Enhancing Compiler-Driven HDL Design with Automatic Waveform Analysis

Frans Skarman¹  Lucas Klemmer²  Oscar Gustafsson¹  Daniel Große²

Linköping University¹ and Johannes Kepler University²
frans.skarman@liu.se¹, lucas.klemmer@jku.at²
A motivating example

Shared device → Arbiter ← Processor

Shared device → Arbiter ← Accelerator
Traditional Waveform Analysis

CLK_I
ADDR_o
DAT_O
STB_O
CYC_O
ACK_I

Addr changed before ACK
Traditional Waveform Analysis

Is powerful for detailed debugging
Traditional Waveform Analysis

Not as powerful for large scale analysis
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)
}

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?
WAL

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

INDEX

top.clk

top.rst

top.data  0xAA  0xBB  0xCC  0xDD
WAL Primer

(print "At " INDEX ": " top.data)  At 0: 0xAA
(step 10)
(print "At " INDEX ": " top.data)
(print "At " INDEX ": " top.data)

At 0: 0xAA

(print "At " INDEX ": " top.data)
WAL Primer

(print "At " INDEX ": " top.data) At 0: 0xAA
(step 10)
(print "At " INDEX ": " top.data)
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)
(define transactions 0)
(whenever (&& (rising clk) ACK STB CYC)
  (inc transactions))
(print "Number of transactions: " transactions)
(in-groups (groups ("ACK", "STB", "CYC")))
(define transactions = 0)
(whenever (&& (rising top.clk) #ACK #STB #CYC)
  (inc transactions))
(print
  CG " performed " transactions " transactions")
(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

\[(\text{in-groups (groups ("ACK", "STB", "CYC"))})\]
\[(\text{define transactions = 0})\]
\[(\text{whenever (&& (rising top.clk) #ACK #STB #CYC)})\]
\[(\text{inc transactions})\]
\[(\text{print CG " performed " transactions " transactions"})\]

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
(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

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

• Group1: top.accelerator.wb_ 
```
Hierarchy Independent WAL

\[
\text{(in-groups (groups \{"ACK", "STB", "CYC"\}))}
\]

\[
(\text{define transactions = 0})
\]

\[
(\text{whenever (&& (rising top.clk) \#ACK \#STB \#CYC)}
\]

\[
(\text{inc transactions})
\]

\[
(\text{print}
\]

\[
CG \ " \text{performed } \text{transactions } \text{transactions}"
\]

- 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")
(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
A Hardware Description Language inspired by software
• Raise the Abstraction Level
A Hardware Description Language inspired by software
• Raise the Abstraction Level
• Without Sacrificing Control
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
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>> {
        // ...
    }
}
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

```rust
#[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 representation
Dependency Management

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

+ wishbone
  | + src
  |   | impl.spade
  |   | swim.toml
Dependency Management

[libraries.wishbone]
git = "https://gitlab.com/..."
branch = "main"
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, ..);
}

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, ..);

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

define 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
...
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,..);
}
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
Back to the example

$ swim plugin analysis
$ swim plugin analysis

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

[ANALYSIS] # wb_harness.wb2
Nr. transactions : 421
Nr. writes       : 421
Avg write latency: 12 clock cycles
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))

23
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

```c
enum State {
    Wait,
    Write{
        address: int<3>,
        data: int<3>
    },
    Read{
        address: int<3>
    }
}
```
And WAL-Spade Integration

```cpp
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 {...}

Wait
Active: 5.9%
Avg.: 6 cycles

Write
Active: 92.59%
Avg.: 124 cycles

Read
Active: 1.51%
Avg.: 7 cycles

53 52 16 16
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
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
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