Amaranth (1, lecture)

Hardware System Design

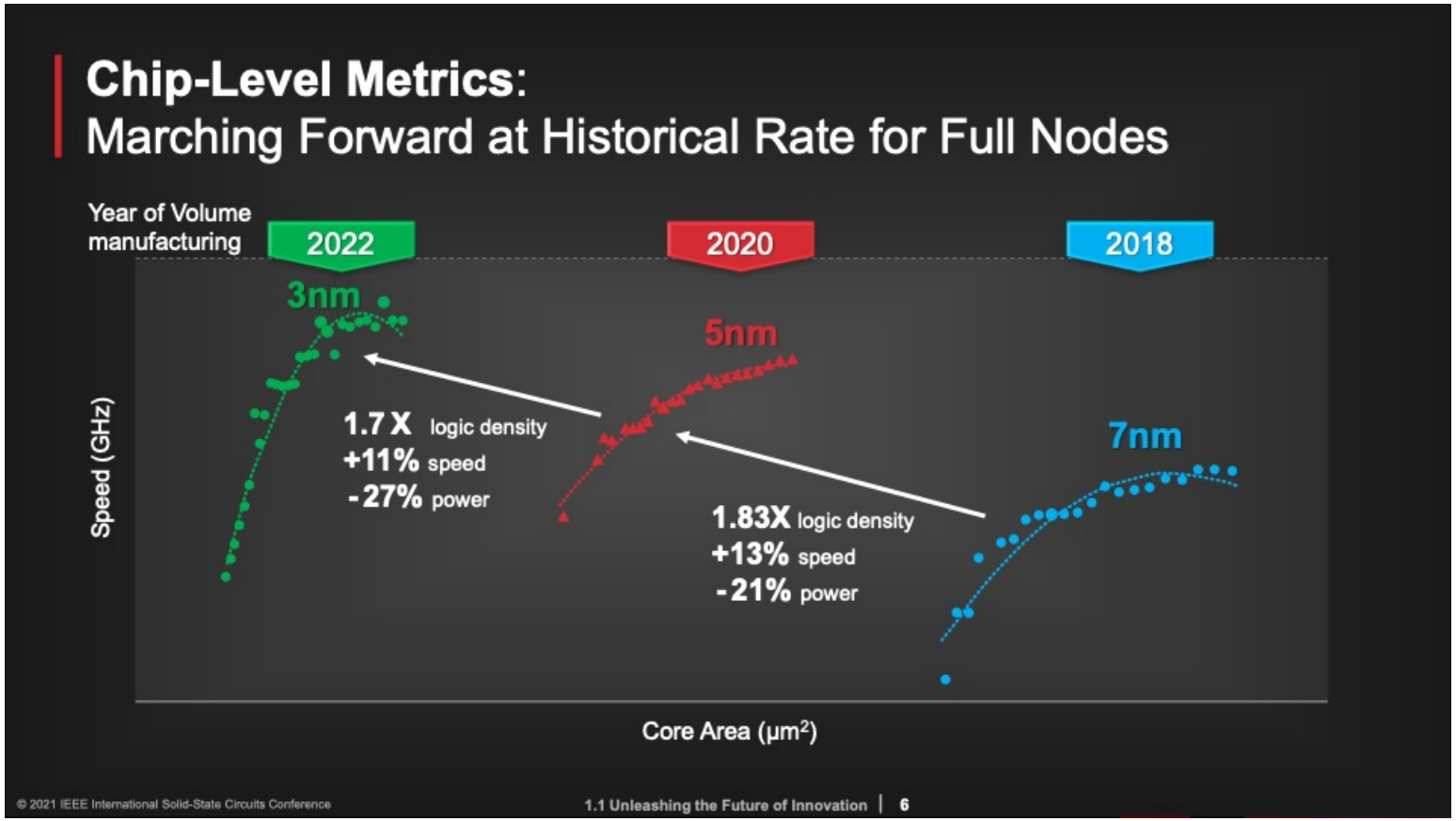
Spring, 2023

Outline

- **Modern HDL**
- Amaranth
  - Introduction
  - Examples & Constructs

Modern HDL

- Technology improves over time…



ISSCC 2021: A Bright Foundry Future - EE Times Asia (eetasia.com)

Modern HDL

- While Verilog and tools have not
  - SystemVerilog is a massive standard and tools are fragmented (making code not portable)
  - No standard libraries
    - Memory
    - FIFO for CDC (clock domain crossing)
  - Not flexible enough
    - Convention is to use Perl script to generate Verilog code

PLUG-AND-PLAY FOSS ML ACCELERATOR : FROM CONCEPT TO CONCEPTION (gatech.edu) Section 3.1.2

Modern HDL

- Chisel
  - HDL on Scala
  - Chisel → FIRRTL (→ Verilog)
  - CHIPS alliance
- Amaranth
  - HDL on Python
  - Amaranth → RTLIL (→ Verilog)
  - Independent open-source project
- And so on…

chipsalliance/chisel (github.com)
m-labs/migen (github.com)
amaranth-lang/amaranth (github.com)
drom/awesome-hdl (github.com)

Modern HDL

- Analogy to C~Python

	C	Python
Binary speed	~100x	1x
Productivity	1x	~10x

	Verilog	Amaranth
HW cycles	1x	1x
Productivity	1x	~10x

- Users focus on frontend code
- Burden passed to backend
  - Optimize HW resource usage
  - Synthesis / place & route tools
- Code critical part with Verilog as needed

Outline

- Modern HDL
- **Amaranth**
  - Introduction
  - Examples & Constructs

Amaranth – introduction

- Migen → nMigen → Amaranth
  - nMigen is fork of Migen (Milkymist Generator)
  - nMigen renamed to Amaranth
- Powerful tools
  - enjoy-digital/litex (github.com) (not covered in this class)

- HDL, not HLS!
  - Hardware Description Language
  - Not High Level Synthesis

Amaranth – example

- Adder
- AdderSync
- TestArray

- Tutorial code
  https://www.notion.so/tutorial-code-1118b39f44324efcaeec85ef3d687c7d?pvs=4

Amaranth – example (1)

```
1     from amaranth import *
  1
  2
  3   class Adder(Elaboratable):
  4       def __init__(self, num_bits):
  5           self.num_bits = num_bits
  6
  7           self.in_a = Signal(num_bits)
  8           self.in_b = Signal(num_bits)
  9           self.out_d = Signal(num_bits)
 10           self.out_ovf = Signal(1)
 11
 12       def elaborate(self, platform):
 13           m = Module()
 14
 15           m.d.comb += [
 16               Cat(self.out_d, self.out_ovf).eq(self.in_a + self.in_b)
 17           ]
 18
 19           return m
```

Amaranth – example (1)

```
1     from amaranth import *
  1
  2
  3   class Adder(Elaboratable):
  4       def __init__(self, num_bits):
  5           self.num_bits = num_bits
  6
  7           self.in_a = Signal(num_bits)
  8           self.in_b = Signal(num_bits)
  9           self.out_d = Signal(num_bits)
 10           self.out_ovf = Signal(1)
 11
 12       def elaborate(self, platform):
 13           m = Module()
 14
 15           m.d.comb += [
 16               Cat(self.out_d, self.out_ovf).eq(self.in_a + self.in_b)
 17           ]
 18
 19           return m
```

Amaranth – constructs

- **`Signal()`** instead of **`reg`** and **`wire`**
  - Automatically assigned by **`m.d.comb`** or **`m.d.sync`**
    - If **`Signal`** is on LHS of **`eq`** at **`m.d.comb`** → **`wire`**
    - If **`Signal`** is on LHS of **`eq`** at **`m.d.sync`** → **`reg`**
    - If **`Signal`** is on LHS of **`eq`** at both → compile error
- **`eq`** instead of **`{assign, <=, =}`**

Amaranth – constructs

- `Signal()` instead of `reg` and `wire`
  - `Signal(shape, reset, reset_less, …)`
    - `shape = Shape(width, signed) castable object or None`

  - 1-bit by default
    - `width=1`
  - Unsigned by default
    - `signed=False`
  - Reset value 0 by default
    - `reset=0`
    - `reset_less=False`

```
>>> Signal().shape()
unsigned(1)
>>> Signal(4).shape()
unsigned(4)
>>> Signal(range(-8, 7)).shape()
signed(4)
```

Language guide(Shape) (amaranth-lang.org)
https://github.com/amaranth-lang/amaranth/blob/ae1aeff0f2b110de3e24e7d1abd0370f06ffe745/amaranth/hdl/ast.py#L52
Language guide(Signal) (amaranth-lang.org)
https://github.com/amaranth-lang/amaranth/blob/ae1aeff0f2b110de3e24e7d1abd0370f06ffe745/amaranth/hdl/ast.py#L947

Amaranth – example (1)

```
1    from amaranth import *
1
2
3    class Adder(Elaboratable):
4        def __init__(self, num_bits):
5            self.num_bits = num_bits
6
7            self.in_a = Signal(num_bits)
8            self.in_b = Signal(num_bits)
9            self.out_d = Signal(num_bits)
10           self.out_ovf = Signal(1)
11
12       def elaborate(self, platform):
13           m = Module()
14
15           m.d.comb += [
16               Cat(self.out_d, self.out_ovf).eq(self.in_a + self.in_b)
17           ]
18
19           return m
```

No sync logic
all wires
+ no synchronous submodules
→ Adder is combinational circuit

Amaranth – example (1)

```python
if __name__ == '__main__':
    num_bits = 4
    dut = Adder(num_bits=num_bits)

    from amaranth.sim import Simulator, Delay, Settle

    def test_case(dut, a, b, d, ovf):
        yield dut.in_a.eq(a)
        yield dut.in_b.eq(b)
        yield Settle()
        yield Delay(1e-6)
        assert (yield dut.out_d == d)
        assert (yield dut.out_ovf == ovf)

    def bench():
        for i in range(2 ** num_bits):
            for j in range(2 ** num_bits):
                try:
                    yield from test_case(dut, i, j,
                                         (i + j) % (2 ** num_bits),
                                         (i + j) // (2 ** num_bits))
                except AssertionError:
                    print(i, j, (i + j) % (2 ** num_bits),
                          (i + j) // (2 ** num_bits))

    from pathlib import Path
    p = Path(__file__)

    sim = Simulator(dut)
    sim.add_process(bench)

    with open(p.with_suffix('.vcd'), 'w') as f:
        with sim.write_vcd(f):
            sim.run()

    from amaranth.back import verilog
    top = Adder(num_bits=num_bits)
    with open(p.with_suffix('.v'), 'w') as f:
        f.write(
            verilog.convert(
                top,
                ports=[top.in_a, top.in_b, top.out_d, top.out_ovf]))
```

Amaranth – example (1)

```python
if __name__ == '__main__':
    num_bits = 4
    dut = Adder(num_bits=num_bits)

    from amaranth.sim import Simulator, Delay, Settle

    def test_case(dut, a, b, d, ovf):
        yield dut.in_a.eq(a)
        yield dut.in_b.eq(b)
        yield Settle()
        yield Delay(1e-6)
        assert (yield dut.out_d == d)
        assert (yield dut.out_ovf == ovf)

    def bench():
        for i in range(2 ** num_bits):
            for j in range(2 ** num_bits):
                try:
                    yield from test_case(dut, i, j,
                                         (i + j) % (2 ** num_bits),
                                         (i + j) // (2 ** num_bits))
                except AssertionError:
                    print(i, j, (i + j) % (2 ** num_bits),
                          (i + j) // (2 ** num_bits))

    from pathlib import Path
    p = Path(__file__)

    sim = Simulator(dut)
    sim.add_process(bench)

    with open(p.with_suffix('.vcd'), 'w') as f:
        with sim.write_vcd(f):
            sim.run()

    from amaranth.back import verilog
    top = Adder(num_bits=num_bits)
    with open(p.with_suffix('.v'), 'w') as f:
        f.write(
            verilog.convert(
                top,
                ports=[top.in_a, top.in_b, top.out_d, top.out_ovf]))
```

Comb. logic → add_process

Amaranth – example (1)

```python
if __name__ == '__main__':
    num_bits = 4
    dut = Adder(num_bits=num_bits)

    from amaranth.sim import Simulator, Delay, Settle

    def test_case(dut, a, b, d, ovf):
        yield dut.in_a.eq(a)
        yield dut.in_b.eq(b)
        yield Settle()
        yield Delay(1e-6)
        assert (yield dut.out_d == d)
        assert (yield dut.out_ovf == ovf)

    def bench():
        for i in range(2 ** num_bits):
            for j in range(2 ** num_bits):
                try:
                    yield from test_case(dut, i, j,
                                         (i + j) % (2 ** num_bits),
                                         (i + j) // (2 ** num_bits))
                except AssertionError:
                    print(i, j, (i + j) % (2 ** num_bits),
                          (i + j) // (2 ** num_bits))

    from pathlib import Path
    p = Path(__file__)

    sim = Simulator(dut)
    sim.add_process(bench)

    with open(p.with_suffix('.vcd'), 'w') as f:
        with sim.write_vcd(f):
            sim.run()

    from amaranth.back import verilog
    top = Adder(num_bits=num_bits)
    with open(p.with_suffix('.v'), 'w') as f:
        f.write(
            verilog.convert(
                top,
                ports=[top.in_a, top.in_b, top.out_d, top.out_ovf]))
```

Amaranth – example (1)

```python
if __name__ == '__main__':
    num_bits = 4
    dut = Adder(num_bits=num_bits)

    from amaranth.sim import Simulator, Delay, Settle

    def test_case(dut, a, b, d, ovf):
        yield dut.in_a.eq(a)
        yield dut.in_b.eq(b)
        yield Settle()
        yield Delay(1e-6)
        assert (yield dut.out_d == d)
        assert (yield dut.out_ovf == ovf)

    def bench():
        for i in range(2 ** num_bits):
            for j in range(2 ** num_bits):
                try:
                    yield from test_case(dut, i, j,
                                         (i + j) % (2 ** num_bits),
                                         (i + j) // (2 ** num_bits))
                except AssertionError:
                    print(i, j, (i + j) % (2 ** num_bits),
                          (i + j) // (2 ** num_bits))

    from pathlib import Path
    p = Path(__file__)

    sim = Simulator(dut)
    sim.add_process(bench)

    with open(p.with_suffix('.vcd'), 'w') as f:
        with sim.write_vcd(f):
            sim.run()

    from amaranth.back import verilog
    top = Adder(num_bits=num_bits)
    with open(p.with_suffix('.v'), 'w') as f:
        f.write(
            verilog.convert(
                top,
                ports=[top.in_a, top.in_b, top.out_d, top.out_ovf]))
```

`Settle()` for value assertion
`Delay()` for time delay
(both for combinational logic only)

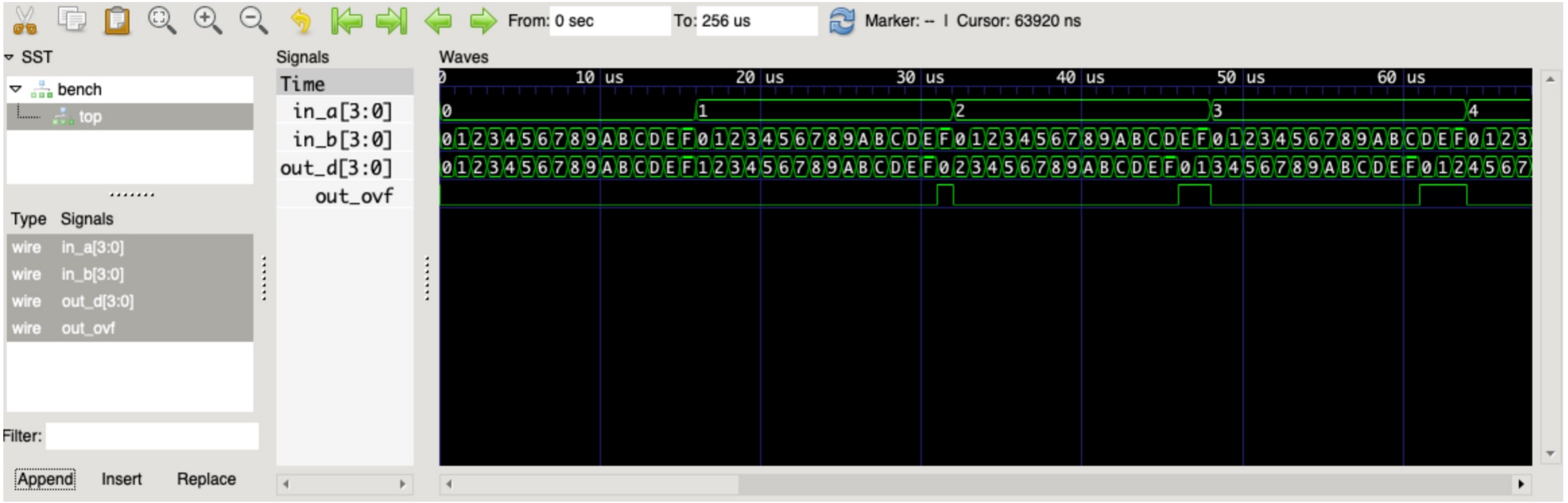
Amaranth – example (1)

```python
if __name__ == '__main__':
    num_bits = 4
    dut = Adder(num_bits=num_bits)

    from amaranth.sim import Simulator, Delay, Settle

    def test_case(dut, a, b, d, ovf):
        yield dut.in_a.eq(a)
        yield dut.in_b.eq(b)
        yield Settle()
        yield Delay(1e-6)
        assert (yield dut.out_d == d)
        assert (yield dut.out_ovf == ovf)

    def bench():
        for i in range(2 ** num_bits):
            for j in range(2 ** num_bits):
                try:
                    yield from test_case(dut, i, j,
                                         (i + j) % (2 ** num_bits),
                                         (i + j) // (2 ** num_bits))
                except AssertionError:
                    print(i, j, (i + j) % (2 ** num_bits),
                          (i + j) // (2 ** num_bits))
    from pathlib import Path
    p = Path(__file__)

    sim = Simulator(dut)
    sim.add_process(bench)

    with open(p.with_suffix('.vcd'), 'w') as f:
        with sim.write_vcd(f):
            sim.run()

    from amaranth.back import verilog
    top = Adder(num_bits=num_bits)
    with open(p.with_suffix('.v'), 'w') as f:
        f.write(
            verilog.convert(
                top,
                ports=[top.in_a, top.in_b, top.out_d, top.out_ovf])
```

Output waveform and verilog

Amaranth – example (1)

- Resulting waveform **`adder.vcd`**



Amaranth – example (1)

- Generated **`adder.v`** (comments omitted)

```
1   /* Generated by Amaranth Yosys 0.25 (PyPI ver 0.25.0.0.post67, git sha1
    e02b7f64b) */
1
2   (* \amaranth.hierarchy  = "top" *)
3   (* top =  1  *)
4   (* generator = "Amaranth" *)
5   module top(in_b, out_d, out_ovf, in_a);
6     wire [4:0] \$1 ;
7     input [3:0] in_a;
8     wire [3:0] in_a;
9     input [3:0] in_b;
10    wire [3:0] in_b;
11    output [3:0] out_d;
12    wire [3:0] out_d;
13    output out_ovf;
14    wire out_ovf;
15    assign \$1  = in_a + in_b;
16    assign { out_ovf, out_d } = \$1 ;
17  endmodule
18
19
```

Amaranth – example (1)

- Generated **`adder.v`** (comments omitted)

```
1   /* Generated by Amaranth Yosys 0.25 (PyPI ver 0.25.0.0.post67, git sha1
    e02b7f64b) */
1
2   (* \amaranth.hierarchy  = "top" *)
3   (* top =  1  *)
4   (* generator = "Amaranth" *)
5   module top(in_b, out_d, out_ovf, in_a);
6     wire [4:0] \$1 ;
7     input [3:0] in_a;
8     wire [3:0] in_a;
9     input [3:0] in_b;
10    wire [3:0] in_b;
11    output [3:0] out_d;
12    wire [3:0] out_d;
13    output out_ovf;
14    wire out_ovf;
15    assign \$1  = in_a + in_b;
16    assign { out_ovf, out_d } = \$1 ;
17  endmodule
18
19
```

Automatically generated wire
Automatic bitwidth extension
(4-bit → 5-bit to handle overflow)

Amaranth – constructs

- Arithmetic operators
  - Arithmetics on Amaranth values **never overflows**
  - **Width** of the arithmetic expression is **always sufficient** to represent **all possible values**

- Width extension
  - On operations that takes two inputs
  - If bitwidth does not match, smaller one is extended
    - Zero-extended for unsigned values
    - Sign-extended for signed values

Language guide(Width extension) (amaranth-lang.org)
Language guide(Arithmetic operators) (amaranth-lang.org)

Q&A on example (1)

Amaranth – example (2)

```
1   from amaranth import *
1
2
3   class AdderSync(Elaboratable):
4       def __init__(self, num_bits, delay=1):
5           self.num_bits = num_bits
6           self.delay = delay
7
8           assert delay >= 0
9
10          self.in_a = Signal(num_bits)
11          self.in_b = Signal(num_bits)
12          self.in_rst = Signal(1, reset_less=True)
13          self.in_en = Signal(1)
14          self.out_d = Signal(num_bits)
15          self.out_ovf = Signal(1)
16
17          if delay == 0:
18              delay = 1
19          self.sim_delay = Signal((num_bits + 1) * delay)
```

```
21      def elaborate(self, platform):
22          m = Module()
23
24          m.d.comb += [
25              Cat(self.out_d, self.out_ovf).eq(
26                  self.sim_delay[-(self.num_bits+1):])
27          ]
28
29          with m.If(self.in_en):
30              if self.delay == 0:
31                  m.d.comb += [
32                      self.sim_delay.eq(self.in_a + self.in_b)
33                  ]
34              else:
35                  m.d.sync += [
36                      self.sim_delay.eq(
37                          self.sim_delay.rotate_left(self.num_bits+1)),
38                      self.sim_delay[:self.num_bits+1].eq(
39                          self.in_a + self.in_b),
40                  ]
41
42          return m
```

Amaranth – example (2)

```python
from amaranth import *


class AdderSync(Elaboratable):
    def __init__(self, num_bits, delay=1):
        self.num_bits = num_bits
        self.delay = delay

        assert delay >= 0

        self.in_a = Signal(num_bits)
        self.in_b = Signal(num_bits)
        self.in_rst = Signal(1, reset_less=True)
        self.in_en = Signal(1)
        self.out_d = Signal(num_bits)
        self.out_ovf = Signal(1)

        if delay == 0:
            delay = 1
        self.sim_delay = Signal((num_bits + 1) * delay)

    def elaborate(self, platform):
        m = Module()

        m.d.comb += [
            Cat(self.out_d, self.out_ovf).eq(
                self.sim_delay[-(self.num_bits+1):])
        ]

        with m.If(self.in_en):
            if self.delay == 0:
                m.d.comb += [
                    self.sim_delay.eq(self.in_a + self.in_b)
                ]
            else:
                m.d.sync += [
                    self.sim_delay.eq(
                        self.sim_delay.rotate_left(self.num_bits+1)),
                    self.sim_delay[:self.num_bits+1].eq(
                        self.in_a + self.in_b),
                ]

        return m
```

AdderSync is synchronous circuit
sim_delay is reg

Amaranth – example (2)

```python
from amaranth import *


class AdderSync(Elaboratable):
    def __init__(self, num_bits, delay=1):
        self.num_bits = num_bits
        self.delay = delay

        assert delay >= 0

        self.in_a = Signal(num_bits)
        self.in_b = Signal(num_bits)
        self.in_rst = Signal(1, reset_less=True)
        self.in_en = Signal(1)
        self.out_d = Signal(num_bits)
        self.out_ovf = Signal(1)

        if delay == 0:
            delay = 1
        self.sim_delay = Signal((num_bits + 1) * delay)

    def elaborate(self, platform):
        m = Module()

        m.d.comb += [
            Cat(self.out_d, self.out_ovf).eq(
                self.sim_delay[-(self.num_bits+1):])
        ]

        with m.If(self.in_en):
            if self.delay == 0:
                m.d.comb += [
                    self.sim_delay.eq(self.in_a + self.in_b)
                ]
            else:
                m.d.sync += [
                    self.sim_delay.eq(
                        self.sim_delay.rotate_left(self.num_bits+1)),
                    self.sim_delay[:self.num_bits+1].eq(
                        self.in_a + self.in_b),
                ]

        return m
```

`Signal` is array of bits
Python indexing supported

Amaranth – example (2)

```
1   from amaranth import *
1
2
3   class AdderSync(Elaboratable):
4       def __init__(self, num_bits, delay=1):
5           self.num_bits = num_bits
6           self.delay = delay
7
8           assert delay >= 0
9
10          self.in_a = Signal(num_bits)
11          self.in_b = Signal(num_bits)
12          self.in_rst = Signal(1, reset_less=True)
13          self.in_en = Signal(1)
14          self.out_d = Signal(num_bits)
15          self.out_ovf = Signal(1)
16
17          if delay == 0:
18              delay = 1
19          self.sim_delay = Signal((num_bits + 1) * delay)
```

```
21      def elaborate(self, platform):
22          m = Module()
23
24          m.d.comb += [
25              Cat(self.out_d, self.out_ovf).eq(
26                  self.sim_delay[-(self.num_bits+1):])
27          ]
28
29          with m.If(self.in_en):
30              if self.delay == 0:
31                  m.d.comb += [
32                      self.sim_delay.eq(self.in_a + self.in_b)
33                  ]
34              else:
35                  m.d.sync += [
36                      self.sim_delay.eq(
37                          self.sim_delay.rotate_left(self.num_bits+1)),
38                      self.sim_delay[:self.num_bits+1].eq(
39                          self.in_a + self.in_b),
40                  ]
41
42          return m
```

Amaranth – constructs

- Runtime branch

```
with m.If(condition):
    m.d.sync += []; m.d.comb += [];
with m.Elif(condition):
    m.d.sync += []; m.d.comb += [];
with m.Else():
    m.d.sync += []; m.d.comb += [];
```

- Compile time branch

  Python `if-elif-else`

Amaranth – example (2)

```
from amaranth import *


class AdderSync(Elaboratable):
    def __init__(self, num_bits, delay=1):
        self.num_bits = num_bits
        self.delay = delay

        assert delay >= 0

        self.in_a = Signal(num_bits)
        self.in_b = Signal(num_bits)
        self.in_rst = Signal(1, reset_less=True)
        self.in_en = Signal(1)
        self.out_d = Signal(num_bits)
        self.out_ovf = Signal(1)

        if delay == 0:
            delay = 1
        self.sim_delay = Signal((num_bits + 1) * delay)
```

```
    def elaborate(self, platform):
        m = Module()

        m.d.comb += [
            Cat(self.out_d, self.out_ovf).eq(
                self.sim_delay[-(self.num_bits+1):])
        ]

        with m.If(self.in_en):
            if self.delay == 0:
                m.d.comb += [
                    self.sim_delay.eq(self.in_a + self.in_b)
                ]
            else:
                m.d.sync += [
                    self.sim_delay.eq(
                        self.sim_delay.rotate_left(self.num_bits+1)),
                    self.sim_delay[:self.num_bits+1].eq(
                        self.in_a + self.in_b),
                ]

        return m
```

Amaranth – example (2)

```python
from amaranth import *


class AdderSync(Elaboratable):
    def __init__(self, num_bits, delay=1):
        self.num_bits = num_bits
        self.delay = delay

        assert delay >= 0

        self.in_a = Signal(num_bits)
        self.in_b = Signal(num_bits)
        self.in_rst = Signal(1, reset_less=True)
        self.in_en = Signal(1)
        self.out_d = Signal(num_bits)
        self.out_ovf = Signal(1)

        if delay == 0:
            delay = 1
        self.sim_delay = Signal((num_bits + 1) * delay)
```

```python
    def elaborate(self, platform):
        m = Module()

        m.d.comb += [
            Cat(self.out_d, self.out_ovf).eq(
                self.sim_delay[-(self.num_bits+1):])
        ]

        with m.If(self.in_en):
            if self.delay == 0:
                m.d.comb += [
                    self.sim_delay.eq(self.in_a + self.in_b)
                ]
            else:
                m.d.sync += [
                    self.sim_delay.eq(
                        self.sim_delay.rotate_left(self.num_bits+1)),
                    self.sim_delay[:self.num_bits+1].eq(
                        self.in_a + self.in_b),
                ]

        return m
```

Output on left (MSB)

Pass right to left
input to output

→ Simulate pipelined adder

Input on right (LSB)

Amaranth – example (2)

```python
if __name__ == '__main__':
    num_bits = 4
    delay = 3

    dut = AdderSync(num_bits, delay=delay)
    dut = ResetInserter(dut.in_rst)(dut)

    from amaranth.sim import Simulator
    from collections import deque

    def test_case(dut, in_a, in_b, dout, ovf, in_rst=0, in_en=1):
        yield dut.in_rst.eq(in_rst)
        yield dut.in_en.eq(in_en)
        yield dut.in_a.eq(in_a)
        yield dut.in_b.eq(in_b)
        yield
        assert (yield dut.out_d == dout)
        assert (yield dut.out_ovf == ovf)

    mask = 0x0001
    for i in range(num_bits - 1):
        mask |= (mask << 1)
```

```python
    def bench():
        dout_history = deque()
        ovf_history = deque()

        for _ in range(delay):
            dout_history.append(0)
            ovf_history.append(0)

        for i in range(2 ** num_bits):
            for j in range(2 ** num_bits):
                added = i + j
                dout = added & mask
                ovf = (added & (0x0001 << num_bits)) >> num_bits
                dout_history.append(dout)
                ovf_history.append(ovf)

                dout = dout_history.popleft()
                ovf = ovf_history.popleft()
                try:
                    yield from test_case(
                        dut, i, j, dout, ovf)
                except AssertionError:
                    print(i, j, dout, ovf)
```

Amaranth – example (2)

`in_rst` works as reset signal without any of your code for synchronous circuit use `reset` value of `Signal`

```python
if __name__ == '__main__':
    num_bits = 4
    delay = 3

    dut = AdderSync(num_bits, delay=delay)
    dut = ResetInserter(dut.in_rst)(dut)

    from amaranth.sim import Simulator
    from collections import deque

    def test_case(dut, in_a, in_b, dout, ovf, in_rst=0, in_en=1):
        yield dut.in_rst.eq(in_rst)
        yield dut.in_en.eq(in_en)
        yield dut.in_a.eq(in_a)
        yield dut.in_b.eq(in_b)
        yield
        assert (yield dut.out_d == dout)
        assert (yield dut.out_ovf == ovf)

    mask = 0x0001
    for i in range(num_bits - 1):
        mask |= (mask << 1)

    def bench():
        dout_history = deque()
        ovf_history = deque()

        for _ in range(delay):
            dout_history.append(0)
            ovf_history.append(0)

        for i in range(2 ** num_bits):
            for j in range(2 ** num_bits):
                added = i + j
                dout = added & mask
                ovf = (added & (0x0001 << num_bits)) >> num_bits
                dout_history.append(dout)
                ovf_history.append(ovf)

                dout = dout_history.popleft()
                ovf = ovf_history.popleft()
                try:
                    yield from test_case(
                        dut, i, j, dout, ovf)
                except AssertionError:
                    print(i, j, dout, ovf)
```

Amaranth – example (2)

```python
if __name__ == '__main__':
    num_bits = 4
    delay = 3

    dut = AdderSync(num_bits, delay=delay)
    dut = ResetInserter(dut.in_rst)(dut)

    from amaranth.sim import Simulator
    from collections import deque

    def test_case(dut, in_a, in_b, dout, ovf, in_rst=0, in_en=1):
        yield dut.in_rst.eq(in_rst)
        yield dut.in_en.eq(in_en)
        yield dut.in_a.eq(in_a)
        yield dut.in_b.eq(in_b)
        yield
        assert (yield dut.out_d == dout)
        assert (yield dut.out_ovf == ovf)

    mask = 0x0001
    for i in range(num_bits - 1):
        mask |= (mask << 1)
```

use any python library in your test code (deque used for simulated delay)

```python
    def bench():
        dout_history = deque()
        ovf_history = deque()

        for _ in range(delay):
            dout_history.append(0)
            ovf_history.append(0)

        for i in range(2 ** num_bits):
            for j in range(2 ** num_bits):
                added = i + j
                dout = added & mask
                ovf = (added & (0x0001 << num_bits)) >> num_bits
                dout_history.append(dout)
                ovf_history.append(ovf)

                dout = dout_history.popleft()
                ovf = ovf_history.popleft()
                try:
                    yield from test_case(
                        dut, i, j, dout, ovf)
                except AssertionError:
                    print(i, j, dout, ovf)
```

Amaranth – example (2)

```python
if __name__ == '__main__':
    num_bits = 4
    delay = 3

    dut = AdderSync(num_bits, delay=delay)
    dut = ResetInserter(dut.in_rst)(dut)

    from amaranth.sim import Simulator
    from collections import deque

    def test_case(dut, in_a, in_b, dout, ovf, in_rst=0, in_en=1):
        yield dut.in_rst.eq(in_rst)
        yield dut.in_en.eq(in_en)
        yield dut.in_a.eq(in_a)
        yield dut.in_b.eq(in_b)
        yield
        assert (yield dut.out_d == dout)
        assert (yield dut.out_ovf == ovf)

    mask = 0x0001
    for i in range(num_bits - 1):
        mask |= (mask << 1)
```

Empty `yield` for clock cycle step for synchronous circuit

```python
    def bench():
        dout_history = deque()
        ovf_history = deque()

        for _ in range(delay):
            dout_history.append(0)
            ovf_history.append(0)

        for i in range(2 ** num_bits):
            for j in range(2 ** num_bits):
                added = i + j
                dout = added & mask
                ovf = (added & (0x0001 << num_bits)) >> num_bits
                dout_history.append(dout)
                ovf_history.append(ovf)

                dout = dout_history.popleft()
                ovf = ovf_history.popleft()
                try:
                    yield from test_case(
                        dut, i, j, dout, ovf)
                except AssertionError:
                    print(i, j, dout, ovf)
```

Amaranth – example (2)

```
48      from pathlib import Path
49      p = Path(__file__)
50
51      sim = Simulator(dut)
52      sim.add_clock(1e-6)
53      sim.add_sync_process(bench)
54      with open(p.with_suffix('.vcd'), 'w') as f:
55          with sim.write_vcd(f):
56              sim.run()
57
58      from amaranth.back import verilog
59      top = AdderSync(num_bits, delay=delay)
60      with open(p.with_suffix('.v'), 'w') as f:
61          f.write(
62              verilog.convert(
63                  top, ports=[
64                      top.in_en, top.in_a, top.in_b, top.out_d, top.out_ovf]))
65
```

For synchronous circuit

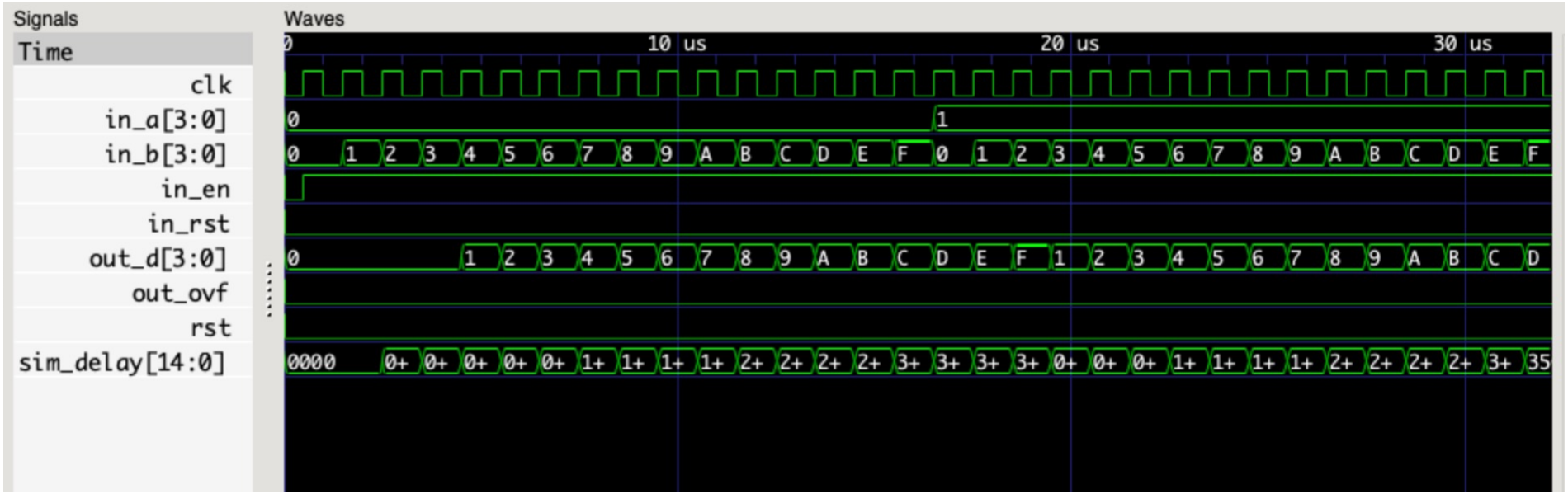
Amaranth – example (2)

```
48      from pathlib import Path
49      p = Path(__file__)
50
51      sim = Simulator(dut)
52      sim.add_clock(1e-6)
53      sim.add_sync_process(bench)
54      with open(p.with_suffix('.vcd'), 'w') as f:
55          with sim.write_vcd(f):
56              sim.run()
57
58      from amaranth.back import verilog
59      top = AdderSync(num_bits, delay=delay)
60      with open(p.with_suffix('.v'), 'w') as f:
61          f.write(
62              verilog.convert(
63                  top, ports=[
64                      top.in_en, top.in_a, top.in_b, top.out_d, top.out_ovf]))
65
```

NOTE no `in_rst`

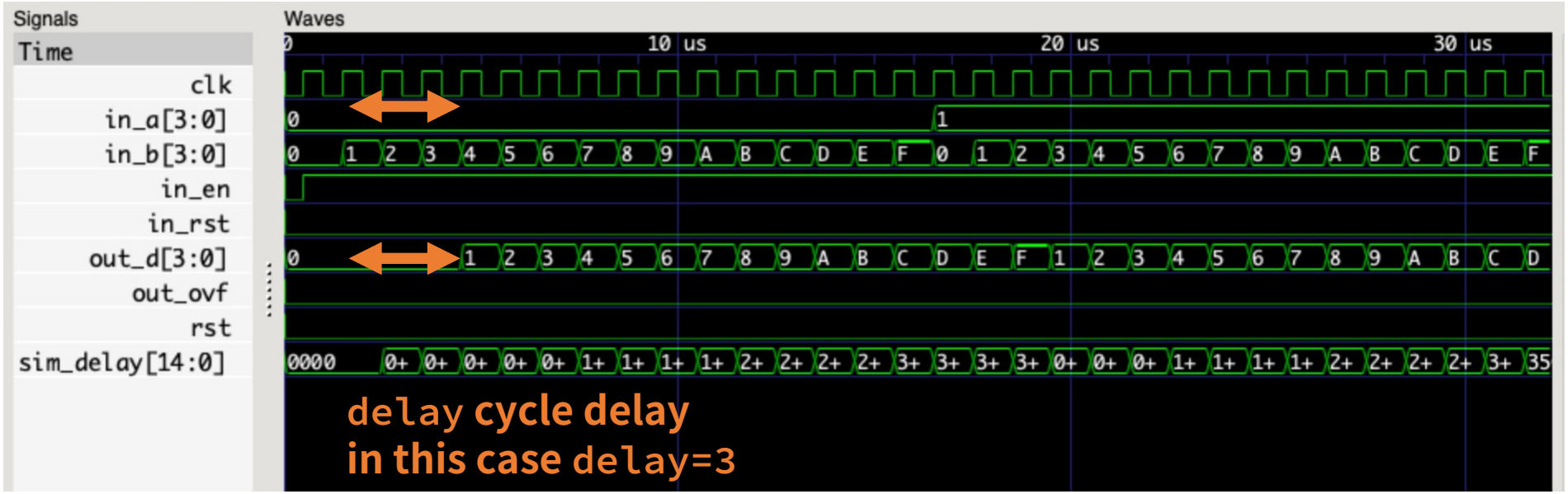
Amaranth – example (2)

- Resulting waveform **`adder_sync.vcd`**



Amaranth – example (2)

- Resulting waveform **`adder_sync.vcd`**



Amaranth – example (2)

- Generated **`adder_sync.v`** (comments omitted)

```
/* Generated by Amaranth Yosys 0.25 (PyPI ver 0.25.0.0.post67, g
e02b7f64b) */

(* \amaranth.hierarchy  = "top" *)
(* top =  1  *)
(* generator = "Amaranth" *)
module top(in_a, in_b, out_d, out_ovf, clk, rst, in_en);
  reg \$auto$verilog_backend.cc:2083:dump_module$2  = 0;
  wire [4:0] \$1 ;
  input clk;
  wire clk;
  input [3:0] in_a;
  wire [3:0] in_a;
  input [3:0] in_b;
  wire [3:0] in_b;
  input in_en;
  wire in_en;
  output [3:0] out_d;
  wire [3:0] out_d;
  output out_ovf;
  wire out_ovf;
  input rst;
  wire rst;
  reg [14:0] sim_delay = 15'h0000;
  reg [14:0] \sim_delay$next ;
  assign \$1  = in_a + in_b;
  always @(posedge clk)
    sim_delay <= \sim_delay$next ;
  always @* begin
    if (\$auto$verilog_backend.cc:2083:dump_module$2 ) begin end
    \sim_delay$next  = sim_delay;
    casez (in_en)
      1'h1:
        begin
          \sim_delay$next [14:5] = sim_delay[9:0];
          \sim_delay$next [4:0] = \$1 ;
        end
    endcase
    casez (rst)
      1'h1:
          \sim_delay$next  = 15'h0000;
    endcase
  end
  assign { out_ovf, out_d } = sim_delay[14:10];
endmodule
```

Amaranth – example (2)

- Generated **`adder_sync.v`** (comments omitted)

```
/* Generated by Amaranth Yosys 0.25 (PyPI ver 0.25.0.0.post67, g
e02b7f64b) */

(* \amaranth.hierarchy  = "top" *)
(* top =  1  *)
(* generator = "Amaranth" *)
module top(in_a, in_b, out_d, out_ovf, clk, rst, in_en);
  reg \$auto$verilog_backend.cc:2083:dump_module$2  = 0;
  wire [4:0] \$1 ;
  input clk;
  wire clk;
  input [3:0] in_a;
  wire [3:0] in_a;
  input [3:0] in_b;
  wire [3:0] in_b;
  input in_en;
  wire in_en;
  output [3:0] out_d;
  wire [3:0] out_d;
  output out_ovf;
  wire out_ovf;
  input rst;
  wire rst;
  reg [14:0] sim_delay = 15'h0000;
  reg [14:0] \sim_delay$next ;
  assign \$1  = in_a + in_b;
  always @(posedge clk)
    sim_delay <= \sim_delay$next ;
  always @* begin
    if (\$auto$verilog_backend.cc:2083:dump_module$2 ) begin end
    \sim_delay$next  = sim_delay;
    casez (in_en)
      1'h1:
        begin
          \sim_delay$next [14:5] = sim_delay[9:0];
          \sim_delay$next [4:0] = \$1 ;
        end
    endcase
    casez (rst)
      1'h1:
          \sim_delay$next  = 15'h0000;
    endcase
  end
  assign { out_ovf, out_d } = sim_delay[14:10];
endmodule
```

Synchronous circuit
→ auto-generated
`clk, rst`

Synchronous circuit
→ auto-generated
`rst` impl.

Q&A on example (2)

Amaranth – example (3)

```python
from amaranth import *
from functools import reduce


class TestArray(Elaboratable):
    def __init__(self, len_arr):
        self.len_arr = len_arr

        self.in_bool = Array([Signal(1, name=f"in_bool_{i}")
                              for i in range(len_arr)])
        self.out_bool = Signal(1)

    def elaborate(self, platform):
        m = Module()

        m.d.comb += [
            self.out_bool.eq(
                reduce(lambda acc, cur: acc & cur, self.in_bool, 1)
            )
        ]

        return m
```

```python
if __name__ == '__main__':
    len_arr = 3
    dut = TestArray(len_arr=len_arr)

    from amaranth.sim import Simulator, Delay, Settle
    import numpy as np

    def test_case(dut, in_bool):
        for i, b in enumerate(in_bool):
            yield dut.in_bool[i].eq(bool(b))
        yield Settle()
        yield Delay(1e-6)

    def bench():
        for i in range(2 ** len_arr):
            yield from test_case(
                dut, np.random.randint(low=0, high=2, size=len_arr))
```

Amaranth – example (3)

```python
from amaranth import *
from functools import reduce


class TestArray(Elaboratable):
    def __init__(self, len_arr):
        self.len_arr = len_arr

        self.in_bool = Array([Signal(1, name=f"in_bool_{i}")
                              for i in range(len_arr)])
        self.out_bool = Signal(1)

    def elaborate(self, platform):
        m = Module()

        m.d.comb += [
            self.out_bool.eq(
                reduce(lambda acc, cur: acc & cur, self.in_bool, 1)
            )
        ]

        return m
```

input `Array`

```python
if __name__ == '__main__':
    len_arr = 3
    dut = TestArray(len_arr=len_arr)

    from amaranth.sim import Simulator, Delay, Settle
    import numpy as np

    def test_case(dut, in_bool):
        for i, b in enumerate(in_bool):
            yield dut.in_bool[i].eq(bool(b))
        yield Settle()
        yield Delay(1e-6)

    def bench():
        for i in range(2 ** len_arr):
            yield from test_case(
                dut, np.random.randint(low=0, high=2, size=len_arr))
```

Amaranth – example (3)

```python
from amaranth import *
from functools import reduce


class TestArray(Elaboratable):
    def __init__(self, len_arr):
        self.len_arr = len_arr

        self.in_bool = Array([Signal(1, name=f"in_bool_{i}")
                              for i in range(len_arr)])
        self.out_bool = Signal(1)

    def elaborate(self, platform):
        m = Module()

        m.d.comb += [
            self.out_bool.eq(
                reduce(lambda acc, cur: acc & cur, self.in_bool, 1)
            )
        ]

        return m
```

Specify `Signal` name
Otherwise, it will be
\Signal$0, \Signal$1, ...

```python
if __name__ == '__main__':
    len_arr = 3
    dut = TestArray(len_arr=len_arr)

    from amaranth.sim import Simulator, Delay, Settle
    import numpy as np

    def test_case(dut, in_bool):
        for i, b in enumerate(in_bool):
            yield dut.in_bool[i].eq(bool(b))
        yield Settle()
        yield Delay(1e-6)

    def bench():
        for i in range(2 ** len_arr):
            yield from test_case(
                dut, np.random.randint(low=0, high=2, size=len_arr))
```

Amaranth – example (3)

```python
from amaranth import *
from functools import reduce


class TestArray(Elaboratable):
    def __init__(self, len_arr):
        self.len_arr = len_arr

        self.in_bool = Array([Signal(1, name=f"in_bool_{i}")
                              for i in range(len_arr)])
        self.out_bool = Signal(1)

    def elaborate(self, platform):
        m = Module()

        m.d.comb += [
            self.out_bool.eq(
                reduce(lambda acc, cur: acc & cur, self.in_bool, 1)
            )
        ]

        return m
```

Using `reduce` as logic

Access `Array` by indexing

```python
if __name__ == '__main__':
    len_arr = 3
    dut = TestArray(len_arr=len_arr)

    from amaranth.sim import Simulator, Delay, Settle
    import numpy as np

    def test_case(dut, in_bool):
        for i, b in enumerate(in_bool):
            yield dut.in_bool[i].eq(bool(b))
        yield Settle()
        yield Delay(1e-6)

    def bench():
        for i in range(2 ** len_arr):
            yield from test_case(
                dut, np.random.randint(low=0, high=2, size=len_arr))
```

Amaranth – example (3)

```python
from amaranth import *
from functools import reduce


class TestArray(Elaboratable):
    def __init__(self, len_arr):
        self.len_arr = len_arr

        self.in_bool = Array([Signal(1, name=f"in_bool_{i}")
                              for i in range(len_arr)])
        self.out_bool = Signal(1)

    def elaborate(self, platform):
        m = Module()

        m.d.comb += [
            self.out_bool.eq(
                reduce(lambda acc, cur: acc & cur, self.in_bool, 1)
            )
        ]

        return m
```

```python
if __name__ == '__main__':
    len_arr = 3
    dut = TestArray(len_arr=len_arr)

    from amaranth.sim import Simulator, Delay, Settle
    import numpy as np

    def test_case(dut, in_bool):
        for i, b in enumerate(in_bool):
            yield dut.in_bool[i].eq(bool(b))
        yield Settle()
        yield Delay(1e-6)

    def bench():
        for i in range(2 ** len_arr):
            yield from test_case(
                dut, np.random.randint(low=0, high=2, size=len_arr))
```

Access `Array` by indexing

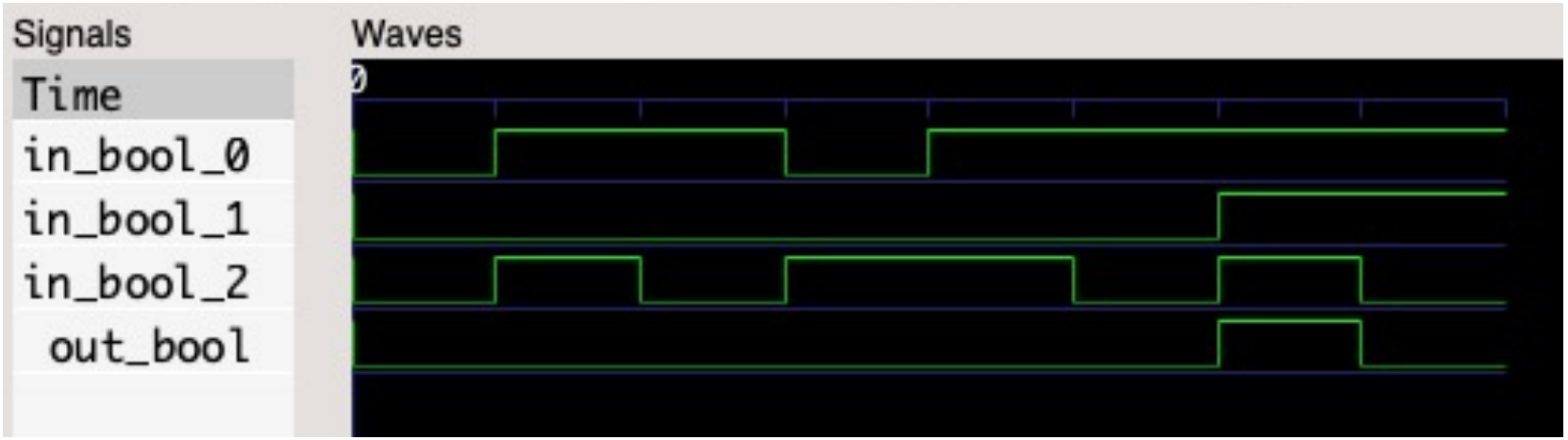
Amaranth – example (3)

```
from amaranth.back import verilog
top = TestArray(len_arr=len_arr)
with open(p.with_suffix('.v'), 'w') as f:
    f.write(
        verilog.convert(
            top,
            ports=[*top.in_bool, top.out_bool]))

```

Need to flatten Array

Amaranth – example (3)

- Resulting waveform `test_array.vcd`



Amaranth – example (3)

- Generated `test_array.v` (comments omitted)

```
1   /* Generated by Amaranth Yosys 0.25 (PyPI ver 0.25.0.0.post67, git sha1
    e02b7f64b) */
1
2   (* \amaranth.hierarchy  = "top" *)
3   (* top =  1  *)
4   (* generator = "Amaranth" *)
5   module top(in_bool_1, in_bool_2, out_bool, in_bool_0);
6     wire \$1 ;
7     wire \$3 ;
8     wire \$5 ;
9     input in_bool_0;
10    wire in_bool_0;
11    input in_bool_1;
12    wire in_bool_1;
13    input in_bool_2;
14    wire in_bool_2;
15    output out_bool;
16    wire out_bool;
17    assign \$3  = \$1  & in_bool_1;
18    assign \$5  = \$3  & in_bool_2;
19    assign out_bool = \$5 ;
20    assign \$1  = in_bool_0;
21  endmodule
22
```

Amaranth – example (3)

- Generated `test_array.v` (comments omitted)

```
/* Generated by Amaranth Yosys 0.25 (PyPI ver 0.25.0.0.post67, git sha1
e02b7f64b) */

(* \amaranth.hierarchy  = "top" *)
(* top =  1  *)
(* generator = "Amaranth" *)
module top(in_bool_1, in_bool_2, out_bool, in_bool_0);
  wire \$1 ;
  wire \$3 ;
  wire \$5 ;
  input in_bool_0;
  wire in_bool_0;
  input in_bool_1;
  wire in_bool_1;
  input in_bool_2;
  wire in_bool_2;
  output out_bool;
  wire out_bool;
  assign \$3  = \$1  & in_bool_1;
  assign \$5  = \$3  & in_bool_2;
  assign out_bool = \$5 ;
  assign \$1  = in_bool_0;
endmodule
```

Array of Signal is flattened

Amaranth – example (3)

- Generated `test_array.v` (comments omitted)

```
1   /* Generated by Amaranth Yosys 0.25 (PyPI ver 0.25.0.0.post67, git sha1
    e02b7f64b) */
1
2   (* \amaranth.hierarchy  = "top" *)
3   (* top =  1  *)
4   (* generator = "Amaranth" *)
5   module top(in_bool_1, in_bool_2, out_bool, in_bool_0);
6     wire \$1 ;
7     wire \$3 ;
8     wire \$5 ;
9     input in_bool_0;
10    wire in_bool_0;
11    input in_bool_1;
12    wire in_bool_1;
13    input in_bool_2;
14    wire in_bool_2;
15    output out_bool;
16    wire out_bool;
17    assign \$3  = \$1  & in_bool_1;
18    assign \$5  = \$3  & in_bool_2;
19    assign out_bool = \$5 ;
20    assign \$1  = in_bool_0;
21  endmodule
22
```

reduce logic

Q&A on example (3)