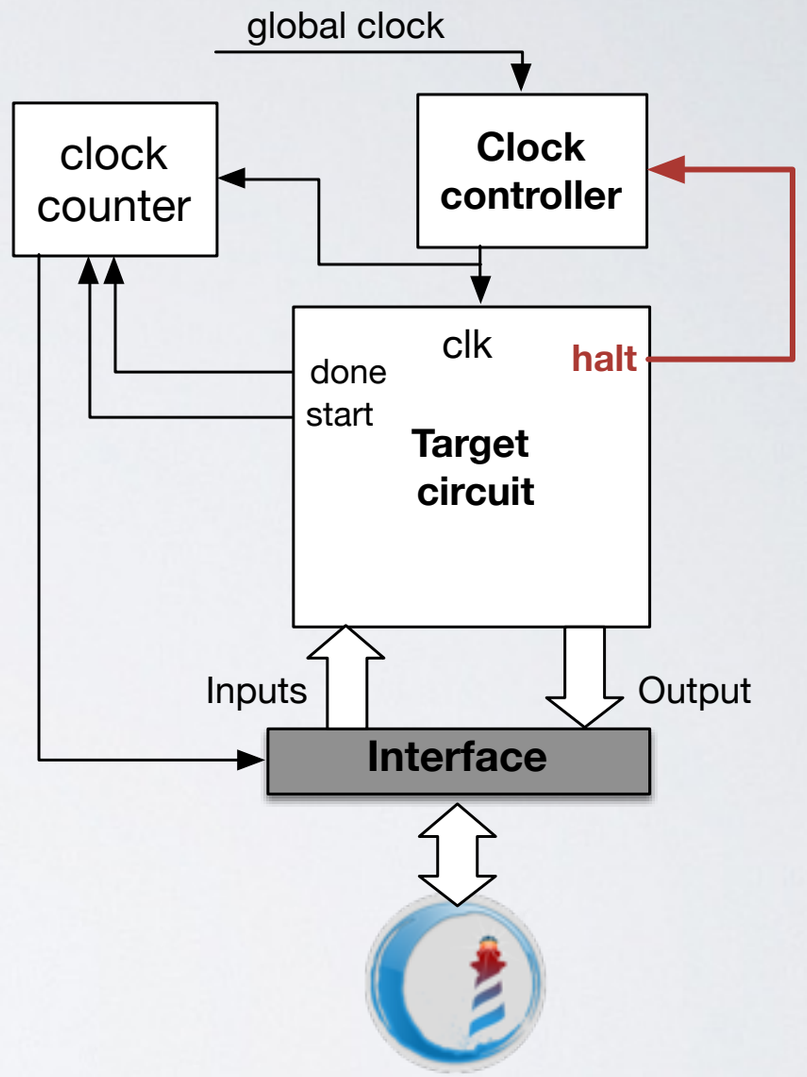
Halting

STATIC BREAKPOINT

A simple counter

```
.....
SimpleCounter>>execute
    <hdl:#combinational>
    {self clk. self reset. self start} onChange:
    [
        self reset = '1' ifTrue:[
            cnt reset:'0'.
        ] ifFalse:[
            self start = '1' ifTrue:[
                cnt reset:'0'.
                done <- false.
            ] ifFalse:[
                cnt = 100 ifTrue:[
                done <- true.
                ] ifFalse:[
                    cnt <- (cnt +1).]
                cnt = 4 ifTrue:[
                    self halt.
                ]
            ]
        ]
    ]
dout <- cnt.
```

Halting

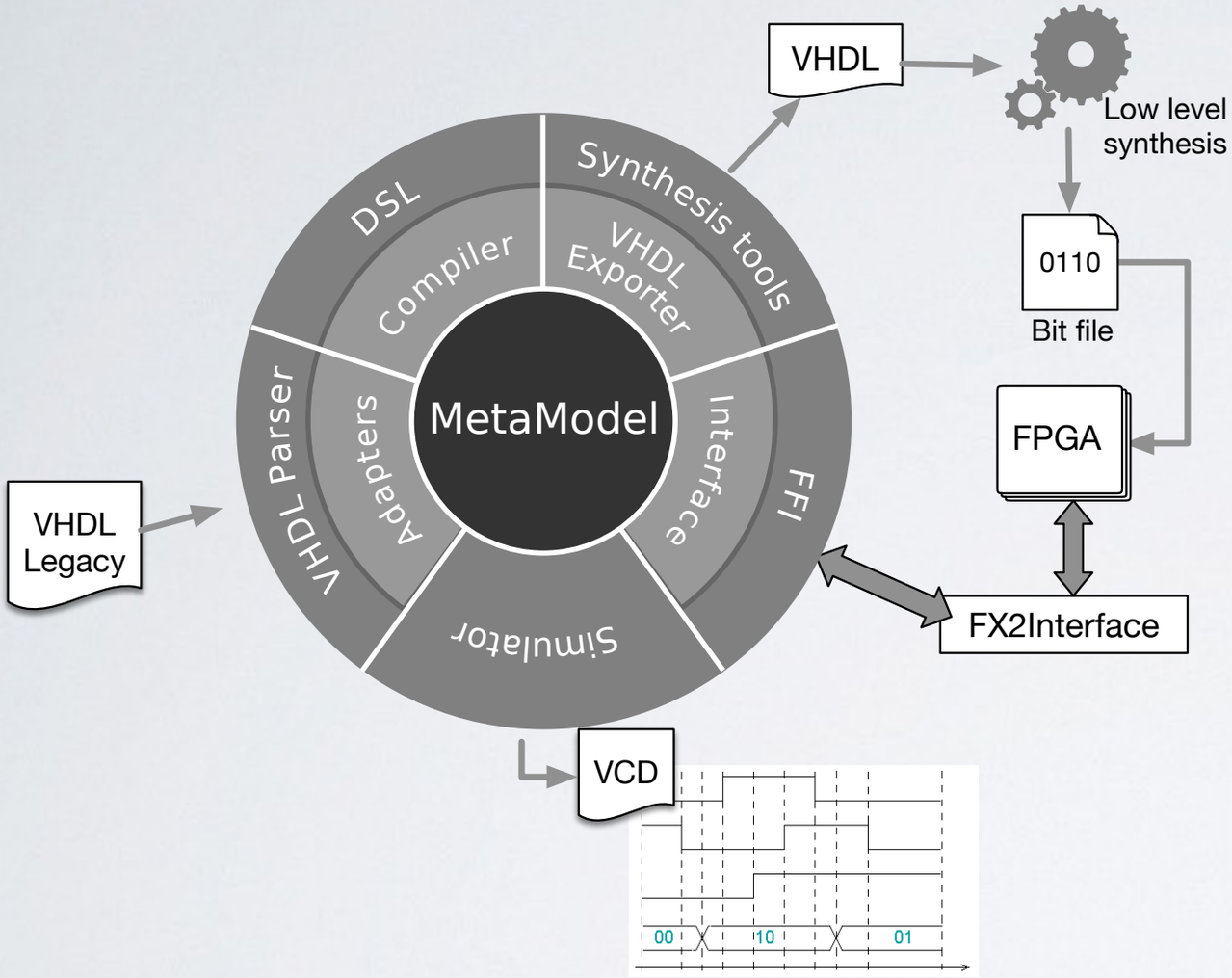


- Need to re-synthesize the circuit when changing

DYNAMIC BREAKPOINT

```
obj :=   SimpleCounter new.

obj setBreakpointOn:#dout value:4.
obj enableBreakpointOn:#dout.

obj execute. "stop at breakpoint”

“set another breakpoint”
obj setBreakpointOn:#dout value:10.
obj resume. "resume the execution”

“stop when out = 10”
obj resume. "Continue”
```

CONCLUSION AND FUTURE WORK



- **Future work**
  - *Short term*
    - Extend the meta-model
    - support software-hardware co-simulation
  - *Long term*
    - Get closer to Smalltalk : Smalltalk→FPGA

A META MODEL SUPPORTING BOTH HARDWARE AND SMALLTALK-BASED EXECUTION OF FPGA CIRCUITS

Le Xuan Sang[1,2]
Loïc Lagadec[1], Luc Fabress[2],
Jannik Laval[2] and Noury Bouraqadi[2]

[1] Lab-STICC, ENSTA Bretagne
[2] Institut Mines-Telecom, Mines Douai

HIGH LEVEL SYNTHESIS FRAMEWORK

1. Source code analysis.
2. Resource allocation.
3. Scheduling.
4. Resource binding
5. **Generate appropriate interface between application and the circuit**
6. **Modelling and simulation FPGA circuit at RTL level.**

Working progress

META-MODEL (1)

- Based on the IEEE 1076.66 RTL synthesis standard (VHDL-87)
- Support almost any VHDL structures in the standard
- Each meta-description in the meta-model capture both structure and behaviour of a circuit element
- Signals and Ports are modelled by objects with history

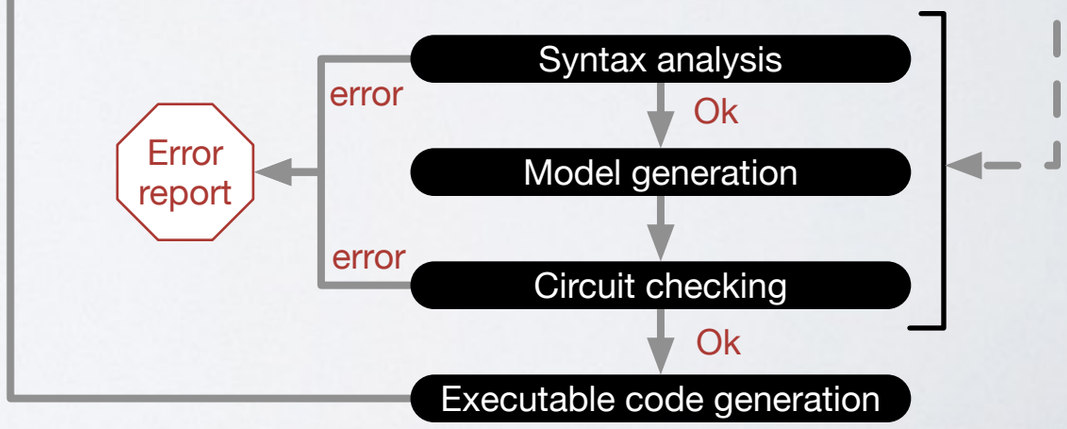
META-MODEL (2)

Dedicated DSL for circuit description

```
HDLSketch subclass: #SimpleALU
    instanceVariableNames: 'r1 r2 r3'

SimpleALU>>setUpPorts
    self in:{#A.#B}of:(LogicVector size:32).
    self in:#opcode of:Logic new.
    self out:#R of:(LogicVector size:32)

SimpleALU>>setUpSignals
    r1 := Signal of:(LogicVector size:32 signed:true).
    r2 := Signal of:(LogicVector size:32 signed:true).
    r3 := Signal of:(LogicVector size:32 signed:true).

SimpleALU>>execute
<hdl:#combinational>
    r1 <- (self A).
    r2 <- (self B).
    self R <- r3.
    {self clk} onChange:[
        self done <- false.
        self clk posedge ifTrue:[
            self opcode caseOf: {
                ['0'] -> [r3 <- (r1 + r2)].
                ['1'] -> [r3 <- (r1 and:r2)].
            }.
            self done <- true.
    ]].
```

DSL

```
HDLSketch class » compile:text
                  classified:aCategory
                  notifying:requestor
    |ast|
    ast := self compiler parse: text.
    (ast hasPragmaNamed: #hdl:) ifTrue:[
        ModelBuilder for:self
            ast:ast notificator:requestor.
        ^self compiledMethodFor:ast selector.
    ].
    ^super compile:text
           classified:aCategory
           notifying:requestor.
```



META-MODEL (3)

- **Reuse of Legacy VHDL**
  - A builtin VHDL Parser (based on PetitParser).
  - Third-party VHDL codes with respect to the IEEE 1076.66 RTL standard can be converted to our circuit-model.
- **Updated**: the converted circuit model now can be simulated in our environment

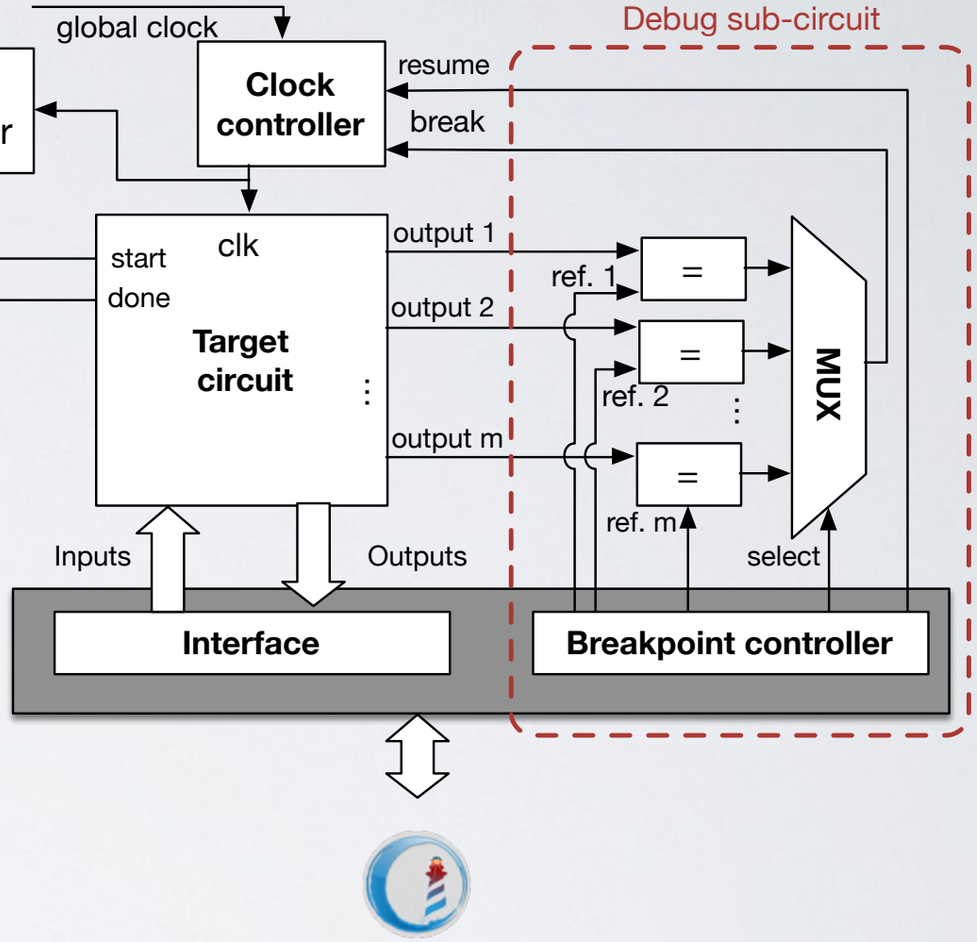
EXAMPLE

A counter from 0 to 100

```
HDLSketch subclass: #SimpleCounter
    instanceVariableNames: 'cnt'

SimpleCounter>>setupPorts
    self out:#dout of:(LogicVector size:32).

SimpleCounter>>setupSignals
    cnt := Signal of:(LogicVector size:32).

SimpleCounter>>execute
    <hdl:#combinational>
    {self clk. self reset. self start} onChange:
[
    self reset = '1' ifTrue:[
        cnt reset:'0'.
    ] ifFalse:[
        self start = '1' ifTrue:[
            cnt reset:'0'.
            done <- false.
        ] ifFalse:[
            cnt = 100 ifTrue:[
            done <- true.
            ] ifFalse:[
                cnt <- (cnt +1).]
        ]
    ]
]
dout <- cnt.
```

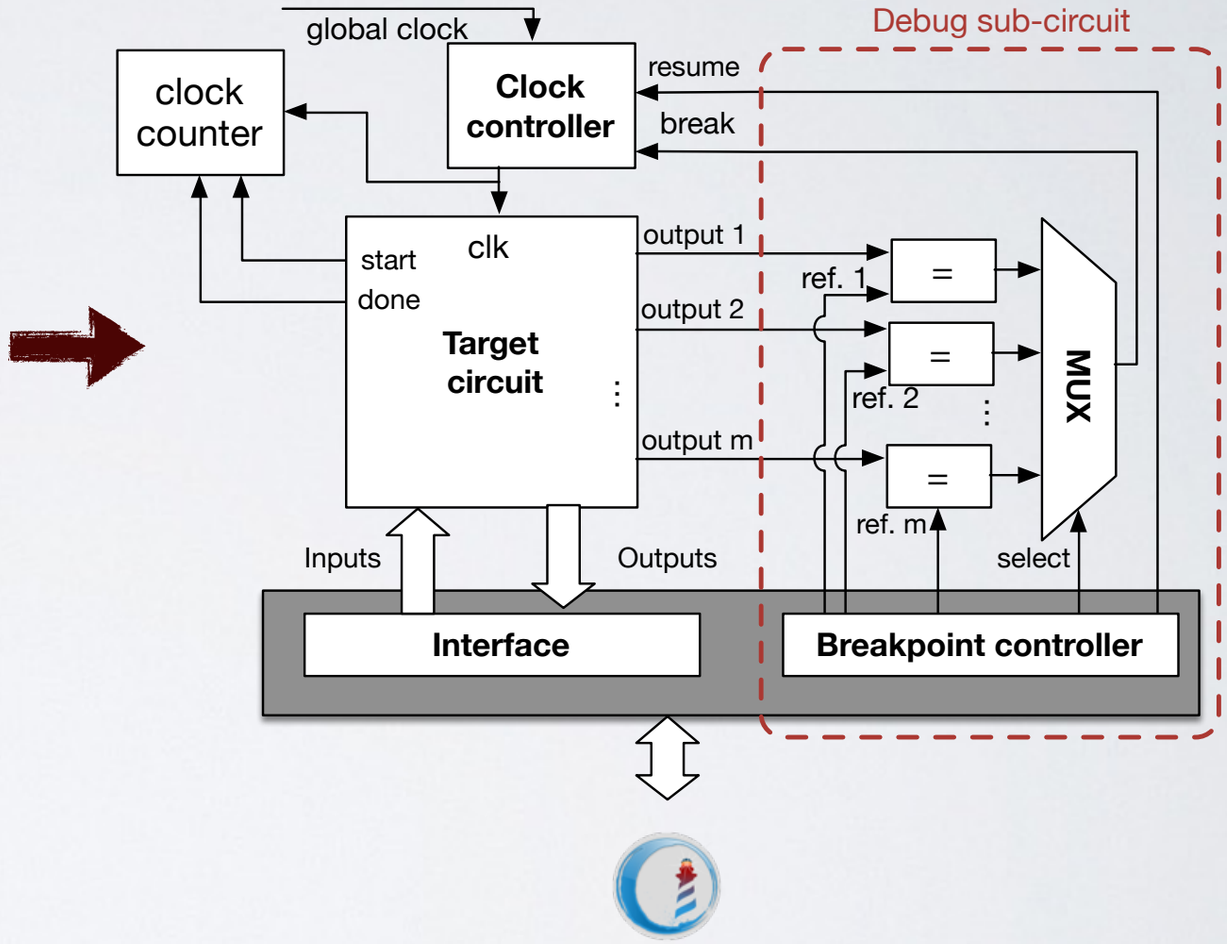
DYNAMIC BREAKPOINT

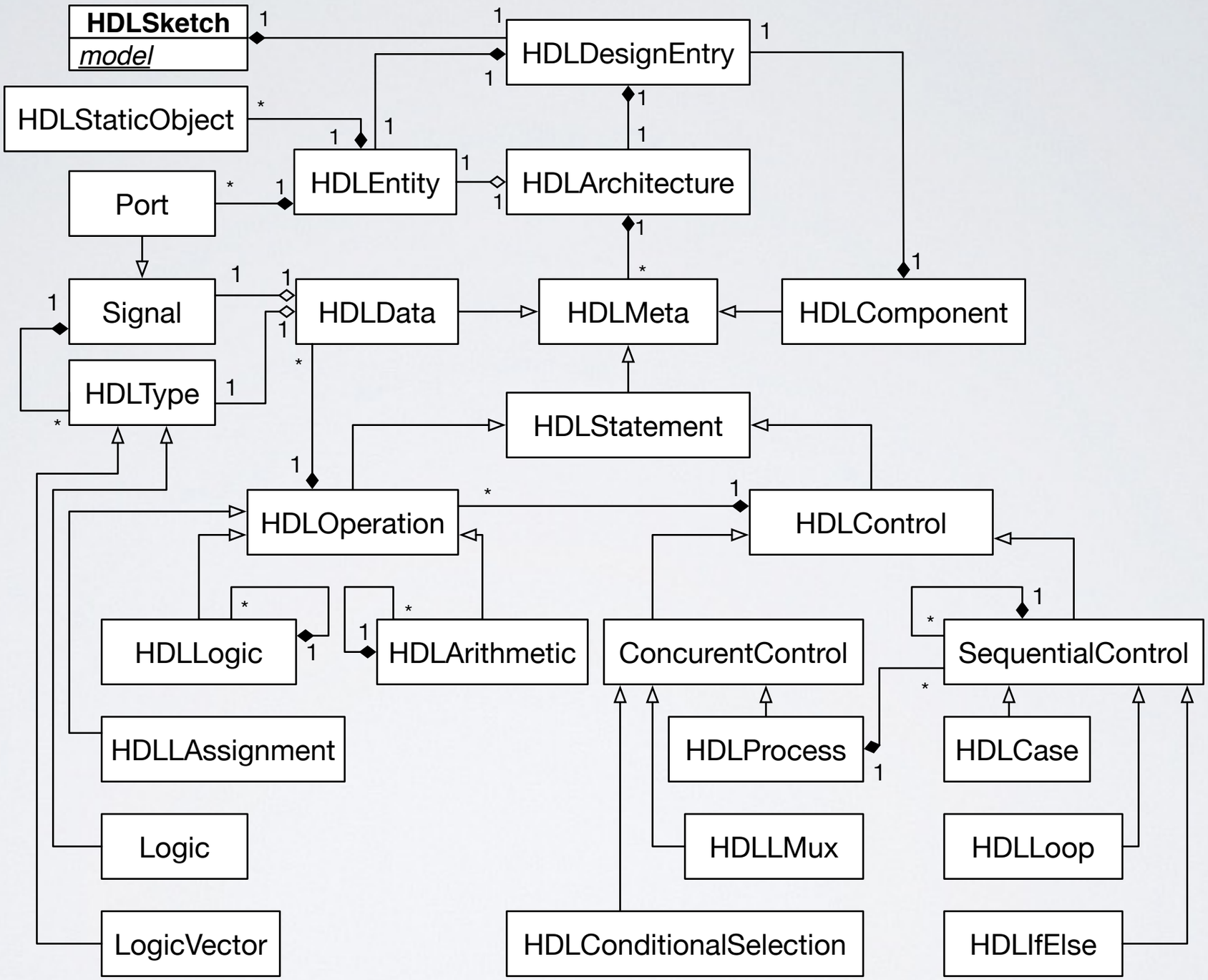
```
obj :=    SimpleCounter new.
obj setBreakpointOn:#dout value:4.
obj enableBreakpointOn:#dout.
obj execute. "stop at breakpoint"
obj resume. "resume the execution"
```



DYNAMIC BREAKPOINT

```
obj :=    SimpleCounter new.

obj setBreakpointOn:#dout value:4.
obj enableBreakpointOn:#dout.

obj execute. "stop at breakpoint"
obj resume. "resume the execution"
```




HDLSketch
model
HDLDesignEntry
HDLStaticObject
HDLEntity
HDLArchitecture
Port
Signal
HDLData
HDLMeta
HDLComponent
HDLType
HDLStatement
HDLOperation
HDLControl
HDLLogic
HDLArithmetic
ConcurentControl
SequentialControl
HDLLAssignment
HDLProcess
HDLCase
Logic
HDLLMux
HDLLoop
LogicVector
HDLConditionalSelection
HDLIfElse