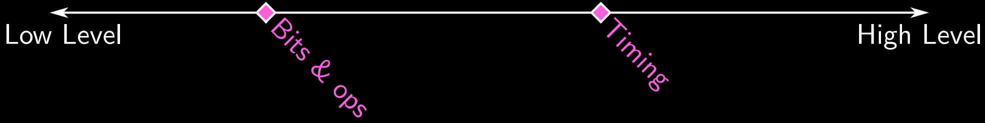
Spade: An Expression-Based HDL With Pipelines

Frans Skarman, Oscar Gustafsson

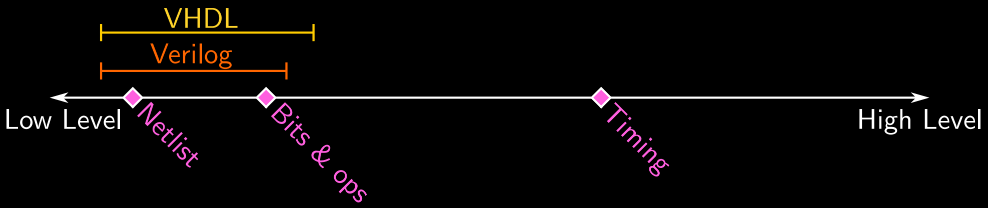
LINKÖPING UNIVERSITY

Abstraction



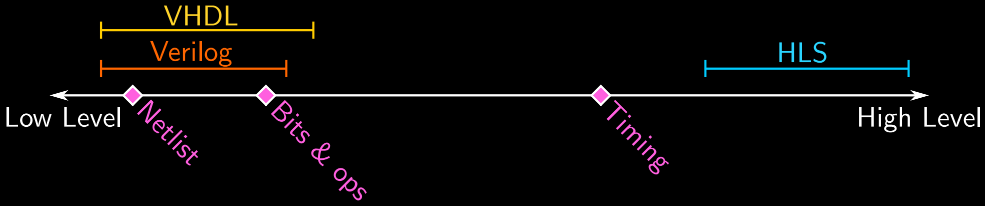
- What do we **have** to think about? What **can** we control?

Abstraction



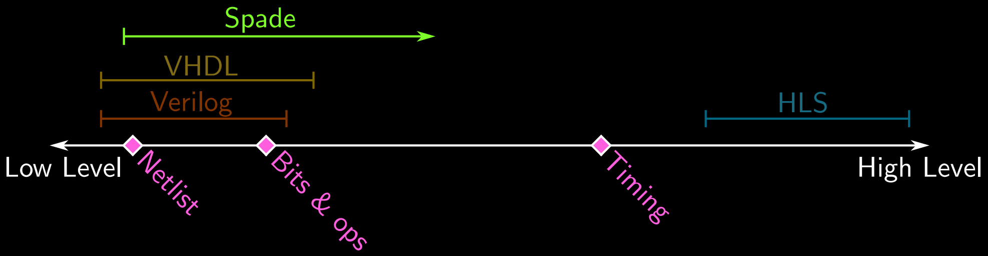
- What do we **have** to think about? What **can** we control?
- Intervals, not points

Abstraction



- What do we **have** to think about? What **can** we control?
- Intervals, not points

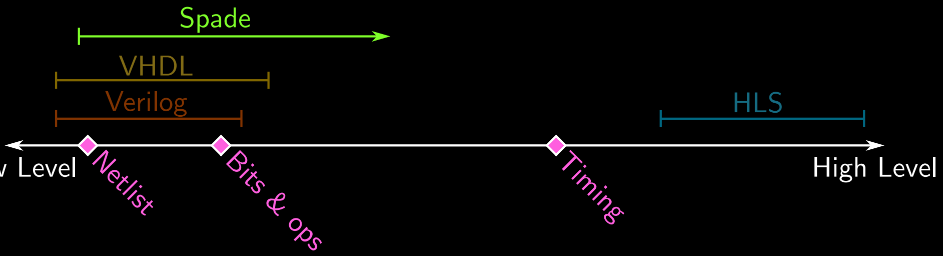
Abstraction



- What do we **have** to think about? What **can** we control?
- Intervals, not points
- Retain control, but allow higher level reasoning

LINKÖPING UNIVERSITY

Abstraction



- What do we **have** to think about? What **can** we control?
- Intervals, not points
- Retain control, but allow higher level reasoning
- ~~Steal~~ Incorporate features from software languages

Hello, World

```
entity counter(
         clk: clock, rst: bool, max: int<20>
    ) -> int<20>
{
    reg(clk) val reset (rst: 0) =
        if val == max {
            0
        } else {
            trunc(val+1)
        };
    val
}
```

LINKÖPING UNIVERSITY

Hello, World

- Inputs separated from outputs

```
entity counter(
        clk: clock, rst: bool, max: int<20>
    ) -> int<20>
{
    reg(clk) val reset (rst: 0) =
        if val == max {
            0
        } else {
            trunc(val+1)
        };
    val
}
```

LINKÖPING UNIVERSITY

Hello, World

- Inputs separated from outputs

```
entity counter(
        clk: clock, rst: bool, max: int<20>
    ) -> int<20>
{
    reg(clk) val reset (rst: 0) =
        if val == max {
            0
        } else {
            trunc(val+1)
        };
    val
}
```

LINKÖPING UNIVERSITY

Hello, World

- Inputs separated from outputs
- Register 'val' clocked by 'clk'

```
entity counter(
        clk: clock, rst: bool, max: int<20>
    ) -> int<20>
{
    reg(clk) val reset (rst: 0) =
        if val == max {
            0
        } else {
            trunc(val+1)
        };
    val
}
```

LINKÖPING UNIVERSITY

Hello, World

- Inputs separated from outputs
- Register 'val' clocked by 'clk'
- Reset to 0 by the rst signal

```
entity counter(
        clk: clock, rst: bool, max: int<20>
    ) -> int<20>
{
    reg(clk) val reset (rst: 0) =
        if val == max {
            0
        } else {
            trunc(val+1)
        };
    val
}
```

Hello, World

- Inputs separated from outputs
- Register 'val' clocked by 'clk'
- Reset to 0 by the rst signal
- New value as a 'function' of the old value

```
entity counter(
        clk: clock, rst: bool, max: int<20>
    ) -> int<20>
{
    reg(clk) val reset (rst: 0) =
        if val == max {
            0
        } else {
            trunc(val+1)
        };
    val
}
```

Hello, World

Key takeaways:

- Expression based semantics, not imperative

```
entity counter(
        clk: clock, rst: bool, max: int<20>
    ) -> int<20>
{
    reg(clk) val reset (rst: 0) =
        if val == max {
            0
        } else {
            trunc(val+1)
        };
    val
}
```

Hello, World

Key takeaways:

- Expression based semantics, not imperative
- Statically typed

```
entity counter(
        clk: clock, rst: bool, max: int<20>
    ) -> int<20>
{
    reg(clk) val reset (rst: 0) =
        if val == max {
            0
        } else {
            trunc(val+1)
        };
    val
}
```

LINKÖPING UNIVERSITY

Hello, World

Key takeaways:

- Expression based semantics, not imperative
- Statically typed
- With type inference

```
entity counter(
        clk: clock, rst: bool, max: int<20>
    ) -> int<20>
{
    reg(clk) val reset (rst: 0) =
        if val == max {
            0
        } else {
            trunc(val+1)
        };
    val
}
```

Hello, World

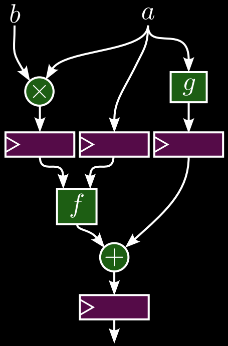
Key takeaways:

- Expression based semantics, not imperative
- Statically typed
- With type inference
- Cycle-to-cycle description

```
entity counter(
        clk: clock, rst: bool, max: int<20>
    ) -> int<20>
{
    reg(clk) val reset (rst: 0) =
        if val == max {
            0
        } else {
            trunc(val+1)
        };
    val
}
```

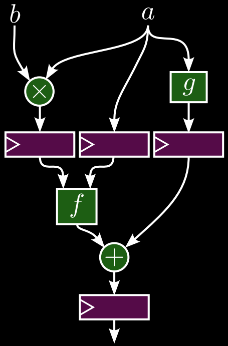
Pipelines

```
pipeline(2) X(clk: clock, a: int<32>, b: int<32>)
    -> int<33> {
    let x = g(a);
    let product = a*b;
reg;

    let sum = x + f(a, product)
reg;
    sum
}
```



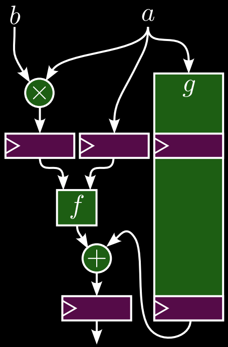
Pipelines

```
pipeline(2) X(clk: clock, a: int<32>, b: int<32>)
    -> int<33> {
    let x = g(a);
    let product = a*b;
reg;

    let sum = x + f(a, product)
reg;
    sum
}
```



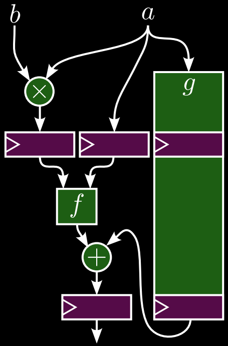
Pipelines

```
pipeline(2) X(clk: clock, a: int<32>, b: int<32>)
    -> int<33> {
    let x = g(a);
    let product = a*b;
reg;

    let sum = x + f(a, product)
reg;
    sum
}
```



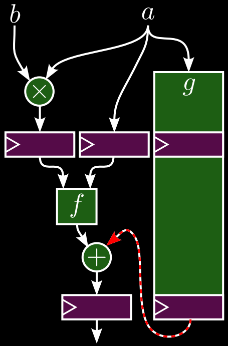
Pipelines

```
pipeline(2) X(clk: clock, a: int<32>, b: int<32>)
    -> int<33> {
    let x = inst(2) g(a);
    let product = a*b;
reg;

    let sum = x + f(a, product)
reg;
    sum
}
```



Pipelines

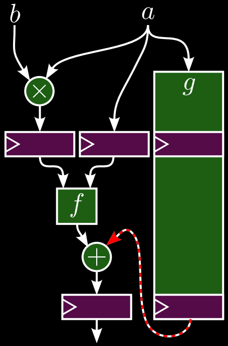
```
pipeline(2) X(clk: clock, a: int<32>, b: int<32>)
    -> int<33> {
    let x = inst(2) g(a);
    let product = a*b;
reg;

    let sum = x + f(a, product)
reg;
    sum
}
```



Pipelines

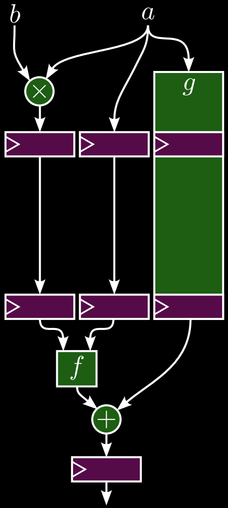
```
pipeline(2) X(clk: clock, a: int<32>, b: int<32>)
    -> int<33> {
    let x = inst(2) g(a);
    let product = a*b;
reg;

    let sum = x + f(a, product)
reg;
    sum
}
```

```
error: Use of x before it is ready
--> src/main.spade:10:19
|
10 | let sum = x + f(a, product);
|                 ^ Is unavailable for another stage
= Requesting x from stage 1
= But it will not be available until stage 2
```



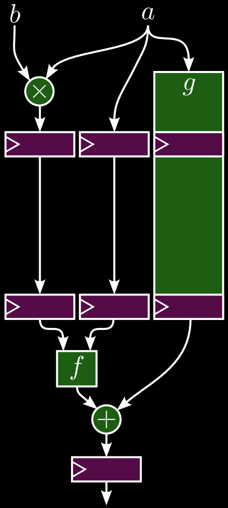
Pipelines

```
pipeline(2) X(clk: clock, a: int<32>, b: int<32>)
    -> int<33> {
    let x = inst(2) g(a);
    let product = a*b;
reg;

    let sum = x + f(a, product)
reg;
    sum
}
```



Pipelines

```
pipeline(3) X(clk: clock, a: int<32>, b: int<32>)
    -> int<33> {
    let x = inst(2) g(a);
    let product = a*b;
reg;
reg;
    let sum = x + f(a, product)
reg;
    sum
}
```



Pipelines

Other features

- Feedback and bypasses

Pipelines

Other features

- Feedback and bypasses
- Built-in dynamic behavior
  - Stalls
  - Flushes
  - Back pressure

Types

- Enum called `Option`

```
enum Option<T> {
    None,
    Some{val: T}
}
```

Types

- Enum called `Option`
- Generic over `T`

```
enum Option<T> {
    None,
    Some{val: T}
}
```

Types

- Enum called `Option`
- Generic over `T`
- Is `Some` in which case `val` is present

```
enum Option<T> {
    None,
    Some{val: T}
}
```

Types

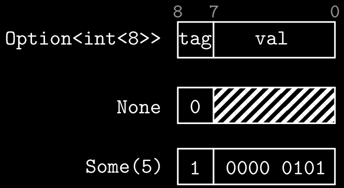
- Enum called `Option`
- Generic over `T`
- Is `Some` in which case `val` is present
- Or `None`

```
enum Option<T> {
    None,
    Some{val: T}
}
```

Types

- Enum called `Option`
- Generic over `T`
- Is `Some` in which case `val` is present
- Or `None`

```
enum Option<T> {
     None,
     Some{val: T}
}
```



More enum examples

- Commands on a bus

```
enum Command {
    Nop,
    Read,
    Write{data: int<32>}
}

struct BusControlSignals {
    access_width: AccessWidth,
    addr: int<32>,
    cmd: Command,
}
```

More enum examples

- Commands on a bus
- Internal instructions

```
enum Insn {
    Set {
        dreg: int<5>,
        val: int<32>
    },
    Add {
        dreg: int<5>,
        lhs: int<5>,
        rhs: int<5>
    },
    Sub {
        dreg: int<5>,
        lhs: int<5>,
        rhs: int<5>
    },
    Jump {
        target: int<32>
    }
}
```

Tooling

A language is nothing without its tools

LINKÖPING UNIVERSITY

Cocotb test benches

```python
# top=peripherals::timer::timer_test_harness

@cocotb.test()
async def timer_works(dut):
    s = SpadeExt(dut)
    clk = dut.clk_i
    await start_clock(clk)

    s.i.mem_range = "(1024, 2048)"
    s.i.addr = "1024 + 0"
    s.i.memory_command = "Command::Write(10)"
    await FallingEdge(clk)

    s.i.addr = "1024 + 4"
    s.i.memory_command = "Command::Read()"
    for i in range(0, 11):
        await FallingEdge(clk)
        s.o.assert_eq(f"{i}")
```

Cocotb test benches

```python
# top=peripherals::timer::timer_test_harness

@cocotb.test()
async def timer_works(dut):
    s = SpadeExt(dut)
    clk = dut.clk_i
    await start_clock(clk)

    s.i.mem_range = "(1024, 2048)"
    s.i.addr = "1024 + 0"
    s.i.memory_command = "Command::Write(10)"
    await FallingEdge(clk)

    s.i.addr = "1024 + 4"
    s.i.memory_command = "Command::Read()"
    for i in range(0, 11):
        await FallingEdge(clk)
        s.o.assert_eq(f"{i}")
```

LINKÖPING
UNIVERSITY

Swim build tool

- Manages dependencies

```
[libraries]
spadev = {path = ".."}
ws2812 = {
    git = "gitlab.com/TheZoq2/ws2812",
    branch = "main"
}


[synthesis]
top = "top"
extra_verilog = [ "src/top_s1.v" ]
command = "synth_ice40"
[pnr]
# ...

[plugins]
loader.git = "..."
loader.args.asm_file = "asm/blinky.asm"
loader.args.template_file = "..."
loader.args.target_file = "..."
```

Swim build tool

- Manages dependencies
- Call build tools

```
[libraries]
spadev = {path = ".."}
ws2812 = {
    git = "gitlab.com/TheZoq2/ws2812",
    branch = "main"
}


[synthesis]
top = "top"
extra_verilog = [ "src/top_s1.v" ]
command = "synth_ice40"
[pnr]
# ...

[plugins]
loader.git = "..."
loader.args.asm_file = "asm/blinky.asm"
loader.args.template_file = "..."
loader.args.target_file = "..."
```

Swim build tool

- Manages dependencies
- Call build tools
- Scriptable via plugins

```
[libraries]
spadev = {path = ".."}
ws2812 = {
    git = "gitlab.com/TheZoq2/ws2812",
    branch = "main"
}


[synthesis]
top = "top"
extra_verilog = [ "src/top_s1.v" ]
command = "synth_ice40"
[pnr]
# ...

[plugins]
loader.git = "..."
loader.args.asm_file = "asm/blinky.asm"
loader.args.template_file = "..."
loader.args.target_file = "..."
```

LINKÖPING UNIVERSITY

Implementation

- Open source, implemented in Rust.

Implementation

- Open source, implemented in Rust.
- Standalone compiler targeting Verilog

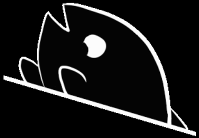
LINKÖPING UNIVERSITY

Implementation

- Open source, implemented in Rust.
- Standalone compiler targeting Verilog
- But it is backend agnostic
    - CIRCT?
    - Calyx?
    - RTLIL?

Thanks for listening!

https://spade-lang.org
frans.skarman@liu.se
mastodon.social/@thezoq2



LINKÖPING UNIVERSITY

What about Chisel? (Or Spinal, Amaranth etc.)

- Much more mature
- Also pushes the abstraction level, but differently
- Spade cannot compete in meta-programming
- But basic hardware description is individual operations on pure bundles of bits
  - No “runtime types”
  - No pattern matching
  - No pipelining
  - Imperative
- Embedding DSLs feel clunky
  - `when.elsewhen.otherwise`
  - Accidental software runtime/hardware runtime mixing
  - Compiler errors

What about Cλash?

- Also much more mature
- Similarly powerful type system
    - Almost too powerful
- No pipelining, no ports
- Haskell is hard

Frans Skarman, Oscar Gustafsson

www.liu.se

LINKÖPING UNIVERSITY