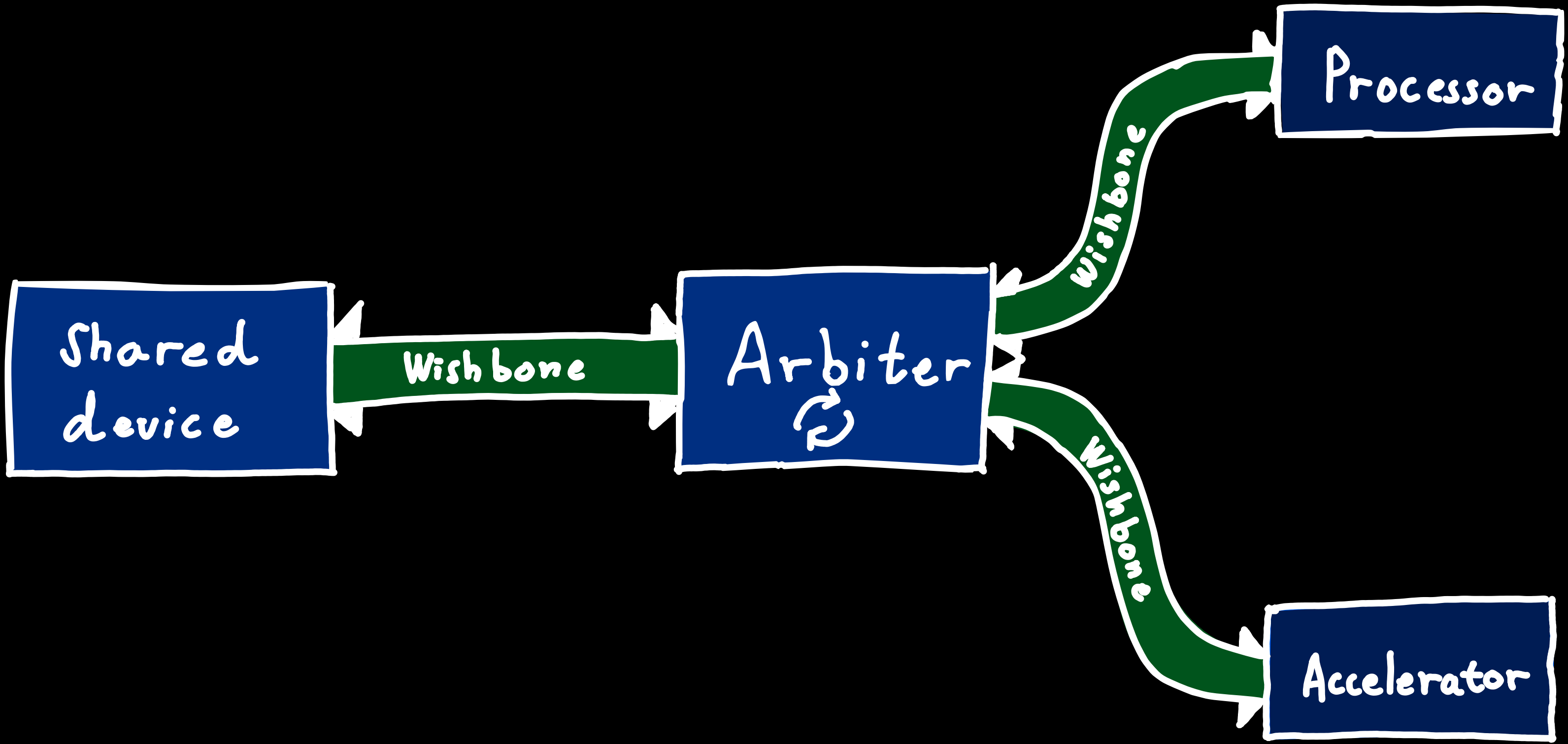
Enhancing Compiler-Driven HDL Design with Automatic Waveform Analysis

Frans Skarman[1] Lucas Klemmer[2] Oscar Gustafsson[1] Daniel Große[2]

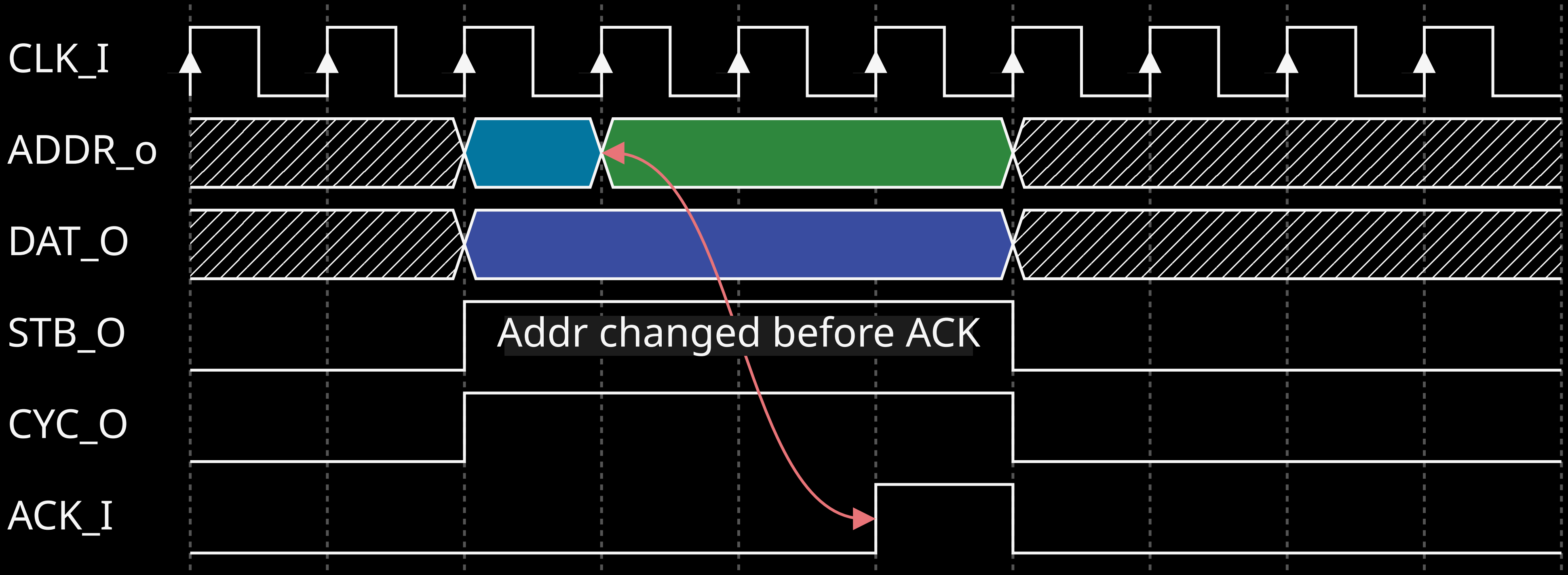
Linköping University[1] and Johannes Kepler University[2]

frans.skarman@liu.se[1], lucas.klemmer@jku.at[2]

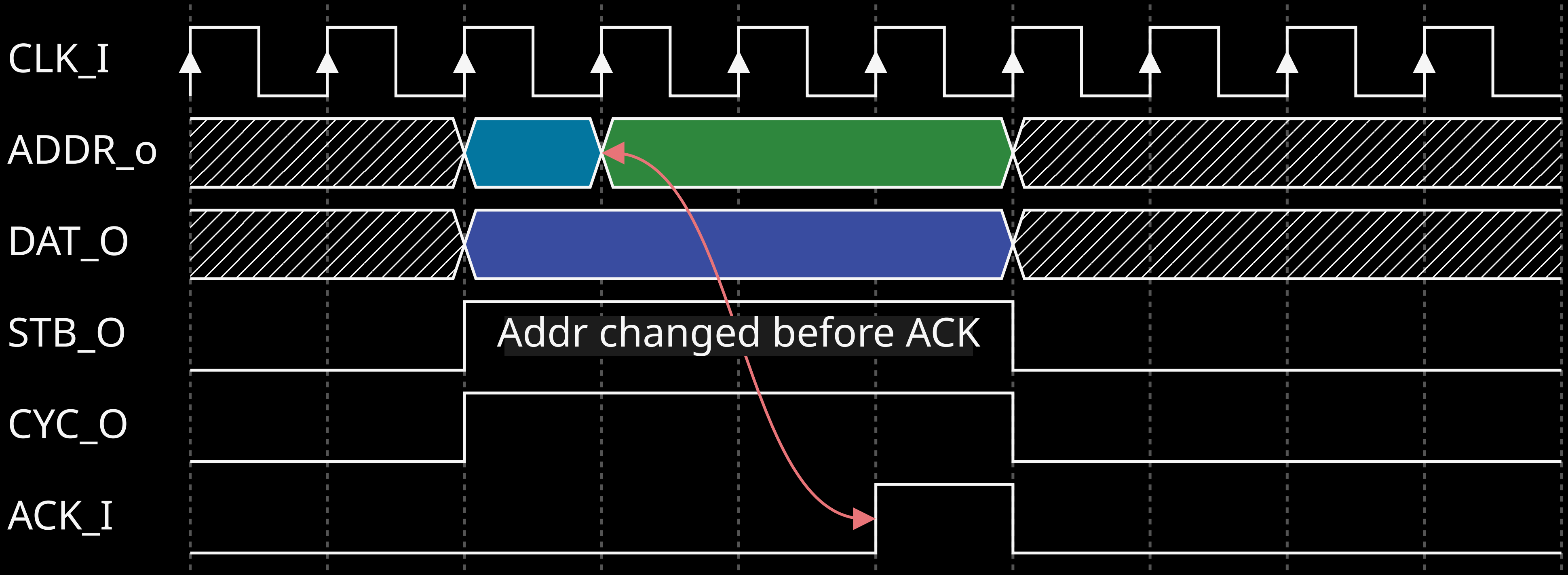
A motivating example



Traditional Waveform Analysis



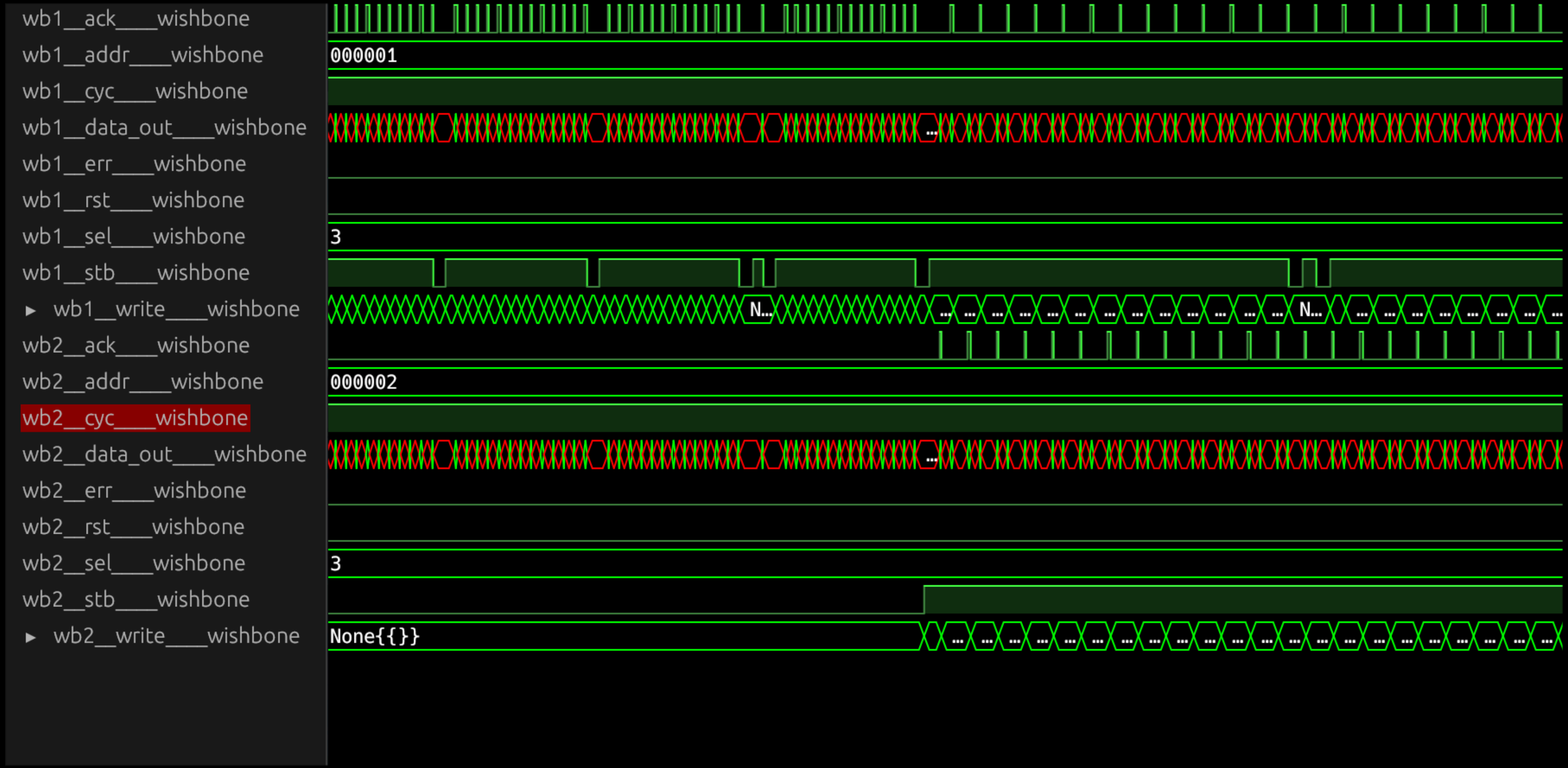
Traditional Waveform Analysis



Is powerful for detailed debugging

Tranditional Waveform Analysis

Not as powerful for large scale analysis



Automation To The Rescue

```
entity wb_harness(clk: clock, rst: bool) -> (int<16>, int<16>)
{
  #[wal_trace(target=wb1, clk=clk, rst=rst))]
  let (wb1, wb1inv) = port;
  #[wal_trace(target=wb2, clk=clk, rst=rst))]
  let (wb2, wb2inv) = port;

  let _ = inst wishbone_arbiter( clk, rst, wb1inv, wb2inv, ...);
  let o1 = inst wishbone_master( clk, rst, wb1, ...);
  let o2 = inst wishbone_master( clk, rst, wb2, ...);
  (o1, o2)
}
```

Automation To The Rescue

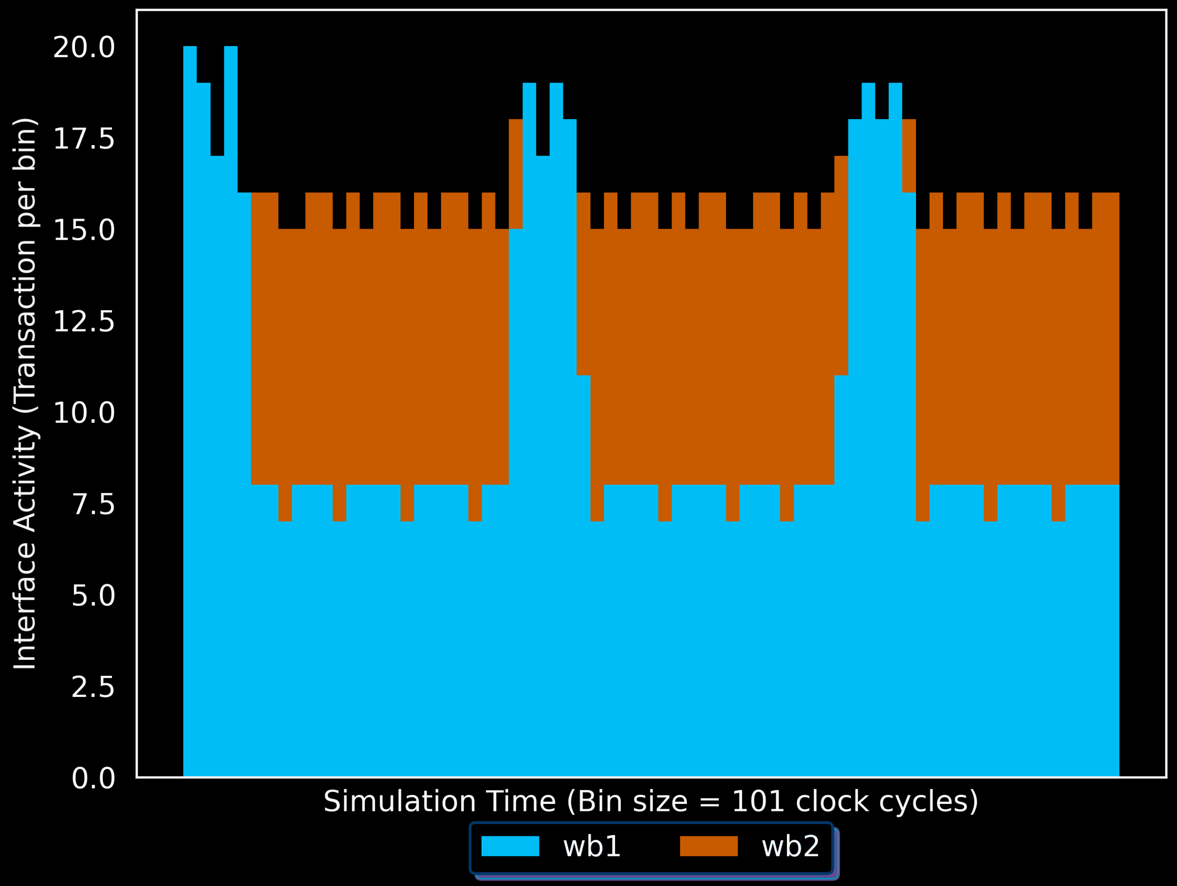
```
entity wb_harness(clk: clock, rst: bool) -> (int<16>, int<16>)
{
  #[wal_trace(target=wb1, clk=clk, rst=rst))]
  let (wb1, wb1inv) = port;
  #[wal_trace(target=wb2, clk=clk, rst=rst))]
  let (wb2, wb2inv) = port;

  let _ = inst wishbone_arbiter( clk, rst, wb1inv, wb2inv, ...);
  let o1 = inst wishbone_master( clk, rst, wb1, ...);
  let o2 = inst wishbone_master( clk, rst, wb2, ...);
  (o1, o2)
}
```

```
$ swim plugin analysis
```

High Level Statistics

```
[ANALYSIS] # wb_harness.wb1
[ANALYSIS]    Nr. transactions : 697
[ANALYSIS]    Nr. reads        : 16
[ANALYSIS]    Avg read latency : 6 clock cycles
[ANALYSIS]    Nr. writes       : 681
[ANALYSIS]    Avg write latency: 9 clock cycles
[ANALYSIS]    Inactive cycles  : 6%
[ANALYSIS] # wb_harness.wb2
[ANALYSIS]    Nr. transactions : 421
[ANALYSIS]    Nr. reads        : 0
[ANALYSIS]    Avg read latency : - clock cycles
[ANALYSIS]    Nr. writes       : 421
[ANALYSIS]    Avg write latency: 12 clock cycles
[ANALYSIS]    Inactive cycles  : 22%
```

And Histograms!



A step back

- What exactly does the annotation do?
- How is the analysis done?
- How did it know the signals are wishbone?
- What language is this?

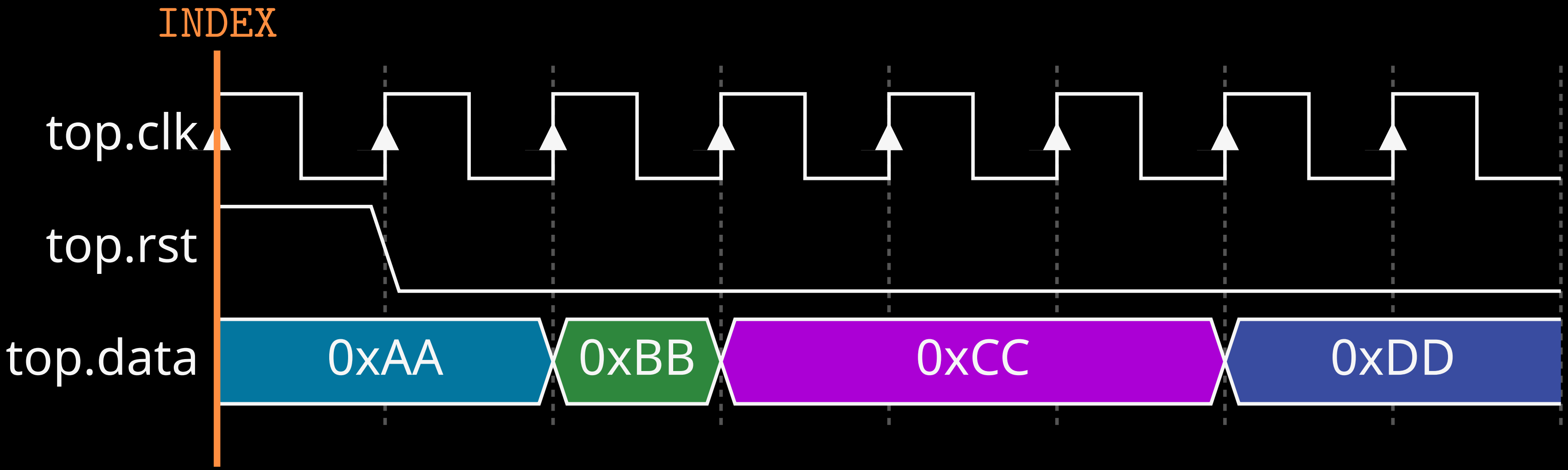
Waveform Analysis Language - WAL

WAL

- Language for complex waveform analysis
- LISP
- With native hardware constructs
  - Timing
  - Signals
  - Hierarchy
- Easy python interoperability

WAL Primer

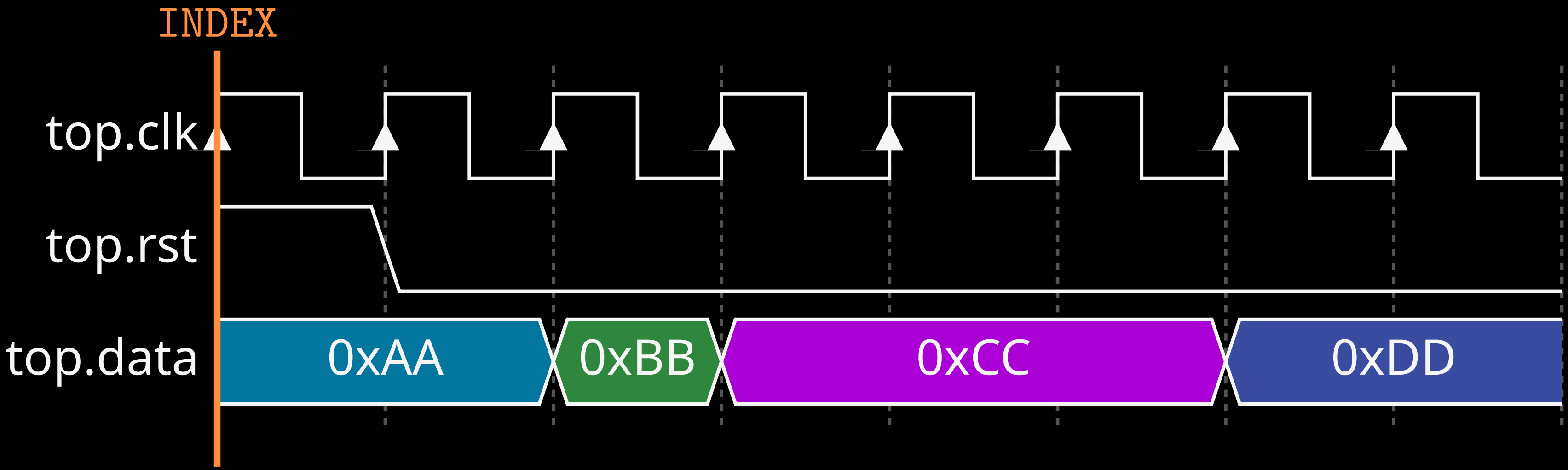
```
(print "At " INDEX ": " top.data)
(step 10)
(print "At " INDEX ": " top.data)
```



WAL Primer

```
(print "At " INDEX ": " top.data)
(step 10)
(print "At " INDEX ": " top.data)
```

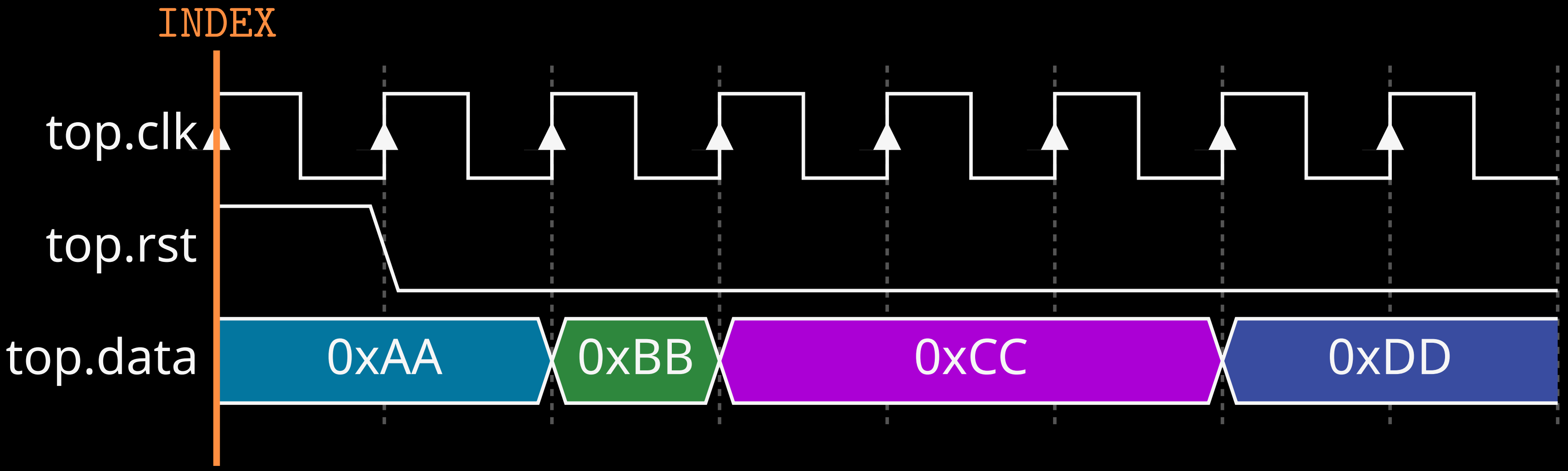
```
At 0: 0xAA
```



WAL Primer

```
(print "At " INDEX ": " top.data)
(step 10)
(print "At " INDEX ": " top.data)
```

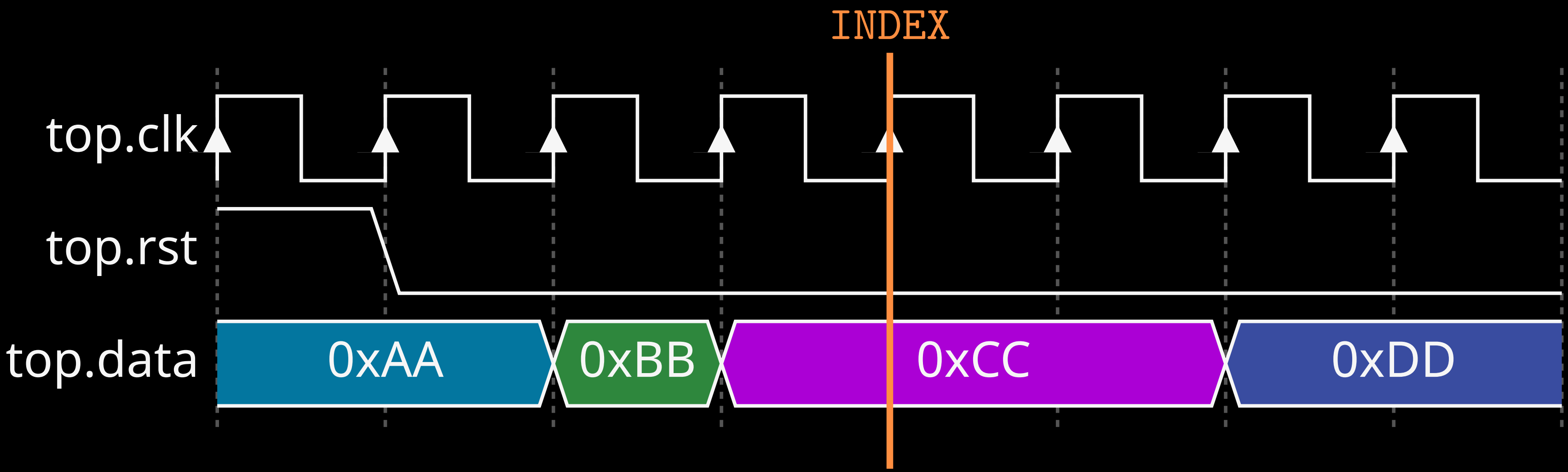
```
At 0: 0xAA
```



WAL Primer

```
(print "At " INDEX ": " top.data)
(step 10)
(print "At " INDEX ": " top.data)
```

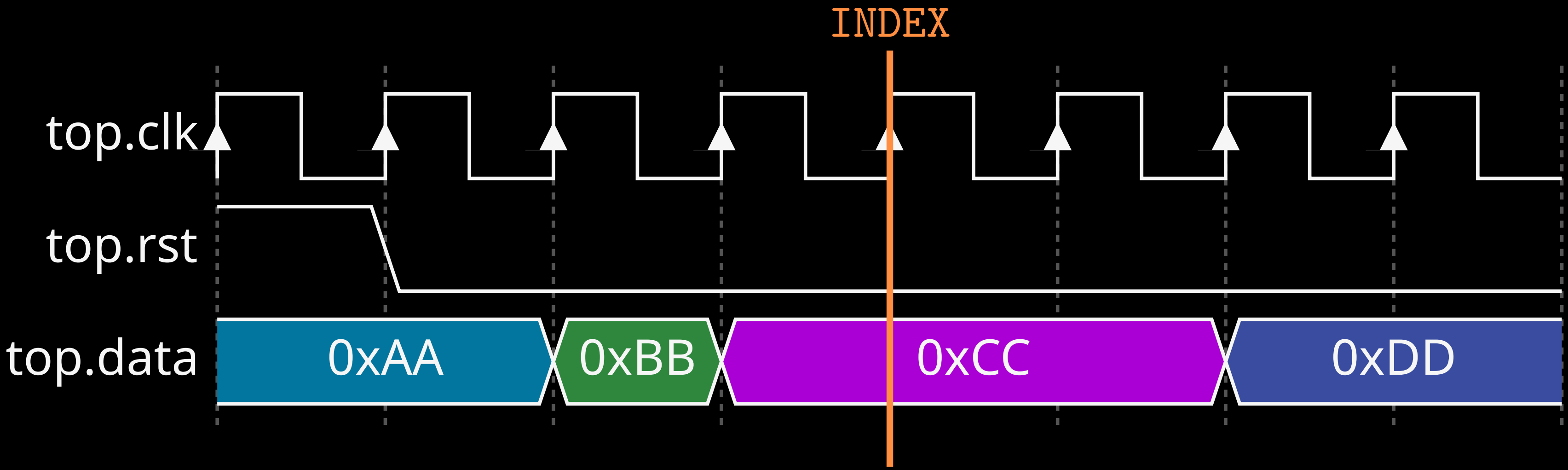
```
At 0: 0xAA
```



WAL Primer

```
(print "At " INDEX ": " top.data)
(step 10)
(print "At " INDEX ": " top.data)
```

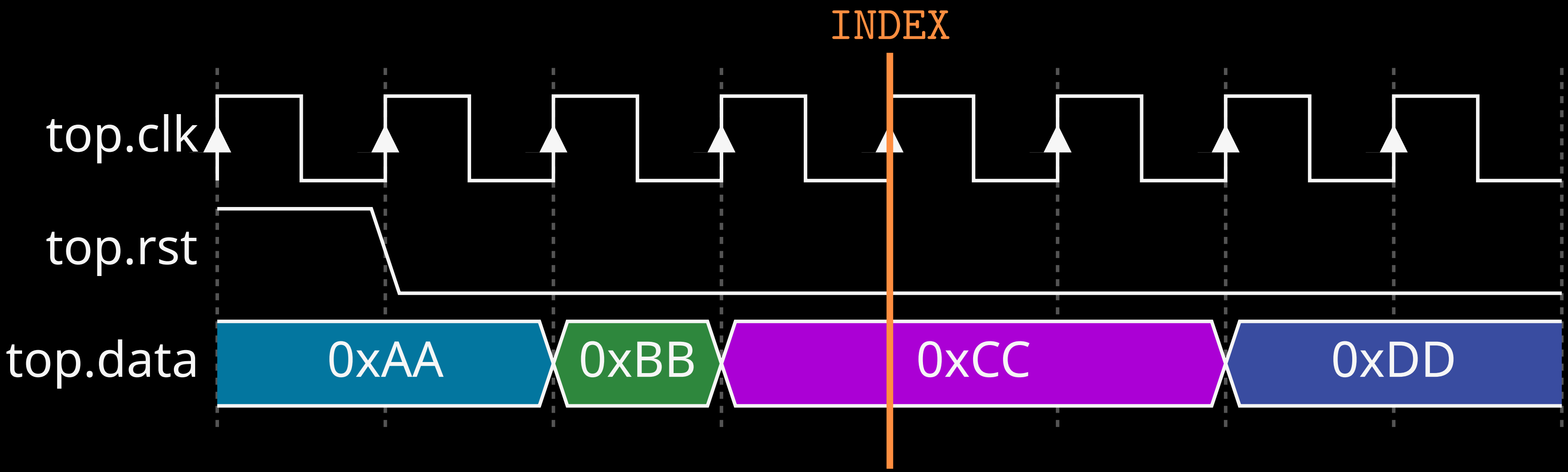
```
At 0: 0xAA
```



WAL Primer

```
(print "At " INDEX ": " top.data)
(step 10)
(print "At " INDEX ": " top.data)
```

```
At 0: 0xAA
At 10: 0xCC
```



Higher level WAL

```
(define transactions 0)
(whenever (&& (rising clk) ACK STB CYC)
  (inc transactions))
(print "Number of transactions: " transactions)
```

Higher level WAL

```
(define transactions 0)
(whenever (&& (rising clk) ACK STB CYC)
  (inc transactions))
(print "Number of transactions: " transactions)
```

Higher level WAL

```
(define transactions 0)
(whenever (&& (rising clk) ACK STB CYC)
  (inc transactions))
(print "Number of transactions: " transactions)
```

Higher level WAL

```
(define transactions 0)
(whenever (&& (rising clk) ACK STB CYC)
  (inc transactions))
(print "Number of transactions: " transactions)
```

Higher level WAL

```
(define transactions 0)
(whenever (&& (rising clk) ACK STB CYC)
  (inc transactions))
(print "Number of transactions: " transactions)
```

Hierarchy Independent WAL

```
(in-groups (groups ("ACK", "STB", "CYC"))
  (define transactions = 0)
  (whenever (&& (rising top.clk) #ACK #STB #CYC)
    (inc transactions))
  (print
    CG " performed " transactions " transactions")
```

Hierarchy Independent WAL

```
(in-groups (groups ("ACK", "STB", "CYC"))
  (define transactions = 0)
  (whenever (&& (rising top.clk) #ACK #STB #CYC)
    (inc transactions))
  (print
    CG " performed " transactions " transactions")
```

Hierarchy Independent WAL

```
(in-groups (groups ("ACK", "STB", "CYC"))
  (define transactions = 0)
  (whenever (&& (rising top.clk) #ACK #STB #CYC)
    (inc transactions))
  (print
    CG " performed " transactions " transactions")
```

```
top.clk
top.accelerator.state
top.accelerator.wb_STB
top.accelerator.wb_ACK
top.accelerator.wb_CYC
top.cpu.pc
top.cpu.wb_STB
top.cpu.wb_ACK
top.cpu.wb_CYC
```

Hierarchy Independent WAL

```
(in-groups (groups ("ACK", "STB", "CYC"))
  (define transactions = 0)
  (whenever (&& (rising top.clk) #ACK #STB #CYC)
    (inc transactions))
  (print
    CG " performed " transactions " transactions")
```

```
top.clk
top.accelerator.state
top.accelerator.wb_STB
top.accelerator.wb_ACK
top.accelerator.wb_CYC
top.cpu.pc
top.cpu.wb_STB
top.cpu.wb_ACK
top.cpu.wb_CYC
```

Hierarchy Independent WAL

```
(in-groups (groups ("ACK", "STB", "CYC"))
  (define transactions = 0)
  (whenever (&& (rising top.clk) #ACK #STB #CYC)
    (inc transactions))
  (print
    CG " performed " transactions " transactions")
```

```
top.clk
top.accelerator.state
top.accelerator.wb_STB
top.accelerator.wb_ACK
top.accelerator.wb_CYC
top.cpu.pc
top.cpu.wb_STB
top.cpu.wb_ACK
top.cpu.wb_CYC
```

- Group1: `top.accelerator.wb_`

Hierarchy Independent WAL

```
(in-groups (groups ("ACK", "STB", "CYC"))
  (define transactions = 0)
  (whenever (&& (rising top.clk) #ACK #STB #CYC)
    (inc transactions))
  (print
    CG " performed " transactions " transactions")
```

```
top.clk
top.accelerator.state
top.accelerator.wb_STB
top.accelerator.wb_ACK
top.accelerator.wb_CYC
top.cpu.pc
top.cpu.wb_STB
top.cpu.wb_ACK
top.cpu.wb_CYC
```

- Group1: `top.accelerator.wb_`
- Group2: `top.cpu.wb_`

Hierarchy Independent WAL

```
(in-groups (groups ("ACK", "STB", "CYC"))
  (define transactions = 0)
  (whenever (&& (rising top.clk) #ACK #STB #CYC)
    (inc transactions))
  (print
    CG " performed " transactions " transactions")
```

Hierarchy Independent WAL

```
(in-groups (groups ("ACK", "STB", "CYC"))
  (define transactions = 0)
  (whenever (&& (rising top.clk) #ACK #STB #CYC)
    (inc transactions))
  (print
    CG " performed " transactions " transactions")
```

But Wait

That was a lot more complex than `#[wal_trace(clk=clk...)]`

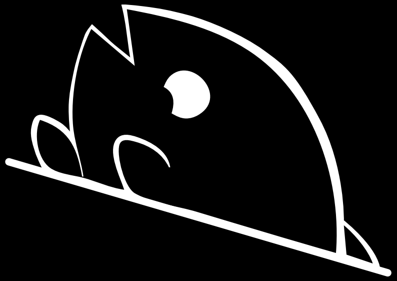
Enhancing Compiler-driven HDL Design with Automatic Waveform Analysis

Enhancing Compiler-driven HDL Design with Automatic Waveform Analysis

Spade

A Hardware Description Language inspired by software

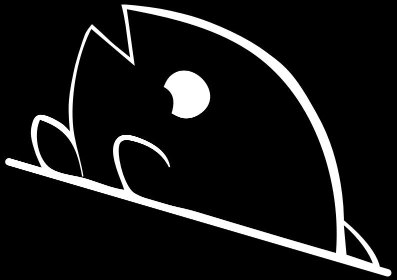
- Raise the Abstraction Level



Spade

A Hardware Description Language inspired by software

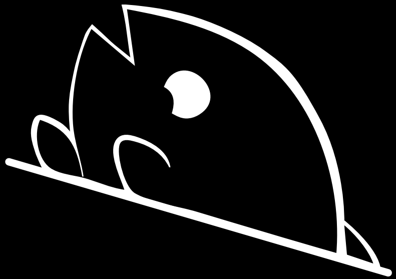
- Raise the Abstraction Level
- Without Sacrificing Control



Spade

A Hardware Description Language inspired by software

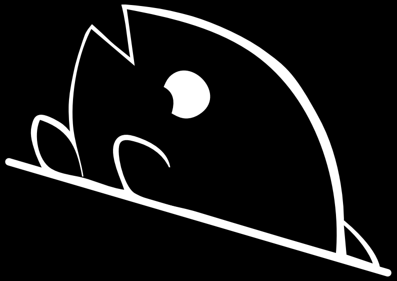
- Raise the Abstraction Level
- Without Sacrificing Control
- Native constructs for hardware
  - Pipelines
  - Ports and Buses
  - Expression Based Semantics



Spade

A Hardware Description Language inspired by software

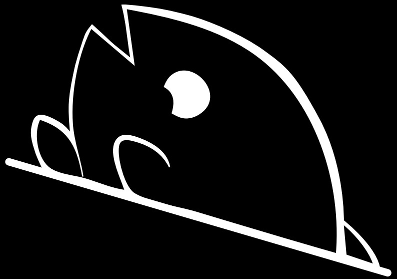
- Raise the Abstraction Level
- Without Sacrificing Control
- Native constructs for hardware
  - Pipelines
  - Ports and Buses
  - Expression Based Semantics
- Powerful type system



Spade

A Hardware Description Language inspired by software

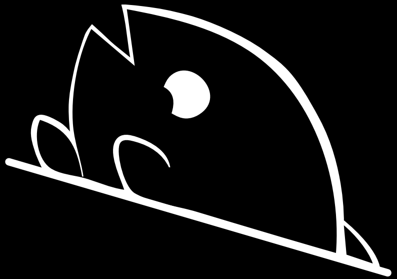
- Raise the Abstraction Level
- Without Sacrificing Control
- Native constructs for hardware
  - Pipelines
  - Ports and Buses
  - Expression Based Semantics
- Powerful type system
- Helpful tooling
  - `swim` build tool



Spade

A Hardware Description Language inspired by software

- Raise the Abstraction Level
- Without Sacrificing Control
- Native constructs for hardware
  - Pipelines
  - Ports and Buses
  - Expression Based Semantics
- **Powerful type system**
- **Helpful tooling**
  - `swim` **build tool**



```
pipeline(5) accelerator(
    clk: clock,
    addr: int<24>,
    opa: int<8>, opb: int<8>, opc: int<16>
    wb: Wishbone
) {
        let product = opa * opb;
    reg*4;
        let sum = product + opc;
    reg;
        wb
            .inst write(addr, sum)
}
```

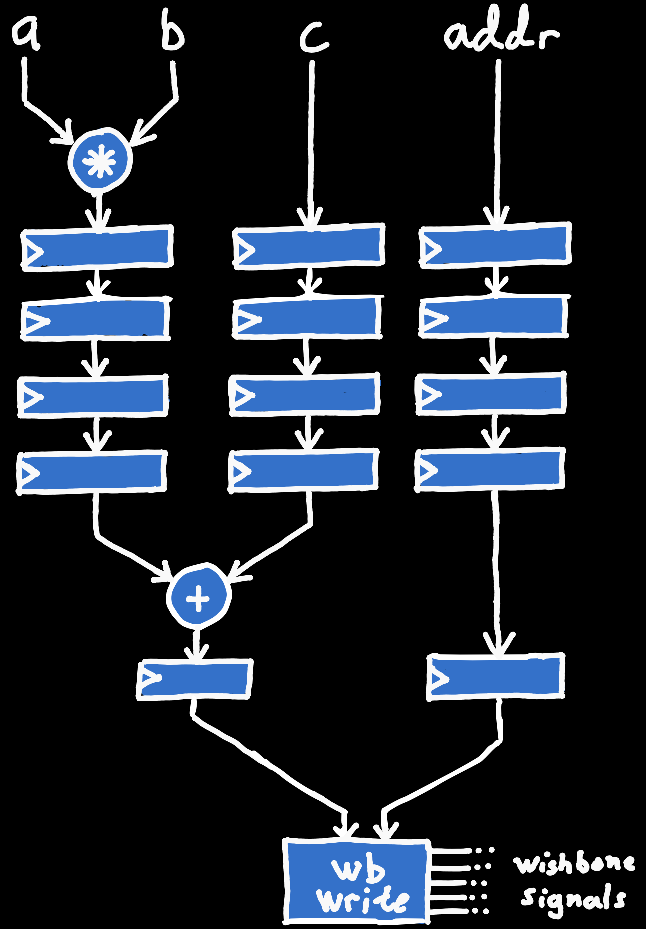
```
pipeline(5) accelerator(
    clk: clock,
    addr: int<24>,
    opa: int<8>, opb: int<8>, opc: int<16>
    wb: Wishbone
) {
        let product = opa * opb;
    reg*4;
        let sum = product + opc;
    reg;
        wb
            .inst write(addr, sum)
}
```

```
pipeline(5) accelerator(
    clk: clock,
    addr: int<24>,
    opa: int<8>, opb: int<8>, opc: int<16>
    wb: Wishbone
) {
        let product = opa * opb;
    reg*4;
        let sum = product + opc;
    reg;
        wb
            .inst write(addr, sum)
}
```

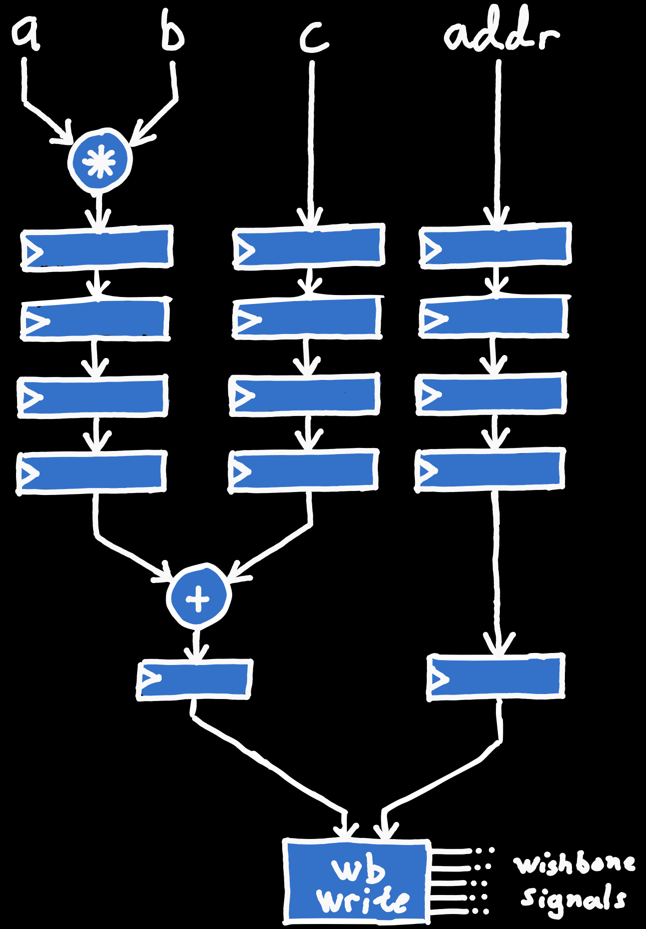
```
pipeline(5) accelerator(
    clk: clock,
    addr: int<24>,
    opa: int<8>, opb: int<8>, opc: int<16>
    wb: Wishbone
) {
        let product = opa * opb;
    reg*4;
        let sum = product + opc;
    reg;
        wb
            .inst write(addr, sum)
}
```

```
pipeline(5) accelerator(
    clk: clock,
    addr: int<24>,
    opa: int<8>, opb: int<8>, opc: int<16>
    wb: Wishbone
) {
        let product = opa * opb;
    reg*4;
        let sum = product + opc;
    reg;
        wb
            .inst write(addr, sum)
}
```

```
pipeline(5) accelerator(
    clk: clock,
    addr: int<24>,
    opa: int<8>, opb: int<8>, opc: int<16>
    wb: Wishbone
) {
        let product = opa * opb;
    reg*4;
        let sum = product + opc;
    reg;
        wb
            .inst write(addr, sum)
}
```



```
pipeline(5) accelerator(
    clk: clock,
    addr: int<24>,
    opa: int<8>, opb: int<8>, opc: int<16>
    wb: Wishbone
) {
        let product = opa * opb;
    reg*4;
        let sum = product + opc;
    reg;
        wb
            .inst write(addr, sum)
}
```



```
struct port Wishbone {
  addr: &mut int<24>,
  write: &mut Option<int<16>>,
  data_out: &int<16>,
  sel: &mut int<2>,
  cyc: &mut bool,
  stb: &mut bool,
  ack: &bool,
  err: &bool
}

impl Wishbone {
  entity read(self, addr: int<24>) -> Option<int<16>> {
    // ...
  }
}
```

Adding Tracing

```
struct port Wishbone {
  addr: &mut int<24>,
  write: &mut Option<int<16>>,
  data: &int<16>,
  ...
}
```

Adding Tracing

```
#[wal_traceable(uses_clk, uses_rst)]
struct port Wishbone {
  addr: &mut int<24>,
  write: &mut Option<int<16>>,
  data: &int<16>,
  ...
}
```

- Tell Spade that a WAL script exists for this type

Adding Tracing

```
#[wal_traceable(uses_clk, uses_rst)]
struct port Wishbone {
  addr: &mut int<24>,
  write: &mut Option<int<16>>,
  data: &int<16>,
  ...
}

wire[..] wb1__addr_wishbone::Wishbone
wire[..] wb1__write_wishbone::Wishbone
wire[..] wb1__data_wishbone::Wishbone
...
```

- Tell Spade that a WAL script exists for this type
- Emit copies of fields with predictable names

Adding Tracing

```
#[wal_traceable(uses_clk, uses_rst)]
struct port Wishbone {
  addr: &mut int<24>,
  write: &mut Option<int<16>>,
  data: &int<16>,
  ...
}

wire[..] wb1__addr_wishbone::Wishbone
wire[..] wb1__write_wishbone::Wishbone
wire[..] wb1__data_wishbone::Wishbone
...
```

- Tell Spade that a WAL script exists for this type
- Emit copies of fields with predictable names
- WAL can discover these signals

Adding Tracing

```
#[wal_traceable(uses_clk, uses_rst)]
struct port Wishbone {
  addr: &mut int<24>,
  write: &mut Option<int<16>>,
  data: &int<16>,
  ...
}

wire[..] wb1__addr_wishbone::Wishbone
wire[..] wb1__write_wishbone::Wishbone
wire[..] wb1__data_wishbone::Wishbone
...
```

- Tell Spade that a WAL script exists for this type
- Emit copies of fields with predictable names
- WAL can discover these signals
- And does not need to know the internal struct representeation

Dependency Management

```toml
[libraries.wishbone]
git = "https://gitlab.com/..."
branch = "main"
```

```
+ wishbone
├─ + src
│   └─ impl.spade
└─ swim.toml
```

Dependency Management

```toml
[libraries.wishbone]
git = "https://gitlab.com/..."
branch = "main"
```

```
+ wishbone
├─ + src
│   └─ impl.spade
├─ + wal
│   ├─ statistics.wal
│   └─ histogram.wal
└─ swim.toml
```

Back to the Example

```
entity cpu(clk: clock, wb: Wishbone) {..}
pipeline(5) accel(
  clk: clock, wb: Wishbone, ..
) {..}

entity wb_harness(clk: clock, rst: bool) {

  let (wb1, wb1inv) = port;

  let (wb2, wb2inv) = port;

  let _ = inst wishbone_arbiter(
    .., wb1inv, wb2inv, ..
  );
  let o1 = inst cpu(.., wb1, ..);
  let o2 = inst(5) accel(.., wb2, ..);
}
```

Back to the Example

```
entity cpu(clk: clock, wb: Wishbone) {..}
pipeline(5) accel(
  clk: clock, wb: Wishbone, ..
) {..}

entity wb_harness(clk: clock, rst: bool) {

  let (wb1, wb1inv) = port;

  let (wb2, wb2inv) = port;

  let _ = inst wishbone_arbiter(
    .., wb1inv, wb2inv, ..
  );
  let o1 = inst cpu(.., wb1, ..);
  let o2 = inst(5) accel(.., wb2, ..);
}
```

Back to the Example

```
entity cpu(clk: clock, wb: Wishbone) {..}
pipeline(5) accel(
  clk: clock, wb: Wishbone, ..
) {..}

entity wb_harness(clk: clock, rst: bool) {


  let (wb1, wb1inv) = port;


  let (wb2, wb2inv) = port;

  let _ = inst wishbone_arbiter(
    .., wb1inv, wb2inv, ..
  );
  let o1 = inst cpu(.., wb1, ..);
  let o2 = inst(5) accel(.., wb2, ..);
}
```

Back to the Example

```
entity cpu(clk: clock, wb: Wishbone) {..}
pipeline(5) accel(
  clk: clock, wb: Wishbone, ..
) {..}

entity wb_harness(clk: clock, rst: bool) {

  let (wb1, wb1inv) = port;

  let (wb2, wb2inv) = port;

  let _ = inst wishbone_arbiter(
    .., wb1inv, wb2inv, ..
  );
  let o1 = inst cpu(.., wb1, ..);
  let o2 = inst(5) accel(.., wb2, ..);
}
```

- Type Inference

Back to the Example

```
entity cpu(clk: clock, wb: Wishbone) {..}
pipeline(5) accel(
  clk: clock, wb: Wishbone, ..
) {..}

entity wb_harness(clk: clock, rst: bool) {

  let (wb1, wb1inv) = port;

  let (wb2, wb2inv) = port;

  let _ = inst wishbone_arbiter(
    .., wb1inv, wb2inv, ..
  );
  let o1 = inst cpu(.., wb1, ..);
  let o2 = inst(5) accel(.., wb2, ..);
}
```

- Opt into the bundled analysis

Back to the Example

```
entity cpu(clk: clock, wb: Wishbone) {..}
pipeline(5) accel(
  clk: clock, wb: Wishbone, ..
) {..}

entity wb_harness(clk: clock, rst: bool) {

  let (wb1, wb1inv) = port;

  let (wb2, wb2inv) = port;

  let _ = inst wishbone_arbiter(
    .., wb1inv, wb2inv, ..
  );
  let o1 = inst cpu(.., wb1, ..);
  let o2 = inst(5) accel(.., wb2, ..);
}
```

```
wire[..]
wb1__addr_wishbone::Wishbone
wire[..]
wb1__write_wishbone::Wishbone
wire[..]
wb1__data_wishbone::Wishbone
...

wire[..]
wb2__addr_wishbone::Wishbone
wire[..]
wb2__write_wishbone::Wishbone
wire[..]
wb2__data_wishbone::Wishbone
...
```

Back to the Example

```
entity cpu(clk: clock, wb: Wishbone) {..}
pipeline(5) accel(
  clk: clock, wb: Wishbone, ..
) {..}

entity wb_harness(clk: clock, rst: bool) {

  let (wb1, wb1inv) = port;

  let (wb2, wb2inv) = port;

  let _ = inst wishbone_arbiter(
    .., wb1inv, wb2inv, ..
  );
  let o1 = inst cpu(.., wb1, ..);
  let o2 = inst(5) accel(.., wb2, ..);
}
```

Back to the Example

```
entity cpu(clk: clock, wb: Wishbone) {..}
pipeline(5) accel(
  clk: clock, wb: Wishbone, ..
) {..}

entity wb_harness(clk: clock, rst: bool) {
  #[wal_trace(clk=clk, rst=rst)]
  let (wb1, wb1inv) = port;
  #[wal_trace(clk=clk, rst=rst)]
  let (wb2, wb2inv) = port;

  let _ = inst wishbone_arbiter(
    .., wb1inv, wb2inv, ..
  );
  let o1 = inst cpu(.., wb1, ..);
  let o2 = inst(5) accel(.., wb2, ..);
}
```

Back to the Example

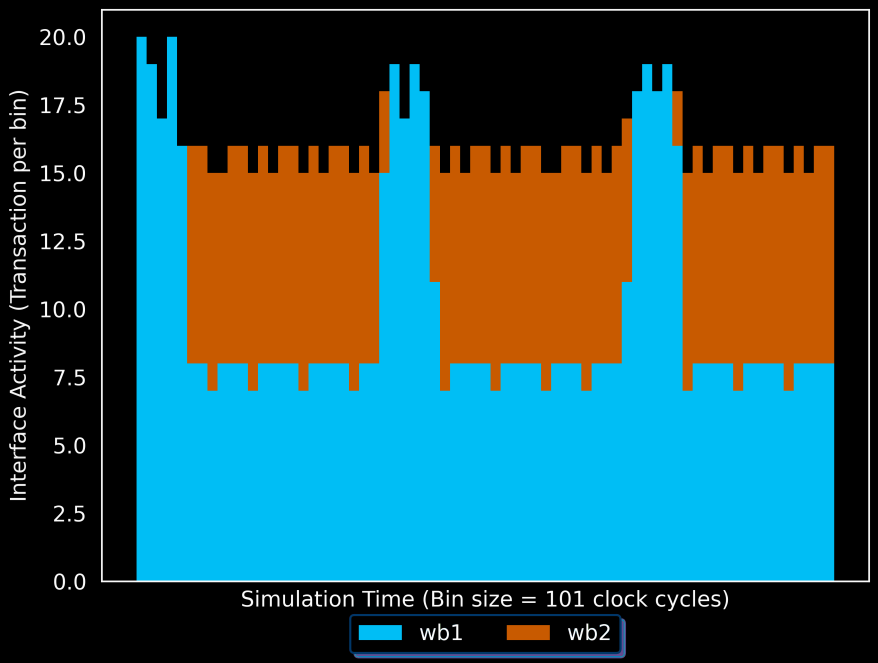
```
entity cpu(clk: clock, wb: Wishbone) {..}
pipeline(5) accel(
  clk: clock, wb: Wishbone, ..
) {..}

entity wb_harness(clk: clock, rst: bool) {
  #[wal_trace(clk=clk, rst=rst)]
  let (wb1, wb1inv) = port;
  #[wal_trace(clk=clk, rst=rst)]
  let (wb2, wb2inv) = port;

  let _ = inst wishbone_arbiter(
    .., wb1inv, wb2inv, ..
  );
  let o1 = inst cpu(.., wb1, ..);
  let o2 = inst(5) accel(.., wb2, ..);
}
```

Back to the example

```
$ swim plugin analysis
```

Back to the example

```
$ swim plugin analysis

[ANALYSIS] # wb_harness.wb1
[ANALYSIS]    Nr. transactions : 697
[ANALYSIS]    Nr. writes       : 681
[ANALYSIS]    Avg write latency: 9
clock cycles
[ANALYSIS]    Inactive cycles  : 6%
[ANALYSIS] # wb_harness.wb2
[ANALYSIS]    Nr. transactions : 421
[ANALYSIS]    Nr. writes       : 421
[ANALYSIS]    Avg write latency: 12
clock cycles
[ANALYSIS]    Inactive cycles  : 22%
```



Improved WAL Ergonomics

```
(define instances
  (spade-struct wishbone::Wishbone
      [clk, rst, ack, stb, cyc]))

(in-spade-structs
  wishbone::Wishbone instances
  (let [transactions 0]
    (whenever (&& (rising #clk) #stb #ack #cyc)
      (inc transactions))
  (log/analysis CG ": " transactions))
```

Improved WAL Ergonomics

```
(define instances
  (spade-struct wishbone::Wishbone
      [clk, rst, ack, stb, cyc])

(in-spade-structs
  wishbone::Wishbone instances
  (let [transactions 0]
    (whenever (&& (rising #clk) #stb #ack #cyc)
      (inc transactions))
  (log/analysis CG ": " transactions))
```

Improved WAL Ergonomics

```
(define instances
  (spade-struct wishbone::Wishbone
      [clk, rst, ack, stb, cyc])

(in-spade-structs
  wishbone::Wishbone instances
  (let [transactions 0]
    (whenever (&& (rising #clk) #stb #ack #cyc)
      (inc transactions)))
  (log/analysis CG ": " transactions))
```

Improved WAL Ergonomics

```
(define instances
  (spade-struct wishbone::Wishbone
      [clk, rst, ack, stb, cyc])

(in-spade-structs
  wishbone::Wishbone instances
  (let [transactions 0]
    (whenever (&& (rising #clk) #stb #ack #cyc)
      (inc transactions)))
  (log/analysis CG ": " transactions))
```

And WAL-Spade Integration

```
enum State {
  Wait,
  Write{
    address: int<3>,
    data: int<3>
  }
  Read{
    address: int<3>
  }
}
```

```
0b00000000

0b01110101
```

And WAL-Spade Integration

```
enum State {
  Wait,
  Write{
    address: int<3>,
    data: int<3>
  }
  Read{
    address: int<3>
  }
}
```

```
(spade/translate top.state)
```

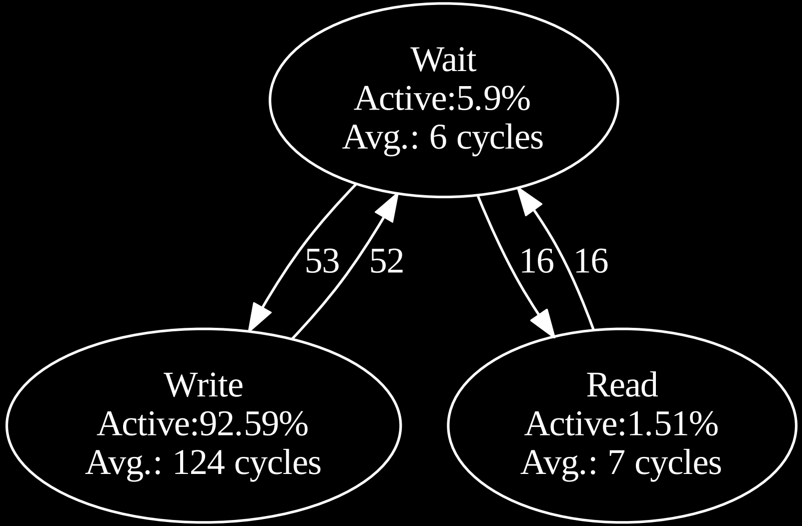
```
0b00000000 ⇒ Wait()

0b01110101 ⇒ Write(6, 5)
```

FSM analysis

```
#[fsm]
reg(clk) state = match state {...}
```

FSM analysis

```
#[fsm]
reg(clk) state = match state {...}
```



Ad-Hoc WAL passes

```
(defun run-pass []
  (in-scope 'proj::main::main
    (whenever
        (= "Mult" (spade/translate "state" ~state))
      (log INDEX ~l "*" ~r "=" ~prod))))
```

- Automatically found by `swim`
- Has access to the same Spade translation features

Conclusions & Contributions

A methodology

- Annotate signals to opt into analysis
- Bundle analysis passes with dependencies
- Let the compiler do the heavy lifting

LINKÖPING UNIVERSITY

JOHANNES KEPLER UNIVERSITÄT LINZ

Conclusions & Contributions

A methodology

- Annotate signals to opt into analysis
- Bundle analysis passes with dependencies
- Let the compiler do the heavy lifting

An implementation

- Extension of Spade and WAL
- Analysis plugin is on gitlab
- Wishbone library with analysis

LINKÖPING UNIVERSITY

JOHANNES KEPLER UNIVERSITÄT LINZ

Conclusions & Contributions

A methodology

- Annotate signals to opt into analysis
- Bundle analysis passes with dependencies
- Let the compiler do the heavy lifting

An implementation

- Extension of Spade and WAL
- Analysis plugin is on gitlab
- Wishbone library with analysis

`spade-lang.org/fdl23`

LINKÖPING UNIVERSITY

JOHANNES KEPLER UNIVERSITÄT LINZ