Spade: An Expression-Based HDL With Pipelines

Frans Skarman, Oscar Gustafsson
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
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
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
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
}
• 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
}
Hello, World

- Inputs separated from outputs

```vhdl
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'

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

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

```vhdl
define 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
}
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
}
```
Key takeaways:

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

```vhdl
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
}
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 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
}
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
Other features

• Feedback and bypasses

• Built-in dynamic behavior
  • Stalls
  • Flushes
  • Back pressure
Types

- Enum called Option

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

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

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

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

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

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

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

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

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

```plaintext
Option<int<8>> tag val

None 0
Some(5) 1 0000 0101
```
More enum examples

- Commands on a bus

```cpp
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

```cpp
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>
    }
}
```
A language is nothing without its tools
# 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

# 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}"
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"

[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

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

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

[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"

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

# ...

Implementation

• Open source, implemented in Rust.
Implementation

• Open source, implemented in Rust.
• Standalone compiler targeting Verilog
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
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 C\lambda\textnormal{ash}?

- 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