Spade - An HDL Inspired by Modern Software Languages

Frans Skarman

Linköping University
Stealing From Software

• **Fearless** refactoring
Stealing From Software

• Fearless refactoring
• Abstractions for hardware
Stealing From Software

- Fearless refactoring
- Abstractions for hardware
- **Low** performance overhead
Stealing From Software

- Fearless refactoring
- Abstractions for hardware
- Low performance overhead
- Great tooling
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
}
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
}
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 produt = a*b;
    reg;
    let sum = x + f(a, product)
    reg;
    sum
}
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
}
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
pipeline(2) X(clk: clock, a: int<32>, b: int<32>) → int<33> {
    let x = \text{inst}(2) g(a);
    let \text{produt} = a \times b;
    reg;
    let \text{sum} = x + f(a, \text{product})
    reg;
    \text{sum}
}
pipelines(3) X(clk: clock, a: int<32>, b: int<32>)
    → int<33> {
    let x = inst(2) g(a);
    let product = a*b;
    reg;
    reg;
    reg;
    let sum = x + f(a, product)
    reg;
    sum
}
Demo?
Camera protocol
Camera protocol
• 1-4 lanes + clock
Camera protocol
- 1-4 lanes + clock
- Find SOT
Camera protocol
• 1-4 lanes + clock
• Find SOT
• Merge lanes
Camera protocol
• 1-4 lanes + clock
• Find SOT
• Merge lanes
• Look for packet headers
Camera protocol
- 1-4 lanes + clock
- Find SOT
- Merge lanes
- Look for packet headers
- Separate short and long packets
Camera protocol

- 1-4 lanes + clock
- Find SOT
- Merge lanes
- Look for packet headers
- Separate short and long packets
- Pixels after packet with header 0x2A
Streams

- 1-4 lanes + clock
- Find SOT
- Merge lanes
- Look for packet headers
- Separate short and long packets
- Pixels after packet with header \(0x2A\)

```vhdl
pipeline(7) csi2(clk, rst, lanes: [uint<8>; 2]) {
    let aligned = inst aligner(clk, rst, lanes);
    reg;
    let merged = merger(aligned);
    reg;
    let headers = merged
        .inst into_packet_headers(clk, rst);
    reg;
    let short_packets = headers
        .into_short_packets();
    let pixel_headers = headers
        .into_long_packets();
    reg;
    let raw_pixels = pixel_headers
        .inst into_pixel_stream(clk, rst, merged);
}
```
Streams

- 1-4 lanes + clock
- Find SOT
- Merge lanes
- Look for packet headers
- Separate short and long packets
- Pixels after packet with header \(0x2A\)

```plaintext
code
pipeline(7) csi2(clk, rst, lanes: [uint<8>; 2]) {
    let aligned = inst aligner(clk, rst, lanes);
    reg;
    let merged = merger(aligned);
    reg;
    let headers = merged .inst into_packet_headers(clk, rst);
    reg;
    let short_packets = headers .into_short_packets();
    let pixel_headers = headers .into_long_packets();
    reg;
    let raw_pixels = pixel_headers .inst into_pixel_stream(clk, rst, merged);
}
```
Streams

- 1-4 lanes + clock
- Find SOT
- Merge lanes
- Look for packet headers
- Separate short and long packets
- Pixels after packet with header \(0x2A\)

```plaintext
def pipeline(7) csi2(clk, rst, lanes: [uint<8>; 2]) {
    let aligned = inst aligner(clk, rst, lanes);
    reg;
    let merged = merger(aligned);
    reg;
    let headers = merged
        .inst into_packet_headers(clk, rst);
    reg;
    let short_packets = headers
        .into_short_packets();
    let pixel_headers = headers
        .into_long_packets();
    reg;
    let raw_pixels = pixel_headers
        .inst into_pixel_stream(clk, rst, merged);
}
```
Streams

- 1-4 lanes + clock
- Find SOT
- Merge lanes
- Look for packet headers
- Separate short and long packets
- Pixels after packet with header 0x2A

```verilog
pipeline(7) csi2(clk, rst, lanes: [uint<8>; 2]) {
    let aligned = inst aligner(clk, rst, lanes);
    reg;
    let merged = merger(aligned);
    reg;
    let headers = merged .inst into_packet_headers(clk, rst);
    reg;
    let short_packets = headers .into_short_packets();
    let pixel_headers = headers .into_long_packets();
    reg;
    let raw_pixels = pixel_headers .inst into_pixel_stream(clk, rst, merged);
}
```
Streams

- 1-4 lanes + clock
- Find SOT
- Merge lanes
- Look for packet headers
- Separate short and long packets
- Pixels after packet with header \(0x2A\)

```plaintext
def pipeline(7) csi2(clk, rst, lanes: [uint<8>; 2]) {
    let aligned = inst aligner(clk, rst, lanes);
    reg;
    let merged = merger(aligned);
    reg;
    let headers = merged
        .inst into_packet_headers(clk, rst);
    reg;
    let short_packets = headers
        .into_short_packets();
    let pixel_headers = headers
        .into_long_packets();
    reg;
    let raw_pixels = pixel_headers
        .inst into_pixel_stream(clk, rst, merged);
}
```
Streams

- 1-4 lanes + clock
- Find SOT
- Merge lanes
- Look for packet headers
- Separate short and long packets
- Pixels after packet with header \text{0x2A}

```plaintext
pipeline(7) csi2(clk, rst, lanes: [uint<8>; 2]) {
    let aligned = inst aligner(clk, rst, lanes);
    reg;
    let merged = merger(aligned);
    reg;
    let headers = merged
        .inst into_packet_headers(clk, rst);
    reg;
    let short_packets = headers
        .into_short_packets();
    let pixel_headers = headers
        .into_long_packets();
    reg;
    let raw_pixels = pixel_headers
        .inst into_pixel_stream(clk, rst, merged);
}
```
Streams

- 1-4 lanes + clock
- Find SOT
- Merge lanes
- Look for packet headers
- Separate short and long packets
- Pixels after packet with header 0x2A

```haskell
pipeline(7) csi2(clk, rst, lanes: [uint<8>; 2]) {
    let aligned = inst aligner(clk, rst, lanes);
    reg;
    let merged = merger(aligned);
    reg;
    let headers = merged
        .inst into_packet_headers(clk, rst);
    reg;
    let short_packets = headers
        .into_short_packets();
    let pixel_headers = headers
        .into_long_packets();
    reg;
    let raw_pixels = pixel_headers
        .inst into_pixel_stream(clk, rst, merged);
}
```
struct LongHeaderStream {
    s: Option<Header>
}

impl LongHeaderStream {
    entity into_pixel_stream(..) → PixelStream {
        reg(clk) num_left reset(rst: 0) =
        match self.s {
            Some(Header$(id: 0x2A, count)) ⇒ count,
            None ⇒ saturating_sub(num_left - 1)
        };
        PixelStream(
            if num_left > 0 { s.s } else { None },
        )
    }
}
struct LongHeaderStream {
    s: Option<Header>
}

impl LongHeaderStream {
    entity into_pixel_stream(..) -> PixelStream {
        reg(clk) num_left reset(rst: 0) =
            match self.s {
                Some(Header$(id: 0x2A, count)) => count,
                None => saturating_sub(num_left - 1)
            };

        PixelStream(
            if num_left > 0 { s.s } else { None },
        )
    }
}
struct LongHeaderStream {
  s: Option<Header>
}

impl LongHeaderStream {
  entity into_pixel_stream(..) -> PixelStream {

    reg(clk) num_left reset(rst: 0) =
    match self.s {
      Some(Header$(id: 0x2A, count)) => count,
      None => saturating_sub(num_left - 1)
    };

    PixelStream(
      if num_left > 0 { s.s } else { None },
    )
  }
}
struct LongHeaderStream {
    s: Option<Header>
}

impl LongHeaderStream {
    entity into_pixel_stream(\_\_\_) \rightarrow PixelStream {
        reg(clk) num_left reset(rst: 0) =
        match self.s {
            Some(Header$(id: 0x2A, count)) \rightarrow count,
            None \rightarrow saturating_sub(num_left - 1)
        };

        PixelStream(
            if num_left > 0 { s.s } else { None },
        )
    }
}
Streams

struct LongHeaderStream {
    s: Option<Header>
}

impl LongHeaderStream {
    entity into_pixel_stream(..) -> PixelStream {
        reg(clk) num_left reset(rst: 0) =
            match self.s {
                Some(Header$(id: 0x2A, count)) => count,
                None => saturating_sub(num_left - 1)
            };

        PixelStream(
            if num_left > 0 { s.s } else { None },
        )
    }
}
A language is nothing without its tools
[libraries]
hdmi = {git = "..", branch="main"}
ulx3s_sdram = {path = "./deps/litedram"}
ecp5stubs = {git = "..", branch = "main"}

[plugins]
sdram = {path = "./deps/litedram"}

[synthesis]
command = "synth_ecp5"
top = "main"
extra_verilog = [
    "src/camera_pll.sv",
    "src/ecp5 pll.sv",
]
[libraries]
hdmi = {git = "..", branch="main"}
ulx3s_sdram = {path = "./deps/litedram"}
ecp5stubs = {git = "..", branch = "main"}

[plugins]
sdram = {path = "./deps/litedram"}

[synthesis]
command = "synth_ecp5"
top = "main"
extra_verilog = [  "src/camera_pll.sv",
                "src/ecp5pll.sv",
            ]
[libraries]
hdmi = {git = "..", branch="main"}
ulx3s_sdram = {path = "./deps/litedram"}
cep5stubs = {git = "..", branch = "main"}

[plugins]
sdram = {path = "./deps/litedram"}

[synthesis]
command = "synth_ecp5"
top = "main"
extra_verilog = [
    "src/camera_pll.sv",
    "src/ecp5pll.sv",
]

Swim - The Spade Build Tool
Swim - The Spade Build Tool

Easy setup

cargo install swim
swim install-tools

git clone "git:gitlab.com/user/project"

swim upload
Conclusion

• **Fearless** refactoring
• **Abstractions** for hardware
• **Low** performance overhead
• **Tooling**
Conclusion

- **Fearless** refactoring
- Abstractions for hardware
- Low performance overhead
- Tooling
Conclusion

• **Fearless** refactoring
  ▶ **Pipelines** for trivial re-timing
  ▶ Strong type systems that catches bugs early
• Abstractions for hardware
• Low performance overhead
• Tooling
Conclusion

• Fearless refactoring
  ‣ Pipelines for trivial re-timing
  ‣ Strong type systems that catches bugs early
• Abstractions for hardware
• Low performance overhead
• Tooling
Conclusion

• Fearless refactoring
  ➤ Pipelines for trivial re-timing
  ➤ Strong type systems that catches bugs early

• Abstractions for hardware
  ➤ Pipelines
  ➤ Memories and ports as primitives
  ➤ Stream interfaces expressible in the language

• Low performance overhead

• Tooling
Conclusion

• Fearless refactoring
  ▶ Pipelines for trivial re-timing
  ▶ Strong type systems that catches bugs early
• Abstractions for hardware
  ▶ Pipelines
  ▶ Memories and ports as primitives
  ▶ Stream interfaces expressible in the language
• **Low** performance overhead
• Tooling
Conclusion

- Fearless refactoring
  - Pipelines for trivial re-timing
  - Strong type systems that catches bugs early
- Abstractions for hardware
  - Pipelines
  - Memories and ports as primitives
  - Stream interfaces expressible in the language
- **Low** performance overhead
  - RTL level description
- Tooling
Conclusion

• Fearless refactoring
  ▶ Pipelines for trivial re-timing
  ▶ Strong type systems that catches bugs early
• Abstractions for hardware
  ▶ Pipelines
  ▶ Memories and ports as primitives
  ▶ Stream interfaces expressible in the language
• Low performance overhead
  ▶ RTL level description
• Tooling
Conclusion

• Fearless refactoring
  ➤ Pipelines for trivial re-timing
  ➤ Strong type systems that catches bugs early

• Abstractions for hardware
  ➤ Pipelines
  ➤ Memories and ports as primitives
  ➤ Stream interfaces expressible in the language

• Low performance overhead
  ➤ RTL level description

• Tooling
  ➤ **Helpful** compiler
  ➤ Powerful **build system**
  ➤ Purpose built **waveform viewer**
Conclusion

• **Fearless** refactoring
  ‣ **Pipelines** for trivial re-timing
  ‣ Strong type systems that **catches bugs early**

• **Abstractions** for hardware
  ‣ **Pipelines**
  ‣ **Memories and ports** as primitives
  ‣ **Stream interfaces** expressible in the language

• **Low** performance overhead
  ‣ **RTL** level description

• **Tooling**
  ‣ **Helpful** compiler
  ‣ **Powerful** build system
  ‣ Purpose built **waveform viewer**

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